IEEE Transactions on Quantum Engineering

Received 18 February 2025; revised 2 October 2025; accepted 3 October 2025; date of publication 20 October 2025; date of current version 21 November 2025.

Digital Object Identifier 10.1109/TQE.2025.3623158

Optimized Quantum Circuit Partitioning Across Multiple Quantum Processors

ENEET KAUR, SHAHROOZ POURYOUSEF, HASSAN SHAPOURIAN, JIAPENG ZHAO, MICHAEL KILZER, RAMANA KOMPELLA, AND REZA NEJABATI

Quantum Lab, Cisco Research, Cisco Systems, Inc., Los Angeles, CA 90404 USA

Corresponding author: Eneet Kaur (e-mail: ekaur@cisco.com).

ABSTRACT This article addresses the challenge of scaling quantum computing by employing distributed quantum algorithms across multiple processors. As quantum circuits grow in complexity, efficiently managing the entanglement resources required for communication between quantum processing units (QPUs) becomes increasingly critical. We propose novel methods to optimize entanglement consumption while accounting for network constraints. Our approach leverages graph partitioning techniques to optimize both qubit teleportation and gate teleportation, minimizing the number of Einstein–Podolsky–Rosen pairs required to execute general quantum circuits. In addition, we formulate an integer linear program to further reduce entanglement requirements by mapping the logical resources of partitioned circuits to the physical constraints of the underlying quantum network. Finally, we analyze the entanglement cost for implementing the quantum Fourier transform across multiple QPUs, employing an approach that exploits the circuit's structure to minimize total entanglement consumption. These contributions provide a pathway to scalable and resource-efficient distributed quantum computing.

INDEX TERMS Quantum communication, quantum computing.

I. INTRODUCTION

Quantum computing has emerged as a promising candidate for solving problems that are classically intractable. Certain quantum algorithms, such as Shor's algorithm [1] and the Harrow-Hassidim-Lloyd (HHL) algorithm [2], are provably faster than their classical counterparts for specific problems [3]. Applications of quantum algorithms span physical simulation, optimization, and finance [1], [4], [5], [6], [7]. In recent years, monolithic quantum computers—i.e., systems built around a single quantum processor—have shown promising results in sampling problems [8], [9]. However, running quantum algorithms that outperform classical ones on general-purpose tasks will require coherent control over a large number of qubits. Scaling up a monolithic quantum processor to that level introduces significant challenges, including qubit decoherence, cross-talk, high error rates, cooling constraints, or physical layout limitations [10], [11].

Distributed quantum computing (DQC) offers a promising path forward by connecting multiple smaller Quantum processing units (QPUs) via a quantum network, as reflected in the roadmaps of industry leaders, such as IBM [12] and IonQ [13]. One of the key advantages of DQC is modularity: when a single quantum processor hits its scalability limits, additional QPUs can be integrated into the system. However, executing quantum circuits across multiple QPUs requires establishing nonlocal gates, which in turn depends on generating entangled Einstein–Podolsky–Rosen (EPR) pairs over quantum communication links between processors.

A core challenge in DQC is deciding how to allocate qubits across QPUs connected by a quantum network. To enable remote gate execution after qubit assigning, two main approaches can be employed: state teleportation and gate teleportation. Both techniques rely on EPR pair generation and classical communication, but they differ in how they transfer quantum information and apply the desired operation. The goals of qubit allocation are twofold: 1) to minimize the total number of entangled pairs (EPR pairs) required for executing nonlocal gates and 2) to avoid overloading links with low fidelity and limited bandwidth. The difficulty of this task is underscored by the fact that even minimizing EPR usage alone is NP-hard [14], making scalable, topology-aware heuristics crucial for real-world quantum data center architectures.

Several circuit partitioning (CP) heuristics have been proposed for DQC, but many treat gate packing [15], [16], state teleportation, or partitioning granularity as separate concerns. This separation (as explained in detail in Section III)

limits the ability to exploit circuit structure, leading to suboptimal reductions in inter-QPU communication. Moreover, some methods introduce significant computational overhead, which restricts their applicability to large-scale workloads. A unified approach that balances teleportation strategies, leverages gate packing, and maintains tractable runtime would be desirable.

Most existing partitioning techniques also assume uniform, all-to-all QPU connectivity and neglect practical constraints, such as fidelity variation, bandwidth limitations, and hierarchical topologies. These assumptions diverge from real-world architectures, including Clos and Dragonfly networks [17], [18], which provide scalable architectures for emerging quantum data centers.

In this article, we make three key contributions toward efficient distributed execution of quantum circuits in modular architectures. First, we introduce a window-based circuit partitioning (WBCP) algorithm that integrates gate packing with intrawindow state and interwindow gate teleportation. This design combines and extends the fine-grained slicing of some previous works, such as fine grained partitioning (FGP)_relaxed overall extreme exchange (roee) [19] with structured gate-packing techniques introduced or exploited in works, such as [15] and [16], optimizing for gate locality and reducing communication overhead. Second, we formulate an integer linear program (ILP) optimization problem to minimize EPR pair usage over low-fidelity links in an heterogeneous network setup. Given a partitioned circuit, the ILP maps logical communication requirements to the physical network, assigning entanglement resources in a way that avoids overloading unreliable or bandwidth-limited connections. Third, we present an analytical model for estimating EPR requirements in the quantum Fourier transform (QFT), a widely used subroutine in quantum algorithms such as Shor's. While QFT circuits can be partitioned across multiple QPUs, their regular structure makes them particularly amenable to structured communication planning. Inspired by window-based reasoning, we derive a closed-form bound for EPR usage under arbitrary QPU partitioning. This result is useful for compilers: it provides a fast, reusable estimate of communication cost in distributed implementations of QFT, and can be modularly embedded into larger algorithms to enable early-stage resource estimation and topology-aware optimization.

The rest of this article is organized as follows. In Section II, we review the basic concepts and notations used throughout the article. Section III provides an overview of related work in quantum CP. In Section IV, we introduce our WBCP algorithm. Section V presents our method for incorporating network characteristics into the QPU assignment in the network using an ILP. Section VI evaluates the performance of both the WBCP and ILP formulation through numerical experiments. Section VII develops an analytical framework to calculate the quantum communication cost of distributing QFT circuits across QPUs. Finally, Section VIII concludes this article.

II. NOTATION AND BASICS

A quantum circuit usually consists of a sequence of one-qubit and two-qubit gates. One-qubit gates operate on individual qubits, performing operations, such as rotations or phase shifts. Two-qubit gates, such as CNOT gates, involve entangling operations between pairs of qubits. In this work, we assume that the input quantum circuit is already decomposed into these basic gate primitives.

Graph and CP: Graph partitioning is a classical problem where the goal is to divide the nodes of a graph into disjoint subsets (partitions) while minimizing the number or total weight of edges crossing between them. This problem is NP-hard, and a wide range of heuristics have been developed to solve it approximately. Common approaches include the Kernighan–Lin (KL) algorithm [20], multilevel partitioners, such as METIS (a multilevel graph-partitioning library) [21], and evolutionary methods, such as genetic algorithms (GA) [22]. KL iteratively improves a given partition by swapping node pairs to reduce the edge cut, although it can be sensitive to the initial partitioning. METIS, in contrast, scales efficiently to large graphs via coarsening and refinement.

In DQC, two-qubit gates may involve qubits that are allocated to different QPUs after CP. These are referred to as remote gates, and their execution requires entanglement. CP aims to divide a quantum circuit across multiple QPUs such that each subcircuit respects local hardware constraints—such as qubit capacity and connectivity—while minimizing the number of inter-QPU operations. The goal of CP is therefore to reduce the EPR pair demand, while maintaining balanced workloads across QPUs.

To enable remote gate execution, two main approaches are commonly employed: state teleportation and gate teleportation. Both techniques rely on EPR pair generation and classical communication, but they differ in how they transfer quantum information and apply the desired operation. In this article, we assume each QPU is equipped with a set of computation or data qubits used to execute the circuit locally, as well as communication qubits that enable entanglement with other QPUs. While we do not explicitly model the number of communication qubits or perform any qubit scheduling, this quantity can indirectly affect entanglement bandwidth—an aspect captured in our ILP-based link optimization.

State teleportation [23] allows the quantum state of a qubit to be transferred to another QPU using an EPR pair and classical communication. The qubit in state $|\psi\rangle$ interacts with one half of the EPR pair, after which both are measured in the X and Z bases. These outcomes are sent classically, and the receiving QPU can apply corresponding Pauli corrections to recover the original state. In our model, the state is first teleported to a communication qubit on the target QPU and then transferred to a computation qubit for local operations. This enables subsequent gates involving that qubit to be executed locally. See Fig. 1 for an illustration.

Gate teleportation [24], [25] provides an alternative: a nonlocal two qubits gate, such as CNOT, can be applied between two qubits A' and B' residing on separate QPUs without

IEEE Transactions on Quantum Engineering

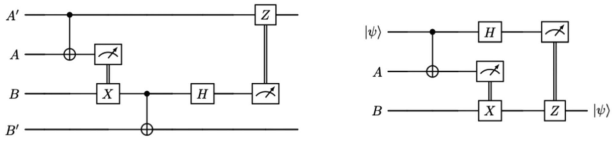

FIGURE 1. (Left) Quantum circuits implementing gate teleportation and (right) state teleportation. Qubits *A* and *B* share an EPR pair. In the left figure, qubits A and *A*′ are on one QPU and B and *B*′ are on another QPU. On the right, A and $|\psi\rangle$ are on one QPU and B is on another QPU.

moving their states. Instead, a shared EPR pair and classical communication are used to perform the gate remotely. The full circuit is shown in Fig. 1.

The choice between gate and state teleportation depends on the structure of the circuit. If a qubit is involved in multiple upcoming gates with qubits located on a remote QPU, then it is often more efficient to teleport its state once and execute those gates locally, rather than repeatedly invoking gate teleportation. This approach minimizes the number of EPR pairs consumed and reduces inter-QPU communication, improving overall resource utilization.

A simplified version of the problem arises when only gate teleportation is used to implement nonlocal CNOT gates, as shown in Algorithm 1. In this case, CP can be directly mapped to a graph partitioning problem. Each qubit q_i in the circuit corresponds to a node in a graph G. For every two qubits gate, such as CNOT gate between qubits q_i and q_j, an edge is added between nodes i and j, with weight equal to the number of such gates. Graph partitioning algorithms can then be used to assign qubits to QPUs in a way that minimizes the total weight of interpartition edges, which corresponds to the number of EPR pairs required.

While this approach is effective for certain circuits, it does not account for state teleportation. As a result, it can overestimate EPR usage in cases where qubit teleportation would be more efficient.

