SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Scalable Hybrid Quantum-Classical Computing with Tensor Networks

Nathaniel Tornow

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Scalable Hybrid Quantum-Classical Computing with Tensor Networks

Skalierbares hybrides Quanten-Klassisches Rechnen mit Tensornetzwerken

Author:	Nathaniel Tornow
Examiner:	Prof. Dr.-Ing. Pramod Bhatotia
Supervisor:	Prof. Dr.-Ing. Pramod Bhatotia
Submission Date:	16.12.2024

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 16.12.2024 Nathaniel Tornow

Acknowledgments

I am grateful to Prof. Pramod Bhatotia for his guidance and mentorship throughout this work, Prof. Christian Mendl for his valuable technical advice and support, and Laura Schulz for her encouragement and leadership at LRZ QCT.

I thank Manos Giortamis, Francisco Romão, Aleksandra Świerkowska, and Oğuzcan Kırmemiş for their collaboration and discussions. I also appreciate the technical insights provided by Lukas Schmitt, Minh Chung, Jorge Evaria, Burak Mete, Qunsheng Huang, and Muhammad Farooqi.

I am grateful for the support of the entire Systems Research Group at TU Munich and the QCT group at LRZ. I also thank the Future-Compute group at LRZ for providing the testbed infrastructure.

Abstract

Circuit knitting offers a promising path to the scalable execution of large quantum circuits by breaking them into smaller sub-circuits whose output is recombined through classical postprocessing. However, current techniques face excessive overhead due to a naive postprocessing method that neglects potential optimizations in the circuit structure. To overcome this, we introduce qTPU, a framework for scalable hybrid quantum-classical processing using tensor networks. By leveraging our hybrid quantum circuit contraction method, we represent circuit execution as the contraction of a hybrid tensor network (h-TN). The qTPU compiler automates efficient h-TN generation, optimizing the balance between estimated error and postprocessing overhead, while the qTPU runtime supports large-scale h-TN contraction using quantum and classical accelerators. Our evaluation shows orders-of-magnitude reductions in postprocessing overhead, a $10^4\times$ speedup in postprocessing, and a $20.7\times$ reduction in overall runtime compared to the state-of-the-art Qiskit-Addon-Cutting (QAC).

Contents

1 Introduction

Quantum computing could enable exponential speedups in certain computations such as quantum simulation or optimization [Kim+23; FGG14]. Thereby, an increasing number of data centers are adding quantum processors (QPUs) to their repertoire of computational resources [Qua24a; Qua24c; Bra24; AI24]. However, current QPUs are still subject to technological challenges due to severe noise causing qubit decay and state decoherence, making it currently impossible to outperform alternative classical methods [Pre18; Tin+24; ZSW20].

To eventually make quantum computing practical, we must employ computations that integrate high-performance classical resources with QPUs, using them as accelerators for specific, suitable tasks [Ale+24; Qua24b; Els+23]. A promising method for scalable hybrid computing is quantum circuit knitting, which facilitates quantum circuit execution by leveraging the strengths of both quantum and classical computing [Pen+20; MF21]. This approach involves decomposing a large quantum circuit into smaller subcircuits, which can be executed with reduced noise on smaller, noisy QPUs. The original circuit's final result is then reconstructed through classical postprocessing (PP).

Theoretically, its divide-and-conquer approach could make circuit knitting a highly suitable mechanism for scaling quantum workloads in data centers with increasingly better QPUs to execute the subcircuits, combined with an abundance of classical resources for the necessary classical PP.

However, the practical applicability of quantum circuit knitting remains greatly hampered by its challenges of prohibitively high exponential sampling and postprocessing (PP) overheads [Pen+20; MF21; Ufr+23a; BPS23; PS22; PS23; Tan+21; BSS16; Bra+24]. One main limitation of current knitting tools is their reliance on a brute-force PP method, which always assumes worst-case overhead [Bra+24]. These approaches fail to exploit efficient structures within decomposed circuits that could drastically lower computational costs and hamper the effective use of tensor processing capabilities of GPUs/ TPUs, making it difficult to scale beyond classical techniques.

To overcome these scalability barriers, we pose the following research question: *Can we design a circuit knitting-based framework that enables large-scale and GPU-accelerated quantum-classical processing of quantum circuits?* Such a framework should be able to outperform classical simulators while producing fewer errors than a single QPU.

In pursuit of this goal, we identify three key challenges:
(1) We must develop a novel technique that significantly reduces PP overhead by exploiting the structure of decomposed circuits. **(2)** We need an automated and scalable procedure to convert large circuits into an optimized format to enable such an efficient technique. **(3)** To enable scalable hybrid circuit co-processing, we require a system capable of efficiently executing quantum and classical workloads across a cluster of hybrid resources.

To address these challenges, in this work, we introduce *qTPU: A framework for large-scale hybrid quantum-classical processing using tensor networks*. To design such a framework, we make the following core contributions:

- We present our core idea of **hybrid quantum circuit contraction**, which enables us to represent the execution of a quantum circuit as a contraction of a *hybrid tensor network (h-TN)*, essentially accelerating the circuit knitting computation using TNs and accelerating PP by orders of magnitude. (§ 4)
- To enable efficient hybrid circuit contraction and reap the full benefits of our h-TN form, we introduce the **qTPU compiler**. The qTPU compiler converts large quantum circuits into efficient h-TNs, while using hyperparameter optimization to find an optimal, user-defined tradeoff between the expected error of circuits on available noisy QPUs and knitting overheads. To do so, we extend well-studied concepts of optimizing TN contraction paths [GK21]. (§ 5)
- Finally, we present the **qTPU runtime**, a system for large-scale contraction of h-TNs using a hybrid cluster of QPUs and GPUs, by building on highly performant tensor processing libraries. The runtime allows for efficient sampling to enable a tradeoff between tolerated error and computational resources. (§ 6)

We implement qTPU in Python, using Qiskit, cotengra, and cuTensorNet [Jav+24; NVI24; GK21]. Our evaluation on an Nvidia A100 GPU shows that, compared to the state-of-the-art framework *Qiskit-Addon-Cutting* [Bra+24], we reduce PP overhead by several orders of magnitude. This results in a PP speedup of 10^4, with an additional GPU speedup of up to $18\times$. Overall, we achieve an average end-to-end runtime speedup of $20.7\times$ and reduce PP time to a fraction of the total runtime.

2 Background and Motivation

2.1 Quantum Circuit Knitting

Circuit knitting allows us to decompose a large quantum circuit into smaller subcircuits that can be executed independently on smaller QPUs, mitigating noise and scaling circuit execution to more than the size of available QPUs [Pen+20; MF21; BSS16; Tan+21; Bra+24; RZB24; YO22]. We show this by the example of Fig. 2.1, where we need to *cut* four two-qubit gates and one qubit wire to dissect the large circuit into four smaller subcircuits with two qubits each, which we run on smaller QPUs. Using classical postprocessing (PP), the large circuit's output can be reconstructed with the subcircuit's results, with the final output being an expectation value of an observable O.

2.1.1 Quasiprobability Decomposition

Circuit knitting is primarily built on the **Quasiprobability Decomposition (QPD)** method, which enables us to decompose qubit wires and multi-qubit gates into a combination of single-qubit unitaries or projective measurements. The core idea of QPD is depicted in Fig. 2.2. Here, we cut a two-qubit gate by representing the circuit as a sum of several circuit instances i, each weighted by a coefficient c_i, where the single-qubit operations A_i and B_i are inserted, replacing the original gate (Fig. 2.2 (b)) [MF21]. Since we only insert single-qubit gates in each instance, we can divide circuits into two smaller subcircuits, which can be executed independently on a smaller QPU. When assuming that the subcircuits estimate O_1 and O_2, respectively, with $O = O_1 \otimes O_2$, we can reconstruct the result by calculating $\langle O \rangle = \sum_{i=0}^{6} c_i \langle O_1^i \rangle \langle O_2^i \rangle$ [MF21].

Similarly, we can cut qubit-wires using QPD (Fig. 2.2 (c)). The only difference to QPD of two-qubit gates is that we need eight instances, and the gates to insert are measurements in different computational bases for A_i and preparation of corresponding states for B_i [Pen+20].

2.1.2 Circuit Knitting Overheads

To generalize the circuit knitting to multiple cuts in a circuit as in Fig. 2.1 (b), we define a coefficient vector $\mathbf{c}_{g_k} = (c_1, c_2, \dots)$ for each decomposed gate g_k, and a *global*

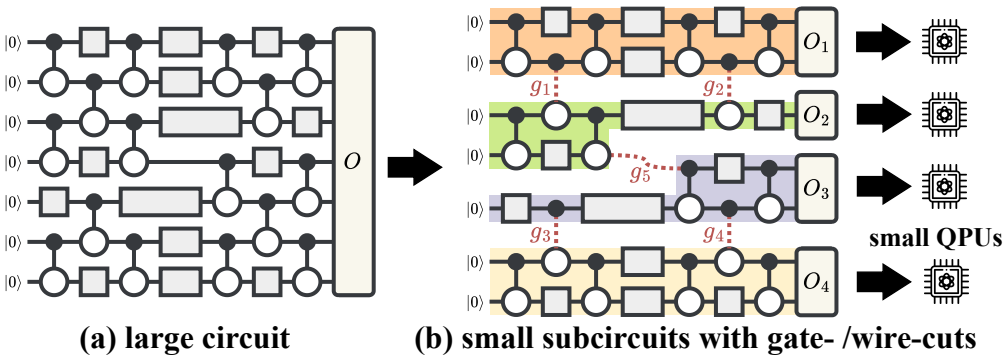


Figure 2.1: Circuit Knitting. *(a) Large quantum circuit. (b) Decomposed quantum circuit with five cuts and four subcircuits that can be run on small QPUs.*

coefficient vector $\mathbf{C} = \bigotimes_{g_k} \mathbf{c}_{g_k}$. To then reconstruct the result $\langle O \rangle$ with circuit knitting, we need to calculate

$$\langle O \rangle = \sum_{c_i \in \mathbf{C}} c_i \prod_{j=1}^{s} \langle O_j^i \rangle \,, \tag{2.1}$$

where s is the number of subcircuits and $\langle O_j^i \rangle$ the result of the jth subcircuit in the ith *global* instance.

Severe overheads greatly limit the scalability of QPD-based circuit knitting [MF21; Pen+20]. These overheads can be categorized into (1) the sampling and (2) the PP overhead:

The **sampling overhead** is the number of samples/shots we must draw from the circuit's instances to estimate the result to a given precision. The sampling overhead is given as $\mathcal{O}(9^k/\varepsilon^2)$ and $\mathcal{O}(16^k/\varepsilon^2)$ total samples for reconstructing the output with a precision ε for k gate/wire cuts [MF21; Pen+20].

