A Hardware-Aware Gate Cutting Framework for Practical Quantum Circuit Knitting

Xiangyu Ren
University of Edinburgh
Edinburgh, United Kingdom
xiangyu.ren@ed.ac.uk

Mengyu Zhang
Tencent Quantum Lab
Shenzhen, China
mengyu_z@hotmail.com

Antonio Barbalace
University of Edinburgh
Edinburgh, United Kingdom
abarbala@ed.ac.uk

Abstract

Circuit knitting emerges as a promising technique to overcome the limitation of the few physical qubits in near-term quantum hardware by cutting large quantum circuits into smaller subcircuits. Recent research in this area has been primarily oriented towards reducing subcircuit sampling overhead. Unfortunately, these works neglect hardware information during circuit cutting, thus posing significant challenges to the follow on stages. In fact, direct compilation and execution of these partitioned subcircuits yields low-fidelity results, highlighting the need for a more holistic optimization strategy.

In this work, we propose a hardware-aware framework aiming to advance the practicability of circuit knitting. Drawing a contrast with prior methodologies, the presented framework designs a cutting scheme that concurrently optimizes the number of gate cuttings and SWAP insertions during circuit cutting. In particular, we leverage the graph similarity between qubits interaction and chip layout as a heuristic guide to reduces potential SWAPs in the subsequent step of qubit routing. Building upon this, the circuit knitting framework we developed has been evaluated on several quantum algorithms, leading to reduction of total subcircuits depth by up to 64% (48% on average) compared to the state-of-the-art approach, and enhancing the relative fidelity up to 2.7×.

ACM Reference Format:
Xiangyu Ren, Mengyu Zhang, and Antonio Barbalace. 2024. A Hardware-Aware Gate Cutting Framework for Practical Quantum Circuit Knitting. In *2024 ACM/IEEE International Conference on Computer-Aided Design (ICCAD), October 27–31, 2024, New Jersey, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Quantum computing has the potential to solve many classically intractable problems. However, the inadequate number of available qubits in current Noisy Intermediate Scale Quantum (NISQ) devices hinders quantum computing from realizing its initially anticipated computational power.

Quantum circuit knitting [7, 23, 28, 29] has been put forward as a technique to solve this problem by running large circuits with more qubits than physically available. Fig. 1 shows a generalization of previous circuit knitting frameworks, e.g., CutQC [33]. First, the original circuit is cut along qubit wires (wire cutting) or 2-qubit gates (gate cutting, not in Figure) to form separated subcircuits. Then, subcircuits are individually compiled and executed on quantum processors. Finally, classical postprocessing: individual execution results are reconstructed into the result of the original circuit.

However, this technique is costly as the overhead of subcircuits sampling (step 3) and classical postprocessing (step 4) scale exponentially with the number of cuts [29]. Hence, to lower such overheads, previous research focuses on minimizing the number of cuts [33, 35]. Other research involving cluster-based hardware [3] also focus on minimizing the number of cuts.

Despite their efficiency in minimizing the number of cuts (step 1), the follow-on compilation (step 2) may counteract the improvements. This is because during compilation a large number of SWAPs may be inserted in the qubit routing stage, which likely extends the depth of each subcircuit and undermines the fidelity.in Qiskit, this is transpilation Fig. 2 shows a comparison of the subcircuits depth before and after compilation. The depth increase is caused by neglecting hardware information during circuit cutting, which makes the routing of the resulting subcircuits onto actual devices challenging. In fact, in previous works, the SWAPs insertion remains

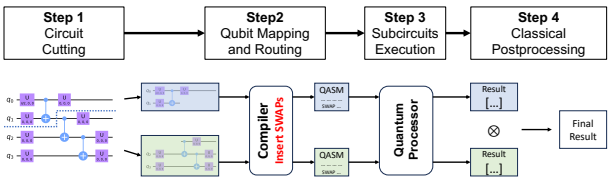


Figure 1: Generalization of previous knitting frameworks.

ICCAD '24, October 27-31, 2024, New Jersey, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

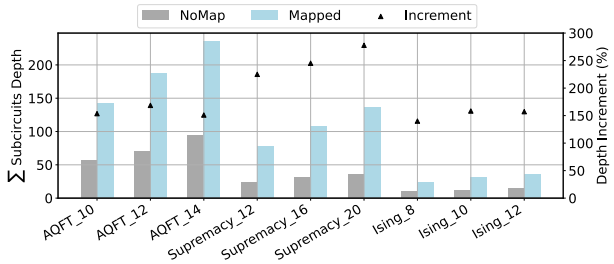


Figure 2: Example of qubit routing overhead, measured by the sum of depth of each subcircuits. "NoMap" shows compilation results with all-to-all qubit connectivity (or before compilation), "Mapped" shows the results when subcircuits are compiled with IBM Lagos (7-qubit) as backend. "Increment" shows the percentage of increased $\sum$ *Subcircuit Depth* from "NoMap" to "Mapped", with its axis on the right.

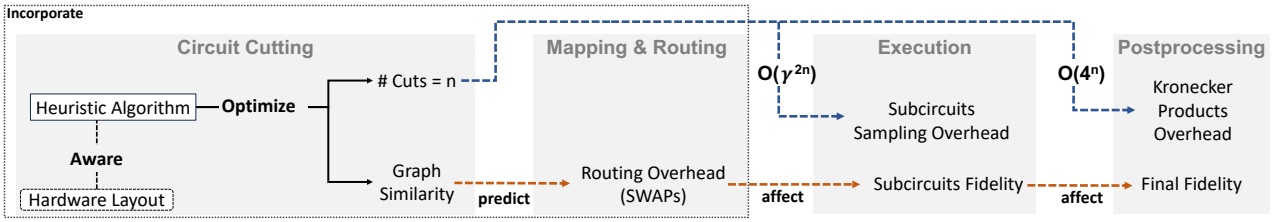


Figure 3: Optimization structure inside our framework. The *number of cuts* affects the sample overhead and postprocessing overhead with an exponential relationship (blue arrows), while *graph similarity* predicts the routing overhead (orange arrows). Being aware of the hardware layout, our algorithm optimize both of them.

unknown till we finish the circuit cutting procedure and start compilation, which makes the final result vulnerable to mapping and routing overheads.

Hardware-aware Circuit Cutting. Instead of investigating new qubit mapping and routing methods – widely studied in recent years, this work proposes to *incorporate* the routing problem (step 2) into the circuit cutting (step 1). This is in sharp contrast to previous works, where such steps execute sequentially, and independently. Our idea is to improve the quality of circuit cutting by considering the routing overhead on the actual hardware during cutting.