III. PREVIOUS WORK

Early heuristics for CP focused primarily on minimizing qubit teleportation under simplified system models. The work of Zomorodi-Moghadam [26] targets the two-partition case, optimizing for the minimum number of qubit teleportations required to execute a circuit. A hypergraph-based approach was proposed in [15], which involves grouping controlled gates with the same control qubit, followed by partitioning using KaHyPar [27]. Nonlocal gates are implemented using the cat entangler and disentangler method [28]. GA were applied in [29] to minimize teleportation costs. The KL algorithm was used in [30] to perform k-way balanced partitioning for $k \geq 2$, and Daei et al.[31] employed integer linear programming to incorporate both gate and qubit teleportations. More recent work of Nikahd et al. [32] explores entanglement-aware execution strategies and was later extended to homogeneous network settings in [33].

Recent work has explored increasingly sophisticated techniques for CP and qubit assignment. A two-step heuristic was introduced in [34], where qubits are first partitioned across k nearly balanced processors, followed by the use of CAT entanglement to implement nonlocal gates. This approach also proposes a heuristic to minimize the number of qubit migrations required, and was later extended to include teleportation-based execution in [35]. Sundaram et al. [36] used bipartite graphs and dynamic programming to reduce the cost of qubit teleportation.

The approach in [37] maps circuits to modular quantum machines one time slice at a time. It optimizes qubit assignments for each slice using a tunable lookahead window that reduces the expected cost of qubit movement in future slices. This work was extended in [38] to incorporate gate packing [32], with GA used to explore the solution space. Other strategies include the use of the Hungarian algorithm for qubit assignment [39] and deep reinforcement learning for partitioning across multicore quantum systems [40].

While recent heuristics have made progress in reducing EPR pair usage through techniques, such as considering both qubit and gate teleportations and gate packing, these approaches typically perform qubit assignments once for the entire circuit. However, some prior work has shown that fine-grained CP, such as layer-by-layer slicing (e.g., FGP [19] and GA [41]), can reduce nonlocal communication compared to coarse-grained, single-shot approaches. On the other hand, the proposed fine-grained CP methods often overlook opportunities for optimization across groups of adjacent layers or gate windows. Furthermore, many existing approaches assume end-to-end connectivity and homogeneous links in the network.

In this work, we go beyond both one-shot and per-layer partitioning by proposing a window-based strategy that partitions the circuit into variable-length windows. Within each window, we reoptimize qubit-to-QPU assignments based on interqubit interactions and the current qubit assigning in the network. This intermediate granularity enables us to benefit from gate locality and state teleportation opportunities without the computational overhead of per-layer optimization. Compared to two state-of-the-art methods and their baselines, our method attains a more favorable flexibility–scalability tradeoff and can yield lower EPR cost and execution delay. In the following section, we present our WBCP algorithm, which is calibrated with fine-grained CP and gate-packing techniques to reduce nonlocal operations and improve entanglement efficiency.

IV. WINDOW-BASED CP

In this section, we propose a WBCP algorithm to reduce EPR pair consumption in distributed quantum circuit execution. The algorithm takes as input a quantum circuit, the number of QPUs, and the computational capacity of each QPU, along with a tunable window length parameter. It produces a dynamic qubit-to-QPU mapping that evolves over time,

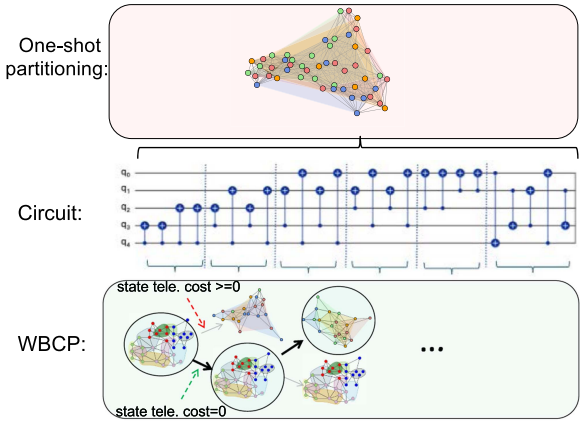

FIGURE 2. **Overview of the WBCP approach compared to one-shot CP approaches, such as KL. The circuit is divided into fixed-length windows. Within each window, qubits are assigned to QPUs using graph partitioning, and nonlocal gates are implemented using gate teleportation. Between windows, qubit assignments can be updated via state teleportation, enabling dynamic reallocation throughout the circuit. In the figure, each subcircuit is associated with two partitionings: one inherited from the previous subcircuit and one computed for the current subcircuit graph. The selected partitioning is indicated by a circle around it.**

Algorithm 1: CP Using Gate Teleportation.

Input: Quantum circuit with qubits $q_1, q_2, \ldots, q_n$ and CNOT gates
Output: Partitioned graph G minimizing inter-partition edge weights
Initialize an empty graph $G = (V, E)$
for each qubit q_i in the circuit **do**
  Add node v_i to V
end for
for each CNOT gate between qubits q_i and q_j **do**
  if edge (v_i, v_j) already exists in E **then**
    Increment the weight of edge (v_i, v_j) by 1
  else
    Add edge (v_i, v_j) to E with weight 1
  end if
end for
Employ a graph partitioning technique to partition graph G into subgraphs $G_1, G_2, \ldots, G_k$ such that the sum of the weights of the edges between the partitions is minimized
Return partitioned graph G

labeling each two-qubit gate as either local or nonlocal based on the required QPU placement.

Unlike static partitioning methods, which assign each qubit to a fixed QPU for the duration of the circuit, such as KL, as shown in Fig. 2, WBCP partitions the circuit into fixed-size windows, each containing a contiguous chunk of two-qubit gates. Within each window, qubits are assigned to QPUs using graph partitioning, and remote gates are typically implemented via gate teleportation. However, when it is more efficient, WBCP can selectively apply state teleportation even within a window to enable local gate execution. Between windows, state teleportation is used more broadly to allow dynamic qubit reassignment. A visual overview is shown in Fig. 2.

In this work, we use a modified version of the KL algorithm that supports unbalanced partition sizes. We also generalize the method to handle more than two partitions through recursive bisection. For each window, qubits are reassigned to QPUs using a locally optimized partitioning step that takes into account the previous window's assignment. Full pseudocode, time complexity, and implementation details are provided in Appendix A.

To discourage excessive qubit movement, we increase the weights of gates between qubits that were previously assigned to the same QPU. This soft bias encourages continuity in qubit placement across windows, without enforcing rigid constraints. A new partition is only adopted if the total EPR cost—accounting for both gate and state teleportations—is lower than reusing the previous one. A high-level description of the algorithm is provided in Algorithm 2.

While our implementation uses a recursive variant of the KL algorithm to partition each window, WBCP is agnostic to the specific partitioning method. Any standard graph partitioning algorithm—such as METIS, spectral partitioning, or flow-based heuristics—can be substituted without modifying the overall structure of the framework. When using KL, we initialize the partition for a given window using the optimal partition from the previous window, which helps reduce qubit movement across windows.

The window length is a key hyperparameter that controls the tradeoff between flexibility and global circuit awareness. Smaller windows allow for finer grained qubit reassignments and more localized optimization, but may miss long-range gate structure. Larger windows capture more global context but reduce the ability to adapt to local variations. In our implementation, we perform a search over a discrete set of window lengths and select the one that minimizes total EPR pair consumption across the evaluated benchmark circuits. A detailed analysis of this tradeoff is presented in Section VI-C.

A. FURTHER OPTIMIZATION

Beyond the core partitioning logic, WBCP supports several optimization modules that further reduce EPR pair usage. We call this version of our algorithm an *Opt_WBCP* algorithm.

1) *Gate packing:* Gate packing groups multiple two-qubit gates that share common qubits within a window, increasing the density of local execution and reducing the need for remote operations. For gate packing, we use the conditions presented in [33, Lemmas 3 and 7]. The gate packing condition checks whether two remote gates acting on the same qubit can be executed using a shared EPR pair, without violating circuit correctness. First, if the gates are directly adjacent, packing is immediately allowed. Otherwise, we examine

Algorithm 2: WBCP.

Inputs: Quantum circuit C, window length w, number of QPUs k
Divide C into subcircuits $\{C_i\}_{i=1}^{\tilde{m}}$, where $\tilde{m} = \lceil T/w \rceil$ and T is the number of two-qubit gates in C
Initialize total entanglement cost $EC \leftarrow 0$
for each subcircuit C_i **do**
  if $i = 1$ **then**
    Run Algorithm 1 on C_1 to obtain initial partition $P_1 = \{P_1^j\}_{j=1}^{k}$
  else
    Construct graph G_i with nodes as qubits in C_i
    for each edge between qubits q_u and q_v in G_i **do**
      if q_u, q_v are in the same partition in P_{i-1} **then**
        Set edge weight $\bar{w}_{uv} = 2 \times \#\text{CNOT}(q_u, q_v)$
      else
        Set edge weight $\bar{w}_{uv} = \#\text{CNOT}(q_u, q_v)$
      end if
    end for
    Partition G_i into k parts to obtain P_{new}
    Compute entanglement costs: $EC_{P_{\text{new}}}^{C_i}$, $EC_{P_{i-1}}^{C_i}$
    Compute number of qubit moves $M = |\text{moved qubits from } P_{i-1} \rightarrow P_{\text{new}}|$
    Update $EC_{P_{\text{new}}}^{C_i} \leftarrow EC_{P_{\text{new}}}^{C_i} + M$
    if $EC_{P_{\text{new}}}^{C_i} \leq EC_{P_{i-1}}^{C_i}$ **then**
      Set $P_i = P_{\text{new}}$
    else
      Set $P_i = P_{i-1}$
    end if
    Add $\min(EC_{P_{\text{new}}}^{C_i}, EC_{P_{i-1}}^{C_i})$ to total EC
  end if
end for
Output: Final qubit-to-QPU mappings $\{P_i\}$, total entanglement cost EC

all intermediate gates between the two candidates and check their effect on the target qubit. If any Hadamard (H) gates are present, then they must appear exactly twice—once to apply and once to undo—otherwise packing is disallowed. For controlled-RZ gates (crz), each gate's rotation angle must be close to π to preserve reversibility; otherwise, packing is rejected. Any single-qubit RZ gate must collectively apply no net rotation on the qubit. Specifically, their summed angle must either be 0 or a multiple of π (i.e., the total angle modulo 2π should be close to 0 or π). If this condition fails, then packing is also disallowed. These checks are essential to ensure that the qubit's quantum state remains suitable for a valid packed CNOT operation and that entanglement reuse does not introduce errors due to intervening operations. We have given the pseudocode of our algorithm for performing gate-packing in a subcircuit in Appendix C.

2) *Intrawindow state teleportation:* This enables temporary teleportation of qubits within a window when gate teleportation is inefficient. This allows the qubits to be moved to a new QPU temporarily and then returned to the original QPU. This contrasts with the dynamic reallocation across windows. The intrawindow state teleportation allows more gates to be executed locally without requiring permanent qubit reassignment. For this, we first check whether a nonlocal two-qubit gate can be bypassed using intrasubcircuit state teleportation by verifying intermediate gate interference. If the two gates are adjacent, then teleportation is trivially allowed. Otherwise, the algorithm iterates through all gates between the two gates and verifies that the teleporting qubit does not interact with qubits in partitions outside of its own or the target partition. Specifically, for each intermediate two-qubit gate, if the teleporting qubit appears and the other qubit belongs to a different partition than either of the two involved, then teleportation is disallowed. If none of these invalid interactions are found, then we permit teleportation across the partition. We have included the pseudocode of the algorithm for this part in Appendix C.
3) *Boundary-aware refinement:* This module applies local adjustments at window edges to reduce cross-window communication. This helps smooth out abrupt partitioning transitions. Specifically, we iteratively reassign boundary qubits to neighboring windows if doing so reduces interwindow EPR cost without violating QPU capacity constraints.