The **postprocessing (PP) overhead** is the overhead that stems from the classical PP of the subcircuit's results. We define the PP overhead as the number of float-multiplications (FLOPs) required to perform the PP following Eq. 2.1. If we use a *naive* implementation of Eq. 2.1, as currently implemented [Bra+24], we require $|\mathbf{C}|(s + n_g - 1)$ FLOPs to postprocess the results, where n_g is the number of QPD-gates/-wires [Bra+24], as we need $|\mathbf{C}|(n_g - 1)$ FLOPs to compute $\mathbf{C}$ and $|\mathbf{C}| \cdot s$ FLOPs to compute Eq. 2.1. We can assume that the number of subcircuits s increases sub-linearly and $|C|$ increases exponentially with the number of cuts. The PP overhead is therefore bounded by $\mathcal{O}(|\mathbf{C}|) \subseteq \mathcal{O}(8^k)$ when cutting k gates or wires.

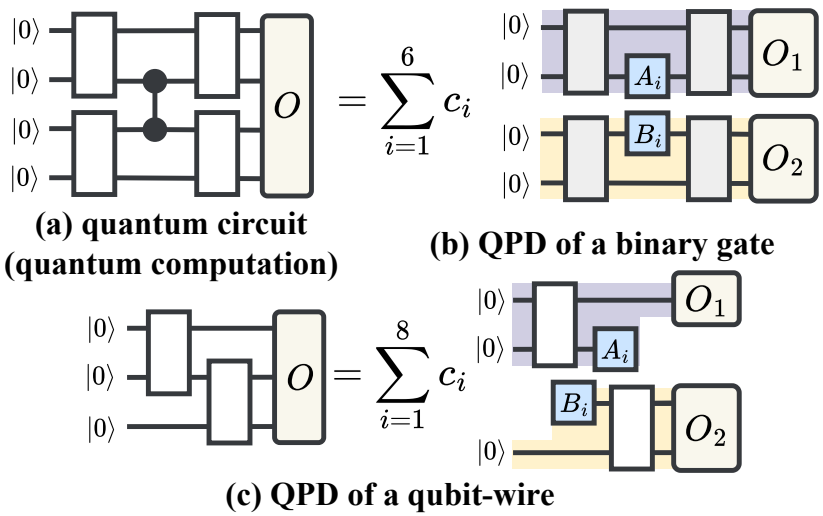


Figure 2.2: Quasiprobability decomposition (QPD). *(a) Quantum computation in the form of a quantum circuit and estimation of the expectation value of the observable* $O = O_1 \otimes O_2$. *(b) QPD of a two-qubit gate. (c) QPD of a qubit wire.*

2.1.3 Problem: Brute-force postprocessing

State-of-the-art circuit knitting methods face a significant challenge due to an unnecessary exponential overhead arising from a naive implementation, failing to account for efficient structures that could reduce this overhead. To highlight the limitations of the current naive method, we use the decomposed circuit in Fig. 2.1 (b) as an illustrative example. Here, we use four gate-cuts ($g_{1\text{-}4}$) and one wire-cut (g_5) to decompose the circuit into $s = 4$ subcircuits. Here, the total PP overhead is $|\mathbf{C}|(s + n_g - 1) = 10368 \cdot (4 + 5 - 1) = 82944$ FLOPs, with an asymptotic PP cost of $\mathcal{O}(|\mathbf{C}|)$ where $|\mathbf{C}| = 10368$ subcircuit instances must be generated.

However, the structure of the decomposed circuit suggests that a significantly more efficient approach to knitting the circuit may be possible. In particular, the circuit structure is relatively sparse; for instance, subcircuits 1 and 2 can be knitted completely independently from subcircuit 4. Additionally, QPD gates g_1 and g_2 only affect subcircuits 1 and 2, leaving subcircuits 3 and 4 unaffected. Thus, creating instances for each global $c_i \in \mathbf{C}$ is unnecessary.

Despite this potential for efficiency, the current naive PP approach still requires us to compute a sum over the entire tensor product of the coefficients $\mathbf{C}$. As a result, we are bound to the worst-case PP overhead, regardless of how sparse the circuit structure

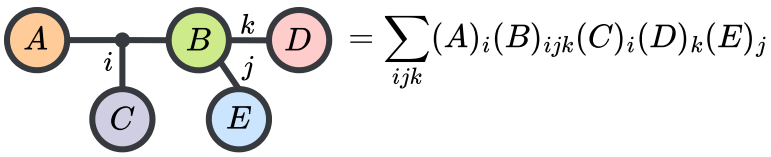


Figure 2.3: Tensor network (TN). *A tensor network of five tensors, connected by two indices and one hyper-index, depicted by a hyperedge connecting the three tensors A, B, and C. The sum on the right represents the contraction of the tensor network, resulting in a single scalar.*

may be. Furthermore, the naive implementation of a sum over every global coefficient complicates efforts to speed up PP using parallel processing on GPUs [Bra+24].

2.2 Tensor Networks

Tensor networks (TNs) enable the efficient storage and manipulation of high-dimensional data, making them a powerful tool for quantum circuit simulation [Orú19; BC17; Orú14; Cic14]. A TN provides a graphical representation of the contraction of multiple tensors. In this representation, vertices represent tensors, and edges represent the indices of those tensors. A contraction occurs across an index when two or more tensors are connected via an edge or hyper-edge. Fig. 2.3 illustrates a TN with five tensors labeled A to E and three indices. When indices connect multiple tensors, we contract by "summing over" the corresponding indices, as indicated by the expression on the right.

The primary reason for the effectiveness of TNs is their ability to efficiently represent quantum states of large systems that exhibit low entanglement [FNW92; Has07]. A quantum circuit execution can also be represented as an equivalent TN, which can be contracted to obtain the result [ZSW20; Gra18; Sei+23; Tin+24; NVI24]. However, the main limitation of this approach is the degree of intra-connectivity between the gates or qubits within the circuit. Higher connectivity tends to increase the complexity of the tensor contractions, impacting performance.

2.2.1 Motivation: Accelerating circuit knitting

In this work, we contend that TNs could also effectively reduce PP overhead by accelerating computation during this phase. Thus, we aim to express circuit knitting as an efficient TN, which can significantly decrease PP costs for suitable circuit cuts. By leveraging TN processing for PP, we can take advantage of existing, highly optimized tensor-processing libraries that allow us to offload computations to GPUs, further enhancing performance.

3 Overview

We aim to develop a data center-scale co-processing framework that enables efficient, hybrid execution of large quantum circuits using QPUs and classical tensor accelerators. To do so, we identify three core design challenges.

3.1 Design Challenges

3.1.1 Challenge #1: Enabling quantum-classical processing

As pointed out in § 2.1, PP of QPD-based circuit knitting suffers from severe exponential overheads, which in many cases, depending on the specific structure of the cuts, can be unnecessarily high due to the currently used naive approach. To enable circuit knitting as a viable hybrid quantum-classical processing technique that could go beyond a handful of circuit cuts, we need a new approach similar to the approach of TNs in quantum simulation that reduces the PP overhead to an acceptable level.

3.1.2 Challenge #2: Efficiency and adaptability

To fully utilize a scalable hybrid processing technique, we need automatic and efficient procedures to transform a quantum circuit into an optimized quantum-classical program. This optimized program should offer an optimal trade-off between (1) PP overhead and (2) other user-specified overheads, such as the error incurred when executing subcircuits on noisy QPUs. In the context of circuit knitting-based processing, this requires identifying efficient circuit partitions that balance the dual objectives of minimizing both quantum and classical overhead while accounting for the expected noise from running subcircuits. Ideally, these circuit-cutting techniques should be near-optimal and capable of scaling to handle large circuits with thousands of qubits and significant depths.

3.1.3 Challenge #3: Large-scale hybrid processing

Finally, to execute the hybrid quantum-classical workflow at scale, we need a system that supports operation on a hybrid cluster of multiple QPUs and GPUs. The challenges

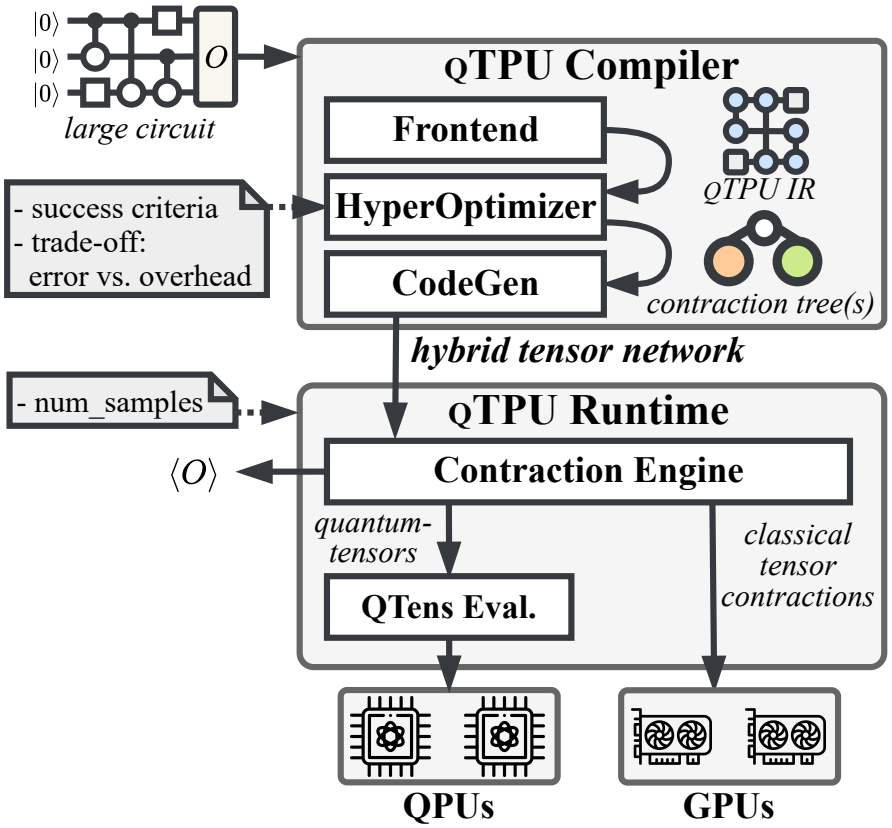


Figure 3.1: qTPU System Overview.

in this context include adapting the computational cost through sampling and efficiently offloading quantum and classical tasks to their respective devices using parallel processing. Additionally, we should be able to leverage existing, highly optimized tensor-computing libraries to enhance performance.

3.2 The qTPU Framework

To tackle the challenges outlined above, we present qTPU, a framework for large-scale quantum-classical co-processing (see Fig. 3.1). qTPU is built upon our core concept of *hybrid quantum circuit contraction* (see Fig. 4.1). To execute a large circuit across a set of QPUs and GPUs, we divide the framework into two components: the *qTPU compiler* and the *runtime*.