Based on this idea, we exploit the fact that graph similarity between **a subcircuit's qubit interaction graph** and **hardware layout** can well predict the number of SWAPs for qubit routing. Setting graph similarity as an optimization objective, we can iteratively improve the quality of circuit cutting. Our hardware-aware circuit knitting framework is depicted in Fig. 3, including relationships and costs for each step. These steps are briefly detailed below.

a) **Circuit Cutting:** Our algorithm co-optimizes two objectives: 1) minimizing the number of gate cuts, 2) maximizing the subcircuit graph similarity to the hardware-layout of a quantum processor. Graph similarity serves as a "heuristic clue" to reduce SWAPs before the qubit mapping & routing step, as shown in the left half of Fig. 3. After dividing logical qubits into different subcircuits, we insert those local single-qubit operations for replacement using Qiskit.

b) **Compilation and Execution:** Each subcircuit is compiled through qubit mapping & routing, where SWAPs insertion are actually performed. Then subcircuits are individually executed on one or more quantum processor. This step is unchanged in our framework, easily integrating in existent frameworks.

c) **Classical Postprocessing:** Finally, we also implement an efficient module for reconstructing results during classical postprocessing. The classical postprocessing can be represented by *tensor network contraction*, and a contraction tree optimizer is utilized to find the optimal sequence for contraction.

Apart from quantum circuit knitting, our design is also beneficial for cutting circuits in other distributed quantum computing scenarios [36], e.g. chiplet architecture [37].

Contributions. Our key contributions can be summarized as:

(1) We introduce a new quantum circuit knitting framework design where the circuit cut stage exploits the quantum hardware-layout to make partitioning decisions, while maintaining compatibility with existent frameworks. This work is the first to consider and co-optimize three distinct types of overhead in circuit knitting tasks, as detailed in Section 2.2;

(2) We specialize such design for the *gate cut* technique – already demonstrated in theory, and introduce a new graph-similarity-based approach to minimize the number of gates to be cut while reducing SWAPs in subcircuits by exploiting the hardware-layout;

(3) We fully implemented the entire framework based on Qiskit, deployed it on a server with GPU, and run subcircuits on IBM quantum processors. We evaluated and compared our framework with the state-of-the-art work CutQC. When running benchmark where the original qubit count is 1.5× – 4× of physical qubits on the chip, our work reduce the subcircuits depth by 48% compared to CutQC. And it shows the relative fidelity up to 2.7× (1.67× on average).

2 Problem Analysis

2.1 Background

Quantum circuit knitting aims to address the constraint of limited physical qubits on NISQ-era quantum computers. The theory behind circuit knitting is quasiprobability simulation [26]: the expected value of the original circuit is obtained by sampling each smaller part of it that we define as subcircuit. Based on the quasiprobability simulation, previous works provide several approaches to partition the quantum circuit into smaller ones. Specifically, this is realized by decomposing a *qubit wire* or a *2-qubit gate* into a set of *single qubit operations*, followed by classical postprocessing to reconstruct the expectation value of the original circuit.

Theoretical Works. Peng et al. proposed *wire cut* [28] and proved its validity. Later, Mitara et al. introduced *gate cut* [23] that is theoretically established to have significantly lower sampling overhead than *wire cut* [29]. Along with *entanglement forging* [12], these techniques are known as *circuit knitting* [29]. Fig. 4 shows the CZ gate being decomposed into an expectation combination of multiple terms of single qubit operations, while in each term the control and target qubits are substituted by single qubit gates and mid-circuit measurements.

Practical Implementations. Tang et al. proposed the CutQC *wire cut* framework [33], with minimizing the classical postprocessing overhead as optimization target, followed by their work ScaleQC further improving the postprocessing of *wire cut* [32]. Brandhofer et al. incorporate *wire cut* and *gate cut* to minimize the sampling overhead [6], while Pawar et al. additionally consider qubit reuse in their work [27]. Chen et al. explored a fast approach for classical

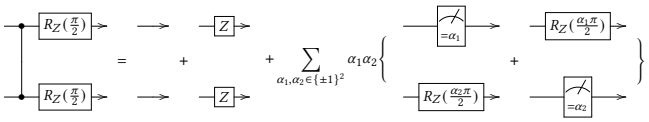


Figure 4: Example of CZ gate cut decomposition. There are four terms of single qubit operations, and each term represents a substitution for the original CZ gate. While the last two terms are associate with α_i coefficients, the mid-circuit measurement boxes stand for "post selection", i.e., the result will be kept only if "mid-circuit measurement result == α_i".

postprocessing of *wire cut* [9], and Tomesh et al. apply the *wire cut* to solve Maximum Independent Set (MIS) problem [35]. While none have addressed the problem of incorporating *gate cut* into a practical framework before, it is important to note that these studies did not take qubit routing overhead into consideration.

Wire Cut and Gate Cut. We have developed our framework using the *gate cut* technique rather than the *wire cut* technique or a hybrid of both, because the *gate cut* protocol is more concisely represented in the graph model (detailed in Section 3.1). This makes it more intuitive for verifying our concept. Additionally, the *gate cut* approach has the potential to benefit from enhancements through classical communication [29], which holds promise for future distributed quantum computing. Note that the gate decomposition protocol in our framework can be switched to a circuit knitting scheme based on classical communication with minor modifications. However, this approach warrants further exploration in conjunction with more mature and advanced quantum control systems in the near future.

2.2 Practical Circuit Knitting Overheads

Existent approaches to circuit knitting are in many cases non-practical due to at least the following overheads, which depends on the number of SWAPs and the number of cuttings.

Circuit Depth and Routing Overhead (depends on #SWAPs). In Fig. 2, we report the sum of subcircuits depth before and after the qubit mapping and routing (step 2, in Fig. 1) targeting a specific hardware. From the graph, we can see that qubit mapping and routing increase the circuit depth from 1.4× to 2.8×, this is due to the SWAP insertions, each including 3 CNOT gates, which account for a large portion of each subcircuit after compilation. A deeper circuit will significantly lower the fidelity of subcircuit, and finally affect the fidelity of original circuit, we represent this relation with the bottom brown arrow in Fig. 3. Therefore, minimizing the routing overhead is critical to improve the fidelity, and the routing overhead is hardware dependent.

Subcircuits Sampling Overhead (depends on #cuttings). Sampling overhead is the number of shots each subcircuit needs to run in order to grantee the accuracy of circuit knitting. Generally, the sampling overhead $O(\gamma^{2n})$ scales exponential with the number of cuts n, while $\gamma = 3$ for *gate cut* and $\gamma = 4$ for *wire cut* [8, 29]. Thus, the lower γ reinforces our decision of selecting *gate cut* protocol, and minimizing the number of cuts n is important to keep the sampling overhead low.