Details for each optimization module are included in Appendixes B, C, and E.

The time complexity of the Opt_WBCP algorithm depends on the number of qubits (n), the circuit size (T), the window size (w), and the number of partitions or QPUs (k). For each window of length w, the algorithm performs partitioning and optimizations. The multiway CP step is based on an extended version of the KL algorithm, which implied that the WBCP for window length w requires $O(\frac{T}{w} n^2 \log k)$. Gate packing and state teleportation optimizations operate on candidate two-qubit gate pairs across different QPUs. In the worst case, the number of candidate pairs is $O(n^2)$, and the optimization cost per window becomes $O(n^2 + w^2)$. Since there are approximately T/w windows, the overall time complexity becomes $O(\frac{T}{w} \cdot (n^2 \log k + n^2 + w^2))$. While the worst case complexity appears high, we note that many of these operations—particularly gate packing and teleportation checks—can be parallelized, keeping runtime manageable. In addition, faster partitioning algorithms can be substituted

to further reduce overhead. Further details of the packing and teleportation steps are provided in Appendix C.

V. NETWORK TOPOLOGY CONSIDERATION

Given the inter-QPU communication demands extracted from CP and a physical network topology with heterogeneous link characteristics—arising from factors, such as limited entanglement resources or variable swapping overhead—we now address the problem of optimally assigning logical QPUs to physical QPUs. This assignment is formulated as an ILP.

The output of WBCP—or any CP scheme—is a logical QPU graph, where edge weights represent the number of EPR pairs required between QPUs over the course of execution. To minimize physical resource usage, we map these logical QPUs onto physical QPUs in the data center, taking into account the heterogeneous quality of interconnects via an ILP formulation.

We formulate the problem of assigning logical QPUs, which is defined by the mapping of the logical QPUs of the circuit to physical QPUs in a quantum data center with heterogeneous link qualities as an ILP.

Let $G' = (V', E')$ represent the logical QPU graph obtained from the circuit, where an edge $j \in E'$ has weight λ_j, denoting the number of EPR pairs required between QPUs $v_1, v_2 \in V'$. Let $G = (V, E)$ be the physical topology of the data center, where an edge $i \in E$ has weight ω_i, representing the cost of establishing entanglement over that physical link.

1) *Variables:*
   a) $y_{uv} \in \{0, 1\}$: Indicates whether logical QPU $u \in V'$ is assigned to physical QPU $v \in V$.
   b) $x_{ij} \in \{0, 1\}$: Indicates whether logical edge $j \in E'$ is mapped to physical edge $i \in E$.
2) *Objective:* Minimize the total weighted EPR usage across the physical topology

$$\min \sum_{i \in E} \sum_{j \in E'} \omega_i \cdot \lambda_j \cdot x_{ij}.$$

3) *Constraints:*
   a) *Vertex mapping constraints:*

$$\sum_{v \in V} y_{uv} = 1 \quad \forall u \in V' \tag{1}$$

$$\sum_{u \in V'} y_{uv} = 1 \quad \forall v \in V. \tag{2}$$

   b) *Edge mapping constraints:*

$$\sum_{j \in E'} x_{ij} = 1 \quad \forall i \in E \tag{3}$$

$$\sum_{i \in E} x_{ij} = 1 \quad \forall j \in E'. \tag{4}$$

   c) *Edge-to-edge consistency constraint (quadratic):* Let edge $i = (u_1, u_2) \in E$, and logical edge $j = (v_1, v_2) \in E'$. Then

$$x_{ij} = y_{u_1 v_1} \cdot y_{u_2 v_2} + y_{u_1 v_2} \cdot y_{u_2 v_1}. \tag{5}$$

To ensure constraint feasibility even when G' is sparse, we pad it to a complete graph by assigning zero EPR weight $\lambda_j = 0$ to noninteracting QPU pairs. This allows all logical edges to participate in the mapping without impacting the optimization cost.

This quadratic constraint ensures that the edge $j \in E'$ can only be mapped to the edge $i \in E$ if the endpoints of j are mapped to the endpoints of i, in either order.

Since standard ILP solvers cannot directly handle quadratic constraints over binary variables, we next show how to linearize (5) using auxiliary variables.

4) *Linearization:* To express the quadratic constraint in (5) using linear constraints, we introduce auxiliary binary variables $z1_{ij}$ and $z2_{ij}$ for each pair (i, j), where $i = (u_1, u_2) \in E$ and $j = (v_1, v_2) \in E'$.
These variables encode the two possible endpoint matchings

$$z1_{ij} = y_{u_1 v_1} \cdot y_{u_2 v_2} \tag{6}$$

$$z2_{ij} = y_{u_1 v_2} \cdot y_{u_2 v_1}. \tag{7}$$

We enforce these products using the following linear constraints:

$$z1_{ij} \le y_{u_1 v_1}, \quad z1_{ij} \le y_{u_2 v_2} \tag{8}$$

$$z1_{ij} \ge y_{u_1 v_1} + y_{u_2 v_2} - 1 \tag{9}$$

$$z2_{ij} \le y_{u_1 v_2}, \quad z2_{ij} \le y_{u_2 v_1} \tag{10}$$

$$z2_{ij} \ge y_{u_1 v_2} + y_{u_2 v_1} - 1. \tag{11}$$

Finally, we replace the quadratic constraint with

$$x_{ij} = z1_{ij} + z2_{ij}. \tag{12}$$

This completes the linearization of the quadratic consistency constraint, making the full formulation suitable for use with standard ILP solvers. For a complexity analysis of this formulation, see Appendix D.

VI. NUMERICAL RESULTS

In this section, we present numerical results evaluating our proposed WBCP algorithm (see Algorithm 2) on a diverse set of benchmark quantum circuits. We compare its performance against three other methods: the baseline KL algorithm, a GA-based partitioning heuristic [41], and FGP_roee [19], two state-of-the-art CP strategies. In this section, we use WBCP to indicate the default WBCP and Opt_WBCP to indicate the optimized WBCP (with gate packing and intrawindow state teleportation).

We note that FGP_roee does not generate results for all circuit configurations. Specifically, it attempts to find a valid qubit assignment at each circuit layer such that all two-qubit gates are executed locally. In circuits with high connectivity or randomized gate placement—such as quantum volume

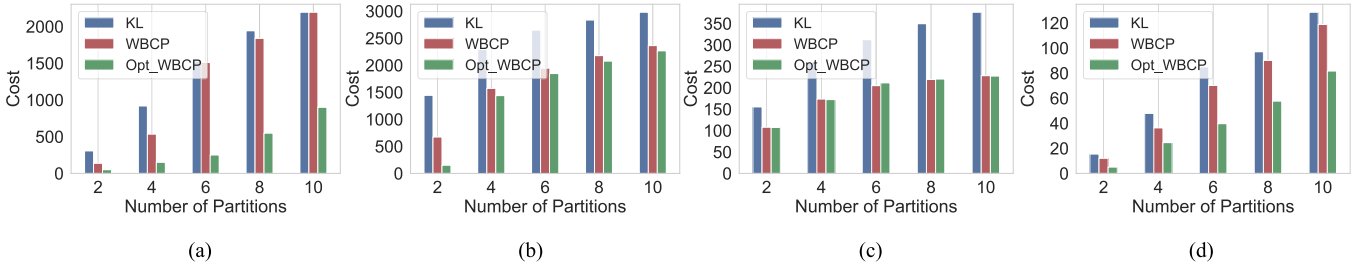

FIGURE 3. EPR cost versus number of QPUs. Opt_WBCP shows lower cost due to gate packing and state teleportation. (a)Circuit: QFT, (b) circuit: QAOA, (c) circuit: QFT, and (d) circuit: Adder.

(QV) circuits—such assignments may not exist. In these cases, FGP_roee either fails to terminate or returns no result within a reasonable timeout, especially when the number of QPUs is large. To ensure a fair comparison, we include FGP_roee only in plots where it successfully completes all data points across the full range of qubit and partition counts.

Our benchmark suite includes the QFT, the quantum approximate optimization algorithm (QAOA) [42], QV circuits [43], the Cuccaro Adder circuit [44], and randomly generated circuits. QFT is a highly structured circuit used in algorithms, such as Shor's algorithm. QAOA is a variational algorithm targeting combinatorial optimization, typically exhibiting a high degree of gate parallelism. QV circuits are designed to benchmark hardware performance with moderate structure, while random circuits stress-tests partitioning heuristics under minimal regularity. These circuits are decomposed into a native gate set {cx, h, rz, crz} using Qiskit's `transpile()` function to ensure compatibility with realistic hardware constraints. For Fig. 5, we use gate set {u, cp} in transpilation as GA algorithm only supports these gates.

In our experiments, we vary both the number of qubits and the number of partitions to assess how each algorithm performs as both the circuit size and number of QPUs increase.

A. EPR COST VERSUS NUMBER OF QPUS

In this experiment, we evaluate how the total EPR cost varies with the number of QPUs. For each quantum circuit in our benchmark set, we vary the number of partitions (i.e., QPUs) from 2 to 8. Our goal is to analyze how the benefits of gate packing and intrawindow state teleportation, introduced in our WBCP algorithm, evolve as the number of QPUs increases, and to compare its performance against KL.

Fig. 3 shows the total EPR cost as a function of the number of partitions (QPUs) for different partitioning algorithms across selected circuits. Each bar in the plot represents one partitioning algorithm, and each point reflects the total number of EPR pairs required for a given number of QPUs. We run the WBCP and Opt_WBCP with different values of window lengths from 50 to the length of the circuit with step size 50 and pick the minimum EPR cost as the algorithm EPR cost. As expected, increasing the number of QPUs generally leads to a higher EPR cost due to more gates becoming nonlocal. However, Opt_WBCP consistently incurs a lower EPR cost compared to KL and default WBCP. This highlights the benefit of window-based optimization and gate packing in preserving gate locality.

B. EPR COST VERSUS NUMBER OF QUBITS

In this experiment, we investigate how the total EPR cost scales with the number of qubits in the circuit. We use our proposed WBCP and Opt_WBCP algorithms along with KL and FGP_roee as baselines, and run each partitioning algorithm on different circuits with varying numbers of qubits while keeping the number of QPUs fixed. For each configuration, we calculate the total number of EPR pairs required to execute the nonlocal gates resulting from the CP.