3.2.1 Hybrid quantum circuit contraction

Hybrid circuit contraction is our approach to minimizing the PP overhead associated with knitting a given circuit. The core idea is to represent the execution of a quantum circuit as a **hybrid tensor network (h-TN)** that consists of quantum and classical tensors. By encoding the execution as an h-TN, we effectively represent the knitting procedure in a more efficient format based on a TN. This technique enables a best-case PP overhead that can outperform current PP-methods by several orders of magnitude, depending on how the original circuit is compiled.

3.2.2 qTPU compiler

As the performance of our hybrid circuit contraction technique depends on the specific structure of a circuit's decomposition, we present the qTPU compiler, which compiles a large quantum circuit into an efficient h-TN while finding a balance between classical overhead and errors caused by QPUs. To do so, the compiler consists of three main components: First, the *frontend* converts the circuit into the qTPU intermediate representation (IR), a TN-like representation of the circuit. Then, the *optimizer* finds an optimal decomposition of the IR using contraction trees. The optimization is based on user-given success criteria and a trades off between expected error and processing overheads. Finally, the *code generator* transforms the decomposed IR into an h-TN.

3.2.3 qTPU runtime

For the large-scale contraction of an h-TN, we present the qTPU runtime. To contract an h-TN, the runtime first applies sampling to reduce the requirements of classical

memory and time resources at the cost of approximations. Then, the *contraction engine* facilitates the hybrid contraction by dividing the contraction into quantum and classical tasks. For the quantum-part of the contraction, quantum-tensors are evaluated by running quantum circuits on a set of QPUs using the *quantum-tensor evaluator*.

4 Hybrid Quantum Circuit Contraction

Our goal is to enable large-scale and accelerated hybrid processing, which is currently unfeasible in circuit knitting due to its naive approach. To address this, we introduce our key concept of *hybrid circuit contraction*, which represents circuit knitting in the form of a *hybrid tensor network (h-TN)*.

4.1 Key Idea: Hybrid Quantum-Tensor Contraction

To grasp the concept behind hybrid circuit contraction, we first need to understand how to represent a QPD of a single gate or wire as an h-TN. We illustrate this relationship in Fig. 4.1. Here, Fig. 4.1 (a) shows the standard QPD formula as a sum over 6 circuit instances i, each decomposed into two subcircuits and weighted by coefficients c_i. We can then express this formula as an h-TN, as shown in Fig. 4.1 (b). To achieve this, we group the coefficients into a classical tensor (CT) $\mathbf{c} = (c_1, \ldots, c_6)$ and the instances of the two subcircuits into two quantum tensors (QTs). A QT is essentially a tensor (in this case, a vector), with its elements representing the instances of the respective subcircuit. To match the QPD formula, the tensors are connected via a hyper-index.

To contract an h-TN, we first *evaluate* each QT by running the instances that yield the outputs $\langle O_j^i \rangle$ for the jth subcircuit in the ith instance. For each QT, the results are written into a corresponding classical tensor (CT) $\langle O_j \rangle$, where the elements represent the results of the respective instances (as shown in Fig. 4.1(c)). Finally, we *contract* this fully classical TN to obtain the final result, $\langle O \rangle$.

Similarly, to represent a wire cut as an h-TN, we can apply the same method using the identity shown in Fig. 2.2(c). However, to be more efficient, we can adopt the approach from CutQC [Tan+21], which performs a wire cut using only four basis measurements A_i and four state preparations B_i, as depicted in Fig. 4.1(d). Unlike the h-TN for a gate, this approach requires a two-dimensional 4×4 CT and four instances A and B for the two QTs, following the CutQC method closely:

$$c_w = \frac{1}{2} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix} \quad \begin{matrix} A = (I,\ Z,\ X,\ Y), \\ B = (|0\rangle,\ |1\rangle,\ |+\rangle,\ |i\rangle) \end{matrix} \tag{4.1}$$

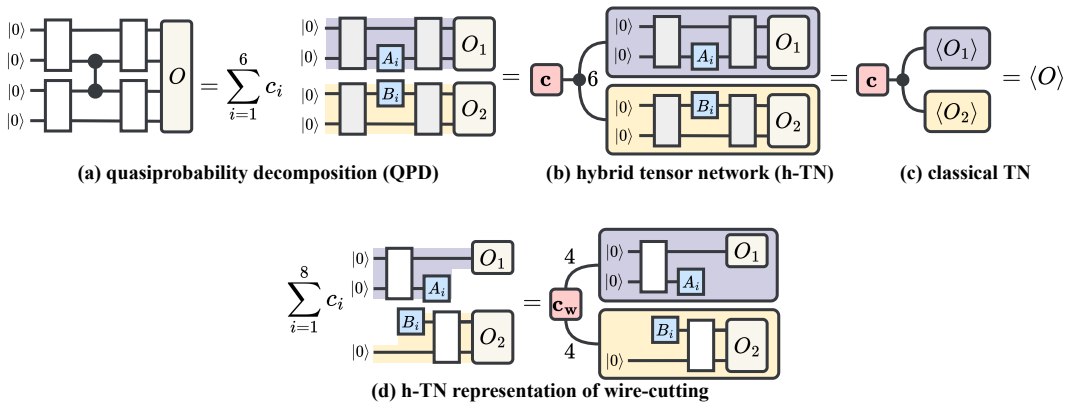


Figure 4.1: Hybrid Quantum Circuit Contraction. *Through Quasiprobability-Decomposition (QPD), we can represent the execution of a quantum circuit as a hybrid tensor network (h-TN) of both quantum and classical tensors.*

This approach drastically reduces the memory complexity and contraction cost with multiple cuts.

4.2 Postprocessing Overhead Analysis

Through hybrid circuit contraction, we can now represent individual circuit cuts as an h-TN. Next, we analyze the impact of this approach on PP overhead when cutting multiple gates or wires, as illustrated by the example in Fig. 4.2.

Here, we consider the same decomposed circuit with 5 cuts that we discussed for the current brute-force knitting in § 2.1. Using our method, this circuit can now be represented as an h-TN, consisting of five classical tensors (CTs), one for each QPD, four quantum tensors (QTs), and one for each subcircuit.

To contract the h-TN, we first evaluate each quantum tensor, yielding the corresponding classical tensors building a classical TN in Fig. 4.2 (c). The classical PP then only consists of contracting the classical TN. The (optimal) cost of contracting the TN of Fig. 4.2 (c) is 884 FLOPs, which is *orders of magnitude* smaller than the cost of the brute-force knitting approach of § 2.1. We obtain this cost by first contracting the tensors of $\langle O_1 \rangle$ and $\langle O_2 \rangle$, as well as $\langle O_3 \rangle$ and $\langle O_4 \rangle$ with their respective connecting classical tensors $\mathbf{c}_{1-4}$, each with a cost of 432 FLOPs. Finally we contract the two resulting tensors and the $\mathbf{c}_5^w$ tensor with 20 FLOPs, resulting in a total of 884 FLOPs.

The PP overhead of hybrid circuit contraction relies entirely on the contraction cost of the final classical TN. This cost heavily depends on the order of tensor contractions performed, making an efficient contraction sequence essential [GK21; NVI24].

In the worst case, contracting a TN is asymptotically equivalent to the current knitting method described in Eq. 2.1 when we fail to identify a more efficient contraction sequence or when the h-TN structure does not allow for it [BC17]. Conversely, in the best case, if the h-TN includes sufficient QTs connected by CTs and we find an optimal contraction sequence, we achieve minimal PP overhead for a given decomposed circuit.

The effectiveness of our approach in reducing PP costs depends on our ability to accelerate the knitting computation by leveraging the sparse structure of the decomposed circuit, akin to existing TN methods for efficient representations of large quantum states [FNW92; Vid03]. This approach occurs by treating the knitting process as a series of pairwise independent knitting operations between subcircuits rather than knitting the entire circuit in one step. For example, subcircuits 1 and 2, as well as subcircuits 3 and 4, are knitted independently through efficient partial contractions.

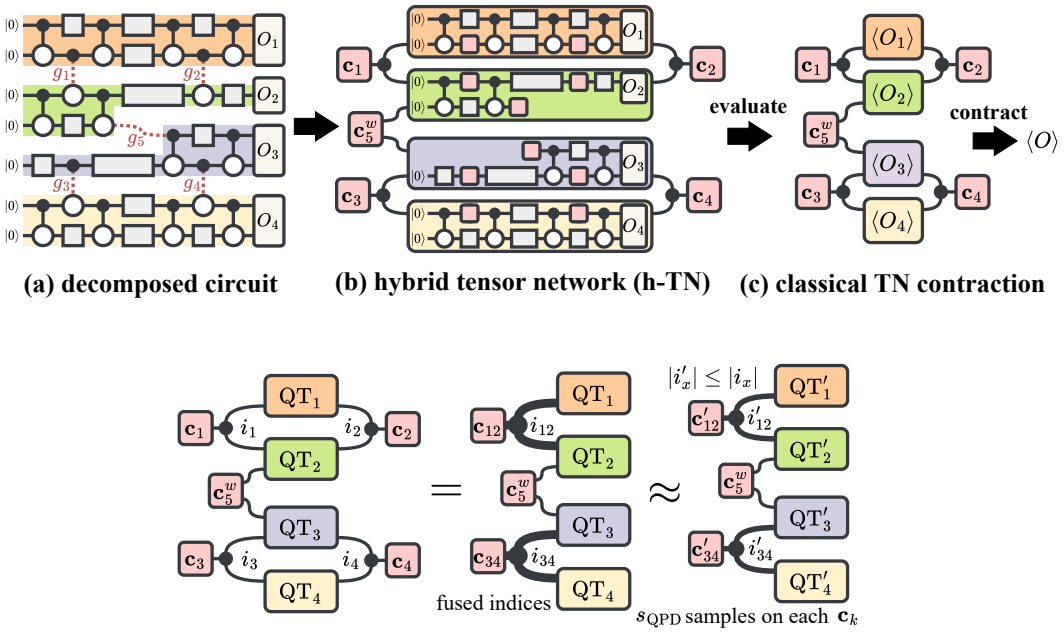
4.3 QPD-Sampling and h-TN Simplification

In addition to its postprocessing (PP) overhead, circuit knitting incurs a significant sampling overhead that determines the total number of samples required to estimate the expectation value $\langle O \rangle$ to an accuracy ε [MF21; Pen+20] (§ 2.1). We can reduce the number of samples s_{QPD} to lessen the overhead, but this comes at the cost of lower-quality approximations.

In the current brute-force knitting approach, we sample s_{QPD} instances and assign them to global circuit instances i, each weighted by its respective coefficient $|c_i|$, where all c_i belong to $\mathbf{C} = \bigotimes \mathbf{c}_k$, the tensor product of all coefficients of QPD gates (see § 2.1) [MF21]. Instances with not-sampled coefficients can be ignored, enabling faster knitting. This process reduces the upper bound of PP to $\mathcal{O}(s_{\text{QPD}})$ [Bra+24].