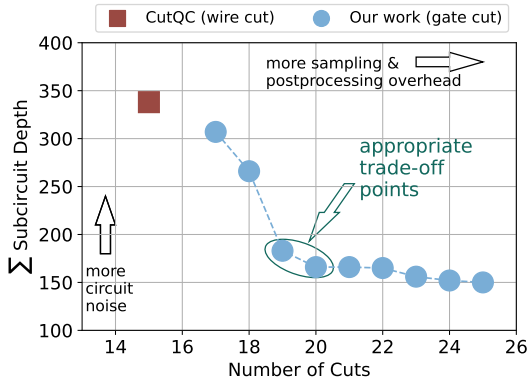


Figure 5: Balancing between different overheads in circuit knitting. In this motivation experiment, we cut and compile a AQFT_16 circuit to the IBM Lagos (7-qubit) quantum hardware. Y-axis shows the sum of subcircuit depth, representing the routing overhead; while X-axis shows the number of cuts related to sampling & postprocessing overhead.

Classical Postprocessing Overhead (depends on #cuttings). In the classical postprocessing step, the result of the original circuit is reconstructed through Kronecker products. State-of-the-art estimates $O(4^n)$ Kronecker products for computation *wire cut* framework, where n is the number of cuts [33, 35]. With our implementation based on tensor network contraction, the Kronecker products computation is also $O(4^n)$ (detailed in Section 4), the same as *wire cut*. Hence, we will focus on comparing the number of cuts n.

2.3 Motivation: A Case Study

Based on the various types of overheads discussed in Section 2.2, our objective is to explore the correlations and trade-offs between them in practical scenarios. Fig. 5 illustrates our analysis by showcasing a search for a circuit cutting solution on one of our benchmark circuits. While the CutQC method provides an optimal solution that minimizes the number of wire cuts (marked by red square scatter points), our framework operates along the Pareto frontier, balancing between qubit routing overhead and sampling/postprocessing overhead (represented by blue circle scatter points). It is important to note that while our solution may not precisely align with the optimal Pareto frontier, it effectively demonstrates the general trends and trade-offs between these overheads. We observe that an appropriate trade-off can lead to significantly improved circuit fidelity.

However, identifying the optimal trade-offs during the circuit cutting of arbitrary quantum circuits is not straightforward. Consequently, there is a pressing need for a strategy that can autonomously determine suitable solutions. In Section 3, we will introduce an algorithm designed to systematically search for circuit cutting solutions that achieve a balanced trade-off between the involved overheads.

3 Framework Design

3.1 Modelling the Circuit Cutting Problem

To reason about the best partitioning of a quantum circuit into subcircuits we decided to use a more abstract model than the circuit

itself: we use the qubit interaction graph [4]. In the qubit interaction graph $G = (V, E)$, each logical qubit l_i is represented by a vertex $v_i \in V$, while 2-qubit gate g_{ij} performed between logical qubits l_i and l_j is represented as edge $e_{ij} \in E$. The weight $W_{e_{ij}}$ of each edge indicates the number of 2-qubit gates applied between the corresponding pair of logical qubits l_i and l_j. An illustrative example of the graph is presented in Fig. 6(a). If the circuit includes multi-qubit gates, these are first transformed into combinations of single- and two-qubit gates prior to graph generation.

Adopting the qubit interaction graph $G = (V, E)$, gate cutting of a circuit corresponds to partitioning G into subgraphs G_{sub} (upper part of Fig. 6(b)). Under these circumstances, the number of cuts is determined by the sum of the weights of the edges that connect between different subgraphs, denoted as:

$$\#Cuts = \sum_{v_{i,j} \in V} W_{e_{ij}} \left[G_{sub}(v_i) \neq G_{sub}(v_j) \right] \tag{1}$$

Where $G_{sub}(v_i)$ specifically refers to the subgraph that contains v_i. On the other hand, routing overhead is associated solely with the two-qubit gates located within a single subgraph, which are referred to as intra-subgraph edges.

3.2 Overview of Optimization Process

Fig. 6 shows the general idea of our optimization. In upper part of Fig. 6(b) are two different cutting solutions A and B of the same qubit interaction graph, derived from the same original logical circuit (upper part of Fig. 6). We can see in Fig. 6(c) that sum of subcircuits depth is largely different between this two solutions after qubit routing. So, an approach to finding a circuit cut with the minimal number of cuts and SWAPs is sought.

For #cuts, it is easy to calculate with Equation. 1. However, for #SWAPs (or routing overhead), it remains unknown till finishing qubit mapping & routing. This requires a circuit cutting algorithm aware of the hardware, hence, the routing overhead. We leverage graph similarity to estimate the routing overhead, as illustrated by the blue arrows connecting between Fig. 6 (a) and (b). Details are explained in the following subsections.

3.3 Exploiting Hardware Layout

Finding a cut with minimal gates cut **and** minimal SWAPs is non-trivial. This is because we cannot normally infer the number of SWAPs during circuit cutting, but the subcircuits need to be compiled first. Indeed, we could compile all subcircuits for each possible cutting solution, but this is unrealistic due to the prolonged time for gate cut decomposition and compilation. Also, the number of SWAPs cannot be defined as a linear function on the circuit; thus, impossible to integrate in a mathematical optimization solver – alike [33].

Graph-Similarity to Predict SWAP insertions. It is vital to explore an approach to estimate SWAP insertions in advance, and measuring graph similarity between **a subcircuit's interaction graph** G_{sub} and **hardware layout** T makes an ideal option. Graph similarity quantifies how closely two graphs resemble each other [18]. There are several approaches to quantify the graph similarity, and we identify Graph Edit Distance (GED) [1] as the most appropriate for our scenario, because of its clarity to describe the difficulty for qubit routing on NISQ scale circuits.

Specifically, we calculate the edit distance from the hardware layout to a subcircuit's interaction graph, denoted as $C_{GED}(G_{sub}, T)$, and use this metric to estimate the SWAPs insertion. SWAP are inserted because of the limited connectivity of physical qubits in the hardware layout. If we have to add more edges to edit hardware layout into interaction graph (higher edit distance), that means the lack of connectivity is more severe, thus more SWAP have to be added to fill the gap. In the graph edit distance calculation function we only set the cost for adding edges to hardware layout graph, and there's no cost for deleting edges or nodes from hardware layout.

3.4 Designing the Searching Heuristic

Building on the concept of Graph Edit Distance introduced in Section 3.3, we propose a heuristic cost function designed to optimize the search for circuit cutting solutions. First, we introduce the *topology distance* (denoted as $dist(p_i, p_j)$), which measures the shortest path between two physical qubits p_i and p_j. Each edge in hardware topology is assigned a distance of 1, and $dist(p_i, p_j)$ depict the number of edges in the shortest path connecting these qubits (see Ref. [20] Section 4.1).

