Quantum Information Processing (2026) 25:118
https://doi.org/10.1007/s11128-026-05136-x

Co-design threading model and circuit cutting for static and adaptive quantum circuits

Waldemir Cambiucci[1] · **Regina Melo Silveira**[1] · **Wilson Vicente Ruggiero**[1]

Received: 17 January 2025 / Accepted: 8 March 2026

Abstract

As quantum computing rapidly evolves, scaling quantum algorithms to utilize multiple quantum processing units (QPUs) becomes crucial for overcoming the limitations of current noisy intermediate-scale quantum (NISQ) devices. This paper focuses on distributed quantum computing (DQC), specifically targeting the challenges associated with circuit cutting and circuit distribution in multi-QPU environments. By leveraging techniques such as hypergraph partitioning and threading models, this paper presents alternative strategies for dividing and managing both static and adaptive quantum circuits across multiple QPUs. A central research question is how the partitioning and distribution of quantum circuits can be optimized to minimize communication overhead and maximize computational performance in multi-QPU systems, for static and adaptive quantum circuits. To answer this question, a hypergraph partitioning method is proposed to effectively segment large static quantum circuits into manageable subcircuits, ensuring minimal inter-QPU gate operations and reduced time for the partitioning process. Additionally, a co-design threading model is presented, tailored for adaptive quantum circuits. This category of circuits can dynamically adjust their sequence of gates in runtime, based on intermediate measurements and classical control flows, creating unique challenges for circuit partition. Finally, to support the coordination between circuit partitions with static and adaptive circuits, we propose a quantum resource manager (QRM) architecture, bridging the gap between partitioning techniques and practical coordination for a scalable quantum computing system with multi-QPU architecture.

Keywords Distributed quantum systems · Adaptive quantum circuits · Quantum resource manager · Circuit cutting · Hypergraphic partitioning · Co-design threading model

✉ Waldemir Cambiucci
waldemir.cambiucci@usp.br

[1] Department of Computing Engineering and Digital Systems, University of São Paulo, São Paulo, Brazil

Published online: 31 March 2026

1 Introduction

Quantum computing holds the promise of unprecedented benefits in accelerating solutions to complex problems by harnessing the principles of quantum mechanics. However, numerous challenges hinder the application of such advantages in all scenarios. These challenges permeate the entire development stack of quantum machines [1], from the physical implementation of qubits to hardware control [2], from control software to real-world applications. This landscape delineates the computational power limitation of current machines, known as noisy intermediate-scale quantum (NISQ) devices [3], which feature few qubits, high error rates, operation latency, low gate fidelity, and short-lived states, thereby hindering the execution of large algorithms.

To overcome these limitations, distributed quantum computing (DQC) emerges as a promising pathway to scalability. DQC is defined as hardware with multiple connected quantum processing units (QPUs). These processors can be connected using direct or indirect quantum links, which create challenges for latency and provisioning. Still, communication between two QPUs can be realized using a shared pair of entangled qubits (ebits) and a teleportation protocol. In this context, entanglement sharing is at least 10 $\times$ slower than local operations inside a partition running in a single QPU. By connecting multiple quantum processing units (QPUs) via quantum or classical communication links, DQC enables the execution of larger circuits that surpass the capacity of individual quantum processors. However, scaling quantum computations across multiple QPUs often involves entanglement-based communication, introducing additional time delays and noise. Efficient inter-QPU communication is essential to maintain the integrity of quantum computations across different hardware resources.

In this context, circuit cutting refers to the process of partitioning large quantum circuits into smaller, manageable subcircuits that can be executed concurrently on distinct QPUs within a modular quantum computing architecture. Unlike approaches that cut circuits into independent fragments and later reconstruct global results via classical post-processing techniques, our work focuses on the dynamic distribution and mapping of virtual qubits to physical qubits across multiple QPUs, ensuring the correct execution of the overall computation while minimizing inter-QPU communication and latency costs. This perspective aligns with the notion of quantum circuit mapping for modular architectures, where operational constraints of current NISQ devices, such as coupling limitations, communication overhead, and dedicated qubit resources for inter-QPU links, play a critical role. Prior studies [4–8] have laid important groundwork in distributed quantum execution and partitioning strategies, but further integration of these techniques with modular architecture and resource-aware coordination remains necessary [56, 57].

Recent works have further advanced the understanding of circuit cutting and mapping in modular and multi-core quantum architectures, providing important context for our approach. BAKER et al. [56] introduced a time-sliced partitioning technique for modular architectures, where circuits are divided both spatially and temporally to fit the constraints of interconnected QPUs while minimizing communication latency. More recently, ESCOFET et al. [57] revisited the mapping of quantum circuits in the emerging multi-core quantum era, highlighting how the allocation of qubits and operations across cores impacts performance and scalability. Our work builds upon

these insights by combining hypergraph-based partitioning for static circuits with a co-design threading model for adaptive circuits, while explicitly addressing the coordination challenges through a quantum resource manager (QRM). By situating our contribution within this line of research, we extend the modular mapping paradigm to accommodate the dynamic behavior of adaptive circuits alongside static ones in distributed, multi-QPU systems.

Two primary approaches for circuit cutting [9] are wire cuts (also known as qubit-wire cuts or temporal cuts) and gate cuts (also known as spatial cuts). Wire cuts reduce the depth of the circuit by segmenting it along qubit lines, allowing the creation of multiple segments executed sequentially with classical post-processing to combine results. Gate cuts reduce the width of the original circuit by creating segments with fewer qubits, partitioning the circuit along specific quantum gates.

These approaches have been primarily applied to static quantum circuits, which have fixed structures and predetermined gate sequences. Static circuits, however, are less adaptable to hardware constraints, leading to inefficiencies and increased communication overhead when partitioned across multiple QPUs [10, 11]. More recently, adaptive quantum circuits [19] dynamically adjust their structure or execution sequence based on intermediate measurements and classical computations. This dynamic behavior introduces additional complexities for circuit partitioning, as the circuit structure evolves during execution, and communication requirements may vary in real time.

Adaptive quantum circuits, while potentially reducing circuit depth and improving convergence rates, creating more complex and high-level quantum circuits, require sophisticated partitioning techniques to manage their dynamic nature effectively. Traditional circuit cutting methods designed for static circuits fail to account for the real-time adaptability inherent in adaptive circuits and do not minimize both state and gate teleportation costs between QPUs.

While graph-based partitioning is already a stablished solution for circuit cutting in static circuits [15, 16], hypergraph-based partitioning offers a promising solution by capturing complex dependencies within adaptive circuits and optimizing communication between QPUs connected via classical or quantum links. By modeling quantum circuits as hypergraphs, a generalization of graphs where qubits are vertices and multi-qubit operations are hyperedges, we can accurately represent the complex interdependencies of adaptive operations and even take benefits from hypergraph heuristics still for static circuits' partitions.

However, identifying all potential branches resulting from adaptive operations and creating corresponding subgraphs is still a complex task. We can use hypergraph representation to select optimal cuts for subcircuits in off-line process [12], but the effectively managing of the adaptive quantum circuit in runtime is still a huge task.

This paper addresses the emerging challenges of quantum circuit cutting in static and adaptive quantum circuits, proposing two different approaches interconnected: first, a reduced heuristic of the hypergraph partitioning process based on Fiduccia—Mattheyses algorithm, to speed up the partitioning task for static circuits; and second, a co-design and threading model approach for adaptive quantum circuit, based on the coordination of selected threads or portions of code that can be executed in distinct QPUs. In this work, both approaches will be orchestrated by a central unit called quantum resource manager, with a proposed architecture.

The remainder of this paper is organized as follows: Section II provides the background and related work, beginning with an exploration of static versus adaptive quantum circuits. Additionally, Section II introduces hypergraph partitioning in quantum computing, covering the basics of hypergraph theory and its previous applications in quantum circuits. Section III outlines the approach and methodology, starting with hypergraph partitioning for static quantum circuits, with explanation of the techniques used and strategies to minimize inter-QPU gate operations and latency. It proceeds with the development of threading models for adaptive quantum circuits, detailing how these models are designed to dynamically adjust gate sequences based on intermediate measurements and classical control flows. The section concludes with the design of the quantum resource manager, describing its architecture within a full-stack quantum computer, the interactions among its functional components, and its role in connecting theoretical partitioning approaches with practical implementation. Section IV presents the experiments and results, starting with benchmarking static quantum circuits and adaptive quantum circuits. Finally, section V concludes the paper with a discussion of the insights gained from the experimental data, evaluating the effectiveness of hypergraph partitioning and threading models. By summarizing the contributions, key findings, and innovations presented, the section ends with final remarks on the optimizing circuit cutting and distribution in multi-QPU systems with co-design approach for threading model with static and adaptive quantum circuits, emphasizing the broader implications of a quantum resource manager for distributed quantum computing.

2 Background and related work

Quantum circuits are the fundamental building blocks of quantum algorithms, consisting of a sequence of quantum gates applied to qubits to manipulate quantum states [13, 14]. The nature of these circuits can be broadly classified into two categories: static quantum circuits and adaptive (or dynamic) quantum circuits [21]. Understanding the differences between these two types is crucial for circuit design, especially in the context of circuit partitioning and distributed quantum computing.

2.1 Static vs. adaptive quantum circuits

A static quantum circuit is characterized by a predetermined, unchanging sequence of quantum operations that are applied to the qubits. The structure and gate sequence are fixed before execution, and there are no modifications or adjustments based on intermediate results during runtime. This rigidity simplifies both the theoretical analysis and practical implementation of the circuit.

Formally, a static quantum circuit C operating on a quantum state $|\psi\rangle$ can be represented as a sequence of unitary operations:

$$C = U_N U_{N-1} \dots U_2 U_1, \tag{1}$$

where each U_i a unitary operator (quantum gate) acting on one or more qubits, and N is the total number of gates in the circuit. The final state after applying for the circuit is:

$$\left|\psi_{fina}\right\rangle = C|\psi\rangle = U_N U_{N-1} \ldots U_2 U_1 |\psi\rangle. \tag{2}$$

So, in static circuits, the sequence of operations does not depend on measurement outcomes or external parameters during execution. The exact operations and their order are known ahead of time, allowing for thorough optimization and error mitigation strategies. This fixed structure facilitates theoretical analysis, such as calculating circuit depth, gate counts, and expected output states in several scenarios.

An adaptive quantum circuit [21], also known as a dynamic quantum circuit [51], incorporates real-time adjustments to its operations based on intermediate measurements and classical computations performed during execution. These circuits blend quantum operations with classical control flow, enabling conditional logic and feedback mechanisms that can modify the circuit's behavior on the fly.

Formally, an adaptive quantum circuit $C^{'}$ can be represented as a sequence of operations that depend on measurement outcomes:

$$C' = U_N(m_{N-1}, .., m_1) U_{N-1}(m_{N-2}, .., m_1) \ldots U_2(m_1) U_{1,} \tag{3}$$

where:

- *Ui(mi-1,..,m1)* are unitary operations conditioned on previous measurement outcomes *mj* for $j < i$.
- m_i denotes the classical measurement result obtained after a measurement operation at step*i*. Measurement outcomes m_i influence subsequent gate operations, making the circuit's evolution dependent on its intermediate results.

Considering this definition, in adaptive quantum circuits [21], gates can be applied conditionally based on measurement outcomes, allowing for branches in the circuit analogous to classical if statements. Measurement results are processed by classical computations that determine future quantum operations, introducing a feedback loop between quantum and classical systems.

Figure 1 illustrates the layout of static and adaptive circuits.

In this context, circuit's path can vary between executions, leading to a potentially vast number of different operation sequences depending on probabilistic measurement

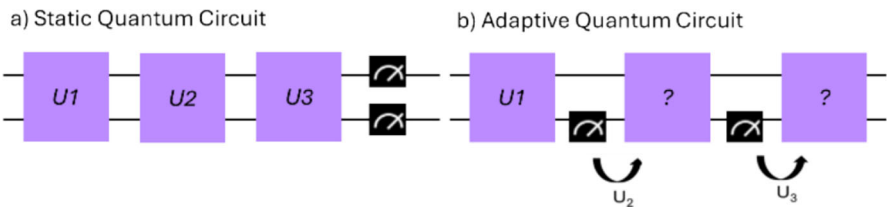


Fig. 1. Circuit diagram for static circuit (**a**) and adaptive circuit (**b**). Conditional operations and classical structures such as "loop" and "switch/cases" can change the sequence of operations in runtime

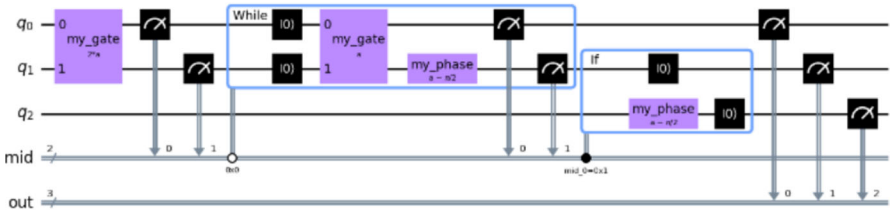


Fig. 2. Circuit diagram for adaptive quantum circuit with structures such as "WHILE" and "IF/ELE" changing the sequence of operations in runtime

outcomes. The adaptive nature of dynamic circuits introduces challenges for circuit partitioning and distributed execution.