In contrast, using our hybrid circuit contraction, we can perform equivalent sampling directly in the form of the h-TN, which still gives an exponential reduction in PP cost, as shown in Fig. 4.2 (d). To do so, we first simplify the h-TN by fusing the indices between every two QTs by computing the tensor product of the CTs between the QTs, giving an equivalent but more compact form of the h-TN. Then, we perform sampling on the individual CTs $\mathbf{c}_x$, assigning samples to $(\mathbf{c}_x)_i$ and weighting by $|(\mathbf{c}_x)_i|$. The samples we record along an index determine the number of shots each circuit instance of an adjacent QT must be performed. For all $(\mathbf{c}_x)_i$ that are not sampled at all, we can truncate the CT and its respective index i_x, as we sketch on the right in Fig. 4.2 (d). This truncation also allows us to reduce the size of the respective QTs along the given index, further reducing PP overhead.

Note that the sampling does not apply to the unique wire-cut tensor (Eq. 4.1), as all instances are equally weighted [Bra+24].



(a) decomposed circuit (b) hybrid tensor network (h-TN) (c) classical TN contraction

(d) QPD-sampling on an h-TN

Figure 4.2: Contraction of a hybrid tensor network (h-TN). *(a) The decomposed circuit of Fig. 2.1. (b) The corresponding h-TN. (c) the evaluated (classical) TN that contracts to the result $\langle O \rangle$. (d) QPD-sampling as a simplification of the h-TN.*

5 qTPU Compiler

To fully leverage our hybrid circuit contraction method (§ 4), we need an efficient process to transform large circuits into optimal h-TNs, balancing contraction overhead and estimated error. We address this with the qTPU compiler, as shown in Fig. 5.1.

The process starts with the frontend converting the circuit into the qTPU IR, a TN-like graph of the circuit's decomposition. The optimizer then applies hyperparameter optimization, generating Pareto-optimal solutions that trade off estimated error and overhead. The error metric can be user-defined, and could range from QPU noise error to error correction overhead. Each solution comprises a partitioned IR and a contraction tree, which dictates the hybrid TN's overhead. We select one of the optimal solutions based on the user's preference and finally generate h-TN with its quantum and classical tensors.

5.1 Frontend and qTPU IR

The compiler frontend converts a quantum circuit into the qTPU IR, which captures the connectivity of circuit operations, allowing for efficient optimization and h-TN generation.

In particular, the IR is a graph consisting of pairs (op_i, q_x) as vertices, where op_i is the operation-index, and q_x a qubit in the circuit. Gate-edges (vertical) exist for two-qubit gates between edges having the same operation op_i, and wire-edges (horizontal) exist between subsequent operations for vertices having the same qubit q_x. Each edge is weighted by the sampling overhead of the respective gate/wire to guide the optimization. Thus, we aim to cut the least amount of gates/wires that have the least amount of sampling overhead, such that we (1) minimize the PP overhead and (2) enable the highest amount of overhead reduction during sampling (§ 4.3).

5.1.1 IR compression

To enable more efficient optimization, we compress the IR, as illustrated in Fig. 5.2. Each node in a compressed graph represents a subgraph of nodes from the uncompressed graph. We implement three main compression techniques: One-qubit compression merges single-qubit vertices into neighboring two-qubit operation vertices (Fig. 5.2 (c)).

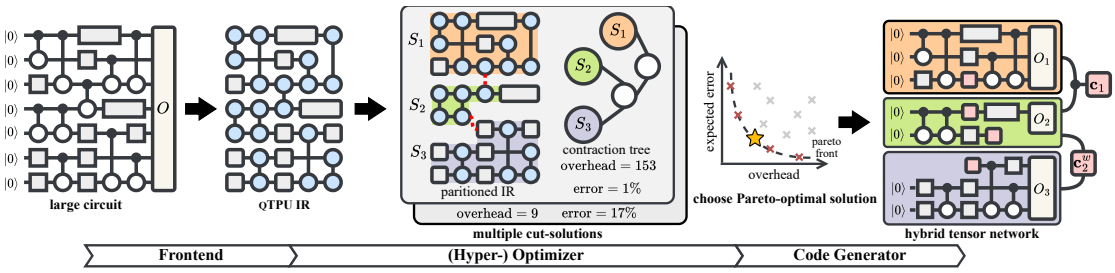


Figure 5.1: Workflow of the qTPU Compiler. *First, the frontend generates the qTPU IR. Then, the hyperoptimizer optimizes the IR using multiobjective optimization between contraction cost and success probability, producing multiple Pareto-optimal solutions, each with a contraction tree that describes a partition of the IR. The code generator generates the corresponding h-TN of user-selected solution.*

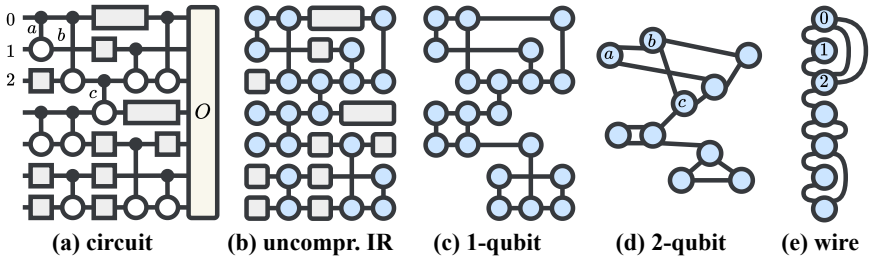


Figure 5.2: Compression of the qTPU IR.

Two-qubit compression merges each vertex of the same operation (Fig. 5.2 (d)). Wire compression merges all vertices corresponding to the same qubit-wire (Fig. 5.2 (e)).

Since single-qubit gates contribute minimally to errors compared to two-qubit operations [Qua24a], one-qubit compression reduces the search space without affecting our ability to find an optimal solution. In contrast, two-qubit and wire compression restrict the search space more aggressively but in a controlled manner: two-qubit compression focuses the search on wire cuts, while wire compression restricts it to gate cuts. This allows users to set compression as a parameter to prioritize wire or gate cuts or to use it as a hyperparameter to accelerate the search in the desired subspace.

5.2 Optimizer

In the second stage of the compiler, the optimizer aims to generate an optimal partition of the IR using a contraction tree. In doing so, it aims to minimize the expected errors of running the resulting subcircuits on (noisy) QPUs and overhead while reaching a

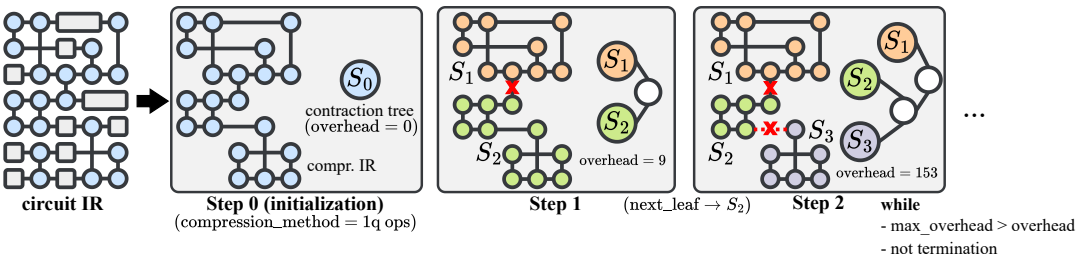


Figure 5.3: Optimizer. *Based on a set of user-defined hyperparameters, we optimize the circuit IR by first applying compression and then iteratively building a contraction tree by (greedily) choosing a tree-leaf to partition until we reach a maximum cost or we reach a terminating condition.*

user-defined success criterion.

5.2.1 Contraction tree

A contraction tree (Fig. 5.1, center) is a binary tree for recursively determining the asymptotic overhead of the qTPU IR [GK21]. In particular, a leaf in the contraction tree describes a subset of vertices in the IR, and each non-leaf vertex describes the contraction between two subsets. In the example of Fig. 5.1, the contraction tree determines that we must contract S_2 and S_3 and then contract the result with S_1.

5.2.2 Hyperparameters

The optimization workflow depends on the following hyper-parameters:

- *Termination*: A function to terminate the optimization. E.g., if the qubit-number is below a given threshold.
- *Max Overhead*: The maximum (asymptotical) PP overhead.
- *Compression*: A qTPU IR compression method to speed up optimization or guide the optimization into the search space of either only wire- or gate-cuts (§ 5.1).
- *Next-Leaf*: A function determining the leaf of the contraction tree to partition next.
- *Graph Partition Params*: Parameters for the iterative graph-partition procedure. Such parameters include the number of partitions or the imbalance factor [Sch+23].

5.2.3 Optimization workflow

To find the optimal decomposition of the IR, we iteratively partition the circuit until we meet a termination criterion. Fig. 5.3 shows an example workflow where we stop when all subcircuits have at most three qubits, the maximum cost is 200, we use the 1q-ops compression method, and our greedy next-leaf function selects the leaf with the most qubits. For graph partitioning, we apply bisection.

We start by compressing the circuit and initializing a contraction tree with a single vertex. In each iteration, we use the next-leaf function to select the leaf for partitioning. For instance, in Fig. 5.3, after step 1, we choose S_2 because it has the most qubits. We then partition the selected subgraph into two or more subgraphs, adding new children to the contraction tree. If the partition produces more than two subsets, we build an optimal subtree from the new nodes [GK21]. We repeat this process until we exceed the overhead limit or meet the termination criterion and return the IR and contraction tree.

5.3 Hyperparameter Optimization

Finding an optimal contraction tree depends heavily on the hyperparameters used in the optimization pipeline (§ 5.2). We use hyper-optimization to identify the best parameter set, aiming to minimize both PP overhead and estimated error. The optimization is constrained by a maximum PP cost and a user-defined termination state [Deb11; Aki+19; GK21].

Running the optimizer with different hyperparameters yields varying results in cost and error. The points where no other solution offers lower cost and error are the Pareto-optimal points, forming the Pareto front [ED18]. These points are optimal solutions for our dual-objective optimization.

To finalize the process, the user supplies a function selecting a point on the Pareto front. By default, we use a function that chooses the point closest to the optimum. This is done by normalizing the points on the Pareto front to $[0, 1]$ for both cost and error, then calculating the Euclidean distance to the ideal point $(0, 0)$, and finally selecting the point with the smallest distance.

5.4 Code Generator

Once the hyper-optimizer returns an optimal compressed IR and contraction tree, the code generator produces the h-TN.

We generate CTs by examining the cut edges in the qTPU IR, spanning two subsets determined by the contraction tree's leaves. For each cut edge, we identify its gate type

and add the corresponding coefficient tensor **c** to the CT set. Wire cuts are handled by adding the two-dimensional tensor from Eq. 4.1. Each index is uniquely identified by the operation's index.

A QT represents multiple sub-circuits, each in a different instance. To optimize storage and processing, we generate one blueprint circuit per QT, incorporating single-qubit placeholder gates at the positions of QPD-decomposed gates associated with classical tensors, allowing us to perform circuit optimization only once for each QT. To create a QT from a subset of the IR corresponding to a leaf in the contraction tree, we iterate through the vertices in the subset, adding the relevant gates to the blueprint circuit. For each cut edge, we insert a placeholder gate to insert instance gates during execution. Finally, we add indices to the QT corresponding to the classical tensors for the cut operations acting on the sub-circuit.