Then, we calculate the graph edit distance utilizing edge weight W_{ij} in interaction graph G_{sub} and topology distance $dist(pi, pj)$ in hardware topology T. Let e_{ij} be a edge that should be add to T, than the cost for this graph editing is

$$C_{GED}(G_{sub}, T) = \sum_{e_{ij} \in E_{add}} W_{ij} \times dist(p_i, p_j) \tag{2}$$

Where E_{add} be the edge to be added. An example is given in Fig. 7 (a) to help understand the calculation of cost.

To evaluate the effectiveness of our heuristic, we compute the correlation between heuristic cost C_{GED} and SWAP counts on a broad range of circuits, and report AQFT_16 benchmark in Fig 7 (b). Results confirm our hypothesis.

Finally, we combine #Cuts and $\sum C_{GED}$ of all subcircuits to construct the overhead heuristic for the circuit cutting solution:

$$C = \#Cuts + \alpha \cdot \sum C_{GED} \tag{3}$$

Where α is a parameter to balance between these two co-optimizing objectives. The parameter α is set uniquely for each circuit according to its characteristics (e.g. gate density [31]), and it can be adjusted to satisfy varying degrees of trade-off between overheads. A systematic approach for determining the value of α is given in Section 5.1.

3.5 Circuit Cutting Algorithm

Based on the heuristic cost function, we are able to assess the routing overheads of a cutting solution. In another word, we enable hardware-aware circuit cutting. With such information, a cutting algorithm that takes both number of cuts and routing overhead into account can now be properly designed.

Overview. Our circuit cutting algorithm is given in Algorithm 1. It utilizes random edge contraction to merge nodes (qubits) into subgraphs (subcircuits) [17], recursively samples a variety of cutting solutions, and returns the solution with minimal cost c. Several constrains are applied to fit the algorithm into the circuit cutting scenario. For size limitation on resulting subgraphs (maximum number of qubits in a subcircuit) [3], our algorithm sets constrain to avoid contraction that leads to merging into an oversize subgraph.

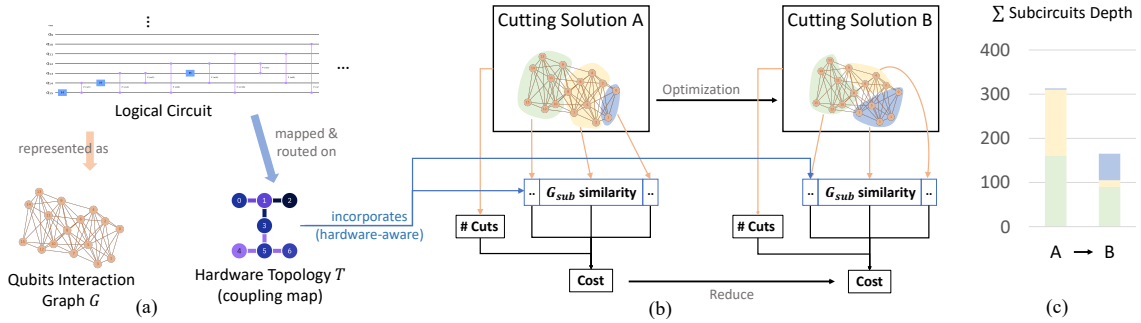


Figure 6: Hardware-aware optimization for circuit cutting. (a) An example of the qubit interaction graph G, with AQFT_16 as the original logical circuit. (Every $W_{e_{ij}}$ has value 1 in this circuit, so we don't draw it on G). The circuit will be cut and compiled to IBM Lagos (hardware topology T). (b) Optimizing the cutting solution with multiple objectives: #Cuts and Graph Similarity, lead to a reduction in subcircuits depth, shown in (c).

Algorithm Details. The major part of algorithm is MERGE_NODES function. Starting from the original qubit interaction graph with n nodes, it randomly merges nodes into subgraphs. It keeps merging until the graph size reaches $\sqrt{n}$, then the function uses the current graph as input and invokes two recursive calls, each have a different merging to cover different solutions. The recursion keeps going until terminate condition ($E_{cst} = \emptyset$) is reached, then we get the solution subgraphs representing subcircuits. When returning from the function, it chooses the recursive calls that returns the solution with lower cost.

3.6 Complexity Analysis.

According to the theory in *Karger-Stein Algorithm* [17], the full program can be repeated for $O(log^2(n_q))$ times to achieve an error probability within $O(1/n_q)$ (n_q is number of qubits). Running the program once need $O(n_q^2 log(n_q))$ time, so the overall complexity is $O(n_q^2 log^3(n_q))$. In comparison with the MIP solver used in CutQC, which have a $O(n_g!)$(n_g is number of 2-qubit gates, and ! is the factorial), our circuit cutting approach can be more scalable.

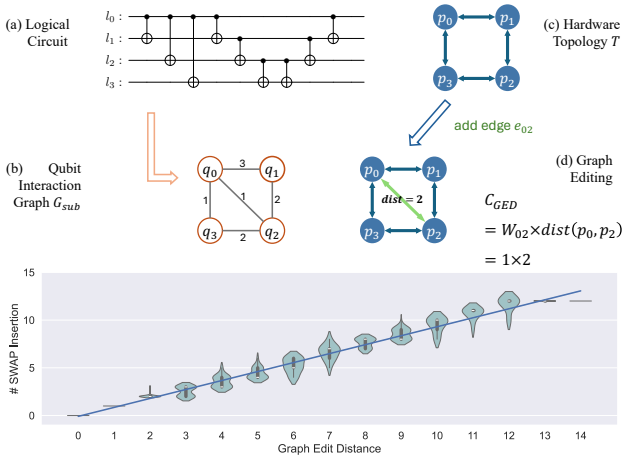


Figure 7: (a) An example of calculating the graph editing cost. (b) Distribution of the number of SWAPs (Y-axis) for different heuristic cost C (X-axis). We randomly sample 1000 subcircuits cut down from the AQFT_16 benchmark, and compile them to IBM Lagos. We can see a linear correlation in this case between edit distance and SWAPs.

4 Classical Postprocessing

Optimizing Recombination Protocol. Classical postprocessing reconstructs the result of the original circuit by merging subcircuits result using Kronecker products. Specifically, the basic protocol to merge results of two subcircuits is defined in Fig. 4. Note that the third term in the protocol will expand into 2×4 sub-terms (each corresponds to a Kronecker product) in respect to each value

Algorithm 1 Circuit Cutting Algorithm

Input: Original qubit interaction graph G_0, hardware layout of backend T, objective weight coefficient α, search iterations r.
Output: Partitioning solution $\{G_{sol}\}$.