Fig. 4 shows the total EPR cost as a function of the number of computational qubits for different partitioning algorithms. While all methods show an increase in EPR cost as circuit size increases, Opt_WBCP yields lower EPR cost compared to KL and FGP_roee for QFT and QV circuits, especially in larger circuits. This demonstrates that WBCP's gate packing and teleportation-aware strategy becomes increasingly effective as the circuit grows in size. However, FGP_roee is doing better than Opt_WBCP for QAOA circuits but with higher computational overhead in search for better qubit assigning for each layer. We notice that as we increase the number of qubits in the QFT circuit, the EPR cost stays same or even decreases in Figs. 4 and 5 at some points. The reason is that as the number of qubits increases in QFT circuits compiled by Qiskit, the total number of controlled-phase gates grows quadratically, with many of them involving small rotation angles between distant qubits. Qiskit applies an approximation threshold to eliminate these small-angle gates, significantly reducing the total number of two-qubit operations.

Fig. 5 compares our proposed Opt_WBCP algorithm against three baselines: FGP_roee, GA-based partitioning, and KL. We use the default implementation of the GA algorithm from [45]. The experiment varies the number of qubits for two benchmark circuits: QFT and QV. We note that FGP_roee occasionally fails to produce results, even after several hours. This occurs because FGP_roee attempts to find a valid qubit assignment for every layer of the circuit such that all two-qubit gates are local, which is sometimes

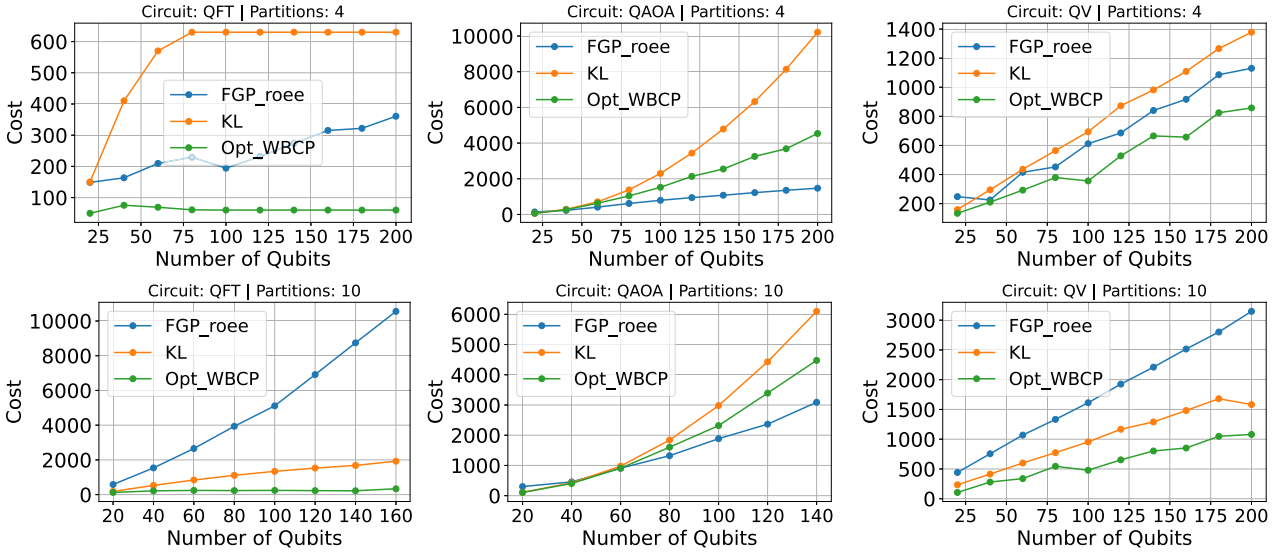

FIGURE 4. EPR cost versus number of qubits for different partitioning strategies. We compare KL, WBCP, Opt_WBCP, and FGP_roee across QFT, QAOA, and QV circuits while fixing the number of QPUs. As circuit size grows, all methods incur higher EPR cost, but Opt_WBCP achieves consistently lower EPR cost for QFT and QV circuits by exploiting gate packing and teleportation-aware optimization. FGP_roee performs slightly better for QAOA circuits.

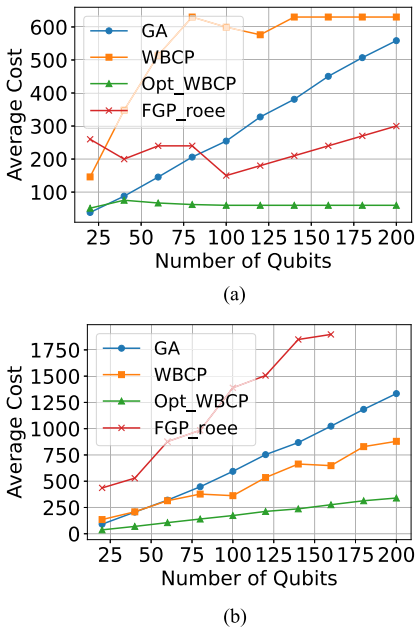

FIGURE 5. Performance comparison of WBCP, Opt_WBCP, FGP_roee [37], and GA-based [41] partitioning on QFT and QV circuits with four QPUs. Opt_WBCP yields the lowest EPR cost across most circuit sizes, highlighting the benefit of combining window-based partitioning with gate packing and intrawindow teleportation. FGP_roee occasionally fails to find a valid solution at larger qubit counts, emphasizing the scalability advantage of theproposed method. (a) Circuit: QFT (b) Circuit: QV.

infeasible. For instance, we were unable to obtain results for QV circuits with more than 160 qubits on four QPUs, as FGP_roee could not find a valid mapping. Across both circuit types and qubit counts, our Opt_WBCP consistently outperforms the other approaches, achieving lower communication cost while maintaining practical runtime and scalability.

C. IMPACT OF WINDOW LENGTH ON WBCP

In this experiment, we evaluate the sensitivity of our WBCP and Opt_WBCP to the choice of window length. Specifically, we analyze how the total EPR cost changes as a function of the window length used during partitioning. We run WBCP on a variety of circuits—including QAOA, QFT, and QV circuits—while sweeping over different fixed window lengths. We compare the performance of WBCP with gate packing against two alternatives: gate packing applied globally without window-based slicing, and the baseline KL algorithm. The gate packing applied globally without window-based slicing is for the point we run the Opt_WBCP algorithm with the highest window length which is the entire circuit. We only consider gate-packing in our WBCP here.

The goal of this experiment is to demonstrate that gate packing is significantly more effective when applied within a window-based slicing framework, as opposed to globally across the entire circuit. Our results, as presented in Fig. 6, show that even fixed-size windows offer near-optimal performance with substantially lower computational overhead compared to more complex strategies (check our results in Appendix E). In particular, our numerical results indicate that the optimal window length follows predictable patterns depending on the circuit type. For instance, in QAOA circuits, the EPR cost-minimizing window length is typically found early in the sweep (e.g., around 400 gates), whereas for QFT circuits, the optimal window length tends to span nearly the entire circuit depth, aligning with their highly structured, layered gate pattern. The limited improvement of Opt_WBCP over WBCP in the QV circuit stems from the random structure of QV circuits. Since the gate placements and interactions do not follow a structured pattern, there is little redundancy or locality for the gate packing subroutine

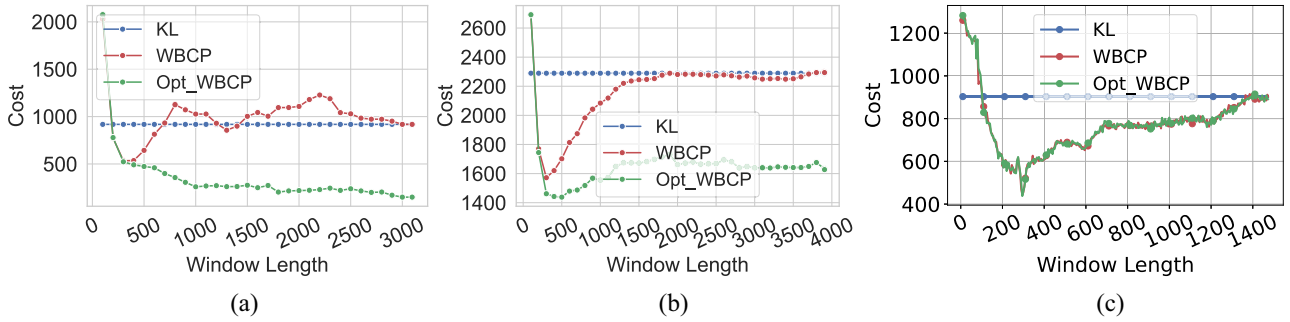

FIGURE 6. EPR cost versus window length for QFT, QAOA, and QV circuits with 100 qubits and eight QPUs. Results show that Opt_WBCP outperforms WBCP and KL, and that the optimal window size varies by circuit type. (a)Circuit: QFT, (b) circuit: QAOA, and (c) circuit: QV.

TABLE 1. Cost Comparison Across Benchmark Circuits for KL, WBCP, Opt_WBCP, and FGP_roee

Circuit type	KL	WBCP	Opt_WBCP	FGP_roee
Adder (118)	16.10	15.38	13.74	90.00
BV (140)	3.00	3.00	1.00	4.06
BV (280)	13.00	13.00	1.00	4.06
Cat (130)	1.73	1.73	1.49	4.00
Cat (260)	1.53	1.53	1.37	4.00
DNN (16)	18.56	14.90	14.22	10.23
Ising (420)	2.64	2.16	2.12	4.06
Square_root (18)	130.00	129.48	85.64	190.00
Wstate (118)	2.96	2.96	2.56	118.00
Wstate (36)	3.40	3.40	2.80	36.00
Wstate (380)	2.0	2.0	2.0	380.00

to exploit. As a result, opportunities for gate packing and boundary refinement are minimal. In such cases, Opt_WBCP behaves very similarly to the baseline WBCP, leading to only marginal performance gains.

In addition, we investigated a dynamic windowing strategy, where the circuit is partitioned into subcircuits of varying window lengths. While this approach allows more flexibility in adapting to circuit structure, our experimental results—provided in Appendix E—show that it does not offer significant improvement over fixed-size windows and incurs higher overhead (because of searching for different lengths of subcircuits). These findings suggest that fixed-size windows seems to be an effective balance between partitioning quality and computational cost, making them a practical and scalable choice for realistic DQC workloads. For more detail on this check Fig. 11 in Appendix E.

Finally, we compare the performance of WBCP and optimized WBCP for different benchmark circuits in Table 1. For each circuit, we run the CP algorithm multiple times and compute and report the average EPR cost. In this table, Bernstein–Vazirani (BV) circuits are oracle-based and benchmark linear structure extraction via a single-query algorithm. Cat and W-state circuits generate entangled states (GHZ and W states, respectively), which are fundamental for distributed quantum protocols and entanglement benchmarking. DNN-inspired circuits reflect parameterized layers used in quantum machine learning and variational algorithms. Ising circuits simulate Ising-type Hamiltonians and are representative of QAOA-style optimization workloads. Finally, square root circuits implement arithmetic operations used in modular computation, often relevant in number-theoretic quantum algorithms, such as Shor's. For all experiments in this table, we partition each circuit across two QPUs, and the number in parentheses indicates the total number of qubits in that circuit instance.