6 The qTPU Runtime

We now describe the qTPU runtime, our scalable system for contracting hybrid TNs.

6.1 Contraction Engine

The contraction engine is the runtime's core component for the contraction of a hybrid TN, running the following steps:
Step 1 (Simplification and sampling): We perform QPD-sampling on the h-TN by first fusing indices through tensor multiplication on CTs between the same QTs and then sampling on the individual CTs a given s_{QPD} times (§ 4.3). This returns a simplified h-TN with truncated CTs and QTs, depending on s_{QPD}. The user can choose the number of samples to enable a tradeoff between accuracy and speed of overall knitting.
Step 2 (QT evaluation): We pass all QTs of the h-TN to the QT evaluator, which returns the respective classical tensor of the expectation values results.
Step 3 (Contraction preparation): We compute the contraction sequence of the evaluated classical TN using hyperoptimization techniques from previous TN optimization work [GK21; NVI24]. Importantly, the contraction sequence of the final classical TN can be computed in parallel with the QT evaluation, as the dimensions of the full TN are already known. This parallelization can significantly reduce the overall computation time for large h-TNs, where both quantum evaluation and contraction-sequence optimizations can be time-consuming.
Step 4 (Classical contraction): Finally, we allocate and contract the classical TN on the available classical accelerators, using a library for contracting classical tensors [NVI24; GK21].

6.2 Quantum Tensor Evaluator

The QT evaluator is responsible for transforming QTs into CTs corresponding to the results of each subcircuit instance. A QT consists of a blueprint circuit with placeholder gates and a tensor of gates to be inserted into the blueprint circuit.

For each QT to evaluate, we first apply the circuit optimization to the blueprint circuit to perform the extensive optimization only once, meaning that each circuit

instance is optimized from one circuit optimization. Then, we generate the circuit instances by inserting the corresponding gates of each element in the instance tensor into the optimized blueprint circuit. We then pass the resulting quantum circuits and the number of shots required by the specific QPD decomposition to an interface capable of executing the quantum circuits and returning the samples. Such an interface could be a single QPU or a cluster of QPUs and/or simulators that can execute the circuits in parallel [Sei+24; Gio+24a]. Finally, we construct and return a classical tensor from the results with the same dimensions as the respective QT.

7 Evaluation

7.1 Methodology

qTPU implementation

We implement qTPU in Python v3.11.9 building on Qiskit v1.1.0 for representing quantum circuits and the Qiskit Addon Cutting (QAC) v0.7.3 for the QPD decompositions of gates. For the compiler, we build on cotengra [GK21] and quimb [Gra18] for the representations of the IR, contraction trees and hybrid TNs, kahypar [Sch20] for efficient graph partitioning and optuna for hyperparameter optimization [Aki+19]. We implement the qTPU runtime using quimb for contractions on CPUs and cuTensorNet on GPUs [Gra18; NVI24].

Framework configuration

Unless otherwise noted, we use the qTPU to compile the circuits into h-TNs with subcircuits with maximal 15 qubits. When running circuit simulations, we use a *BackendSamplerV2* primitive on top of a GPU-accelerated *AerSimulator* performing 20,000 shots per circuit.

Experimental setup

We conduct our experiments on an Intel Xeon Platinum 8360Y CPU with 72 cores (144 HT), 2.40GHz, 362GB RAM, and an NVIDIA A100 80GB PCIe GPU.

Benchmarks

We use four commonly used ansätze as benchmarks: (a) Variational Quantum Eigensolver (VQE) with an efficient $SU(2)$ ansatz and linear entanglement, (b) Quantum Machine Learning (QML) with a ZZ-feature map and pairwise entanglement, (c) Quantum Approximate Optimization Algorithm (QAOA-I) for a random clustered graph with 70% intra-cluster and 30% inter-cluster connection probabilities [Low+23], and (d) QAOA-II for a random clustered graph with 70% intra-cluster connections and p edges between adjacent clusters [Per+21].

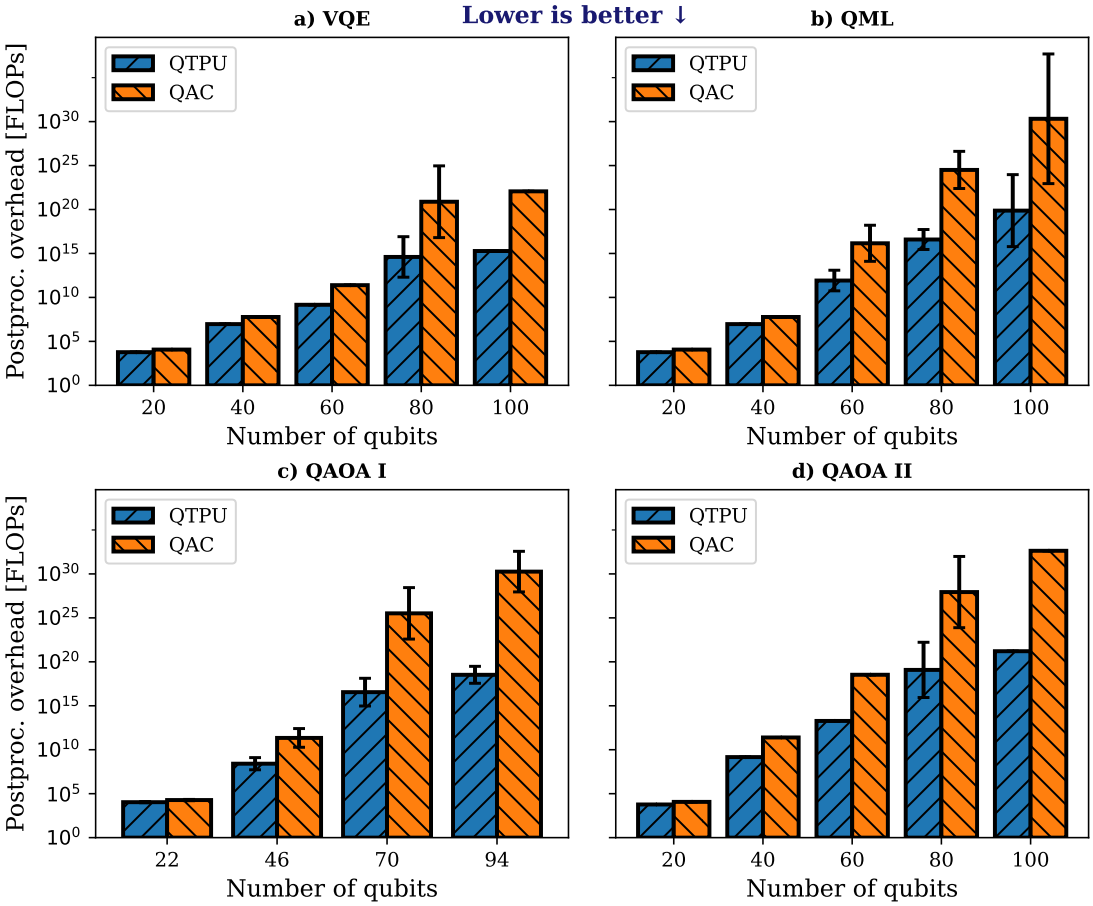


Figure 7.1: Postprocessing overhead. *Classical postprocessing overhead in FLOPs for exact result reconstruction.*

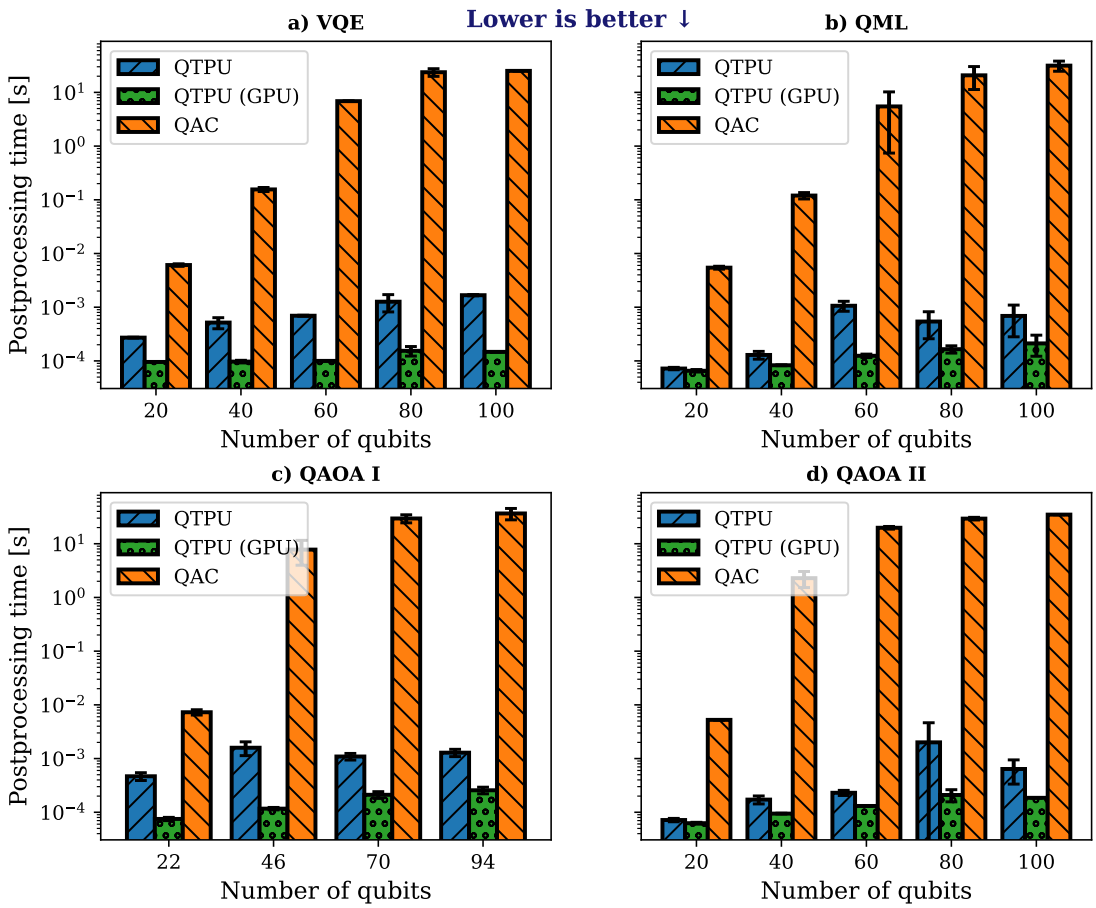


Figure 7.2: Postprocessing time. *Classical postprocessing time for exact result reconstruction of qTPU on CPU and GPU vs. approximate reconstruction using QAC. We limit QAC at 10^5 QPD-samples, as it would otherwise not terminate in a reasonable time.*

Baselines

We use the current state-of-the-art circuit-knitting framework, the "Qiskit Addon Cutting" (QAC) v0.7.3, as our main baseline [Bra+24]. QAC is the only existing QPD-based knitting framework that, like qTPU, allows for both wire and gate cutting. We also compare against classical GPU-accelerated TN simulation cuTensorNet [NVI24].