$C_{min} \leftarrow \infty$
repeat r **times**
  $(C, \{G_{sub}\}) \leftarrow$ MERGE_NODES(G_0)
  if $C < C_{min}$ **then**
    $C_{min} \leftarrow C$
    $\{G_{sol}\} \leftarrow \{G_{sub}\}$
  end if
end
function MERGE_NODES(interaction graph $G = (V, E)$)
  $E_c \leftarrow$ edges connecting between subgraphs
  $E_{cst} \leftarrow$ edges in E_c that comply constrain
  if $E_{cst} = \emptyset$ **then**
    $C = \#Cuts + \alpha \cdot \sum C_{GED}(G_{sub}, T)$
    return $(\{G_{sub}\}, C)$
  end if
  $n \leftarrow |V|$
  while $|V| > \sqrt{n}$ **do**
    $e \leftarrow$ random sample from E_{cst}
    $G' \leftarrow G$ apply edge contraction on e
  end while
  $(G_1, c_1) \leftarrow$ MERGE_NODES(G')
  $(G_2, c_2) \leftarrow$ MERGE_NODES(G')
  return $\min_{c_i}\{(G_1, c_1), (G_2, c_2)\}$
end function

combination of the coefficients $\alpha_1\alpha_2$. Due to our observation that each coefficient α_i is only related to one side of the result in these sub-terms, we can split the combined coefficient $\alpha_1\alpha_2$ on each side, and the following Kronecker product would do recombination for them. Thus, on each side of the result, the coefficient will calculate with subcircuit results ahead of Kronecker product, shrinking 4 sub-terms into one. Our preprocessing on the coefficients improve the number of Kronecker products of each merging from 10 (= $2+2\times4$) to 4, shown in Fig. 8.

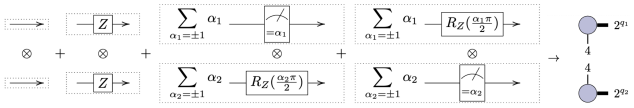


Figure 8: Classical postprocessing: merging two subcircuits result using tensor contraction. The notation on the right is a tensor network notation, representing the contraction of these two subcircuit result.

Acceleration with Tensor Contraction. Furthermore, the merging operation in Fig. 8 is identical to the operation of tensor contraction. Kronecker product is a special version of tensor product when we group the q indices each with the 2-dimension, into one single index with 2^q-dimension (q=num of qubits). Assume that in both upper and lower side of Fig. 8, the 4 result vectors (each with 2^{q_1} or 2^{q_2}-dimension) can be seen as tensor in 4×2^q shape, the whole merging operation is abstracted into the tensor contraction on the right of Fig. 8. Correspondingly, if there are n gate cuts between two subcircuits, the result tensor of each side would be in shape $4_1\times\cdots\times4_n\times2^q$, who has n legs with 4-dimension and one leg with 2^q-dimension in tensor network notation. With such abstraction, we can translate the classical postprocessing problem into a tensor network to searching a better solution. After translation, we use NVIDIA cuTensorNet framework [34] to enable an efficient classical postprocessing.

5 Evaluation

Herein we are trying to answer if the proposed hardware-aware circuit cutting framework improves the state-of-the-art in terms of number of cuts, circuit depth, and fidelity, on a set of major quantum benchmarks, on real quantum hardware. Furthermore, in Section 5.3, we test the scalability of our algorithm to determine its effectiveness with circuits of significantly larger scales. In Section 5.4, we enhance the QUEKO benchmark [31] to generate circuits that have known optimal cutting solutions and qubit mappings, which we then use to study the optimality of our framework. Finally, in Section 5.5, we provide details on the classic postprocessing overhead when executed on a GPU.

5.1 Evaluation Methodology

Framework Implementation: We extended Qiskit [30], we adopted the NetworkX [14] Python package to implement our circuit cutting algorithm, and use Qiskit's QuantumCircuit to partition and generate the subcircuits. To run subcircuits on real-hardware, we use Qiskit transpiler for qubit mapping and routing, and selected the SABRE layout method [20] because of its speed and scalability.

Algorithm Configuration: We do provide an approach to decide the value of parameter α automatically, integrating the *gate density* feature [31]. Suppose the original circuit has N logical qubits, M_2 two-qubit gates, and l as the length of longest dependency chain, then the two-qubit gate density is $d_2 = 2M_2/(N\cdot l)$. We observe from experiments that multiples of d_2 are a good fit for the value of α. Thus, we use two configurations in our evaluation: $\alpha = 0.2d_2$ and $\alpha = 0.5d_2$.

Experiment Platform: Our experiments are performed on a CentOS 7 server featuring dual Intel Xeon Platinum 8255C CPUs for a total of 48 cores/96 threads at 2.5 GHz, 384 GB of RAM, and a NVIDIA V100 GPU with 5120 CUDA cores at 1.2 GHz. The Python version is 3.10.12.

Hardware Backends: We use four IBM quantum computers (free access with Open Plan [15]): 7-qubit IBM Lagos, 16-qubit IBM Guadalupe, 27-qubit IBM Mumbai, and 127-qubit IBM Brisbane as backends. These hardware are all heavy-hexagon structure, so we add a fictitious 20-qubit IBM Tokyo (retired) backend, which has square structure, to prove the generalizability of our algorithm for different hardware layouts. The original circuits and subcircuits are run from 32K to 1M shots, depending on the circuit size and sampling overhead.

Benchmarks: We benchmark and compare with state-of-the-art our approach on four quantum algorithms that show potential for quantum advantages according to previous research:

(1) *Supremacy:* Google random circuit for quantum supremacy[2];
(2) *AQFT:* Approximate Quantum Fourier Transform[5];
(3) *QAOA:* Hardware efficient ansatz for Quantum Approximate Optimization Algorithm[25];
(4) *Ising*: 2-local Hamiltonian Simulation with Ising Model [19].