While the execution path can vary due to probabilistic measurement outcomes influencing subsequent operations, adaptive circuits can be more complex to analyze due to the branching structure and dependencies on measurement results. And for the execution in runtime, we require specialized hardware that supports mid-circuit measurements, fast classical processing, and the ability to apply gates conditionally during execution. Platforms as IBM's Heron series [17] and Quantinuum H1-1 series [18] already support adaptive quantum circuits in different scenarios, with operands such as "IF/ELSE", "SWITCH", "FOR", and "WHILE".

Conditional operations create non-trivial dependencies between different parts of the circuit, complicating partitioning strategies. In a distributed setup, measurement outcomes need to be communicated between QPUs to coordinate subsequent operations, potentially increasing latency. And allocating qubits and gates across QPUs requires careful consideration to ensure that conditional operations can be executed without excessive inter-QPU communication.

Figure 2 illustrates an adaptive quantum circuit, with a random combination of classical structures such as "WHILE" and "IF/ELSE", responsible for changing the sequence of operations to be execute in runtime.

2.2 Distributed quantum computing (DQC) and multi-QPU systems

Distributed quantum computing (DQC) [10] represents a strategic approach to overcoming the current limitations of individual quantum processing units (QPUs) by interconnecting multiple QPUs, thereby enabling the execution of larger and more complex quantum algorithms. This methodology is driven by the recognition that the quantum systems available today, known as noisy intermediate-scale quantum (NISQ) devices, are limited by a relatively small number of qubits, high error rates, and coherence time constraints. DQC aims to extend computational capacity and fault tolerance by leveraging the power of multiple QPUs working in concert, distributing computational tasks, and managing entanglement across these interconnected units.

The foundations of DQC are drawn from both classical distributed computing principles and quantum mechanics. Just as classical distributed systems use interconnected processing units to achieve scalability and fault tolerance, DQC uses multiple QPUs

to handle larger quantum workloads that surpass the capabilities of a single QPU. Key to this paradigm is the ability to partition a quantum circuit into subcircuits, which can then be assigned to different QPUs [11]. This necessitates efficient inter-QPU communication, which often requires classical channels for data exchange alongside quantum channels for entanglement and teleportation. The interdependence of these channels demands a high level of coordination, where classical control is closely coupled with quantum operations to ensure coherent and consistent circuit evolution.

For static quantum circuits, where the sequence of operations is predetermined and does not depend on runtime measurements, DQC can be highly effective. Static circuits benefit from well-established partitioning strategies that aim to minimize the number of inter-QPU operations, thereby reducing communication overhead and latency. Techniques such as hypergraph partitioning can be used to analyze the circuit structure and identify segments that can be executed independently. The goal is to keep most operations localized within individual QPUs and minimize the number of gates that require synchronization across QPUs. In this context, the advantage of DQC is clear: it can expand the practical problem size that a quantum system can solve by using parallelism and by circumventing the qubit number limitations of a single QPU.

Adaptive quantum circuits [21], on the other hand, introduce additional complexity to the distributed quantum computing model. Adaptive circuits are characterized by their dynamic nature, where the sequence of gates can change during execution depending on intermediate measurement outcomes and classical control logic, such as "IF-ELSE" statements or "WHILE" and "FOR" loops. This adaptability implies that the distribution of the circuit cannot be fully predefined and must instead adapt as the computation proceeds. For DQC involving adaptive circuits, the key challenge lies in the need for real-time communication between QPUs. Measurement results from one QPU might need to be shared with others to determine subsequent operations, leading to increased latency and the need for sophisticated synchronization mechanisms [20].

The impact of DQC on adaptive circuits is profound, as it necessitates new strategies for circuit segmentation and execution coordination. Unlike static circuits, adaptive circuits cannot rely solely on static partitioning heuristics, as the circuit topology evolves during computation. Instead, a threading model [21, 22] can be employed, where each QPU handles a specific thread or sequence of operations, and control is dynamically transferred between threads based on classical results. In this context, a quantum resource manager (QRM) plays a crucial role, acting as an orchestrator that coordinates the execution flow between QPUs, initializing the qubits allocation, and managing the synchronization of classical data to ensure consistency across the distributed system.

To address the challenges of DQC for adaptive quantum circuits, a co-design threading model is proposed in this research, where developers collaborate closely with the quantum resource manager to identify portions of the circuit that can be segmented and distributed across multiple QPUs. This approach allows developers to explicitly indicate parts of the circuit that are suitable for parallel execution, optimizing both resource utilization and communication patterns. By combining human insight into circuit structure with automated orchestration by the QRM, the co-design model enhances the ability to efficiently manage adaptive behaviors, reducing inter-QPU latency and improving the overall performance of distributed adaptive quantum circuits.

2.3 Circuit cutting techniques

Circuit cutting has emerged as a crucial strategy in quantum computing to enable the execution of large-scale quantum circuits on hardware with limited qubit counts and coherence times. By partitioning a complex quantum circuit into smaller, more manageable fragments, circuit cutting allows for the simulation of larger circuits than what current quantum computers can natively handle. Recent advancements in circuit cutting techniques have focused on improving efficiency [52], optimizing partitioning process [7, 8], minimizing communication costs [34], and mitigating errors inherent in noisy quantum systems [32].

In the generalized circuit partitioning approach [23], the authors introduce a graph-based method for optimizing circuit partitioning in distributed quantum computing (DQC), where quantum processing units (QPUs) are interconnected via shared entanglement to enable teleportation of both states and gates. By jointly optimizing gate and state teleportation costs, unlike previous methods that focus on one or the other, the authors minimize entanglement-based communication that increases processing time. Using gate grouping and a genetic algorithm, their approach reduces entanglement bit (EBIT) costs and improves time scaling across various circuits compared to existing methods.

Integrated quantum reuse and circuit cutting (IQRC) [24] is an innovative approach that combines qubit reuse with circuit cutting to optimize resource utilization. By strategically reusing qubits and minimizing the number of cuts, IQRC achieves a reduction in the number of required cuts by approximately 34% on average. This method not only conserves valuable quantum resources but also decreases the classical post-processing overhead associated with reconstructing the full circuit's output from its fragments.

Maximum likelihood fragment tomography (MLFT) [25] addresses the classical computational overhead in circuit cutting by employing a maximum likelihood estimation framework. This technique reconstructs the most probable distribution of the original quantum circuit output based on measurements from its fragments. MLFT minimizes the amount of classical computation needed for post-processing, thus making it more practical to work with larger circuits.

Majumdar and Wood [26] explored the application of error mitigation strategies, such as readout error mitigation and dominant eigenvalue truncation (DEVT), to enhance circuit cutting performance in noisy environments. These techniques aim to suppress the errors that accumulate during quantum computations, improving the fidelity of the results obtained from circuit fragments.

Rotation-inspired circuit cut optimization (RICCO) [27] is a method that reduces post-processing overhead by optimizing the placement of cuts within a quantum circuit. Inspired by rotation-based optimization techniques, RICCO strategically selects cut locations that minimize the complexity of reconstructing the circuit's output. This results in more efficient computations and less demanding classical resources for post-processing.

Entanglement-based criteria for circuit fragmentation [28] is an approach that involves developing entanglement-based criteria to guide the fragmentation of quantum circuits. By minimizing the loss of qubit entanglement during circuit cutting, this method preserves the quantum correlations necessary for accurate computations. Taking entanglement into account improves the circuit's performance and reliability.

With the virtual distillation combined with circuit cutting [29], Li and colleagues investigated the combination of virtual distillation—a technique used to mitigate errors by effectively purifying quantum states—with circuit cutting. This hybrid approach shows promise in reducing the impact of noise on quantum computations, thereby improving the accuracy of results obtained from current noisy quantum computers.

The recent advancements in circuit cutting techniques underscore a trend toward integrating resource optimization and error mitigation to make large-scale quantum computations more feasible on near-term hardware.

Also, graph and hypergraph methodologies have become instrumental in advancing circuit cutting techniques in quantum computing. By representing quantum circuits as graphs or hypergraphs, these approaches enable systematic partitioning that minimizes communication overhead, preserves entanglement, and optimizes resource allocation.

The FitCut approach [30] involves transforming quantum circuits into weighted graphs. This representation allows the application of community detection algorithms to identify tightly connected subgraphs (communities) within the circuit.

While primarily focused on qubit reuse, Pawar et al. [24] also incorporated graph-based techniques in their circuit cutting strategy. By representing the circuit as a graph, they identified opportunities to reuse qubits without increasing the number of cuts significantly. This integration led to a reduction in the total number of cuts required, optimizing both quantum and classical resources.

Andrés-Martínez et al. [12] developed a method for the automated distribution of quantum circuits using hypergraph partitioning. By representing circuits as hypergraphs, they automated the fragmentation process to distribute computational tasks effectively across different quantum processors.

Duncan et al. [31] employed ZX-calculus, a graphical language for quantum mechanics, to simplify quantum circuits using graph-theoretic methods. By transforming circuits into ZX-diagrams, they applied graph simplification techniques to reduce circuit complexity. This process can lead to more efficient circuit cutting by identifying redundancies and optimizing the circuit structure before fragmentation.

FragQC [32] uses a weighted graph representation of quantum circuits to facilitate efficient fragmentation. By assigning weights based on gate complexities and qubit interactions, FragQC identifies optimal cut locations that balance the computational load and minimize communication between fragments.

Lowe et al. [33] proposed the use of randomized measurements in conjunction with graph-based methods for fast circuit cutting. By representing circuits as graphs and applying randomized measurement techniques, they reduced the overhead associated with reconstructing the output from circuit fragments. This approach accelerates the circuit cutting process while maintaining result accuracy.

Davarzani et al. [7] presents a dynamic programming algorithm that minimizes communication in distributed quantum computation by optimally partitioning quantum circuits into smaller, low-capacity units. By converting circuits into bipartite

graph models and applying dynamic programming, the method significantly reduces the number of required quantum teleportation on benchmark circuits, addressing key challenges in developing near-term large-scale quantum computers.

The integration of graph and hypergraph approaches in circuit cutting has markedly advanced methods to partition quantum circuits. These techniques address key challenges such as communication overhead, resource allocation, and computational efficiency. By leveraging the structural properties of quantum circuits through graph and hypergraph representations, we optimized partitioning strategies for large-scale algorithms on near-term hardware. However, the impact of new adaptive quantum circuits on graph and hypergraph partitioning remains underexplored. Adaptive circuits introduce dynamism that static graph models do not fully capture. While significant progress has been made in optimizing static quantum circuits through various circuit cutting techniques, there is a noticeable gap in the incorporation of adaptive quantum circuits into these strategies.

Building on the rich landscape of circuit cutting techniques outlined above, our work extends these approaches by addressing the unique challenges posed by both static and adaptive quantum circuits in a distributed, multi-QPU setting. While many of the reviewed methods focus on optimizing static circuit partitioning, mitigating noise, or reducing classical post-processing, they generally assume fixed circuit structures and do not fully account for the dynamic behavior of adaptive circuits, where gate sequences depend on intermediate measurements. Our proposed framework complements and advances this body of work by combining hypergraph-based partitioning for static circuits with a co-design threading model specifically designed for adaptive circuits, coordinated through a quantum resource manager. This integration bridges the gap between state-of-the-art static circuit cutting strategies and the emerging need to efficiently distribute adaptive quantum workloads across modular quantum architectures.

2.4 Graph and hypergraph for quantum circuits

Quantum circuits can be effectively modeled using mathematical structures such as graphs and hypergraphs. These representations facilitate the analysis, optimization, and partitioning of circuits, which is particularly important in the context of distributed quantum computing and adaptive quantum circuits.