D. ILP NUMERICAL RESULTS

In this section, we compare the solution proposed in Section V to the approach of randomly assigning QPUs in a quantum data center. To model the physical connectivity of the quantum data center, we assign weights ω_i to the edges of the graph G, where each weight represents the infidelity of EPR pair generation along that link. These weights are drawn from a beta distribution with parameters $\alpha = \beta$, applied to an upper triangular matrix to reflect symmetric connectivity between QPUs, as is typically expected in a data center setting. By varying the value of $\beta > 0$, we control the skewness of the distribution: small values produce highly variable link qualities, while large values yield near-uniform link quality across the network. This captures hardware asymmetries arising from differences in fiber length, number of swapping in data centers, switch depth, or calibration quality between QPU connections.

We generate the logical EPR demand graph G', where each edge weight λ_i represents the number of EPR pairs required between pairs of logical QPUs. The weights are sampled using a parameterized exponential transformation that biases values toward either end of the allowed range. By adjusting a bias factor, we can control how sharply the distribution concentrates around low and high values. This allows us to explore both uniform and highly imbalanced communication patterns within a single modeling framework.

If both ω_i (link fidelity) and λ_i (EPR demand) are uniformly or randomly distributed, then aligning logical and physical topologies becomes a trivial task—and the optimizer offers no real advantage over a random assignment. Our experimental setup intentionally introduces structure into both graphs to evaluate the ILP's ability to exploit alignment opportunities.

To evaluate how skewness in EPR demand affects the benefit of ILP optimization, we fix the physical topology graph G using a Beta distribution with parameters $\alpha = \beta \in$

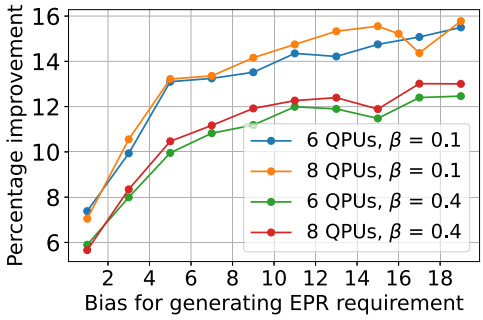

FIGURE 7. We plot the percentage improvement of the ILP objective function relative to random QPU assignment. The *x*-axis shows the bias factor used to generate EPR demands (weights λ_i), while different curves correspond to different values of the Beta distribution parameter β used to model link infidelity (weights ω_i). Higher improvement indicates that the optimizer is better able to exploit the nonuniformity in both communication demands and physical link quality.

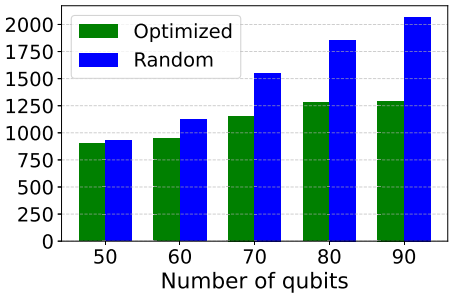

FIGURE 8. We compare the value of the objective function for a random QPU assignment with that of an optimized QPU assignment for QFT circuits.

{0.1, 0.4}, and vary the bias factor that controls the skew in logical communication demands (i.e., edge weights in G'). Fig. 7 reports the percentage improvement in communication cost achieved by the ILP over a random QPU assignment, plotted against this bias factor. As the communication demand becomes increasingly imbalanced, the ILP is able to reduce communication cost by better aligning high-demand links with lower cost physical connections. This trend confirms our earlier observation: optimization is only meaningful when there is structure to exploit. Fig. 8 presents a case study on QFT circuits across varying qubit counts. The EPR demands are extracted using the gate-based graph partitioning algorithm, and the resulting distribution reflects the compiled structure of the circuit. Specifically, QFT circuits compiled by Qiskit apply an approximation threshold that eliminates small-angle controlled-phase rotation gates, which in turn introduces an imbalance in the number of remote gates between different QPU pairs. When combined with a skewed physical topology, this discrepancy enables the ILP to improve communication cost relative to a random assignment.

VII. METHODS FOR IMPLEMENTING DISTRIBUTED QFT

The QFT [46] is a cornerstone component in many quantum algorithms, including Shor's factoring algorithm, quantum phase estimation, and various applications in quantum chemistry and simulation. Due to its structured nature, QFT circuits offer an opportunity to derive analytical bounds for distributed execution across multiple QPUs.

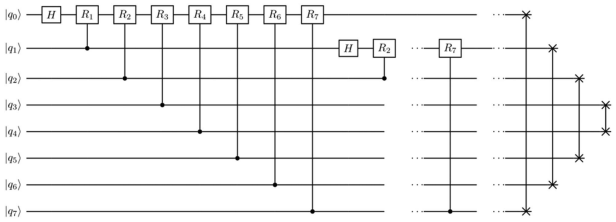

FIGURE 9. Implementation of QFT circuit on eight qubits.

In this section, we present methods for implementing QFT in a distributed setting. While the heuristic approach in Algorithm 2 is an option, we leverage the inherent structure of the QFT circuit to reduce the number of EPR pairs. Due to its regular structure and predictable gate layout, the QFT can be studied analytically in distributed settings. Since QFT frequently appears as a modular subroutine in larger quantum algorithms, an optimized distributed version can be reused directly in a plug-and-play manner.

For initial work for distributed QFT for two partitions see [47] and [48]. Neumann et al. [48] calculated the number of required EPR pairs to be $n/2$ for a QFT(n), where n refers to the number of qubits in the circuit, without implementing the swaps explicitly, and instead relying on logical bookkeeping. In this work, they used the concept of grouping multiple two-qubit gates and one EPR pair to implement this group of gates, also called as cat-entangler and cat-disentangler [49].

We outline a method to implement QFT(n) on two processors and extend it to handle the case of $k \geq 3$ processors. At the culmination of our implementation, the qubit positions are such that the swap gates typically required at the end of the QFT circuit are no longer necessary. Our approach relies on qubit teleportation as well as the cat-entangler and cat-disentangler techniques introduced in [49].

A QFT circuit has the following structure: For all i in $[0, n-1]$, apply a Hadamard gate H on the ith qubit, followed by controlled rotations (CR) with i as the target qubit and $j > i$ as the control qubit. Finally, swap qubits i and $n-i-1$. See Fig. 9 for a QFT circuit on eight qubits.

We assume that the hardware natively supports CR gates. If this is not the case, then they can be implemented using an ancilla-based approach or decomposed into two CX gates and a single-qubit rotation.

A. QFT DISTRIBUTED ON TWO QPUS

In this section, we distribute QFT(n) to two processors. We assume that both the quantum processors have $\lceil \frac{n}{2} \rceil + 1$ computational qubits. The processors are also equipped with one communication qubit each. The algorithm laid out can be

IEEE Transactions on Quantum Engineering

Algorithm 3: QFT Distributed on Two QPUs.

Initialize QPU1 with qubits $[1, \frac{n}{2}]$
Initialize QPU2 with qubits $[\frac{n}{2}+1, n]$
For all i in $[1, \frac{n}{2}]$, apply a H gate on the i^{th} qubit, followed by the controlled rotations with i as the target qubit and $\bar{k} \geq i$ as the control qubit, where $\bar{k} \in [i, \frac{n}{2}]$
for $j \in [\frac{n}{2}+1, n]$ **do**
    Generate entanglement between communication qubits of QPU1 and QPU2
    Transfer EPR pair to computational qubits
    Teleport qubit j to computational qubit of QPU1
    Apply all controlled rotation gates between qubits $[1, n-j+1]$ and qubit j, where j is the control qubit.
    Generate entanglement between computational qubits of QPU1 and QPU2
    Teleport qubit $n-j+1$ to QPU2
    Apply all controlled rotation gates between qubit $n-j+1$ and qubits $[j+1, n]$, where $n-j+1$ is the target qubit.
end for
For all i in $[\frac{n}{2}+1, n]$, apply a H gate on the i^{th} qubit, followed by the controlled rotations with i as the target qubit and $j \geq i$ as the control qubit. The order is inherently reversed so there is no need of the swap gates.

Algorithm 4: Distributed QFT for Multiple QPUs.

Input: Number of qubits n, number of QPUs k
Divide n qubits into k QPU, each with $m = \lceil \frac{n}{k} \rceil$ qubits.
for $i = 1$ to $k/2$ **do**
    Apply the gates in the i QPU.
    for $j = 0$ to $m-1$ **do**
        Move qubit j of i QPU from $i \to i-1 \to i-2 \to 1 \to i+1 \to i+2 \ldots \to k-i+1$
        Apply CR gates with qubit j as the target in QPUs $i-1, i-2, 1, i+1, i+2 \ldots, k-i+1$.
        Select qubit l from the original qubits in block $k-i+1$
        Move qubit l to QPU i. Apply CR gate with l as control in QPU i.
    end for
end for
for $i = k/2$ to 1 **do**
    Apply all gates in i QPU.
    for $j = 0$ to $m-1$ **do**
        Move qubit j from QPU $i \to i-1 \to i-2 \ldots \to 1 \to i$
        Apply CR gates with qubit j as the target in QPUs $i-1, i-2, i-3 \ldots, 1$.
    end for
end for

generalized to the case when the processors have more communication or computation qubits allowing for parallelization. The pseudocode of the method for distributing QFT on two QPUs is given in Algorithm 3. In this algorithm, we label the two QPUs as QPU1 and QPU2. The qubit numbering reflects the inherent numbering in the QFT circuit and not the qubit locations in the QPU.

Typically, the QFT reverses the order of qubits, which would require swap gates to correct the order for subsequent computations. However, in this distributed approach, as qubits are teleported between the QPUs and the corresponding gate operations required for the QFT are applied, the final qubit states emerge in the correct order without the need of additional swaps.

The method presented in Algorithm 3 requires the use of n EPR pairs throughout the execution of the algorithm. If the swapping order is not a concern, then there exists a more efficient method described in [48] that requires only $\frac{n}{2}$ EPR pairs. Adding swapping to the protocol [48] would increase the count of EPR pairs to $3n/2$. The reason being that $n/2$ swaps would require n EPR pairs.

B. QFT DISTRIBUTED ON MULTIPLE QPUS

The pseudocode for distributing the QFT across k QPUs is presented in Algorithm 4. The algorithm begins by dividing the n qubits into k blocks, each handled by one QPU, with $m = \lceil \frac{n}{k} \rceil$ qubits per block. We assume that k is even for simplicity, although the analysis can be extended to odd k.

The execution proceeds in two passes. In the first pass (forward direction), for each QPU indexed i from 1 to $k/2$, we apply local gates, then begin routing qubits across QPUs to enable all required CR interactions. Each qubit j in QPU i is moved leftward and rightward across neighboring QPUs—$i-1, i-2, \ldots, 1$ and $i+1, i+2, \ldots, k-i+1$—with CR gates applied at each step. In addition, a qubit from QPU $k-i+1$ is routed back to QPU i to complete mirrored interactions.