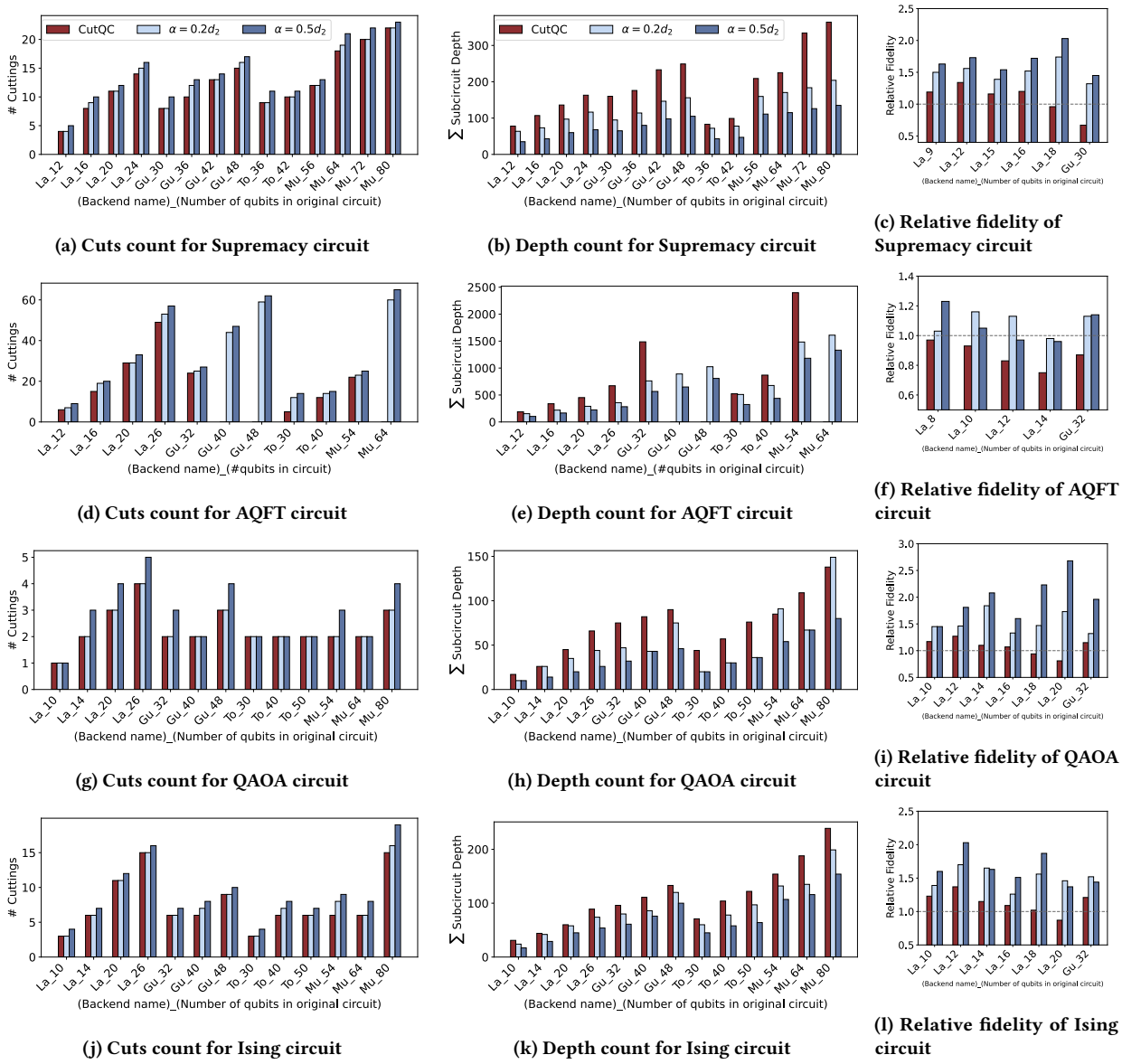
We set the size of original circuit from 1.5× to 4× of the physical qubits available on the selected hardware backends.

Metrics: We use the following metrics to evaluate the overheads introduced in Sect. 2.2.

(1) *Number of cuts.* To establish a fair comparison that mitigate the impact of using different classical computing platforms and quantum hardware, we directly choose the number of cuts n as metric, to evaluate the sampling overhead $O(\gamma^{2n})$ and postprocessing overhead $O(4^n)$.
(2) *Sum of subcircuits' depth.* The routing overhead of a subcircuit is reflected on its depth. Because the same original circuit will have different cutting solution in different frameworks, each of the subcircuit's depth is unmatched and incomparable. However, we can compare the sum of the depths of all subcircuits.
(3) *Relative fidelity on quantum computer.* We select the IBM Lagos (7-qubit) and IBM Guadalupe (16-qubit) quantum computer as the hardware backend to run circuit knitting, and compare its relative fidelity with directly running the original circuit on IBM Mumbai (27-qubit).

5.2 Results

Our evaluation results are shown in Fig. 9. We use the state-of-the-art CutQC as a baseline, instead of its improved version ScaleQC [32], because the source code of ScaleQC is still not available. Specifically, we test our framework with two configurations: $\alpha = 0.2d_2$ and $\alpha = 0.5d_2$ as introduced in Section 5.1. With IBM quantum computers as the backends, three metrics are evaluated in four benchmarks. The x-axis in Fig. 9 is in the format of [Backend Name]_[Original circuit qubits].



(a) Cuts count for Supremacy circuit

(b) Depth count for Supremacy circuit

(c) Relative fidelity of Supremacy circuit

(d) Cuts count for AQFT circuit

(e) Depth count for AQFT circuit

(f) Relative fidelity of AQFT circuit

(g) Cuts count for QAOA circuit

(h) Depth count for QAOA circuit

(i) Relative fidelity of QAOA circuit

(j) Cuts count for Ising circuit

(k) Depth count for Ising circuit

(l) Relative fidelity of Ising circuit

Figure 9: Evaluation. Row 1,2,3,4 are result for Supremacy, AQFT, QAOA, Ising benchmark respectively. Column 1 is the number of cuts, column 2 is sum of depth of subcircuits, column 3 is the relative fidelity. In the X-axis, La_n, Gu_n, To_n, Mu_n stand for IBM chip Lagos, Guadalupe, Tokyo, Mumbai running n-qubit size original circuit. It is notably that three groups of data of CutQC is blank for the AQFT benchmark, because solution is not feasible.

For Metrics (1) and (2), the scale of the original circuits is limited to 80 qubits, constrained by the ability of baseline framework [33]. For Metric (3), the calculation of relative fidelity depends on noise-free results obtained from simulations; therefore, the limitation is limited by the capability of current quantum simulators, which is approximately 33 qubits.

It is notably that in AQFT benchmark, CutQC have "solution not feasible" during the run in 16_40, 16_48, and 27_64, so the corresponding results are left blank in Figure 9d and 9e.

Cuts and Depth. Compared to CutQC as baseline, our framework with the $\alpha = 0.2d_2$ configuration increases the number of cuts by 3% on average, but reduces the subcircuits depth from 25% to 64% (34% on average; 43%, 47%, 29%, 17% respectively for benchmark Supremacy, AQFT, QAOA, and Ising). The $\alpha = 0.5d_2$ configuration increases the number of cuts by 13% on average compared to baseline, but it reduces the subcircuits depth from 25% to 64% (48% on average; 57%, 53%, 48%, 36% respectively for benchmark Supremacy, AQFT, QAOA, and Ising). Based on these results, our framework effectively reduces circuit depth while accommodating a manageable

increase in the number of cuts, demonstrating scalability to larger circuit sizes.