2.4.1 Graph representation of quantum circuits

Graph G = (V, E) is a mathematical structure consisting of vertices V, representing qubits in the quantum circuit, and edges E, representing binary (two-qubit) quantum gates that connect pairs of qubits. In this representation:

- An edge e = (u, v) in E corresponds to a binary quantum gate, such as CNOT, CZ, CP, or SWAP, acting on qubits u and v.
- The graph captures the pairwise interactions between qubits, allowing for the visualization and analysis of the circuit's connectivity.

Besides being well used for circuit partitioning, the graph representation presents limitations, such as:

- Inability to represent multi-qubit gates while standard graphs cannot naturally represent gates that act on more than two qubits, such as the Toffoli (CCNOT) gate or other multi-qubit operations.
- Complex interdependencies, while quantum circuits often involve complex entanglement and operations that cannot be accurately captured by simple pairwise edges.

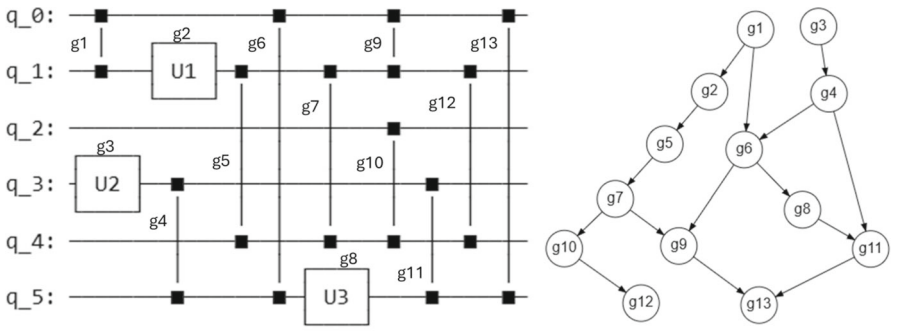
In this context, a Directed Acyclic Graph (DAG) [35] is a finite graph consisting of vertices and directed edges, where the edges have direction, and there are no cycles, that is, it's impossible to start at any vertex v and follow a consistently directed sequence of edges that eventually loops back to v. In the context of quantum circuits, we have:

- Vertices (Nodes): represent quantum operations (gates) or events, such as the application of a gate, measurements, or the initialization of qubits.
- Directed edges (Arrows): indicate the flow of quantum information and dependencies between operations. An edge from vertex u to vertex v implies that the operation at v depends on the outcome or completion of the operation at u.
- Acyclic Nature: ensures that there are no circular dependencies, which aligns with the temporal ordering of operations in a quantum circuit, future operations depend on past ones, but not vice versa.

DAGs provide a clear visual representation of the circuit's flow, aiding in the analysis and debugging of complex circuits. Figure 3 illustrates a static quantum circuit represented as a Direct Acyclic Graph, as defined before.

DAGs are often used to represent the temporal and causal structure of static quantum circuits, allowing for optimization techniques such as gate reordering, parallelization, and critical path analysis. They help identify dependencies between operations, which is crucial for understanding which gates can be executed in parallel and which must wait for others to complete.

The analysis of the quantum circuit depicted in Fig. 3, its corresponding gates-to-operation mapping, and the derived DAG representation highlights how dependencies



a) Static quantum circuit with 6 qubits, 3 unitary gates, and 10 binary gates.

b) DAG graph representing the static quantum circuit.

Fig. 3. Random static circuit with 6 qubits, 3 unitary gates, and 10 binary gates (on left) and the DAG equivalent (on right)

and temporal ordering between gates can be effectively captured using a directed acyclic graph. The DAG formalism clearly models gate-level dependencies and ensures acyclic execution paths, which is crucial for circuit scheduling and partitioning. However, DAGs are inherently limited to representing pairwise dependencies (edges between two nodes), while many quantum operations naturally involve multi-qubit interactions that induce more complex constraints. To overcome this limitation, hypergraph representation becomes advantageous, as it allows modeling multi-qubit gates and shared resources as hyperedges connecting multiple nodes simultaneously. This richer structural representation enables more accurate partitioning strategies that minimize inter-QPU communication and better preserve entanglement, thus motivating our exploration of hypergraph-based approaches for distributed quantum circuit partitioning.

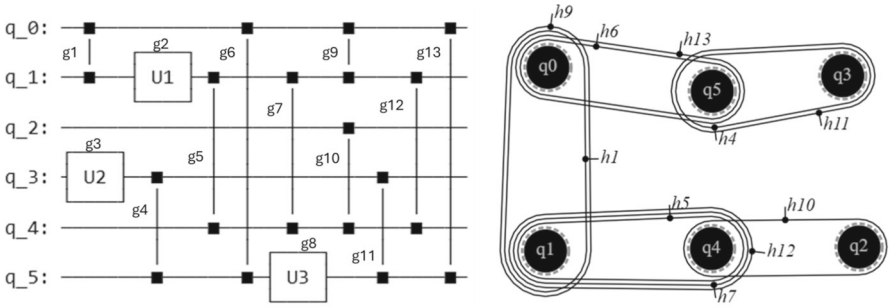
2.4.2 Hypergraph representation of quantum circuits

A primal hypergraph H = (V, E) [16] generalizes the concept of a graph by allowing edges, called hyperedges, to connect any number of vertices. In this definition, vertices V represent qubits, like graph vertices. And hyperedges E represent multi-qubit gates or operations involving multiple qubits simultaneously. In the hypergraph representation, a hyperedge can connect more than two vertices, naturally representing gates like "*CCNOT*" or "*CSWAP*" operations that act on multiple qubits. Hypergraphs can model intricate interdependencies, such as tripartite entanglement, quantum error correction codes, and adaptive operations that involve conditional logic based on measurement outcomes. We can recognize advantages of hypergraph representation against graph representation, such as:

- Expressiveness: accurately models operations involving any number of qubits.
- Adaptability: captures the dynamic behavior of adaptive quantum circuits by including vertices and hyperedges that represent adaptive structures and control flows.
- Complex interdependencies: reflects the true complexity of quantum circuits, including higher-order interactions and dependencies.

Figure 4 illustrates a static quantum circuit represented as a primal hypergraph, when qubits are vertices and gates are hyperedges, supporting more than two vertices per hyperedges. In this picture, we have only binary operations in the input quantum circuit, all hyperedges are formed with only two vertices in this case.

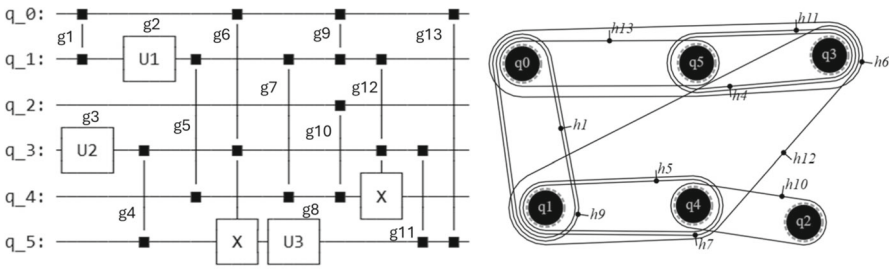
Finally, because hypergraphs are generalization of graphs, we can support complex circuits with gates for more than two qubits. In this case, our hypergraph representation can incorporate the elements of this circuit, with hyperedges, as we have illustrated in Fig. 5.



a) Static quantum circuit with 6 qubits, 3 unitary gates, and 10 binary gates.

b) Hypergraph representation for the same circuit with 6 vertices and 10 hyperedges.

Fig. 4. Random static circuit (**a**) with 6 qubits and only binary operations and the equivalent hypergraph (**b**) with 6 vertices and 10 hyperedges. In this case, we have all the hyperedges with binary connections between vertices



a) Static quantum circuit with 6 qubits, 3 unitary gates, 8 binary gates, and 2 trinary gates (ccx).

b) Hypergraph representation for the same circuit from a) with 6 nodes (qubits) and 10 hyperedges (gates).

Fig. 5. Random static circuit (**a**) with 6 qubits and 10 n-qubits gates, including binary (cz) and ternary gates (ccx), and the hypergraph representation (**b**) for the static quantum circuit with 6 qubits and 10 n-qubits gates, including hyperedges for two CCX gates, the CCX q0,q3,q5 (hyperedge h6) and CCX q1,q3,q4 (hyperedge h12)

While we have hypergraph as a powerful tool for quantum circuit representation, we can work with algorithms to translate quantum circuits into hypergraph representation.

Algorithm 1 implements the baseline circuit normalization variant for fully expanded static quantum circuits described in QASM. It parses the circuit, extracts qubit–gate associations and produces a structural hypergraph-ready representation used as input to the incidence matrix construction stage.

Algorithm 1 Pseudo-code for circuit normalization (variant A): static qasm-based structural normalization and hypergraph preparation.

```
1: Input:
2:    QASM_File_Path: String representing the path to the .qasm file
3: Output:
4:    H = (V, E): Primal hypergraph where
5:       V = set of qubits (vertices)
6:       E = set of quantum gates (hyperedges)
7: Initialization:
8:    V = ∅
9:    E = ∅
10: Read and parse the QASM file:
11:   Open QASM_File_Path for reading
12:   for each line in QASM file do
13:      Trim leading and trailing whitespace from line
14:      if line starts with "qreg" then
15:         Extract the qubit register name and size n
16:         for i from 0 to n-1 do
17:            qubit_id = "q[" + i + "]"
18:            Add qubit_id to V
19:      else if line matches quantum gate operation pattern then
20:         Extract gate type and list of qubits involved, QGates
21:         if QGates is not empty then
22:            Create a hyperedge e connecting all qubits in QGates
23:            Add hyperedge e to E
24:      else
25:         Continue to next line
26:   Close QASM file
27: Return H = (V, E)
```

Algorithm 1 implements the step to parse any static quantum circuit into a primal hypergraphic representation, that can be used in the partitioning process of static circuits using hypergraphic heuristics. This algorithm is planned to be executed on the classical side of a full-stack quantum computer.

Following this way, we have an efficient approach to translating static quantum circuits into hypergraphic representations. We can use this baseline to explore alternatives for future adaptive quantum circuits.

2.4.3 Hypergraph partitioning heuristics

Partitioning a quantum circuit is essential for executing it across multiple quantum processing units (QPUs) in a distributed quantum computing environment. The goal

is to divide the circuit into subcircuits that can be run independently while minimizing inter-QPU communication and maintaining efficient resource utilization. Adopting hypergraph partitioning enhances the segmentation process of quantum circuits, offering benefits over traditional graph-based methods. Hypergraphs provide a better representation of operations involving multiple qubits and complex interconnections, accurately capturing the nuances of quantum gate interactions, especially important for adaptive circuits.

In the essence, the objective of a hypergraph partitioning is to partition the hypergraph H into k disjoint sub hypergraphs H_1, H_2, …, H_k such that each sub hypergraph corresponds to a subcircuit assigned to a specific QPU. We can define some constraints in this process, such as:

- Balance: the size of each partition should respect the qubit capacity of each QPU.
- Minimize cut size: reduce the number of hyperedges that are "cut" (i.e., hyperedges that span multiple partitions), as these represent gates requiring inter-QPU communication.
- Minimize the total weight of the cut hyperedges: reducing the total weight of the cut hyperedges.

While exact solutions to graph and hypergraph partitioning problems are computationally intractable (NP-hard), we can use several heuristics to find near-optimal solutions efficiently.

Based on the literature, the following methods and algorithms are mentioned:

For graph partitioning, Spectral Bisection [36], Kernighan–Lin (KL) algorithm [37], Metis [38], Diffusion-Based Methods [39], Greedy Algorithms [40], Genetic Algorithms [41], Stoer–Wagner (SW) algorithm [42], and Force-Directed Methods [43].

For hypergraph partitioning, techniques such as Recursive Bisection [44, 45], K-way Partitioning [45], the Fiduccia–Mattheyses (FM) algorithm [46], Multilevel Hypergraph Partitioning [47], Hypergraph Balancing [15], and Min-Cut Algorithms [48] are available.

Two prominent heuristics are Kernighan–Lin (KL) [37] for graph partitioning and Fiduccia–Mattheyses (FM) [46] for hypergraph partitioning, used in this research.