The second pass (reverse direction) performs a symmetric sweep. For i descending from $k/2$ to 1, qubits in each QPU are again routed leftward through $i-1, i-2, \ldots, 1$, and CR gates are applied. In this phase, a cat-entangler-based implementation could be used to further optimize entanglement usage.

This method completes the QFT without requiring explicit final swap gates, as the routing and gate sequence preserve the correct output order by construction.

1) EPR CALCULATION

The total number of EPR pairs required by the Algorithm 4 is determined by considering the movement and interaction of qubits across QPUs.

For the first loop, we can calculate the EPR pairs as

$$E^1_{\text{total}} \leq m \sum_{i=1}^{k/2} (k+1-i). \tag{13}$$

Observe that the number of blocks visited by any qubit from the ith block is $k-i$. Then, qubits from $k-i$ blocks must return to the original ith block.

If we use CAT entangler and disentangler for the second loop, then we have

$$E^2_{\text{total}} \leq m \sum_{i=1}^{k/2} (i-1) \tag{14}$$

and then

$$E_{\text{total}} \leq \frac{mk^2}{2} \tag{15}$$

where E^2_{total} is the total EPR pairs for the second loop and E_{total} is the total EPR pairs. The inequality in the above equation comes from the fact that the number of qubits in each QPU are upper bounded by m.

We also generalize the distributed QFT algorithm presented in [48] to accommodate multiple QPUs. Let us consider k QPUs or blocks. Each block $i \in [1, k]$ must interact with $k-i$ other blocks to apply the remote CR gates, where each block contains m qubits. Every interaction results in the exchange of m EPR pairs. Consequently, the total number of EPR pairs required can be calculated as $\frac{k(k-1)m}{2}$. While this approach does offer better EPR consumption compared to the one in Algorithm 4, it does not account for the final swap gates. For a fair comparison, we account for the swap gates in the extension of [48, Algorithm]. The EPR pair consumed by including the swap gate is $\frac{k(k-1)m}{2} + n = \frac{mk^2}{2} + \frac{mk}{2}$, where we assume $n = mk$. We then find that the algorithm introduced in Algorithm 4 uses less number of EPR pairs in comparison. We summarize the results as follows: given an n-qubit QFT circuit partitioned into m QPUs, the number of EPR pairs required by the described method is $\frac{nk}{2}$. In contrast, the number of EPR pairs required when generalizing the method from [48] is $\frac{nk}{2} + \frac{n}{2}$. See Fig. 10 for the plots.

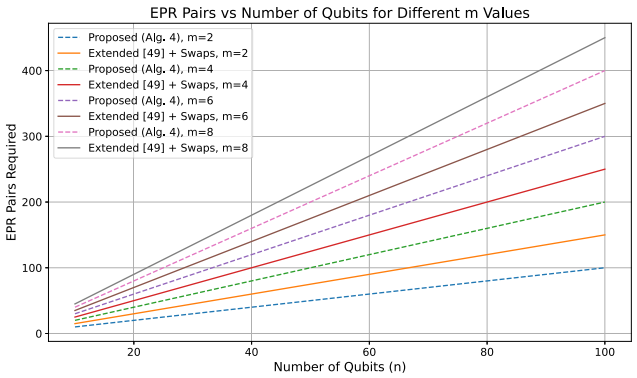

FIGURE 10. Comparison of EPR pair requirements for distributed QFT across multiple QPUs. We plot the total number of EPR pairs required as a function of the number of qubits *n*, assuming a fixed number of QPUs *k*. The dashed line represents our proposed method (see Algorithm 4), while the solid line shows the EPR usage for the extended method based on [48], including the cost of final swap gates. Our approach consistently requires fewer EPR pairs, particularly as *n* increases.

VIII. CONCLUSION

This work proposes a novel approach to optimizing qubit allocation in quantum data centers. By utilizing graph partitioning techniques and integer linear programming, the algorithm efficiently minimizes the EPR pairs required to execute quantum circuits across multiple QPUs. This method not only tackles the challenge of reducing overall EPR pair usage but also strategically alleviates the burden on low-fidelity and low-capacity links. In addition, we present a technique to optimize EPR demands for a QFT circuit distributed across multiple QPUs.

Together, these contributions form a pipeline for distributed quantum execution: The WBCP algorithm partitions the circuit in order to minimize the number of nonlocal gates, the ILP maps the circuit's qubits and their communication demands onto the physical network in a cost-aware manner, and the distributed QFT analysis offers closed-form insights for subroutines used commonly in quantum algorithms. When used jointly, these techniques provide a blueprint for scalable, network-aware DQC compiler. Future work will extend this model to support more complex quantum data center architectures, including heterogeneous QPUs, dynamic network topologies, and scheduling constraints driven by entanglement generation success rates. In addition, it would be valuable to formally characterize how the optimal window length varies with circuit structure and algorithmic class, potentially leading to adaptive partitioning strategies tailored to workload properties.

APPENDIX A
MODIFIED KL ALGORITHM

To solve the graph partitioning problem introduced in Section II, we use a modified version of the classical KL algorithm [20]. While the original KL algorithm assumes balanced partitions, our version supports unbalanced splits, allowing for partitions of size n_1 and n_2, where $n_1 + n_2 = n$, and n is the number of nodes in the graph.

A. UNBALANCED BIPARTITIONING

The algorithm starts with an initial partition and iteratively refines it by swapping node pairs to reduce the total edge cut between partitions. In our unbalanced setting, a constraint is introduced to preserve the specified partition sizes. Before each swap, we validate that the resulting partition maintains $|P_1| = n_1$ and $|P_2| = n_2$.

Time complexity: The modified KL algorithm has a worst case time complexity of $O(n^2)$ per iteration. The added checks for size balance introduce negligible overhead and do not change the asymptotic complexity.

Algorithm 5: Modified KL Algorithm for Unbalanced Graph Partitioning.

Input: Graph $G = (V, E)$, desired partition sizes n_1, n_2 such that $n_1 + n_2 = n$
Output: Two partitions P_1 and P_2 minimizing total edge weight, respecting size constraints
Step 1: Initial Partitioning
Divide graph G into partitions P_1 and P_2 with sizes $|P_1| = n_1, |P_2| = n_2$
repeat
  Step 2: Gain Calculation
  for each node $v_i \in P_1$ **do**
    Compute the gain g_i of moving v_i to P_2
  end for
  for each node $v_j \in P_2$ **do**
    Compute the gain g_j of moving v_j to P_1
  end for
  Step 3: Node Swapping with Balance Enforcement
  while there is a node pair (v_i, v_j) that improves gain **and** preserves partition sizes **do**
    Check if swapping $v_i \in P_1$ and $v_j \in P_2$ keeps $|P_1| = n_1, |P_2| = n_2$
    if swap is valid **then**
      Swap v_i and v_j
    end if
  end while
  until no further improvement in edge cut size
  Return final partitions P_1 and P_2

B. RECURSIVE EXTENSION FOR MULTIWAY PARTITIONING

To extend KL beyond bipartitioning, we apply it recursively. At each step, the graph is split into two parts, and the algorithm is applied to each subgraph until the desired number of partitions k is reached. If k is even, then we split evenly. If k is odd, then we assign approximately half the partitions to each side (e.g., $\lceil k/2 \rceil$ and $\lfloor k/2 \rfloor$), and continue the recursion accordingly. This strategy ensures that the recursion proceeds in a balanced fashion even when the target number of partitions is not a power of two.

An optional parameter allows users to specify custom partition sizes at each step, enabling support for nonuniform qubit distributions across QPUs. The overall time complexity of the recursive KL algorithm is $O(n^2 \log k)$, where n is the number of nodes and k is the number of partitions.

C. PSEUDOCODE

The pseudocode for the unbalanced KL algorithm is shown in Algorithm 5.

APPENDIX B
WBCP: DETAILED ALGORITHM

This section provides a detailed step-by-step description of the WBCP algorithm introduced in Section IV. The algorithm dynamically assigns qubits to QPUs across fixed-length windows in the circuit to minimize EPR pair usage.

a) *Inputs:* A quantum circuit C, window length w, and number of QPUs k.
b) *Window decomposition:*
   i) Divide C into subcircuits $\{C_i\}_{i=1}^{\tilde{m}}$, where each window contains at most w two-qubit gates.
   ii) Let $\tilde{m} = \lceil T/w \rceil$, where T is the total number of two-qubit gates.
c) *First window (initialization):*
   i) Construct a graph G_1 with one node per qubit and edges representing CNOT gates.
   ii) Edge weights are equal to the number of CNOTs between each pair of qubits.
   iii) Apply a graph partitioning algorithm (e.g., KL and METIS) to obtain the initial partition P_1.
d) *Subsequent windows:*
   i) For each window C_i $(i > 1)$:
      A) Construct a new graph G_i using the same qubit-to-node mapping.
      B) For each edge between qubits q_u and q_v:
      C) If q_u and q_v were colocated in P_{i-1}, then set edge weight to $2 \times \#\text{CNOT}(q_u, q_v)$.
      D) Otherwise, set weight to $\#\text{CNOT}(q_u, q_v)$.
      E) Apply the partitioning algorithm to obtain a candidate partition P_{new}.
      F) Count the number of qubits moved between P_{i-1} and P_{new}.
      G) Compute:

$$\text{EC}_{P_{\text{new}}}^{C_i} = \text{inter-QPU gate cost} + \#\text{qubits moved}.$$

      H) Compare with the reuse cost $\text{EC}_{P_{i-1}}^{C_i}$. Choose the lower cost option.
e) *Total cost accumulation:*
   i) Add the selected cost for each window to the total entanglement cost EC.
   ii) Store the corresponding qubit assignment P_i for that window.
f) *Output:*
   i) The total entanglement cost EC.
   ii) A sequence of qubit-to-QPU partitions $\{P_i\}_{i=1}^{\tilde{m}}$.
   iii) A gate-by-gate labeling of which operations are local versus nonlocal.

A. COMPLEXITY ANALYSIS

The computational complexity of the WBCP algorithm is dominated by the graph partitioning operation performed within each window. In every window, a graph is constructed where nodes represent qubits and weighted edges reflect the

number of CNOT gates between them. This graph is then partitioned into k subsets corresponding to QPUs. Since this step is executed once per window, its cost determines the overall runtime of the algorithm.

Let n denote the number of qubits and T the total number of two-qubit gates in the circuit. For a given window length w, the circuit is divided into approximately $\tilde{m} = \lceil T/w \rceil$ subcircuits. In each window, a graph of at most n nodes is constructed and partitioned. Since the number of qubits remains fixed across the circuit, the size of the graph per window is upper bounded by n, while the number of such graphs scales with T/w. Let k be the number of QPUs.