Relative Fidelity. For relative fidelity on quantum computers, the $\alpha = 0.2d_2$ and $\alpha = 0.5d_2$ configurations have an average result of 1.54× and 1.73× for Supremacy, 1.13× and 1.1× for AQFT, 1.55× and 1.98× for QAOA, 1.5× and 1.67× for Ising, and it's 1.39× and 1.65× totally for all benchmarks. Compared to CutQC, we have an improvement up to 231%. Our framework have relative fidelity > 1 on almost every benchmark, while CutQC have negative results in part of the benchmarks. Therefore, we identified that our framework enhance the practicality of circuit knitting by reducing routing overhead.

5.3 Scaling Up in Circuit Size

In Section 5.2 the evaluation of #cuts and subcircuits depth are limited to circuits up to 80 qubits, constrained by the capabilities of baseline framework [33]. In complementary to this, experiment results on larger circuits (ranging from 100 to 1000 qubits) are presented in Figure 10, using IBM Brisbane (127-qubit) for the hardware topology. Our observations indicate that while setting $\alpha = 0.5k$ results in a slight increase in #Cuts compared to $\alpha = 0.2k$, it considerably reduces the depth of subcircuits. These results effectively demonstrate that as circuit size increases to larger scale (1000 qubits), our algorithm can still successfully find suitable trade-off solutions aligned with the Pareto frontier.

5.4 Optimality Study

We have enhanced the QUEKO [31] to evaluate the optimality of our algorithm. Specifically, we created several "virtual quantum hardware" by connecting together multiple instances of IBM Lagos topology, illustrated in Fig. 11. Based on these virtual device topology, we employed QUEKO to generate logical circuits with known optimal cutting and mapping solutions. Subsequently, we tested our framework ($\alpha = 0.2k$) on these circuits and compared the results with the baseline. The data presented in Table1 show that our approach is more likely to find the optimal solution when considering multiple overheads.

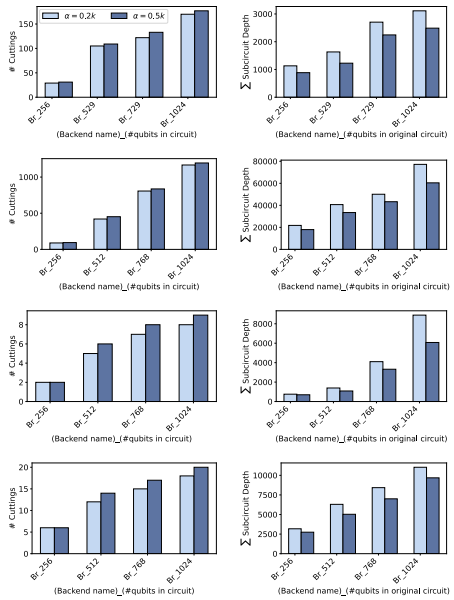


Figure 10: Evaluation. Row 1,2,3,4 are result for Supremacy, AQFT, QAOA, Ising benchmark respectively. Column 1 is the number of cuts, column 2 is sum of depth of subcircuits. In the X-axis, Br_n stand for IBM chip Brisbane (127-qubit) running n-qubit size original circuit.

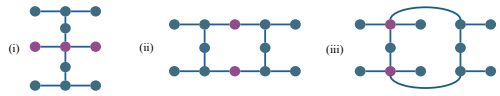


Figure 11: The topology of virtual devices created by connecting multiple IBM Lagos chips.

Table 1: Optimality Study. The result data is presented as the form of [CutQC | Our Work].

Original Circuit Generated From	#Cuts	Subcircuits Depth Added Compared to Optimal Solution
(i)	1 | 1	0% | 0%
(ii)	2 | 2	13% | 0%
(iii)	2 | 2	21% | 0%

5.5 Classical Postprocessing

We tested the performance of our classical postprocessing approach, described in Section 4, on the benchmarks we used for the evaluation (see Section 5.1). The results are shown in Fig. 12. For QAOA and Ising benchmarks, we only recombine the expectation value of each subcircuits, instead of recombining the full measurement results, so the overheads are quite low compared with other algorithms. For Approximate-QFT benchmark, we adopt the "Dynamic Definition" approach purposed in [33] when $qubits \geq 50$, to reduce the overhead, while we don't apply any optimization for the Supremacy benchmark. Results follow the same trends as previous works, our postprocessing approach seem faster than [33] – likely, because of using a GPU vs CPU.

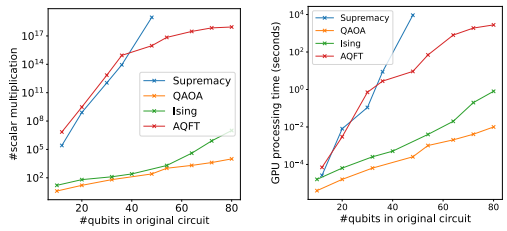


Figure 12: Overheads associated with result recombination in postprocessing. (a) illustrates the number of scalar multiplications to be performed in postprocessing, while (b) shows the postprocessing time on a NVIDIA V100 GPU.

6 Related Works

CutQC[33] is a framework that cuts a large quantum circuit in subcircuits and execute them on (small) quantum computers using the *wire cut* protocol [28]. ScaleQC [32] is an extension of CutQC's post-processing module, integrating tensor network contraction specific for *wire cut*. Our work focuses on *gate cut* protocol. Qiskit Circuit

Knitting Toolbox (CKT) [13] provides out-of-the-box circuit knitting protocols, but the core cutting solution have to be instructed manually by the user. Moreover, CKT's postprocessing only calculates the result of observable expectation, thus only supports applications like VQE and QAOA. Baker et al. propose a time-sliced circuit cutting[3], while Tomesh et al. [35] design a *wire cut* framework for QAOA programs. No previous work integrates hardware layout information during circuit cutting, like we propose.

FCM [24] utilizes wire cutting to improve the fidelity of Measurement Based Quantum Computing (MBQC), while our framework is compatible with various type of quantum backends. TopGen [10] generates topology-aware subcircuits for VQA algorithm. JigSaw [11] and Quixote [16] are error mitigation frameworks that measure only part of the qubits and reconstructs the result with higher fidelity; similarly, Li et al. [21, 22] leverage circuit cutting to enhance the effectiveness of error mitigation techniques, while we focus on optimizing circuit cutting and compilation themselves to improve the result. Bandic et al. utilize graph theory-based metrics to characterize quantum circuits[4].

7 Conclusion

Developing methods to fully exploit current NISQ devices with their limited number of qubits has been an active area of research. In this work, we propose a hardware-aware circuit knitting framework with graph similarity optimization, to facilitate the identification of better cutting schemes that can significantly reduce circuit depth. We fully implemented the entire framework and performed evaluations using real quantum processors. Our results surpass those of approaches that perform cutting and compilation independently, pioneering pathways for practical circuit knitting. As quantum hardware continues to advance, our techniques will be invaluable for future distributed quantum computing scenarios.