The Kernighan–Lin algorithm (KL) [37] heuristic is designed for partitioning graphs into two balanced subsets. It starts with an initial partition of the graph's vertices into two subsets. The heuristic iteratively swaps pairs of vertices between the subsets to reduce the cut size and uses a greedy approach to maximize the decrease (or minimize the increase) in the total edge cut weight at each iteration. KL is broadly used in several experiments for circuit partitioning, besides few limitations, such as:

- Binary partitioning: naturally supports only two partitions, requiring extensions for multi-way partitioning.
- Graph structure: not directly applicable to hypergraph; adapting it to hypergraphs may not capture the complexities of multi-qubit interactions.

Several works already explored KL heuristics in partitioning challenges of quantum circuits [8, 49, 50]. While working on native gates-set based only on unitary and binary gates, standard graphs are good alternatives for the circuit representation in the

partitioning of small circuits. KL algorithm has a worst-case time complexity of O(n3), where n is the number of nodes in the graph. The complexity comes from having to evaluate gain values for every possible swap between the two subsets, followed by repeated iterations until reaching convergence. This cubic complexity limits its application to smaller or medium-sized problems, making it unsuitable for partitioning large quantum circuits.

Fiduccia–Mattheyses algorithm (FM) [46] is designed for efficient multi-way partitioning of hypergraphs. The heuristic starts with an initial partitioning of the hypergraph's vertices into k subsets. Using a priority queue (gain bucket) to select vertices that, when moved to a different partition, yield the greatest reduction in cut size. Each vertex is moved individually rather than swapping pairs, allowing for more flexibility and efficiency. In this way, FM employs balance constraints to ensure partitions remain within desired size limits.

The FM heuristic is a linear-time algorithm with a time complexity of O(E), scaling linearly with the number of hyperedges E, making it highly efficient and scalable for large hypergraphs in distributed environments. By leveraging hypergraph representations and appropriate partitioning heuristics, we can optimize the execution of quantum circuits across multiple QPUs, minimizing communication overhead and improving overall computational efficiency.

3 Approach and methodology

In this research, we used different approaches and methodologies to capture the challenges from static and adaptive quantum circuits, in the process of circuit partitioning. The next sections explore each approach.

3.1 Hypergraph partitioning for static quantum circuits

We introduce three innovative approaches for applying the Fiduccia–Mattheyses (FM) heuristic to partition quantum circuits more efficiently and effectively:

1. Incidence Matrix of a Quantum Circuit as a Graph for Partitioning with FM Heuristic.
2. Variation of FM Heuristic with Minimum and Maximum Constraints for Balanced and Unbalanced Partitions.
3. Expanded Hypergraph Representation for Circuit Partition with FM Heuristic.

3.1.1 Incidence matrix for graphs

The incidence matrix of a quantum circuit provides a convenient way to represent the circuit as a graph, which can then be efficiently partitioned using the FM heuristic. In this graph representation, each node corresponds to a gate in the circuit, and an edge is placed between two nodes if the corresponding gates act on the same qubit.

Applying the FM heuristic to this graph aims to minimize the number of edges crossing between partitions, which directly translates to reducing inter-QPU communication. By grouping together gates that frequently interact, the partitioning minimizes communication overhead and improves the efficiency of distributed execution.

Algorithm 2 constructs the explicit combinatorial hypergraph used for partitioning from a normalized circuit representation produced by Algorithm 1. It generates the hypergraph incidence matrix, defines vertices and hyperedges, and assigns consistent baseline weights required for all downstream partitioning methods.

Algorithm 2 Pseudo-code for incidence matrix generation and baseline weight assignment.

```
1: INPUT:
2:    Hypergraph H=(V,E), where V is the vertex (qubit) set and E is the
hyperedge (quantum gate) set
3: OUTPUT:
4:    Incidence Matrix M representing the Hypergraph H
5: INITIALIZE:
6:    Create an empty matrix M of size |V| x |E|
7: FOR each hyperedge e in E:
8:    FOR each vertex v involved in e:
9:       Set M[v, e] = 1
10: RETURN Incidence Matrix M
```

Algorithm 2 formalizes the standard construction of the incidence matrix representing the quantum circuit as a hypergraph. This step is included for methodological completeness and to explicitly bridge the circuit abstraction with the partitioning stage. In practice, the implemented version used in our experiments extends this construction by assigning weights to hyperedges based on gate counts or aggregated gate costs, ensuring consistency across all partitioning methods evaluated.

It is also worth noting that the structural incidence matrix can become large and sparse for circuits with many qubits or multi-qubit operations, which may affect downstream partitioning efficiency. Beyond the minimal weighting scheme employed in this work, more sophisticated cost models—such as hardware-aware fidelities, noise-dependent weights, or decomposition-based interaction costs remain promising extensions. These richer heuristics could further enhance the expressiveness of the hypergraph representation and open new opportunities for optimization in distributed quantum computing scenarios.

3.1.2 Variation of FM for balanced/unbalanced partitioning

The traditional FM heuristic aims to create balanced partitions, where each subset contains roughly an equal number of nodes. However, in a multi-QPU system, it may be advantageous to create unbalanced partitions based on the computational capabilities or qubit availability of each QPU. We introduce a variation of the FM heuristic that incorporates minimum and maximum constraints on partition sizes. This

allows for greater flexibility in managing workload distribution, enabling the creation of both balanced and unbalanced partitions depending on the system requirements. For example, a more powerful QPU could handle a larger portion of the circuit, while a less capable QPU takes a smaller subset, thereby optimizing overall resource utilization.

Algorithm 3 performs hypergraph partitioning on the explicit incidence matrix hypergraph produced by Algorithm 2. It applies a reduced variant of the Fiduccia—Mattheyses heuristic to compute optimized partitions under balanced or unbalanced size constraints, serving as the core partitioning engine of the proposed framework.

Algorithm 3 Pseudo-code for hypergraph partitioning from constructed hypergraph: reduced FM heuristic with balanced and unbalanced constraints.

```
1. INPUT:
2.    Hypergraph H(V,E) extracted from Algorithm 2 (with E
representing hyperedges)
3.    Minimum and Maximum Partition Sizes (Min_Size, Max_Size)
4.    Initial Partition P1, P2
5. OUTPUT:
6.    Optimized Partition P1, P2
7. PARTITIONING HEURISTIC:
8.    Initialize Gain Buckets for P1 and P2
9.    While there are nodes to be moved:
10.       Select node v from P1 or P2 with maximum gain
11.       IF moving v maintains partition size constraints
                                  (Min_Size <= |P1|, |P2| <= Max_Size):
12.          Move node v to opposite partition
13.          Update Gain Buckets for affected nodes
14.       ELSE:
15.          Skip node v and select the next highest gain node
16.    END WHILE
17. RETURN optimized partitions P1, P2
```

Algorithm 3 consumes only the fully constructed hypergraph generated by Algorithm 2 and does not operate on normalized circuit representations or intermediate hypergraph forms.

In our work, "reduced FM" refers to the variant formalized in Algorithm 3 and differs from classical Fiduccia–Mattheyses in several practical ways. Instead of considering moves of any vertex, we restrict candidates to those on the cut frontier (touching a boundary edge/net), which focuses effort where cuts can shrink and reduces runtime. We also enforce an explicit capacity window at selection time, every move must satisfy the hard bounds Min_Size $\leq$|P1|, |P2|$\leq$ Max_Size (Algorithm 3, lines 11–15), rather than allowing temporary imbalance with later repair. The cost model is circuit-aware: the (hyper)graph is derived from the circuit, with qubits as vertices and (hyper)edge weights reflecting the number/cost of two-qubit interactions, so the

cut directly proxies inter-QPU communication. For convergence control, we retain gain buckets and best-prefix rollback as in FM but add early stopping when no further positive cumulative gain is observed. Finally, to stabilize results for benchmarking, we adopt deterministic tie-breaking and fixed random seeds, an engineering choice beyond classical description.

3.1.3 Expanded hypergraph for FM heuristic

In this approach, we extend the representation of a quantum circuit as a hypergraph by adding groups of gates and dependencies as additional hyperedges. Typically, qubits are vertices, and multi-qubit gates are hyperedges connecting them. We enhance this model by introducing extra hyperedges to represent logical groupings of gates and dependencies beyond direct qubit interactions. These additional hyperedges help the FM heuristic better capture gate dependencies, improving partition efficiency. This expanded hypergraph representation captures both spatial and temporal correlations, enabling more effective segmentation of circuits across QPUs and reducing complex inter-QPU communication. Furthermore, this method allows adaptive construction from dynamic circuits during analysis and partitioning.

The Algorithm 4 implements an extended circuit normalization variant designed for adaptive programs and high-level circuit descriptions. It expands control structures, groups semantically related gates, and generates an enriched structural hypergraph-ready representation prior to incidence matrix construction.

Algorithm 4 Pseudo-code for circuit normalization (variant B): pattern-aware and adaptive normalization with expanded hypergraph construction.

```
1: INPUT:
2:     Quantum Circuit C with Qubits Q and Gates G
3: OUTPUT:
4:     Expanded Hypergraph H with Vertices V and Hyperedges E
5: INITIALIZE:
6:     Create an empty hypergraph H
7:     Create vertices V representing each qubit in Q
8: FOR each gate g in G:
9:     IF g is CZ or CX with the same control qubit across different target qubits:
10:        Create a new hyperedge e for group of gates with same control qubit
11:        Add all affected qubits to hyperedge e
12:        Add hyperedge e to H
13:    ELSE IF g is part of a control structure (IF, WHILE, FOR):
14:        Create a new hyperedge e for the control structure
15:        Add all qubits involved in the structure to hyperedge e
16:        Add hyperedge e to H
17:    ELSE:
18:        Create a standard hyperedge for gate g connecting relevant qubits
19:        Add hyperedge to H
20: RETURN Expanded Hypergraph H
```

In Algorithm 4, we have the following sections: lines 1–2 (INPUT), the algorithm takes as input a quantum circuit, including a set of qubits and a set of gates. Lines 3–4 (OUTPUT), the output is an expanded hypergraph representing the quantum circuit. Lines 5–7 (INITIALIZE), the algorithm initializes an empty hypergraph and creates vertices for each qubit. Lines 8–19, we have a loop through each gate in the circuit: Lines 9–12: If the gate is CZ or CX (illustrated here as an example) with the same control qubit for different target qubits, create a hyperedge that groups these gates together. Lines 13–16: If the gate is part of an adaptive control structure (e.g., IF, WHILE, FOR), create a hyperedge to represent the control logic. Lines 17–19: Otherwise, create standard hyperedge connecting the qubits involved in the gate. At the end, line 20 returns the expanded hypergraph that captures both spatial and temporal correlations of the circuit.

It is important to note that algorithm 2 operates exclusively on the normalized circuit representations generated by the normalization variants (Algorithms 1 and 4) as presented here and is the only stage that produces the full incidence matrix hypergraph used for partitioning.

3.2 Threading model for adaptive quantum circuits

Threading models in classical distributed systems provide frameworks for managing concurrent execution across multiple processing units. In a common approach, threading models can classify applications in two categories: single-threaded and multi-threaded, to manage access and availability to shared resources and ensure thread safety in runtime. This is still a good alternative to differentiation of portions of code safe for parallelism and isolation. Key concepts related to threading models are:

- Logical separation: it divides tasks into logical units (threads) that can be executed independently or in coordination.
- Synchronization mechanisms: it ensures safe access to shared resources through locks, semaphores, or message passing.
- Scalability and performance: it optimizes resource utilization and minimizes communication overhead to enhance performance.

As we discussed before, adaptive quantum circuits present several challenges when partitioning for multi-QPU environments:

- Dynamic gate sequences: intermediate measurements and classical control instructions can alter the sequence and structure of subsequent gates, making it difficult to predict and partition the circuit at compile time.
- Dependency management: adaptive operations introduce dependencies between different parts of the circuit, complicating the identification of independent subcircuits that can be executed concurrently on separate QPUs.
- Communication overhead: the need for real-time communication between QPUs to handle adaptive control flow increases latency and potential error rates, undermining performance gains from distribution.
- Complex partitioning logic: traditional graph and hypergraph partitioning fails to account for the evolving topology of adaptive circuits, requiring more sophisticated approaches to identify optimal partitions dynamically in a cut finding automatically approach.
- Resource allocation: efficiently allocating quantum and classical resources across multiple QPUs while maintaining state consistency is non-trivial.
- Synchronization: finally, ensuring that intermediate measurements and conditional operations are correctly synchronized across QPUs to maintain the integrity of the computation is still a complex task.

To address the challenges of partitioning adaptive quantum circuits, we propose adopting a threading model inspired by classical distributed systems. This approach involves treating logical constructs within the quantum circuit as threads, which can be assigned to different QPUs.