Assuming the use of the KL algorithm, which has worst case complexity $\mathcal{O}(n^2 \log k)$ per call, the total runtime of WBCP scales as $\mathcal{O}((T/w) \cdot n^2 \log k)$. Alternative partitioners, such as `METIS`, offer lower asymptotic complexity for large graphs (typically $\mathcal{O}(n \log n)$), but may sacrifice fine-grained control over qubit allocation. The actual runtime and quality of the partitioning depend on the specific algorithm chosen.

In our experiments, we evaluate the performance of WBCP across a range of window lengths w. We find that small window sizes, which allow for fine-grained qubit reallocation, and very large windows, which capture broader circuit structure, tend to produce the lowest EPR costs. In contrast, intermediate window sizes often underperform, as they fail to fully exploit either local adaptability or global coherence. We note that this is an empirical observation rather than a formal result, based on a sweep over a range of window lengths in increments of 5–10, which revealed consistent trends across benchmarks. But it suggests that quantum circuits often exhibit either predominantly local gate structure or predominantly global entanglement patterns, making extreme window lengths more effective.

APPENDIX C
GATE-PACKING AND TELEPORTATION OPTIMIZATION DETAILS

We detail the algorithm used for gate packing in each subcircuit. The input is G a directed graph where it represents the subcircuit's gate connectivity. Each edge (i, j) has a weight indicating how many times a two-qubit gate (e.g., CNOT) occurs between qubits i and j in this subcircuit. It is used to identify communication requirements across QPUs. The algorithm reduces the number of required EPR pairs by identifying multiple remote gates between the same pairs of qubits across QPUs and attempting to pack them into a single entanglement usage. P is the partition list, which is an array (or dictionary) that assigns each qubit to a partition (i.e., a QPU). For example, if $P[3] = 1$, it means qubit 3 is assigned to QPU 1. s (subcircuit index) identifies the current subcircuit being processed within the overall window-based partitioning algorithm. This helps index into global structures tracking gates across subcircuits. $\mathcal{I}$ (global gate indices) is a nested dictionary where $\mathcal{I}[s][(i, j)]$ gives a list of global indices for all instances of a two-qubit gate between qubits i and j in subcircuit s. This allows the algorithm to sort and schedule gates in the correct order.

It performs gate-packing by the following.

1) Grouping qubits by their assigned partitions.
2) Detecting remote two-qubit gates between qubits across partitions.
3) Sorting such gates by their execution order (based on global indices).
4) Iteratively checking whether consecutive gates can be packed together, i.e., executed using the same EPR pair without violating scheduling or partition constraints.
5) If packing is successful, then the edge weight in the dependence graph is reduced, and the gate is marked as entanglement-waived.

The time complexity of the gate packing algorithm can be broken down into two main components. First, building the mapping of all potential cross-partition two-qubit gates involves nested loops over pairs of qubits across different partitions, leading to a worst case complexity of $\mathcal{O}(n^2)$ where n is the number of qubits. Then, for each qubit, the algorithm performs gate packing checks over pairs of gates in a window, costing $\mathcal{O}(w^2)$. Therefore, the total worst case time complexity of the algorithm is $\mathcal{O}(n^2 + w^2)$. Despite the asymptotic bound, we observe low empirical cost in practice, owing to the sparsity of cross-partition edges, small window sizes, and the fact that these checks can be parallelized.

Next, we describe our algorithm for intrasubcircuit state teleportation. The underlying principle is that if a qubit participates in multiple nonlocal gates with the same remote qubit, it may be more efficient to temporarily relocate its state to the remote QPU and then return it, provided that the surrounding circuit structure and gate dependencies permit such movement. This allows multiple interactions to be performed with a single teleportation round, reducing the overall entanglement cost.

The time complexity of the intrasubcircuit state teleportation algorithm is dominated by two main components: identifying candidate qubit pairs across different partitions and evaluating teleportation opportunities. Constructing the partition-to-qubit mapping requires $\mathcal{O}(n)$ time, where n is the number of qubits. Checking all cross-partition qubit pairs to identify candidate gate interactions incurs a worst case cost of $\mathcal{O}(n^2)$. For each gate in the subcircuit with w gates, we check the condition for the state teleportation with all $w - 1$ other gates in the subcircuit, which will cost $\mathcal{O}(w^2)$. The overall time complexity becomes $\mathcal{O}(n + n^2 + w^2)$.

APPENDIX D
ILP COMPLEXITY ANALYSIS

The ILP formulation introduced in Section V produces a large binary optimization problem due to the need to model pairwise mappings between all logical and physical QPUs, as well as their interconnections.

Algorithm 6: Gate Packing Optimization for a Subcircuit.

Input: Directed graph G, partition list P, subcircuit index s, global gate indices $\mathcal{I}$
Output: Updated graph G and list of entanglement-waived gates

Group qubits by partition: $Q_p \leftarrow$ qubits in each partition $p \in P$
Initialize empty mapping: $gate\_map \leftarrow$ gates between qubit q and other partitions
for all partitions p_1 and p_2 where $p_1 \neq p_2$ **do**
  for all qubits q_1 in Q_{p_1} and q_2 in Q_{p_2} **do**
    if edge (q_2, q_1) exists in G **then**
      Add (q_2, q_1) to $gate\_map[q_1][p_2]$
    end if
  end for
end for
Initialize list of waived gates: $\mathcal{W} \leftarrow \emptyset$
for all qubits q with entries in $gate\_map$ **do**
  for all partitions p', gate list L in $gate\_map[q]$ **do**
    if $|L| > 1$ **then**
      Sort L by global gate index from $\mathcal{I}[s]$
      for all consecutive gate pairs (g_i, g_{i+1}) in L **do**
        Retrieve global indices t_i, t_{i+1} from $\mathcal{I}[s]$
        if CheckPackingConditionst_i, t_{i+1}, P **then**
          Decrease weight of edge g_{i+1} in G
          Add g_{i+1} to $\mathcal{W}$
          Remove t_{i+1} from $\mathcal{I}[s][g_{i+1}]$
        end if
      end for
    end if
  end for
end for
return $G, \mathcal{W}$

Algorithm 7: Intrasubcircuit State Teleportation.

Input: DiGraph G, partition assignment $\mathcal{P}$, subcircuit gate indices $\mathcal{I}$, subcircuit index s
Output: Updated DiGraph G with reduced entanglement weights

Initialize partition-to-qubits map $\mathcal{Q}$
for each qubit q **do**
  Add q to its partition's list in $\mathcal{Q}$
end for
Initialize candidate gate dictionary $\mathcal{C}$
for each pair (q_1, q_2) of qubits across different partitions **do**
  if (q_1, q_2) or (q_2, q_1) in G **then**
    Add (q_1, q_2) to $\mathcal{C}$ for teleportation consideration
  end if
end for
for each qubit q and target partition p in $\mathcal{C}$ **do**
  Let $\mathcal{G}$ be the list of candidate gates (q, q')
  if $|\mathcal{G}| > 2$ **then**
    Sort $\mathcal{G}$ by their global gate index
    for $i = 2$ to $|\mathcal{G}| - 1$ **do**
      Let $g_1 = \mathcal{G}[i-2]$, $g_2 = \mathcal{G}[i]$
      Get global indices idx_1, idx_2
      if TeleportationCondition$(idx_1, idx_2, \mathcal{P})$ **then**
        Reduce weight of edge g_2 in G
        Mark g_2 as teleported
      end if
    end for
  end if
end for
returnUpdated G

Let $k = |V(G)| = |V(G')|$ be the number of QPUs, and let $e = \binom{k}{2}$ denote the number of edges in each fully connected graph after zero-weight padding.

A. VARIABLES

The formulation contains the following.

1) k^2 binary variables y_{uv}, encoding logical-to-physical QPU assignments.
2) e^2 binary variables x_{ij}, encoding edge mappings.
3) $2e^2$ auxiliary binary variables $z1_{ij}$ and $z2_{ij}$, used to linearize the quadratic edge consistency constraints.

This results in a total of $\mathcal{O}(k^4)$ binary variables.

B. CONSTRAINTS

The ILP enforces the following.

1) $2k$ constraints to ensure one-to-one node mappings between G' and G.
2) $2e$ constraints to enforce that each logical edge is mapped to a physical edge and vice versa.
3) $7e^2$ constraints associated with the linearization of the quadratic consistency condition between edge and node assignments.

Hence, the total number of constraints also scales as $\mathcal{O}(k^4)$.

C. SCALABILITY

The size of the ILP grows quadratically with the number of QPUs. In practice, we find that solving instances with up to $k = 10$ QPUs is feasible using off-the-shelf ILP solvers, such as Gurobi or CPLEX. Larger instances quickly become intractable due to the number of binary variables and constraints.

The following optimizations may help improve scalability.

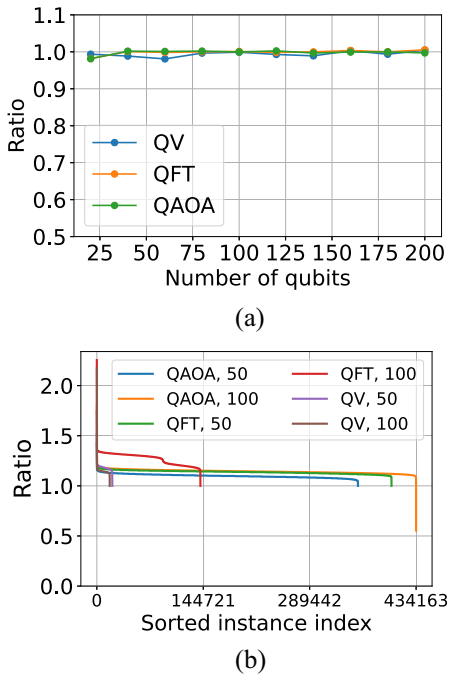

FIGURE 11. Effect of window-boundary refinement and dynamic windowing on EPR cost. (a) Ratio of optimized WBCP with window boundary handling to default WBCP. (b) Ratio of EPR cost in adaptive (or dynamic) WBCP to default WBCP cost for different circuit types and number of qubits. Number of partitions is 4.

1) *Symmetry breaking:* Many logical QPUs may be structurally interchangeable. Adding constraints to eliminate symmetric solutions can significantly reduce the search space.
2) *Preprocessing:* Removing edges with zero demand from consideration and mapping them to fixed assignments in advance can simplify the ILP before solving.
3) *Decomposition:* If the physical architecture exhibits a hierarchical structure (e.g., racks connected via clusters), then a graph partitioning or clustering approach can first divide the mapping problem into smaller regions. Independent ILPs can then be solved within each region, reducing overall problem size.

While these optimizations are not implemented in this work, they represent promising directions for future exploration.

APPENDIX E
WINDOW BOUNDARY LENGTH AND DYNAMIC WINDOW LENGTHS

In this experiment, we evaluate two extensions to our WBCP algorithm: boundary-aware partition refinement and a dynamic window length strategy. All results are obtained using a four-QPU setup.

Fig. 11(a) shows the ratio of the total EPR cost in optimized WBCP with boundary-aware refinement to the cost of the default WBCP (without refinement) as a function of the number of qubits in the circuit. While the boundary adjustment module occasionally yields improvements, the overall gain is modest. This indicates that the default partitioning already aligns well with circuit structure in many cases, and the refinement has limited additional impact.