Metrics

We evaluate qTPU based on the following metrics.

1. *Postprocessing (PP) overhead*: The number of float operations (FLOPs) required for classical PP.
2. *Runtime*: Runtime in seconds for either (1) classical PP or (2) full knitting, including QT-evaluation.
3. *Number of two-qubit gates*: The relative number of two-local gates of the circuit (lower is better). If a circuit contains multiple subcircuits, we report the maximum of all subcircuits.
4. *Estimated Error*: Estimated error of running a circuit, assuming a uniform error model of 10^{-4} and 10^{-3} for single- and two-qubit gates, respectively [Qua24a]. For multiple subcircuits, we provide the maximal error.

7.2 Postprocessing Analysis

To show qTPU's capability to mitigate exponential PP overhead, we use our compiler to compile benchmarks ranging from 20 to 100 qubits, reducing them to subcircuits of no more than 15 qubits, requiring 2 to 10 cuts. We then measure both the PP overhead in FLOPs and the total PP time. We pose the following research questions (RQs).

— RQ1: Postprocessing overhead

How does qTPU's approach impact the PP overhead of circuit knitting?
First, we compute the PP overhead, assuming all QPD coefficients are sampled, and compare this with the PP overhead used by QAC (see Fig. 7.1). We observe only a linear overhead increase for qTPU's PP overhead as qTPU allows accelerated tensor-based PP. At the same time, we see a high exponential growth in overhead for the PP of QAC. In total, qTPU improves overhead by $10^{15}\times$ on average and up to $10^{17}\times$.

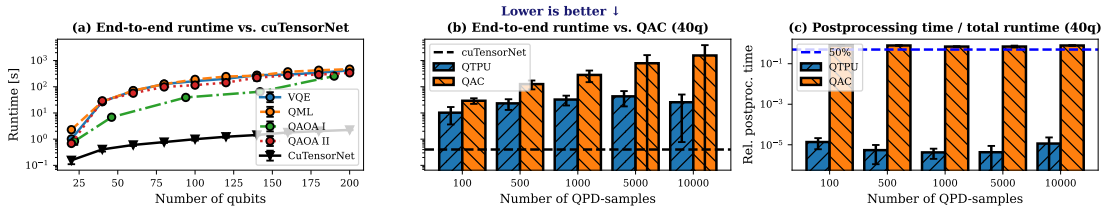


Figure 7.3: End-to-end runtime. *(a) End-to-end h-TN contraction time using a 15-qubit statevector simulator QPU vs. TN-simulation. (b) End-to-end h-TN contraction time depending on the number of QPD-samples vs. QAC knitting. (c) Portion of time spent on postprocessing relative to full runtime.*

RQ1 takeaway

qTPU reduces the PP overhead by orders of magnitude compared to state-of-the-art knitting as it avoids exponential overheads through its h-TN contraction.

— RQ2: Postprocessing time

By how much can qTPU reduce PP time?
Correlated with the reduction in overhead, we see a significant speedup in PP time compared to QAC. We need to run QAC with an approximation using 10^5 QPD-samples, as it would otherwise not terminate for more than ∼50 qubits. Despite only approximating the result with QAC, our results, shown in Fig. 7.2, indicate an average speedup of $10^{4.3}\times$.

As we can offload the PP onto a GPU, we can obtain a further speedup by up to 18.5× compared to qTPU's CPU PP. Importantly, even when the theoretical overhead is the same, e.g., for the 20-qubit benchmarks, qTPU still achieves a speedup of $\sim 57\times$ through its tensor processing.

RQ2 takeaway

qTPU's enables a speedup in PP of $10^{4.3}\times$ on average compared to QAC. Since qTPU's PP consists of tensor contractions, we can offload the process to the GPU, allowing a further speedup of up to 18.5×.

7.3 End-to-end Runtime Analysis

To show the practicality of qTPU as a valuable method for large-scale hybrid computations, we need to analyze its end-to-end runtime and its ability to scale beyond purely

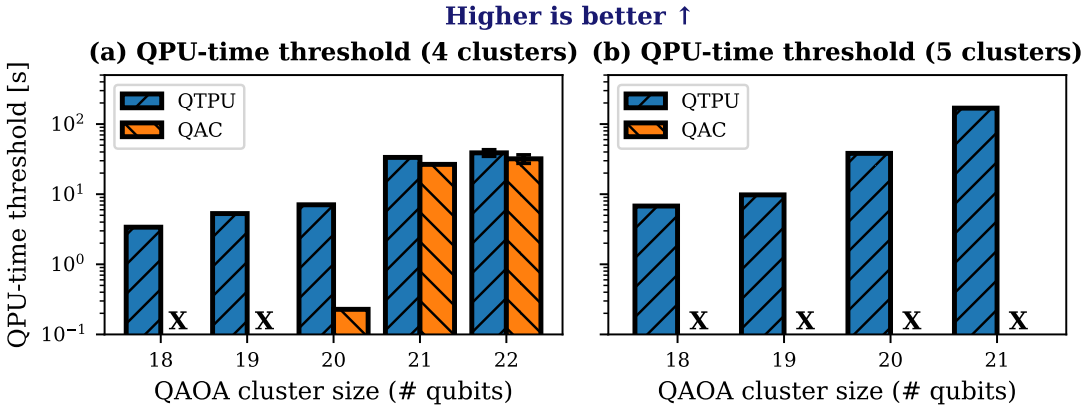


Figure 7.4: Runtime threshold analysis. *The maximal allowed time to spend on circuit evaluation during knitting to outperform classical TN-simulation. An X indicates that postprocessing exceeds full TN-simulation time.*

classical simulation techniques by using QPUs and GPUs. To this end, we examine the runtime of qTPU to contract an h-TN using a 15-qubit state vector simulator as a mock QPU and compare it to QAC and cuTensorNet.

— RQ3: End-to-end scalability

How does qTPU's end-to-end runtime scale for large circuits?
First, to demonstrate qTPU's scalability, we conduct end-to-end simulations of benchmarks from 20 to 200 qubits, as shown in Fig. 7.3 (a). qTPU scales linearly with the number of qubits and subcircuits, taking ∼30 seconds for the 40-qubit benchmark with three subcircuits and ∼400 seconds for the 200-qubit benchmark with more than 12 subcircuits.

The TN-simulator based on cuTensorNet achieves sub-second runtimes by running benchmarks directly in GPU memory. In contrast, qTPU simulates hundreds of subcircuits sequentially during h-TN contraction, though this could be improved by using real QPU(s) to run subcircuits faster.

Despite this, qTPU shows comparable scalability to efficient TN-contraction-based simulation techniques, enabling the execution of circuits that cannot be run with QAC or state vector simulation alone due to their exponential complexity.

RQ3 takeaway

qTPU achieves linear scaling for the end-to-end simulation of large quantum circuits that would not be possible to run with a single state-vector simulator or QAC.

— RQ4: Runtime-dependence on QPD-sampling

How does end-to-end runtime behave depending on the number of QPD-samples compared to QAC?
To reduce runtime during knitting, we can perform QPD-sampling on the coefficients to approximate the result, in return for losing accuracy as we use fewer samples (§ 4.3). To show how sampling affects overall runtime scaling compared to QAC, we run our benchmarks of 40 qubits and show the average end-to-end runtime over all benchmarks in Fig. 7.3 (b).

We observe an exponential increase in runtime for QAC as we increase the number of QPD samples, as it directly depends on the number of global coefficients sampled. However, in qTPU, since we sample directly in the efficient h-TN form, QPD-sampling only slightly affects runtime as the number of samples increases exponentially. By leveraging its efficient form, qTPU can still speed up overall runtime by orders of magnitude with an average of 20.7× and up to 36.3×.

RQ4 takeaway

As qTPU leverages its efficient h-TN form during QPD sampling, its runtime is comparatively unaffected by the total number of QPD samples. It enables a significant speedup over QAC by 20.7× on average.

— RQ5: Bottleneck analysis

Where does the main bottleneck in end-to-end runtime lie in qTPU compared to QAC?
To investigate entirely where our approach's bottleneck compared to QAC is, we analyze the relative time spent on PP compared to the end-to-end runtime. We show the results in Fig. 7.3 (c), using the same benchmarks as the previous result. In QAC, we observe that classical PP dominates the overall runtime of knitting, with 75.7% on average and up to 79.5%. For qTPU, however, PP only occupies a fraction of 10^{-3}% on average of overall runtime, virtually eliminating the imposed bottleneck of PP. Therefore, the bottleneck of circuit knitting is only the evaluation of subcircuits on QPUs, where the sampling overhead remains unchanged.

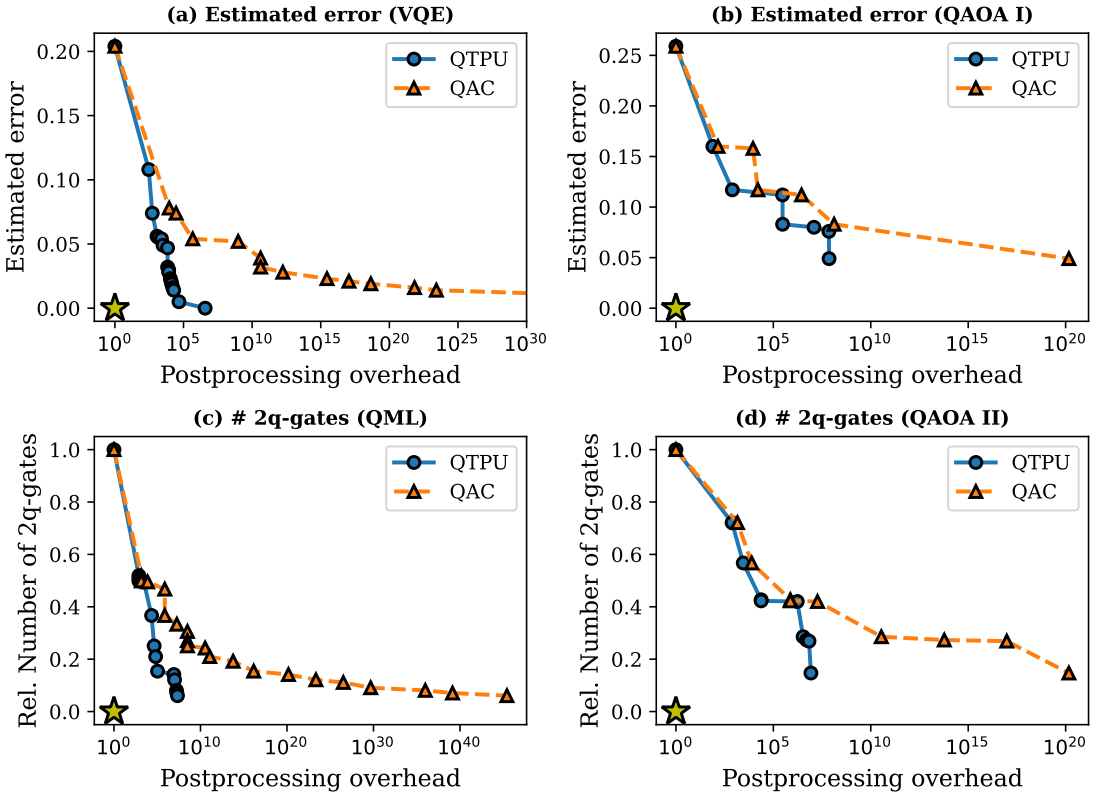


Figure 7.5: Compilation tradeoff. *Pareto-optimal circuit compilations of 100-qubit benchmarks explored by qTPU's compiler.*

RQ5 takeaway

qTPU allows us to eliminate the PP-bottleneck of circuit knitting, as it now only occupies a fraction of the total circuit knitting time.