The key components of this approach are as follows:

- Logical Thread Identification.
- Quantum Resource Manager.

3.2.1 Logical thread identification

Given the complexity of automatically identifying optimal partitions for adaptive circuits, logical thread identification is a critical step in the partitioning process. In this approach, developers annotate quantum circuits to explicitly mark segments or functions that can be executed as independent threads. These annotations serve as guides for determining portions of the circuit that are suitable for distribution across multiple QPUs. By allowing developers to use their expertise and domain knowledge to identify reusable gate sequences, conditional branches (such as IF-ELSE blocks), and loop constructs (FOR or WHILE loops), this approach makes the identification of potential threads more efficient and accurate.

Algorithm 5 illustrates an adaptive quantum circuit with primitives in OpenQASM [53, 54], proposed in this research as indicatives for different threads.

Algorithm 5 Example of adaptive quantum circuits with primitives for threading indication.

```
1: OPENQASM 3.0;
2: include "qelib1.inc";
3: qreg q[6];
4: creg c[3];

5: // Thread 1: Initialization and Preliminary Operations
6: #pragma qpu_begin QPU1
7: h q[0]; h q[1]; h q[2];
8: x q[0]; x q[2];
9: cx q[0], q[1]; cx q[1], q[2];
10: #pragma qpu_end
11: #pragma qpu_begin QPU2
12: h q[3]; h q[4]; h q[5];
13: x q[4]; x q[5];
14: cx q[3], q[4]; cx q[4], q[5];
15: #pragma qpu_end

16: // Thread 2: Conditional Operations (IF-ELSE Block)
17: if (c[0]) {
18: // Thread 2A: Execute on QPU1 since it involves q[0] and q[1]
19: #pragma qpu QPU1
20: cx q[0], q[1];
21: while (c[0] < 2) {
22: x q[2];
23: }
24: } else {
25: // Thread 2B: Execute on QPU1 for q[1]
26: #pragma qpu QPU1
27: z q[1];
28: }

29: // Thread 3: Hadamard Gates (Independent across QPUs)
30: #pragma qpu_begin QPU1
31: for (int i = 0; i < 3; i++) {
32: h q[i];
33: }
34: #pragma qpu_end
35: #pragma qpu_begin QPU2
36: for (int i = 3; i < 6; i++) {
37: h q[i];
38: }
39: #pragma qpu_end

40: // Thread 4: Measurement Phase
41: #pragma qpu_begin QPU1
42: measure q[0] -> c[0];
43: measure q[1] -> c[1];
44: measure q[2] -> c[2];
45: #pragma qpu_end
46: #pragma qpu_begin QPU2
47: measure q[3] -> c[0];
48: measure q[4] -> c[1];
49: measure q[5] -> c[2];
50: #pragma qpu_end
```

In Algorithm 5, we have several indicators and threading primitives proposed to support the circuit segmentation between QPU1 and QPU2. Each directive indicates the boundaries of logical segments (or threads) that can be executed independently by different QPUs. Following Algorithm 5, we have:

- Thread 1—Initialization & local entanglement (Lines 6–15): QPU1 (Lines 6–10) prepares q[0..2] with H/X and entangles locally via cx q[0],q[1] and cx q[1],q[2] (updated to keep all gates intra-QPU1). QPU2 (Lines 11–15) analogously prepares q[3..5] and entangles via cx q[3],q[4] and cx q[4],q[5]. Because no gate spans qubits across QPUs, both blocks can execute independently and in parallel, minimizing inter-QPU communication.
- Thread 2—Conditional (adaptive) operations on QPU1 (Lines 17–28): A classical predicate c[0] selects between two branches, both bound to QPU1: either reinforce entanglement with cx q[0],q[1] and repeatedly apply x q[2] while the condition holds, or apply z q[1]. This illustrates adaptive control flow driven by classical data while preserving qubit locality on QPU1.
- Thread 3—Parallel Hadamards (Lines 30–39): Each QPU independently applies H gates to its own subset (q[0..2] on QPU1; q[3..5] on QPU2). This stage is fully parallel and maintains the partitioned execution model.
- Thread 4—Measurement (Lines 41–50): QPU1 measures q[0..2] into c[0..2]; QPU2 measures q[3..5] to the same classical bits. This demonstrates result extraction on each device; in practice, implementations may map to distinct classical registers or aggregate results to avoid overwriting.

Note that, the algorithm as written assumes that this inter-QPU gate is either handled through pre-established entanglement between the QPUs or deferred to an inter-QPU communication protocol. In practice, operations within each QPU (on qubits it owns exclusively) are independent, while operations that span QPUs introduce synchronization and communication.

The benefit of this method is that it allows adaptive circuits to be partitioned more deliberately and effectively by using manual insight. Logical thread identification in a multi-QPU environment reduces the dependence on automatic partitioning algorithms, which often struggle with the complexity of adaptive logic. As a result, workloads are more optimized, and the scalability of the quantum system is enhanced.

3.2.2 Quantum resource manager (QRM)

In this research, to support the partitioning process with static and adaptive quantum circuits, we proposed a quantum resource manager (QRM) layer, responsible for orchestrating the execution of partitioned threads across multiple QPUs.

Figure 6 illustrates a full-stack quantum computer architecture, where proposed the QRM layer to support circuit cutting with hypergraph partition and threading model in a multi-QPU environment.

Here's an overview of each layer:

- Programming layer: this is where quantum algorithms are designed. Developers create quantum programs and annotate logical threads to identify potential segments

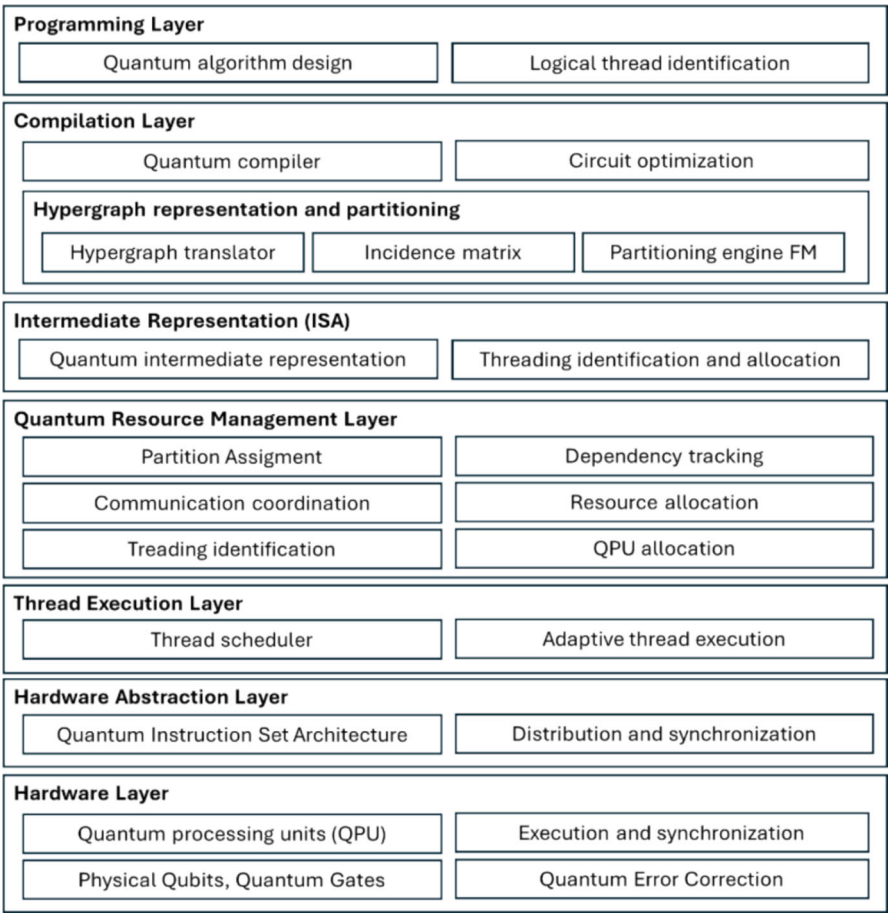


Fig. 6. Layers of a quantum resource manager (QRM) as proposed in this work, to support and coordinate adaptive quantum circuits in full-stack quantum computers

for distribution across multiple QPUs. This is possible for static and adaptive quantum circuits. It involves the developer's insight in determining which parts of the quantum circuit can benefit from parallel execution.

- Compilation layer: quantum circuits undergo compilation, optimization, and transformation into intermediate representations here. Key components include the quantum compiler that translates high-level quantum algorithms into machine-compatible circuits and a partitioning engine that uses hypergraph techniques to partition the circuit. It represents a crucial step for preparing the quantum circuits for distribution.
- Intermediate Representation (ISA): this layer acts as a bridge between high-level quantum circuits and the actual hardware instructions. It involves identifying different threads (parts of the circuit) and allocating them to appropriate QPUs.

- Quantum Resource Management Layer: this layer is managed by the QRM and involves multiple roles:
  1. Partition Assignment allocates quantum threads to QPUs, ensuring efficient distribution based on resource constraints.
  2. Dependency Tracking ensures that execution across QPUs maintains the correct sequence, respecting inter-thread dependencies.
  3. Communication Coordination manages data and state transfers between QPUs to minimize latency and maintain coherence.
  4. Thread Identification and Allocation identify suitable segments for threading and assigns them to available QPUs for optimal execution.

Resource Allocation dynamically manages quantum and classical resources, ensuring efficient use and avoiding resource contention.

QPU Allocation selects the best QPU for executing specific circuit segments, balancing workload and ensuring smooth, efficient execution across all available QPUs.

- Thread Execution Layer: this layer is responsible for executing the divided tasks (threads) across the different QPUs. It schedules threads, manages how they adapt when measurements are taken, handles the results from measurements, and keeps quantum states synchronized across the QPUs.
- Hardware Abstraction Layer: this layer bridges the quantum instructions from resource management to the actual hardware. It includes the quantum instruction set architecture (QISA) for translating logical operations into executable instructions for QPUs, ensuring compatibility between software commands and physical quantum hardware.
- Hardware Layer: this is the physical layer consisting of quantum hardware—quantum processing units (QPUs), physical qubits, and quantum gates. It also incorporates quantum error correction, which is essential for mitigating the effects of errors that arise during quantum computations.

3.3 Algorithms and execution in the full-stack architecture

Figure 6 presents the full-stack architecture of a distributed quantum computer and highlights where each stage of our method is executed. In this architecture, the Compilation Layer, specifically the subsection labeled Hypergraph representation and partitioning, is responsible for invoking Algorithms 1–4.

The layers above and below consume or propagate the resulting artifacts but do not execute these algorithms directly.

3.3.1 Hypergraph translation (Algorithms 1 & 4).

The leftmost box in this subsection, Hypergraph translator, is where Algorithm 1 (standard translation) and Algorithm 4 (expanded and adaptive translation) are executed.

These algorithms take the optimized quantum circuit produced by the quantum compiler and circuit optimization boxes in the Compilation Layer and convert it into the structural hypergraph representation used downstream.

- Algorithm 1 is invoked when the circuit is static or already expressed in fully expanded gate form.
- Algorithm 4 is invoked when the circuit contains composite gate patterns or adaptive constructs requiring structural unfolding.

For the static execution, algorithms 1 or 4 are called once, right after compilation. For adaptive execution, the adaptive quantum program (Programming Layer) triggers new compilation windows after each measurement event. In these cases, the Compilation Layer is re-entered, and the Hypergraph translator calls Algorithm 4 (or Algorithm 1 when applicable) to regenerate the active subcircuit fragment.

So, algorithms 1 and 4 represent alternative normalization variants within the same compilation stage; exactly one of them is invoked depending on whether the input circuit is static or adaptive.

3.3.2 Incident matrix construction (Algorithm 2)

The central box of the same subsection, Incidence matrix, is where Algorithm 2 is executed.

This stage consumes the structural hypergraph produced by Algorithm 1/4 and builds the explicit incidence matrix H used for cost modeling and partitioning.

- On static mode: Algorithm 2 is executed once.
- On adaptive mode: Algorithm 2 is executed at every re-compilation step, since the hypergraph must reflect the currently active subcircuit.

3.3.3 Hypergraph partitioning (Algorithm 3)

The rightmost box in this subsection, Partitioning engine FM, is responsible for invoking Algorithm 3, our reduced FM hypergraph partitioner.

Algorithm 3 takes the incidence matrix produced by Algorithm 2 and computes the partition assignment that minimizes communication and cross-QPU entanglement costs.