References

[1] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In Proceedings of the International Conference on Pattern Recognition Applications and Methods - Volume 1, ICPRAM 2015, page 271–278, Setubal, PRT, 2015. SCITEPRESS - Science and Technology Publications, Lda.

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. Nature, 574(7779):505–510, 2019.

[3] Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. Time-sliced quantum circuit partitioning for modular architectures. CF '20, page 98–107. ACM, 2020.

[4] Medina Bandic, Carmen G Almudever, and Sebastian Feld. Interaction graph-based characterization of quantum benchmarks for improving quantum circuit mapping techniques. Quantum Machine Intelligence, 5(2):40, 2023.

[5] Adriano Barenco, Artur Ekert, Kalle-Antti Suominen, and Päivi Törmä. Approximate quantum fourier transform and decoherence. Phys. Rev. A, 54:139–146, Jul 1996.

[6] Sebastian Brandhofer, Ilia Polian, and Kevin Krsulich. Optimal partitioning of quantum circuits using gate cuts and wire cuts. IEEE Transactions on Quantum Engineering, 5:1–10, 2024.

[7] Sergey Bravyi, Graeme Smith, and John A. Smolin. Trading classical and quantum computational resources. Phys. Rev. X, 6:021043, Jun 2016.

[8] Lukas Brenner, Christophe Piveteau, and David Sutter. Optimal wire cutting with classical communication. arXiv preprint arXiv:2302.03366, 2023.

[9] Daniel Chen, Betis Baheri, Vipin Chaudhary, Qiang Guan, Ning Xie, and Shuai Xu. Approximate quantum circuit reconstruction. In 2022 IEEE International Conference on Quantum Computing and Engineering (QCE), pages 509–515, 2022.

[10] Jinglei Cheng, Hanrui Wang, Zhiding Liang, Yiyu Shi, Song Han, and Xuehai Qian. Topgen: Topology-aware bottom-up generator for variational quantum circuits. arXiv preprint arXiv:2210.08190, 2022.

[11] Poulami Das, Swamit Tannu, and Moinuddin Qureshi. Jigsaw: Boosting fidelity of nisq programs via measurement subsetting. MICRO-54, pages 937–949, 2021.

[12] Andrew Eddins, Mario Motta, Tanvi P. Gujarati, Sergey Bravyi, Antonio Mezzacapo, Charles Hadfield, and Sarah Sheldon. Doubling the size of quantum simulators by entanglement forging. PRX Quantum, 3:010309, Jan 2022.

[13] Luciano Bello et al. Circuit Knitting Toolbox. https://github.com/Qiskit-Extensions/circuit-knitting-toolbox, 2023.

[14] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[15] IBM Quantum. Access Plans: Access quantum computing. https://www.ibm.com/quantum/pricing, 2024.

[16] Enhyeok Jang, Seungwoo Choi, and Won Woo Ro. Quixote: Improving fidelity of quantum program by independent execution of controlled gates. In 2023 60th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2023.

[17] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. J. ACM, 43(4):601–640, jul 1996.

[18] Danai Koutra, Ankur P. Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. In Technical Report of Carnegie-Mellon-Universit, 2011.

[19] Lingling Lao and Dan E. Browne. 2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms. ISCA '22, page 351–365. ACM, 2022.

[20] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. ASPLOS '19, page 1001–1014, New York, NY, USA, 2019. ACM.

[21] Peiyi Li, Ji Liu, Alvin Gonzales, Zain Hamid Saleem, Huiyang Zhou, and Paul Hovland. Qutracer: Mitigating quantum gate and measurement errors by tracing subsets of qubits. arXiv preprint arXiv:2404.19712, 2024.

[22] Peiyi Li, Ji Liu, Hrushikesh Pramod Patil, Paul Hovland, and Huiyang Zhou. Enhancing virtual distillation with circuit cutting for quantum error mitigation. In 2023 IEEE 41st International Conference on Computer Design (ICCD), pages 94–101. IEEE, 2023.

[23] Kosuke Mitarai and Keisuke Fujii. Constructing a virtual two-qubit gate by sampling single-qubit operations. New Journal of Physics, 23(2):023021, feb 2021.

[24] Zewei Mo, Yingheng Li, Aditya Pawar, Xulong Tang, Jun Yang, and Youtao Zhang. Fcm: A fusion-aware wire cutting approach for measurement-based quantum computing. 2024.

[25] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, et al. Quantum optimization using variational algorithms on near-term quantum devices. Quantum Science and Technology, 3(3):030503, 2018.

[26] Hakop Pashayan, Joel J. Wallman, and Stephen D. Bartlett. Estimating outcome probabilities of quantum circuits using quasiprobabilities. Phys. Rev. Lett., 115:070501, Aug 2015.

[27] Aditya Pawar, Yingheng Li, Zewei Mo, Yanan Guo, Youtao Zhang, Xulong Tang, and Jun Yang. Integrated qubit reuse and circuit cutting for large quantum circuit evaluation. arXiv preprint arXiv:2312.10298, 2023.

[28] Tianyi Peng, Aram W. Harrow, Maris Ozols, and Xiaodi Wu. Simulating large quantum circuits on a small quantum computer. Phys. Rev. Lett., 125:150504, Oct 2020.

[29] Christophe Piveteau and David Sutter. Circuit knitting with classical communication. IEEE Transactions on Information Theory, pages 1–1, 2023.

[30] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[31] Bochen Tan and Jason Cong. Optimality study of existing quantum computing layout synthesis tools. IEEE Transactions on Computers, July 2020.

[32] Wei Tang and Margaret Martonosi. ScaleQC: A scalable framework for hybrid computation on quantum and classical processors. arXiv preprint arXiv:2207.00933, 2022.

[33] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. CutQC: using small quantum computers for large quantum circuit evaluations. ASPLOS '21, page 473–486. ACM, 2021.

[34] The cuQuantum development team. Nvidia cuquantum sdk (v23.10.0), 2023.

[35] Teague Tomesh, Zain H. Saleem, Michael A. Perlin, Pranav Gokhale, Martin Suchara, and Margaret Martonosi. Divide and conquer for combinatorial optimization and distributed quantum computation, 2021.

[36] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. Autocomm: A framework for enabling efficient communication in distributed quantum programs. MICRO-55, pages 1027–1041, 2022.

[37] Hezi Zhang, Keyi Yin, Anbang Wu, Hassan Shapourian, Alireza Shabani, and Yufei Ding. Compilation for quantum computing on chiplets. arXiv preprint arXiv:2305.05149, 2023.