In Fig. 11(b), we evaluate the effectiveness of a dynamic window strategy, where the circuit is divided into subcircuits with varying window lengths rather than using a fixed window size throughout. For each subcircuit, we randomly select a window length and apply WBCP. We then compute the ratio of the cost in the dynamic window approach to the cost in the default static-window approach and plot the distribution of these ratios. A ratio below 1 indicates that the dynamic approach performs better.

Despite testing over 400 000 random window length combinations, we observe that dynamic windowing rarely outperforms the fixed-window strategy. This suggests that the static windowing scheme offers a strong balance between simplicity and performance, and that dynamic tuning provides little additional benefit in practice.

ACKNOWLEDGMENT

The authors thank Don Towsley, Stephen DiAdamo, and Troy Sewell for the insightful discussions.

REFERENCES

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997, doi: 10.1137/S0097539795293172.
[2] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Phys. Rev. Lett.*, vol. 103, Oct. 2009, Art. no. 150502, doi: 10.1103/PhysRevLett.103.150502.
[3] A. K. Lenstra, *The Development of the Number Field Sieve*. Berlin, Germany: Springer 1993, doi: 10.1007/BFb0091534.
[4] G. H. Low and I. L. Chuang, "Hamiltonian simulation by qubitization," *Quantum*, vol. 3, Jul. 2019, Art. no. 163, doi: 10.22331/q-2019-07-12-163.
[5] A. Peruzzo et al., "A variational eigenvalue solver on a photonic quantum processor," *Nature Commun.*, vol. 5, no. 1, Jul. 2014, Art. no. 4213, doi: 10.1038/ncomms5213.
[6] R. Ur Rasool, H. F. Ahmad, W. Rafique, A. Qayyum, J. Qadir, and Z. Anwar, "Quantum computing for healthcare: A review," *Future Internet*, vol. 15, no. 3, Feb. 2023, Art. no. 94, doi: 10.3390/fi15030094.
[7] S. Woerner and D. J. Egger, "Quantum risk analysis," *NPJ Quantum Inf.*, vol. 5, no. 1, Feb. 2019, Art. no. 15, doi: 10.1038/s41534-019-0130-6.
[8] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 2019, doi: 10.1038/s41586-019-1666-5.
[9] M. DeCross et al., "Computational power of random quantum circuits in arbitrary geometries," *Phys. Rev. X*, vol. 15, May 2025, Art. no. 021052. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.15.021052
[10] H. J. Manetsch et al., "A tweezer array with 6100 highly coherent atomic qubits," *Nature*, 2025, doi: 10.1038/s41586-025-09641-4.
[11] L. J. Stephenson et al., "High-rate, high-fidelity entanglement of qubits across an elementary quantum network," *Phys. Rev. Lett.*, vol. 124, Mar. 2020, Art. no. 110501, doi: 10.1103/PhysRevLett.124.110501.
[12] IBM, "IBM quantum roadmap," 2024. Accessed: Aug. 7, 2024. [Online]. Available: https://www.ibm.com/roadmaps/quantum/
[13] IonQ, "IonQ achieves critical first step towards developing future quantum networks," 2024. Accessed: Aug. 7, 2024. [Online]. Available: https://ionq.com/news/ionq-achieves-critical-first-step-towards-developing-future-quantum-networks

[14] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory Comput. Syst.*, vol. 39, no. 6, pp. 929–939, Oct. 2006, doi: 10.1007/s00224-006-1350-7.

[15] P. Andres-Martinez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Phys. Rev. A*, vol. 100, no. 3, 2019, Art. no. 032308, doi: 10.1103/PhysRevA.100.032308.

[16] P. Andres-Martinez et al., "Distributing circuits over heterogeneous, modular quantum computing network architectures," *Quantum Sci. Technol.*, vol. 9, no. 4, 2024, Art. no. 045021, doi: 10.1088/2058-9565/ad6734.

[17] H. Shapourian, E. Kaur, J. Zhao, M. Kiltzer, R. Kompella, and R. Nejabati, "Quantum data center infrastructures: A scalable architectural design perspective," 2024, *arXiv:2501.05598*, doi: 10.48550/arXiv.2501.05598.

[18] D. Sakuma et al., "An optical interconnect for modular quantum computers," 2024, *arXiv:2412.09299*, doi: 10.48550/arXiv.2412.09299.

[19] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures," in *Proc. 17th ACM Int. Conf. Comput. Front.*, 2020, pp. 98–107, doi: 10.1145/3387902.3392617.

[20] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Techn. J.*, vol. 49, no. 2, pp. 291–307, 1970, doi: 10.1002/j.1538-7305.1970.tb01770.x.

[21] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Jan. 1998, doi: 10.1137/S1064827595287997.

[22] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 841–855, Jul. 1996, doi: 10.1109/12.508322.

[23] 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, pp. 1895–1899, Mar. 1993, doi: 10.1103/PhysRevLett.70.1895.

[24] D. Gottesman and I. L. Chuang, "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations," *Nature*, vol. 402, no. 6760, pp. 390–393, Nov. 1999, doi: 10.1038/46503.

[25] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio, "Optimal local implementation of nonlocal quantum gates," *Phys. Rev. A*, vol. 62, Oct. 2000, Art. no. 052317, doi: 10.1103/PhysRevA.62.052317.

[26] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand, "Optimizing teleportation cost in distributed quantum circuits," *Int. J. Theor. Phys.*, vol. 57, no. 3, pp. 848–861, Dec. 2017, doi: 10.1007/s10773-017-3618-x.

[27] S. Schlag et al., "High-quality hypergraph partitioning," *ACM J. Exp. Algorithmics*, vol. 27, pp. 1–39, 2023, doi: 10.1145/3529090.

[28] A. Yimsiriwattana and S. J. Lomonaco, "Generalized GHZ states and distributed quantum computing," 2004, *arXiv:quant-ph/0402148*, doi: 10.48550/arXiv.quant-ph/0402148.

[29] M. Houshmand, Z. Mohammadi, M. Zomorodi-Moghadam, and M. Houshmand, "An evolutionary approach to optimizing communication cost in distributed quantum computation," 2019, *arXiv:1910.07877*, doi: 10.48550/arXiv.1910.07877.

[30] O. Daei, K. Navi, and M. Zomorodi-Moghadam, "Optimized quantum circuit partitioning," *Int. J. Theor. Phys.*, vol. 59, no. 12, pp. 3804–3820, Nov. 2020, doi: 10.1007/s10773-020-04633-8.

[31] E. Nikahd, N. Mohammadzadeh, M. Sedighi, and M. S. Zamani, "Automated window-based partitioning of quantum circuits," *Physica Scripta*, vol. 96, no. 3, Jan. 2021, Art. no. 035102, doi: 10.1088/1402-4896/abd57c.

[32] J.-Y. Wu et al., "Entanglement-efficient bipartite-distributed quantum computing," *Quantum*, vol. 7, Dec. 2023, Art. no. 1196, doi: 10.22331/q-2023-12-05-1196.

[33] P. Andres-Martinez et al., "Distributing circuits over heterogeneous, modular quantum computing network architectures," *Quantum Sci. Technol.*, vol. 9, no. 4, 2024, Art. no. 045021, doi: 10.1088/2058-9565/ad6734.

[34] R. G Sundaram, H. Gupta, and C. R. Ramakrishnan, "Efficient distribution of quantum circuits," in *35th Int. Symp. Distrib. Comput. (DISC 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2021, pp. 41:1–41:20, doi: 10.4230/LIPIcs.DISC.2021.41.

[35] R. G. Sundaram, H. Gupta, and C. R. Ramakrishnan, "Distribution of quantum circuits over general quantum networks," 2022, *arXiv:2206.06437*, doi: 10.48550/arXiv.2206.06437.

[36] Z. Davarzani, M. Zomorodi-Moghadam, M. Houshmand, and M. Nouri-baygi, "A dynamic programming approach for distributing quantum circuits by bipartite graphs," 2020, *arXiv:2005.01052*, doi: 10.48550/arXiv.2005.01052.

[37] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures," in *Proc. 17th ACM Int. Conf. Comput. Front.*, May 2020, pp. 98–107, doi: 10.1145/3387902.3392617.

[38] F. Burt, K.-C. Chen, and K. K. Leung, "Generalised circuit partitioning for distributed quantum computing," in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*, Montreal, QC, Canada, 2024, pp. 173–178, doi: 10.1109/QCE60285.2024.10273.

[39] P. Escofet, A. Ovide, C. G. Almudever, E. Alarcón, and S. Abadal, "Hungarian qubit assignment for optimized mapping of quantum circuits on multi-core architectures," *IEEE Comput. Archit. Lett.*, vol. 22, no. 2, pp. 161–164, Jul.–Dec. 2023, doi: 10.1109/LCA.2023.3318857.

[40] A. Pastor, P. Escofet, S. Ben Rached, E. Alarcón, P. Barlet-Ros, and S. Abadal, "Circuit partitioning for multi-core quantum architectures with deep reinforcement learning," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2024, pp. 1–5, doi: 10.1109/ISCAS58744.2024.10557956.

[41] F. Burt, K.-C. Chen, and K. K. Leung, "Generalised circuit partitioning for distributed quantum computing," in *Proc. IEEE Int. Conf. Quantum Comput. Eng.*, vol. 2, 2024, pp. 173–178, doi: 10.1109/QCE60285.2024.10273.

[42] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014, *arXiv:1411.4028*, doi: 10.48550/arXiv.1411.4028.

[43] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Phys. Rev. A*, vol. 100, no. 3, 2019, Art. no. 032328, doi: 10.1103/PhysRevA.100.032328.

[44] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," 2004, *arXiv:quant-ph/0410184*, doi: 10.48550/arXiv.quant-ph/0410184.

[45] F. Burt, "DISQCO: Distributed quantum circuit optimization," 2024. Accessed: May 14, 2025. [Online]. Available: https://github.com/felix-burt/DISQCO

[46] D. Coppersmith, "An approximate fourier transform useful in quantum factoring," 2002, *arXiv:quant-ph/0201067*, doi: 10.48550/arXiv.quant-ph/0201067.

[47] A. Yimsiriwattana and S. J. Lomonaco, Jr., "Distributed quantum computing: A distributed Shor algorithm," in *Quantum Information and Computation II*. Bellingham, WA, USA: SPIE, doi: 10.1117/12.546504.

[48] N. M. P. Neumann, R. van Houte, and T. Attema, *Imperfect Distributed Quantum Phase Estimation*. Berlin, Germany: Springer, 2020, pp. 605–615, doi: 10.1007/978-3-030-50433-5_46.

[49] A. Yimsiriwattana and S. J. Lomonaco, Jr, "Generalized GHZ states and distributed quantum computing," 2004, *arXiv:quant-ph/0402148*, doi: 10.48550/arXiv.quant-ph/0402148.