For the KL baseline comparison, the same incidence matrix H produced by Algorithm 2 is used as input before projecting it to a pairwise graph.

3.3.4 Interaction with quantum resource manager layer

The quantum resource management layer does not execute Algorithms 1–4. Instead, it consumes the partitions produced by Algorithm 3 and provides:

- partition assignment across QPUs,
- dependency tracking,
- communication coordination,
- resource allocation and threading identification,
- QPU allocation.

These are runtime responsibilities that occur after the Compilation Layer has executed Algorithms 1–3.

So, we have:

- Algorithms 1 and 4 are entry-point transformations. The most important clarification here is: we can use Alg.1 or Alg. 4 with the rest of the pipeline. While using Alg. 1, we are not taking benefits from the group/template expansion in the normalized gate list. While using Alg.4, we are considering templates of groups of gates, expanding the final normalized gate list.
- Algorithm 2 is the canonical hypergraph constructor:
- Algorithm 3 is the consumer of Algorithm 2's output

4 Experiments and results

4.1 HyPAQ framework

We consolidated the many algorithms presented in this work in a framework called HyPAQ—Hypergraph Partitioning for Adaptive Quantum Circuits. This framework consolidates the algorithms for hypergraph translation, incidence matrix construction, groups port for expanded hypergraph, and the variation for FM heuristic for balanced and unbalanced partition.

We also have algorithms for recognition of threads identification, and hypergraph analysis for adaptive quantum circuits, simulating the behavior of the quantum resource management (QRM) presented in this research. The public link is the following: https://github.com/hypaq/.

4.2 Benchmark circuits

To test the impact of HyPAQ approach in real scenarios, we select a series of benchmark circuits, in static and adaptive versions.

For static circuits, we selected the following benchmark algorithms:

- Variational Quantum Eigensolver (VQE): VQE is a hybrid quantum–classical algorithm that uses parameterized quantum circuits to find the ground state energy of a Hamiltonian. By iteratively optimizing circuit parameters through classical feedback based on measurement results, VQE adapts its operations dynamically, making it adaptable to different quantum systems and noise levels.
- Quantum Phase Estimation (QPE): QPE is a fundamental quantum algorithm that estimates the eigenvalues (phases) of a unitary operator. It typically requires several qubits proportional to the desired precision, utilizing a quantum register for the estimation and applying the quantum Fourier transform.
- Random Circuits (RND): random circuits are quantum circuits composed of randomly selected quantum gates applied to a set of qubits.
- For adaptive circuits, we selected the following list of algorithms:
- Variational Quantum Eigensolver (VQE), in the same approach as used for static benchmark circuits.

- Iterative Quantum Phase Estimation (IQPE): IQPE is an efficient method for estimating the eigenvalues (phases) of a unitary operator using a minimal number of qubits, typically two qubits. It works by repeatedly applying controlled unitary operations and updating the phase estimate based on measurements. These adaptive steps create dependencies that HyPAQ's hypergraph-based partitioning can efficiently handle.
- Random Circuits (RND): random circuits are quantum circuits composed of randomly selected quantum gates applied to a set of qubits. In this adaptive version, each circuit presents a combination of several classical instructions, such as "FOR", "WHILE", "IF", and "ELSE".

4.3 Experiments with static circuits

We conducted three experiments to evaluate the proposed methods. The first series focused on assessing the performance and impact of the reduced FM heuristic (presented in Algorithm 3) as implemented in the HyPAQ framework, under increasing complexity for both full and random circuits. The second series explored the effectiveness of the *reduced FM* heuristic on benchmark circuits with sizes of 16, 24, 32, 40, and 48 qubits. Finally, the third series analyzed the influence of the threading model on adaptive quantum circuits, reporting the results of the proposed threading-based segmentation approach.

For static circuits, we benchmark an optimized variant of the Kernighan–Lin (KL) heuristic [37] against multiple configurations of the reduced FM heuristic integrated into the HyPAQ framework. In our usage, "KL" denotes a KL baseline engineered in the spirit of modern partitioners [38, 48]: the circuit is modeled as a weighted interaction graph (qubits as unit-weight vertices; edge weights proportional to the number/cost of two-qubit gates), subject to QPU capacity constraints with a small imbalance ϵ. To speed and stabilize the classical KL swap routine without changing its objective, we restrict candidate swaps to the cut frontier and employ early stopping with a patience parameter p. These refinements preserve KL's logic while reducing search overhead, providing a strong and reproducible baseline for circuit cutting comparisons.

In this context, reduced FM refers to a streamlined version of the traditional Fiduccia–Mattheyses (FM) graph partitioning heuristic, adapted within HyPAQ to support faster bipartitioning of large circuits while maintaining balanced or intentionally unbalanced partitions, depending on the scenario. This adaptation selectively reduces the number of passes and limits candidate moves to improve runtime efficiency without significantly sacrificing cut quality. The experiments considered two types of static circuits, full circuits and random circuits, to evaluate the trade-offs between partitioning speed and cut performance across different circuit structures.

A full circuit is a configuration where every qubit is connected to every other qubit through possible two-qubit operations. This means that all qubits interact, creating maximum entanglement across the entire circuit. In a full circuit with "n" qubits, each qubit pairs with all others, resulting in the highest number of two-qubit gates. For instance, with 3 qubits (q0, q1, q2), there would be gates between q0 and q1, q0 and q2, and q1 and q2. With 4 qubits, each qubit interacts with every other one, such as q0-q1, q0-q2, q0-q3, and so on. This structure provides maximum connectivity, making it

ideal for studying the effects of circuit partitioning and entanglement in both balanced and unbalanced partitions. Full circuits are often used to analyze quantum algorithms that require strong interactions between all qubits.

A random circuit is a type of circuit where the sequence of gate operations is chosen randomly. This randomness involves selecting the quantum gates and their qubit targets from a predefined set, effectively creating a unique circuit each time. Random circuits are frequently used as a benchmark to evaluate the performance of quantum processors because they produce complex quantum states that are challenging to simulate on classical computers, making them an excellent test for demonstrating quantum supremacy or the computational advantage of quantum systems.

For each family of benchmark circuits, such as VQE, QPE, IQE, and RND, there are circuits ranging for 16, 24, 32, 40, and 48 qubits, implemented in Qiskit platform with OpenQASM 3 for native set of gates from IBM Heron processor, that is currently supporting adaptive circuits. Doing this way, we could explore the OpenQASM 3 language and the primal adaptive hypergraph representation for each circuit, preparing the series of experiments using hypergraph partitioning heuristics, such as classic KL, reduced FM, mid-cut approach, and HyPAQ approach.

We generated a series of benchmark circuits using the MQT Bench tool with various configurations [55]. Accordingly, we selected circuit sizes ranging from 4 to 120 qubits, as this reflects the practical limits of the tool, which is unable to generate valid benchmark circuits with more than 120 qubits for certain configurations.

Figure 7 compares the optimized Kernighan–Lin (KL) baseline with our reduced FM heuristic on static quantum circuits, full and random, over 4 to 120 qubits. Here, "KL" denotes the optimized KL described in Sec. IV-C, and "reduced FM" is the variant formalized in Algorithm 3. Reduced FM adapts to the input structure: when the interaction hypergraph contains only pairwise nets, it is collapsed to an ordinary graph

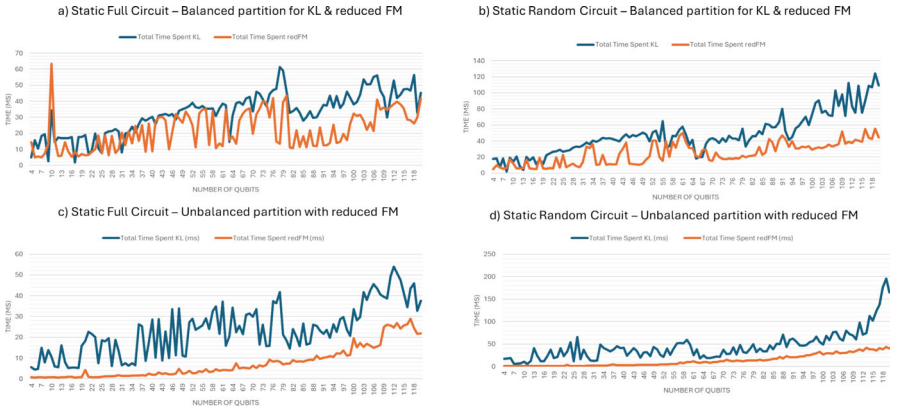


Fig. 7. Partitioning time (ms) for bipartitioning static Full (4–120 qubits) and Random (4–120) circuits using KL (optimized Kernighan–Lin; Sec. IV-C) and HyPAQ (reduced FM; Algorithm 3). Results are averaged over multiple runs. Panels (**a**) and (**b**) show balanced bipartitions ($\approx$ 50–50), and panels (**c**) and (**d**) show unbalanced bipartitions ($\approx$ 75–25). Cut sizes are reported separately to assess partition quality alongside runtime

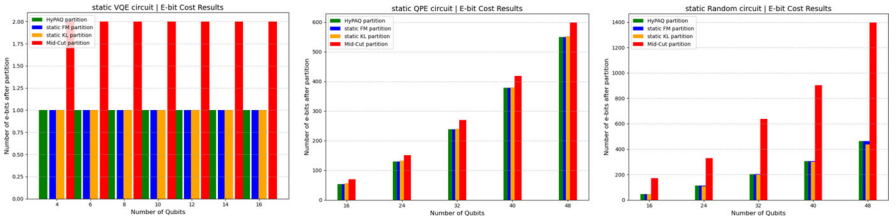


Fig. 8. EBITS cost for bipartitioning static VQE (4–16 qubits), QPE (16–48), and Random (16–48) circuits using KL (optimized Kernighan–Lin; Sec. IV-C), FM (classical Fiduccia–Mattheyses), HyPAQ (reduced FM; Algorithm 3), and Mid-Cut. All results use balanced bipartitions ($\approx$ 50–50) and are averaged over multiple runs; lower is better

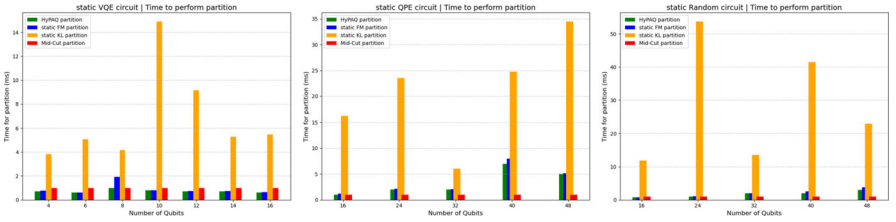


Fig. 9. Partitioning time (ms) for the same static VQE (4–16), QPE (16–48), and Random (16–48) benchmarks using KL (optimized), FM (classical), HyPAQ (reduced FM; Algorithm 3), and Mid-Cut under balanced bipartitions ($\approx$ 50–50). Results are averaged over multiple runs

and the graph variant is applied; otherwise, the hypergraph variant is used, preserving the objective while reducing runtime. Panels (a) and (b) report balanced bipartitions, whereas panels (c) and (d) consider unbalanced bipartitions ($\approx$ 75–25 split). Across all settings, reduced FM attains lower partitioning time than KL, particularly as qubit count grows, while delivering comparable cut sizes (reported separately). This behavior aligns with the design choices detailed in Sec. IV-C (frontier-focused search and early stopping), which make reduced FM more scalable on large circuits.

Figures 8 and 9 report results for four approaches: KL (the optimized Kernighan–Lin baseline, Sec. IV-C), FM (classical Fiduccia–Mattheyses), HyPAQ (our reduced FM variant, Algorithm 3), and Mid-Cut (a median-cut baseline). In terms of partition quality (Fig. 8), HyPAQ and FM achieve the lowest ebit costs across VQE, QPE, and Random circuits, with only minor, circuit-dependent variations between them. KL generally incurs higher ebit costs, especially for QPE and Random, while Mid-Cut performs worse and degrades as the number of qubits increases.

In terms of runtime (Fig. 9), HyPAQ and FM are the fastest and scale smoothly with problem size; KL is substantially slower and exhibits spikes at larger sizes; Mid-Cut is time-competitive but trades off significant partition quality. All results concern static circuits and balanced bipartitions and are averaged over multiple runs; detailed cut-size statistics are reported separately.