— RQ6: Outperforming classical simulation

In which scenarios can qTPU's hybrid execution enable outperforming classical TN-simulation? We demonstrate that qTPU enables the execution of large circuits that cannot be efficiently run with TN-based techniques. Using QAOA II benchmarks with increasing cluster sizes makes them increasingly difficult to simulate due to memory constraints. We conduct benchmarks on four and five QAOA clusters, ranging from 18 to 22 qubits, measuring the simulation time with cuTensorNet and the postprocessing (PP) time for qTPU and QAC. The differences in TN-simulation time and PP are shown in

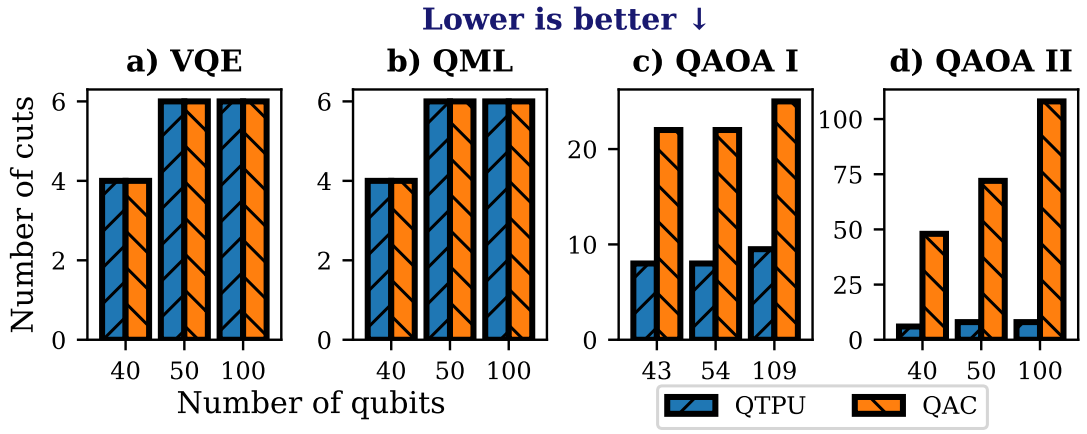


Figure 7.6: Compiler Optimality. *Number of cuts after compiling a circuit to have ∼25% of the original number of qubits.*

Fig. 7.4, indicating the maximum time available for subcircuit evaluation to outperform cuTensorNet.

We find that qTPU sets a threshold runtime for QPUs between 3.38 seconds and 168.7 seconds, which increases rapidly as cluster sizes complicate TN simulation. This provides a realistic opportunity to surpass TN simulation as the number of clusters increases since QPU runtime depends solely on circuit depth. However, reaching this threshold is significantly more difficult for QAC's PP, as it takes, on average, 9.2 times longer than the full TN simulation, hindering any chance of surpassing simulation through knitting.

RQ6 takeaway

For clustered benchmarks that become difficult to simulate classically, qTPU's hybrid processing allows a realistic chance to outperform classical simulation because of its negligible PP time.

7.4 Compiler Analysis

Finally, we discuss the performance of our compiler. In doing so, we focus on its novelty in enabling us to explore a tradeoff between expected error and overhead. Finally, we also compare the compiler against QAC's automatic cut-finder.

— RQ7: Error-overhead tradeoff

How does the qTPU compiler enable an optimal trade-off between error and overhead?

The qTPU compiler employs hyper-parameter optimization on various qTPU-specific parameters to compute circuit decompositions, each representing a tradeoff between estimated error in subcircuits and PP cost. To showcase this, we run our hyper-optimizer with 200 trials for 100-qubit benchmarks and record the Pareto-optimal solutions. For example, we use (1) the estimated error and (2) the relative number of two-qubit gates as error measures. The results are shown in Fig. 7.5, including the respective overheads of QAC.

For the simpler benchmarks of VQE and QML, we observe a steep Pareto front. Notably, the qTPU compiler provides solutions with nearly zero estimated error by cutting the circuit into subcircuits of only a few gates, enabled by the linearly increasing overhead, reaching a maximum of approximately $\mathcal{O}(10^8)$ FLOPs, allowing most of the circuit to run classically.

In contrast, for the more complex clustered QAOA benchmarks, we observe a flatter curve, as more cuts are needed to achieve low estimated error, making it difficult to reach minimal errors with reasonable overhead.

The corresponding curves of QAC's cost show an exponential increase in PP cost as we attempt to reduce expected error, closely following the previous results in § 7.2. This enables qTPU to come significantly closer to the optimum (yellow star).

RQ7 takeaway

qTPU's compiler facilitates the exploration of the trade-off space between PP cost and user-defined error estimation, significantly improving on previous work focusing solely on minimizing overhead. During its exploration, the optimizer even enables exponential error reduction with only a linear increase in overhead.

— RQ8: Compiler efficiency

How does qTPU's compiler compare to QAC's automatic cut-finder to determine efficient cuts?

To answer this question, we compile benchmarks with 40-100 qubits, cutting the circuit into subcircuits that are $\sim 25\%$ of the original size. We present the results, showing the number of cuts in the compiler's solution in Fig. 7.6, and compare them to QAC's automatic cutter [She+24]. We observe that the simpler benchmarks, such as VQE and QML, perform comparably to QAC. However, as the benchmarks increase in complexity, particularly with highly clustered interconnectivity of gates like in QAOA, our hyperparameter optimization enables us to identify efficient cuts where QAC struggles.

RQ8 Takeaway

qTPU's compiler can efficiently cut complex quantum circuits where single graph-partitioning methods fail. At worst, qTPU is on par with existing methods.

8 Related Work

8.1 Quantum circuit knitting

The Qiskit Addon Cutting (QAC) serves as the current de facto framework for circuit knitting, integrating prior work on quantum gate decomposition (QPD) and circuit cutting [TM22; MF21; Pen+20], and, like qTPU, it supports both gate and wire cuts. Ongoing research aims to reduce sampling overhead by identifying lower-overhead decompositions for multi-qubit gates or enabling classical communication between quantum processing units (QPUs) [BPS23; SPS23; Ufr+23b; Ufr+23a; Low+23]; in contrast, we focus on minimizing postprocessing (PP) overhead while enabling large-scale hybrid processing with classical accelerators. Other studies explore efficient circuit cuts, targeting either wire or gate cuts [Tan+21; CSR23] or both simultaneously [She+24; BPK23], relying on graph partitioning routines designed to fit circuits to specific QPU sizes. Our work facilitates scalable wire and gate cutting while optimizing the trade-off between error and overhead through hyperparameter tuning and efficient TN optimization. Recent studies have explored using TNs to speed up PP [TM22; RZB24], focusing only on wire or gate cutting. In contrast, we present a holistic approach through a compiler that optimizes for h-TN processing.

8.2 Tensor network methods

qTPU draws inspiration from well-studied TN methods, establishing connections between techniques used in classical TN optimization and those implemented in our framework. In particular, the optimizer within the qTPU compiler resembles the standard approach for determining an optimized contraction sequence of a classical TN, making it highly suitable for optimizing our hybrid circuit contraction [GK21; NVI24; SJ20]. The compression of our IR parallels rank-simplification optimization while sampling on a hybrid TN can be likened to low-rank approximations [BC17]. Furthermore, prior research has introduced the concept of a "hybrid TN" for performing quantum ground state optimizations [Yua+21; Sch+24], which focuses on offloading parts of an iterative TN algorithm onto a quantum processing unit (QPU), presenting an approach that contrasts with ours.

8.3 Hybrid computing

A related branch of research focuses on architectures for scheduling and running complex quantum-classical algorithms in hybrid clusters [Che+24; Gio+24b; Gio+24a; Sal+22; WDN24; Sch+]. Our hybrid circuit contraction enforces a straightforward processing model, requiring only a simple execution model to execute subcircuits on QPUs. However, we can easily expand our runtime by integrating it with sophisticated schedulers.

9 Conclusion

Circuit knitting holds promise for scaling quantum computing, but its exponential brute-force postprocessing limits its potential. To address this, we introduced qTPU, a scalable framework for hybrid circuit contraction, representing quantum circuits as hybrid tensor networks (h-TNs) to accelerate circuit knitting. Our qTPU compiler generates hyper-optimized h-TNs, balancing error and overhead, while the runtime efficiently contracts these networks on GPUs and QPUs. Evaluation shows qTPU significantly improves postprocessing and runtime by $10^4\times$ and $20.7\times$ on average, respectively. We believe our approach is a crucial step toward making hybrid quantum-classical processing useful.

List of Figures

Bibliography

[AI24] G. Q. AI. *Google Quantum AI*. https://quantumai.google/. Accessed: 2024-10-03. 2024.

[Aki+19] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. "Optuna: A next-generation hyperparameter optimization framework." In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[Ale+24] Y. Alexeev, M. Amsler, M. A. Barroca, S. Bassini, T. Battelle, D. Camps, D. Casanova, Y. J. Choi, F. T. Chong, C. Chung, et al. "Quantum-centric supercomputing for materials science: A perspective on challenges and future directions." In: *Future Generation Computer Systems* 160 (2024), pp. 666–710.

[BC17] J. C. Bridgeman and C. T. Chubb. "Hand-waving and interpretive dance: an introductory course on tensor networks." In: *Journal of physics A: Mathematical and theoretical* 50.22 (2017), p. 223001.

[BPK23] S. Brandhofer, I. Polian, and K. Krsulich. "Optimal Partitioning of Quantum Circuits using Gate Cuts and Wire Cuts." In: *arXiv preprint arXiv:2308.09567* (2023).

[BPS23] L. Brenner, C. Piveteau, and D. Sutter. "Optimal wire cutting with classical communication." In: *arXiv preprint arXiv:2302.03366* (2023).