Taken together, our static circuit experiments establish a clear baseline for partitioning under fixed gate lists and stable interaction graphs: the reduced FM variant

(HyPAQ) and classical FM deliver the best ebit costs and the most favorable time–scaling, while the optimized KL remains competitive in quality but is significantly slower, and Mid-Cut, though time-competitive, consistently sacrifices partition quality. These results hold full, random, VQE, and QPE benchmarks, for both balanced and (when tested) unbalanced bipartitions, and they validate the value of frontier-focused search and early stop on circuit-derived (hyper)graphs. However, the static setting assumes a fixed topology with no mid-circuit measurements or feed-forward, and thus avoids runtime effects, latency, outcome-dependent control, and reconfiguration, that dominate practical multi-QPU workloads. In the next section we therefore turn to adaptive quantum circuits, where classical outcomes steer subsequent quantum operations, partitions evolve over time, and scheduling must reconcile locality with on-the-fly communication and synchronization. This shift introduces new challenges, incremental (re)partitioning, thread-aware scheduling, outcome-conditioned cut management, and explicit handling of remote two-qubit operations, that motivate our co-design of threading primitives and partitioning within the HyPAQ framework.

4.4 Experiments with adaptive circuits

Transitioning to adaptive quantum circuits, we investigate the impact of threading models using benchmark algorithms on both full circuit and random circuits style. We explore several combinations of threading primitives for partitioning the circuit across two QPUs. Figure 10 illustrates the results of our experiments, where each data point represents a fixed circuit size (ranging from 16 to 48 qubits) executed under different numbers of QPUs (from 2 up to 16). The x-axis in Fig. 10 refers to the number of QPUs used, not the number of qubits, and the circuits are partitioned accordingly.

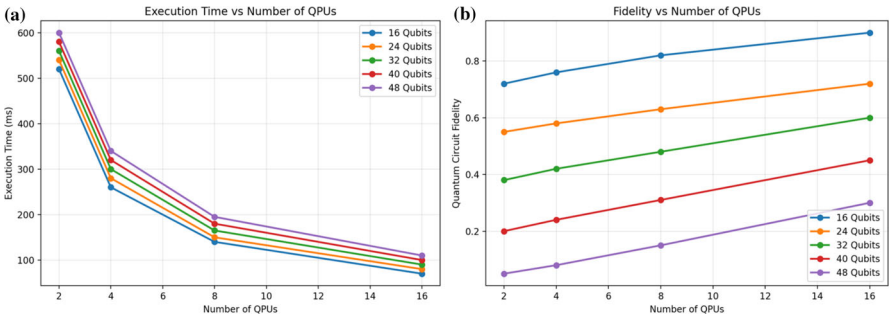


Fig. 10. Performance of adaptive random circuits as a function of the number of QPUs and circuit size. Panel (**a**) shows the average execution time (in milliseconds) as the number of QPUs increases from 2 to 16, for circuits of sizes 16, 24, 32, 40, and 48 qubits. The results demonstrate how parallelizing the workload across more QPUs reduces execution time, with diminishing returns as the number of QPUs approaches the circuit size. Panel (**b**) reports the quantum circuit fidelity for the same configurations, showing that fidelity improves with more QPUs, particularly for smaller circuits, as increased parallelism reduces inter-QPU communication overhead. Communication costs were modeled assuming a uniform penalty per inter-partition gate, reflecting realistic NISQ-era constraints such as entanglement generation and classical coordination latency

For the estimation of time, here are some gate times extracted from the *ibmq_manila* backend:

- Single-Qubit Gates: approximately 35 ns
- CNOT Gate (CX): ranges from 300 to 500 ns depending on the qubit pair.
- Measurement time: approximately 7000 ns (7 μs).

To evaluate the impact of partitioning an adaptive quantum circuit without access to a real-world multi-QPU system, we simulated the behavior and estimated key performance indicators such as execution time, communication overhead, and circuit fidelity in a multi-QPU environment. To do that, we considered three components:

- Modeling communication and computation costs: assigning costs to both quantum operations within QPUs and communication between QPUs to simulate performance metrics.
- Estimating cross-QPU communication overheads: modeling the overhead involved in coordinating across different QPUs.
- Fidelity reduction modeling: modeling how fidelity changes with an increasing number of inter-QPU communications, reflecting errors introduced by needing to entangle and disentangle across QPUs.

Using this approach, it was possible to execute several experiments, comparing results for different combinations of circuits. This approach can be calibrated in the future, against real numbers from different platforms. Every simulation can take three steps:

- Segmenting the circuit based on co-design threading approach, where each segment of the circuit is mapped to a specific QPU by the developer.
- Defining metrics such as execution time and fidelity loss for the partitioned circuit, depending on the number of QPUs. We simulate communication costs that arise when multiple QPUs need to interact to exchange quantum states.
- Running experiments to simulate the effect of partitioning on communication overhead and overall fidelity.

The performance metrics to evaluate are the execution time (the time for circuit execution, considering both within-QPU operations and cross-QPU operations), communication overhead (the delay due to sending quantum states between different QPUs), and the fidelity (the expected loss in fidelity when increasing the number of inter-QPU operations).

Algorithm 6 illustrates the method to estimate the impact of co-design threading modeling with adaptive quantum circuit using primitives in OpenQASM, proposed in this work as indicatives for different threads. As results, this algorithm returns the impact for execution time and fidelity after the circuit segmentation for different combinations of #qpus and #threads.

Algorithm 6 Estimating impact of co-design threading approach with adaptive quantum circuits.

```
INPUT:
folder_path = "path/to/qasm/files" // Path containing multiple QASM files.
num_qpus_config = [2, 4, 8, 16] // Maximum number of QPUs available for partitioning.
communication_penalty_factor = 1.5 // Overhead multiplier for cross-QPU communication.
fidelity_degradation_factor = 0.02  // Fidelity loss per cross-QPU communication.
base_execution_time = 1000 // Base execution time in milliseconds for a single QPU without segmentation
INITIALIZE:
    Create a 2D array execution_times to store execution time results
    Create a 2D array fidelity_results to store fidelity results
FOR each QASM file in folder_path:
    Parse the QASM file to extract quantum circuit characteristics:
      num_qubits = Extract number of qubits from QASM
      num_gates = Extract total number of gates from QASM
      circuit_depth = Extract circuit depth from QASM
      num_binary_gates = Count number of two-qubit gates (e.g., cx, cz) in the QASM file
     num_threads = Count "#pragma thread_session" primitives // Use this to define QPU allocation
    IF num_threads > len(num_qpus_config):
       num_threads = len(num_qpus_config) // Limit threads to the number of available QPUs
    FOR each available_qpus in num_qpus_config:
       IF available_qpus >= num_threads:
          num_qpus_used = num_threads   // Use the number of threads as the number of QPUs
       ELSE:
          CONTINUE  // Skip this configuration since available QPUs are fewer than required threads
       qubits_per_qpu = num_qubits / num_qpus_used  // Calculate qubits per QPU
       // Estimate cross-QPU communications
       cross_qpu_communications = Calculate cross-QPU operations based on num_binary_gates, num_qubits, and qubits_per_qpu
       // Model the execution time
       intra_qpu_time = base_execution_time / num_qpus_used * (1 + num_gates / num_qubits) // Increase based on gate density
       cross_qpu_time = communication_penalty_factor * cross_qpu_communications * (circuit_depth / num_qpus_used) // Penalty scales with communication depth
       total_execution_time = intra_qpu_time + cross_qpu_time
       Store total_execution_time in execution_times
       // Model the fidelity of the quantum circuit
       base_fidelity = 0.95  // Assume a high initial fidelity
34 fidelity_loss = cross_qpu_communications * fidelity_degradation_factor
       final_fidelity = base_fidelity - fidelity_loss
       Store max(final_fidelity, 0) in fidelity_results
OUTPUT:
    execution_times
    fidelity_results
```

Algorithm 6 estimates the impact of a co-design threading approach on adaptive quantum circuits, focusing on execution time and fidelity when distributed across multiple QPUs. The algorithm analyzes quantum circuits represented as QASM files and evaluates different configurations of QPU allocations to optimize performance. Following Algorithm 6 we have:

- INPUT: The inputs are detailed in lines 2–6, including a folder path to QASM files, a list of QPU configurations ("[2, 4, 8, 16]"), a communication penalty factor ("1.5"), a fidelity degradation factor ("0.02"), and a base execution time ("1000 ms"). These inputs guide the partitioning process and impact calculations regarding the execution cost and fidelity of quantum circuits.
- INITIALIZE: In lines 8–9, the algorithm initializes two 2D arrays, "execution_times" and "fidelity_results", to store results for execution time and fidelity, respectively. For each QASM file (lines 10–16), the algorithm extracts circuit characteristics such as the number of qubits, gates, circuit depth, and binary (two-qubit) gates. The "thread_session" directive is also counted to determine the number of threads, which helps decide QPU allocation.
- Execution Flow: For each QPU configuration (line 19), the algorithm checks if the available QPUs meet the thread requirements (lines 20–23). It calculates the qubits allocated per QPU (line 24) and estimates cross-QPU communications (line 26) based on the extracted characteristics. The intra-QPU execution time and the additional time penalty due to cross-QPU communications are then modeled (lines 28–30), resulting in the total execution time that is stored in "execution_times" (line 31). Fidelity is modeled starting from a base value of "0.95" (line 33), and losses are estimated based on cross-QPU communication (lines 34–35), with results saved in "fidelity_results" (line 36).
- OUTPUT: The output consists of the arrays "execution_times" and "fidelity_results" (lines 38–39). These provide insights into how different QPU configurations affect the performance and fidelity of the quantum circuits. This helps evaluate optimal partitioning strategies to reduce execution time and maintain high fidelity across multiple QPUs.

By using primitives such as "thread_session" inside the quantum circuit, the developer's insight dictates how to partition the circuit. This ensures:

- Manual optimization: the developer may have a better understanding of which segments should be parallelized to minimize cross-QPU interactions.
- Reduced complexity: no need for hypergraph partitioning or other sophisticated automatic segmentation techniques with adaptive quantum circuits; the number of threads strictly dictates the number of QPUs.
- Potential overhead: if the available QPUs exceed the number of threads, the system will not use additional QPUs, potentially leaving some computational resources idle.

Figure 10 examines adaptive random circuits as we increase the number of QPUs from 2 to 16 for circuit sizes 16–48. In all runs we assume identical QPUs, a uniform penalty per cross-QPU interaction, and a partitioning strategy that tries to keep strong interacting qubits together.

From Fig. 10, panel (a) shows the expected runtime effect: adding QPUs lets us split the work, so each device executes a shorter slice of the circuit and total time drops. The benefit is large when going from very few QPUs to a moderate number, then tapers off once the critical path is already short, hence the diminishing returns beyond ~ 8 QPUs. Panel (b) shows fidelity improving as we add QPUs. Two opposing factors drive this curve. On the plus side, less work per QPU means fewer local errors accumulate on any single device. On the minus side, every cut between partitions creates cross-QPU communication that can introduce extra errors. In our experiments, we used a simple model with a uniform penalty for each cross-QPU interaction, and we partitioned the circuits so that strongly interacting qubits tend to stay together. Under these settings, the "less work per QPU" benefit dominates across the tested range, which is why the fidelity rises almost linearly.

In the 16-qubit case, partitioning keeps cross-QPU interactions relatively low even when each QPU holds roughly a single qubit, so fidelity continues to improve up to 16 QPUs. This is not a general rule. For larger or denser circuits, splitting into many tiny partitions typically creates more cross-QPU interactions, and the communication penalty grows; in those cases, fidelity will usually peak before reaching the number of qubits. In practice, there is a hardware- and circuit-dependent "sweet spot" in the QPU count where fidelity is maximized.

We run all partitioning experiments and simulations in a local machine with 32 GB RAM of memory, 12th Gen Intel® Core ™ i7 1270p, 2200 MHz, 12 Core(s), and 16 Logical Processors. For the estimation of execution time of each quantum gate, we used the average time per instruction from IBM.

5 Discussion and conclusions

Partitioning adaptive quantum circuits in multi-QPU environments presents significant challenges due to the dynamic nature of gate sequences and dependencies introduced by intermediate measurements and classical control flow. Traditional hypergraph partitioning methods, while effective for static circuits, fall short in accommodating the real-time adaptability required by adaptive quantum circuits. This paper proposes a threading-based partitioning model inspired by classical distributed systems, wherein logical segments within the quantum program are treated as threads managed by a quantum resource manager.

Figure 10 illustrates that increasing the number of QPUs results in considerable improvements in both execution time and fidelity for adaptive quantum circuits, particularly when thread sessions are manually defined by the developer. In graph (a), execution time decreases significantly with the number of QPUs, especially from 2 to 8, demonstrating the benefits of parallelizing workloads across QPUs. However, there is a diminishing return beyond a certain point, suggesting that once enough QPUs are available to handle all thread sessions, adding more processing units provides minimal additional speedup. This observation underscores the importance of an efficient threading model that allows for balanced segmentation to fully leverage available hardware resources.

In graph (b) from Fig. 10, fidelity consistently improves as more QPUs are introduced. Increased QPU availability helps to minimize cross-QPU communication, which is known to degrade quantum state fidelity due to noise and error. Smaller circuits maintain high fidelity across all QPU counts, whereas larger circuits see improvements limited by the complexity of inter-qubit dependencies and cross-QPU operations. The use of developer-assisted thread segmentation minimizes the number of operations spanning QPU boundaries, highlighting the impact of careful thread allocation on maintaining quantum state integrity.

Our threading approach leverages developer-assisted annotations and a quantum resource manager to effectively partition and executes adaptive quantum circuits across multiple QPUs. These findings are supported by the qualitative understanding that arbitrary or uninformed partitioning strategies often fail to effectively minimize inter-QPU communication and balance workload, which can degrade both performance and fidelity. Our results with informed manual thread allocation demonstrate how careful segmentation can mitigate these issues by reducing communication overhead and improving parallelism. We acknowledge, however, that direct empirical comparisons with automatic or arbitrary partitioning approaches were not included in this work. As such, we frame this observation as an informed interpretation of the observed trends rather than a quantitative claim. Systematic evaluation of manual, heuristic-based, and arbitrary partitioning methods remains an important direction for future research, to further validate and refine these insights.

The integration of quantum resource management within a multi-QPU architecture enhances this approach by dynamically managing resource allocation, reducing communication overhead, and ensuring state consistency across distributed quantum computations.

Future work will involve implementing and benchmarking the proposed threading model in real-world multi-QPU systems, focusing on optimizing thread allocation and communication protocols. The results from Fig. 10 suggest a deeper investigation into the thresholds of QPU utilization, aiming to identify the best way to distribute quantum workloads for maximum efficiency and fidelity.

Developing developer tools and guidelines will also be crucial to facilitate the practical adoption of this model in quantum computing. Additionally, future research will explore advanced error correction methods and adaptive resource scaling mechanisms to enhance the robustness, scalability, and reliability of the threading model in larger and more complex quantum environments.

Acknowledgements This research was supported in part by the Laboratory of Computer Networks and Architecture (LARC) and a laboratory inside the Computing and Digital Systems Department of Polytechnic School from University of São Paulo, Brazil (PCS-EPUSP).

Author contributions W.C. wrote the main manuscript text and prepared figures. All authors reviewed the manuscript.

Funding The Article Processing Charge (APC) for the publication of this research was funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) (ROR identifier: 00x0ma614).

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Bandic M, Feld S, Almudever CG (2022) Full-stack quantum computing systems in the NISQ era: algorithm-driven and hardware-aware compilation techniques. In: 2022 design, automation & test in Europe conference & exhibition (DATE) (pp. 1-6). IEEE
2. Riesebos, L et al. (2022) Modular software for real-time quantum control systems. In: 2022 IEEE International Conference on Quantum Computing and Engineering (QCE). IEEE, p. 545–555
3. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
4. Monroe, C., et al.: Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. Phys. Rev. A **89**(2), 022317 (2014)
5. Loke SW. From distributed quantum computing to quantum internet computing: an Overview. arXiv preprint arXiv:2208.10127, 2022.
6. Yimsiriwattana A, Lomonaco JR, Samuel J. Distributed quantum computing: A distributed Shor algorithm. In: Quantum Information and Computation II. SPIE, 2004. p. 360–372. https://arxiv.org/abs/quant-ph/0403146.
7. Davarzani, Z., et al.: A dynamic programming approach for distributing quantum circuits by bipartite graphs. Quantum Inf. Process. **19**, 1–18 (2020)
8. Daei, O., Navi, K., Zomorodi-Moghadam, M.: Optimized quantum circuit partitioning. Int. J. Theor. Phys. **59**(12), 3804–3820 (2020)
9. Piveteau, C., Sutter, D.: Circuit knitting with classical communication. IEEE Trans. Inform. Theory. **70**(4), 2734–2745 (2023)
10. Barral D. et al. Review of distributed quantum computing. from single qpu to high performance quantum computing. arXiv preprint arXiv:2404.01265, 2024.
11. Caleffi, M., et al.: Distributed quantum computing: a survey. Comput. Netw. **254**, 110672 (2024)
12. Andres-Martinez, P., Heunen, C.: Automated distribution of quantum circuits via hypergraph partitioning. Phys. Rev. A **100**(3), 032308 (2019)
13. Nie J, Zi W, Sun X. Quantum circuit for multi-qubit Toffoli gate with optimal resource. arXiv preprint arXiv:2402.05053, 2024.
14. Michael Nielsen and Isaac Chuang: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
15. Schlag, S., et al.: High-quality hypergraph partitioning. ACM J. Exper. Algorithm. **10**(27), 1–39 (2023)
16. Ouvrard, X. Hypergraphs: an introduction and review. arXiv preprint arXiv:2002.05014, 2020.
17. Javadi-Abhari, A. et al. Quantum computing with Qiskit. arXiv preprint arXiv:2405.08810, 2024.
18. Iqbal, M., et al.: Topological order from measurements and feed-forward on a trapped ion quantum computer. Commun. Phys. **7**(1), 205 (2024)
19. Foss-Feig, M. et al. Experimental demonstration of the advantage of adaptive quantum circuits. arXiv preprint arXiv:2302.03029, 2023.
20. Cheng, X., Chen, X., Cao, K., Zhu, P., Feng, S., Guan, Z.: Optimization of the transmission cost of distributed quantum circuits based on merged transfer. Quant. Inform. Proc. **22**(5), 187 (2023)

21. Hayashi, A. et al. Enabling Multi-threading in Heterogeneous Quantum-Classical Programming Models. In: 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2023. p. 509–516.
22. Mintz, T.M., McCaskey, A.J., Dumitrescu, E.F., Moore, S.V., Powers, S., Lougovski, P.: Qcor: a language extension specification for the heterogeneous quantum-classical model of computation. J. Emerg. Technol. Comput. Syst. (2020). https://doi.org/10.1145/3380964
23. Burt F, Chen KC, Leung KK. Generalised circuit partitioning for distributed quantum computing. arXiv preprint arXiv:2408.01424, 2024.
24. Pawar, A. et al. Integrated qubit reuse and circuit cutting for large quantum circuit evaluation. arXiv preprint arXiv:2312.10298, 2023.
25. Perlin, MA. et al. Quantum circuit cutting with maximum likelihood tomography. arXiv preprint arXiv:2005.12702, 2020.
26. Majumdar, R. and Christopher J. Wood. “Error mitigated quantum circuit cutting.” (2022).
27. G. Uchehara, T. M. Aamodt and O. Di Matteo, "Rotation-inspired circuit cut optimization," 2022 IEEE/ACM Third International Workshop on Quantum Computing Software (QCS), Dallas, TX, USA, 2022, pp. 50–56, https://doi.org/10.1109/QCS56647.2022.00011.
28. Hart, M., & McAllister, J. (2024). Quantum circuit cutting minimising loss of qubit entanglement. In CF ‘24: Proceedings of the 21st ACM International Conference on Computing Frontiers (pp. 207 - 214).
29. Li, Peiyi et al. Enhancing Virtual Distillation with Circuit Cutting for Quantum Error Mitigation. In: 2023 IEEE 41st International Conference on Computer Design (ICCD). IEEE, 2023. p. 94–101.
30. Kan, S. et al. Scalable Circuit Cutting and Scheduling in a Resource-constrained and Distributed Quantum System. arXiv preprint arXiv:2405.04514, 2024.
31. Duncan, R., et al.: Graph-theoretic simplification of quantum circuits with the ZX-calculus. Quantum **4**(4), 279 (2020)
32. Basu, S., et al.: FragQC: an efficient quantum error reduction technique using quantum circuit fragmentation. J. Syst. Softw. **214**, 112085 (2024)
33. Lowe, A., et al.: Fast quantum circuit cutting with randomized measurements. Quantum **7**, 934 (2023)
34. Zomorodi-Moghadam, M., Houshmand, M., Houshmand, M.: Optimizing teleportation cost in distributed quantum circuits. Int. J. Theor. Phys. **57**, 848–861 (2018)
35. Wang Y., Kleinberg, J. From graphs to hypergraphs: Hypergraph projection and its remediation. arXiv preprint arXiv:2401.08519, 2024.
36. Mcsherry, F. (2001). Spectral partitioning of random graphs. Proceedings 2001 IEEE International Conference on Cluster Computing, 529–537.
37. kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970). https://doi.org/10.1002/j.1538-7305.1970.tb01770.x
38. Karypis, G. & Kumar, V. (1995). METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0.
39. Pirani M, Mitra A, Sundaram S. A survey of graph-theoretic approaches for analyzing the resilience of networked control systems. arXiv preprint arXiv:2205.12498, 2022.
40. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
41. Reeves, C. (2003). Genetic Algorithms. In: Glover, F., Kochenberger, G.A. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 57. Springer, Boston, MA. https://doi.org/10.1007/0-306-48056-5_3
42. Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM **44**(4), 585–591 (1997). https://doi.org/10.1145/263867.263872
43. Frick A, Ludwig A, Mehldau H (1994) A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In: International symposium on graph drawing (pp. 388-403). Berlin, Springer Berlin Heidelberg
44. Karypis, G., Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1998)
45. Schlag, S. et al. K-way hypergraph partitioning via n-level recursive bisection. In: 2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics, 2016. p. 53–67. https://arxiv.org/abs/1511.03137.
46. Fiduccia CM, Mattheyses RM (1982) A Linear-Time Heuristic for Improving Network Partitions," 19th Design Automation Conference, pp. 175–181, https://doi.org/10.1109/DAC.1982.1585498.

47. Umur Acikalin U, Caskurlu B. Multilevel memetic hypergraph partitioning with greedy recombination. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2022. p. 168–171. https://arxiv.org/abs/2204.03730.
48. Mak, W.-K., Wong, D.F.: A fast hypergraph min-cut algorithm for circuit partitioning. Integration **30**(1), 1–11 (2000). https://doi.org/10.1016/S0167-9260(00)00008-0. (**ISSN 0167-9260**)
49. Crampton, O. et al. A Genetic Approach to Minimising Gate and Qubit Teleportations for Multi-Processor Quantum Circuit Distribution. arXiv preprint arXiv:2405.05875, 2024.
50. Patil, S.V., Kulkarni, D.B.: Graph partitioning using heuristic kernighan-lin algorithm for parallel computing. In: Deshpande, P., Abraham, A., Iyer, B., Ma, K. (eds.) Next Generation Information Processing System. Advances in Intelligent Systems and Computing, Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-4851-2_30
51. Fang, K. et al. Dynamic quantum circuit compilation. arXiv preprint arXiv:2310.11021, 2023.
52. Brandhofer, S., Polian, I., Krsulich, K.: Optimal partitioning of quantum circuits using gate cuts and wire cuts. IEEE Trans. Quant. Eng. **26**(5), 1 (2023)
53. Cross, AW. et al. Open quantum assembly language. arXiv preprint arXiv:1707.03429, 2017.
54. Cross, A., et al.: OpenQASM 3: a broader and deeper quantum assembly language. ACM Trans. Quantum Comput. **3**(3), 1–50 (2022)
55. Quetschlich, N., Burgholzer, L., Wille, R.: MQT Bench: benchmarking software and design automation tools for quantum computing. Quantum **7**, 1062 (2023)
56. Baker JM, Duckering C, Hoover A, Chong FT (2020) Time-sliced quantum circuit partitioning for modular architectures. In Proceedings of the 17th ACM International Conference on Computing Frontiers (CF '20). Association for Computing Machinery, New York, pp. 98–107. https://doi.org/10.1145/3387902.3392617.
57. Escofet, P., Ovide, A., Bandic, M., Prielinger, L., van Someren, H., Feld, S., Alarcon, E., Abadal, S., Almudever, C.: Revisiting the mapping of quantum circuits: entering the multi-core era. ACM Trans. Quantum Comput. **6**(1), Article 4 (2025). https://doi.org/10.1145/3655029