[Bra+24] A. M. Brańczyk, A. Carrera Vazquez, D. J. Egger, B. Fuller, J. Gacon, J. R. Garrison, J. R. Glick, C. Johnson, S. Joshi, E. Pednault, C. D. Pemmaraju, P. Rivero, I. Shehzad, and S. Woerner. *Qiskit addon: circuit cutting*. https://github.com/Qiskit/qiskit-addon-cutting. 2024. DOI: 10.5281/zenodo.7987997.

[Bra24] A. Braket. *Quantum Cloud-Computing Service – Amazon Braket – AWS*. https://aws.amazon.com/de/braket/. Accessed: 2024-10-03. 2024.

[BSS16] S. Bravyi, G. Smith, and J. A. Smolin. "Trading Classical and Quantum Computational Resources." In: *Phys. Rev. X* 6 (2 June 2016), p. 021043. DOI: 10.1103/PhysRevX.6.021043.

[Che+24] K.-C. Chen, X. Li, X. Xu, Y.-Y. Wang, and C.-Y. Liu. "Quantum-hpc framework with multi-gpu-enabled hybrid quantum-classical workflow: Applications in quantum simulations." In: *arXiv preprint arXiv:2403.05828* (2024).

[Cic14] A. Cichocki. "Tensor networks for big data analytics and large-scale optimization problems." In: *arXiv preprint arXiv:1407.3124* (2014).

[CSR23] W. Cambiucci, R. M. Silveira, and W. V. Ruggiero. "Hypergraphic partitioning of quantum circuits for distributed quantum computing." In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 2. IEEE. 2023, pp. 268–269.

[Deb11] K. Deb. "Multi-objective optimisation using evolutionary algorithms: an introduction." In: *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 2011, pp. 3–34.

[ED18] M. T. Emmerich and A. H. Deutz. "A tutorial on multiobjective optimization: fundamentals and evolutionary methods." In: *Natural computing* 17 (2018), pp. 585–609.

[Els+23] A. Elsharkawy, X.-T. M. To, P. Seitz, Y. Chen, Y. Stade, M. Geiger, Q. Huang, X. Guo, M. A. Ansari, C. B. Mendl, et al. "Integration of quantum accelerators with high performance computing–a review of quantum programming tools." In: *arXiv preprint arXiv:2309.06167* (2023).

[FGG14] E. Farhi, J. Goldstone, and S. Gutmann. "A quantum approximate optimization algorithm." In: *arXiv preprint arXiv:1411.4028* (2014).

[FNW92] M. Fannes, B. Nachtergaele, and R. F. Werner. "Finitely correlated states on quantum spin chains." In: *Communications in mathematical physics* 144 (1992), pp. 443–490.

[Gio+24a] E. Giortamis, F. Romão, N. Tornow, and P. Bhatotia. "QOS: A Quantum Operating System." In: *arXiv preprint arXiv:2406.19120* (2024).

[Gio+24b] E. Giortamis, F. Romão, N. Tornow, D. Lugovoy, and P. Bhatotia. "Orchestrating Quantum Cloud Environments with Qonductor." In: *arXiv preprint arXiv:2408.04312* (2024).

[GK21] J. Gray and S. Kourtis. "Hyper-optimized tensor network contraction." In: *Quantum* 5 (2021), p. 410.

[Gra18] J. Gray. "quimb: a python library for quantum information and many-body calculations." In: *Journal of Open Source Software* 3.29 (2018), p. 819. DOI: 10.21105/joss.00819.

[Has07] M. B. Hastings. "An area law for one-dimensional quantum systems." In: *Journal of Statistical Mechanics: Theory and Experiment* 2007.08 (2007), P08024. DOI: 10.1088/1742-5468/2007/08/P08024.

[Jav+24] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta. *Quantum computing with Qiskit*. 2024. DOI: 10.48550/arXiv.2405.08810. arXiv: 2405.08810 [quant-ph].

[Kim+23] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. Van Den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, et al. "Evidence for the utility of quantum computing before fault tolerance." In: *Nature* 618.7965 (2023), pp. 500–505.

[Low+23] A. Lowe, M. Medvidović, A. Hayes, L. J. O'Riordan, T. R. Bromley, J. M. Arrazola, and N. Killoran. "Fast quantum circuit cutting with randomized measurements." In: *Quantum* 7 (2023), p. 934.

[MF21] K. Mitarai and K. Fujii. "Constructing a virtual two-qubit gate by sampling single-qubit operations." In: *New Journal of Physics* 23.2 (2021), p. 023021.

[NVI24] NVIDIA. *cuTensorNet: A High-Performance Library for Tensor Network Computations*. Accessed: 2024-10-03. 2024.

[Orú14] R. Orús. "A practical introduction to tensor networks: Matrix product states and projected entangled pair states." In: *Annals of physics* 349 (2014), pp. 117–158.

[Orú19] R. Orús. "Tensor networks for complex quantum systems." In: *Nature Reviews Physics* 1.9 (2019), pp. 538–550.

[Pen+20] T. Peng, A. W. Harrow, M. Ozols, and X. Wu. "Simulating Large Quantum Circuits on a Small Quantum Computer." In: *Phys. Rev. Lett.* 125 (15 Oct. 2020), p. 150504. DOI: 10.1103/PhysRevLett.125.150504.

[Per+21] M. A. Perlin, Z. H. Saleem, M. Suchara, and J. C. Osborn. "Quantum circuit cutting with maximum-likelihood tomography." In: *npj Quantum Information* 7.1 (2021), p. 64.

[Pre18] J. Preskill. "Quantum Computing in the NISQ era and beyond." In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79.

[PS22] C. Piveteau and D. Sutter. "Circuit knitting with classical communication." In: *arXiv preprint arXiv:2205.00016* (2022).

[PS23] C. Piveteau and D. Sutter. "Circuit knitting with classical communication." In: *IEEE Transactions on Information Theory* (2023), pp. 1–1. DOI: 10.1109/TIT.2023.3310797.

[Qua24a] I. Quantum. *IBM Quantum Computing*. `https://www.ibm.com/quantum`. Accessed: 2024-10-03. 2024.

[Qua24b] I. Quantum. *IBM Quantum Roadmap*. `https://www.ibm.com/roadmaps/quantum/`. Accessed: 2024-10-18. 2024.

[Qua24c] M. A. Quantum. *Azure Quantum*. `https://quantum.microsoft.com/`. Accessed: 2024-10-03. 2024.

[RZB24] X. Ren, M. Zhang, and A. Barbalace. "A Hardware-Aware Gate Cutting Framework for Practical Quantum Circuit Knitting." In: *arXiv preprint arXiv:2409.03870* (2024).

[Sal+22] M. Salm, J. Barzen, F. Leymann, and B. Weder. "Prioritization of compiled quantum circuits for different quantum computers." In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2022, pp. 1258–1265.

[Sch+] M. Schulz, L. Schulz, M. Ruefenacht, and R. Wille. "Towards the Munich Quantum Software Stack." In: ().

[Sch+23] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders. "High-quality hypergraph partitioning." In: *ACM Journal of Experimental Algorithmics* 27 (2023), pp. 1–39.

[Sch+24] J. Schuhmacher, M. Ballarin, A. Baiardi, G. Magnifico, F. Tacchino, S. Montangero, and I. Tavernelli. "Hybrid tree tensor networks for quantum simulation." In: *arXiv preprint arXiv:2404.05784* (2024).

[Sch20] S. Schlag. "High-Quality Hypergraph Partitioning." PhD thesis. Karlsruhe Institute of Technology, Germany, 2020.

[Sei+23] P. Seitz, I. Medina, E. Cruz, Q. Huang, and C. B. Mendl. "Simulating quantum circuits using tree tensor networks." In: *Quantum* 7 (2023), p. 964. ISSN: 2521-327X. DOI: 10.22331/q-2023-03-30-964.

[Sei+24] P. Seitz, M. Geiger, C. Ufrecht, A. Plinge, C. Mutschler, D. Scherer, C. B. Mendl, et al. "SCIM MILQ: An HPC Quantum Scheduler." In: *arXiv preprint arXiv:2404.03512* (2024).

[She+24] I. Shehzad, E. Pednault, J. Garrison, C. Johnson, B. Fuller, and J. Glick. "Automated cut finding and circuit knitting on large quantum circuits." en-US. In: Sept. 2024.

[SJ20] F. Schindler and A. S. Jermyn. "Algorithms for tensor network contraction ordering." In: *Machine Learning: Science and Technology* 1.3 (2020), p. 035001.

[SPS23] L. Schmitt, C. Piveteau, and D. Sutter. "Cutting circuits with multiple two-qubit unitaries." In: *arXiv preprint arXiv:2312.11638* (2023).

[Tan+21] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi. "Cutqc: using small quantum computers for large quantum circuit evaluations." In: *Proceedings of the 26th ACM International conference on architectural support for programming languages and operating systems*. 2021, pp. 473–486.

[Tin+24] J. Tindall, M. Fishman, E. M. Stoudenmire, and D. Sels. "Efficient tensor network simulation of ibm's eagle kicked ising experiment." In: *PRX Quantum* 5.1 (2024), p. 010308.

[TM22] W. Tang and M. Martonosi. "ScaleQC: A Scalable Framework for Hybrid Computation on Quantum and Classical Processors." In: *arXiv preprint arXiv:2207.00933* (2022).

[Ufr+23a] C. Ufrecht, L. S. Herzog, D. D. Scherer, M. Periyasamy, S. Rietsch, A. Plinge, and C. Mutschler. *Optimal joint cutting of two-qubit rotation gates*. 2023. arXiv: 2312.09679 [quant-ph].

[Ufr+23b] C. Ufrecht, M. Periyasamy, S. Rietsch, D. D. Scherer, A. Plinge, and C. Mutschler. "Cutting multi-control quantum gates with ZX calculus." In: *arXiv preprint arXiv:2302.00387* (2023).

[Vid03] G. Vidal. "Efficient classical simulation of slightly entangled quantum computations." en. In: *Physical Review Letters* 91.14 (Oct. 2003). arXiv:quant-ph/0301063, p. 147902. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.91.147902.

[WDN24] M. Wang, P. Das, and P. J. Nair. "Qoncord: A Multi-Device Job Scheduling Framework for Variational Quantum Algorithms." In: *arXiv preprint arXiv:2409.12432* (2024).

[YO22] T. Yamamoto and R. Ohira. "Error suppression by a virtual two-qubit gate." In: *arXiv preprint arXiv:2212.05493* (2022).

[Yua+21] X. Yuan, J. Sun, J. Liu, Q. Zhao, and Y. Zhou. "Quantum simulation with hybrid tensor networks." In: *Physical Review Letters* 127.4 (2021), p. 040501.

[ZSW20] Y. Zhou, E. M. Stoudenmire, and X. Waintal. "What limits the simulation of quantum computers?" In: *Physical Review X* 10.4 (2020), p. 041038.