SWIFTN: Accelerating Quantum Circuit Simulation through Tensor Optimization

Seunghwan Kim[*1], Changjong Kim[*1], Alex Sim[2], Kesheng Wu[2], Houjun Tang[2], and Sunggon Kim[†1]

[1]Seoul National University of Science and Technology, South Korea
[2]Lawrence Berkeley National Laboratory, USA

***Abstract*—Quantum computers are evolving at a rapid pace and are considered next-generation computers with high computational capabilities. However, due to the unique characteristics of qubits, state-of-the-art quantum computers are vulnerable to noise caused by qubit instability. To overcome this, high-performance computing (HPC) systems are utilized for quantum circuit simulations to evaluate complex quantum algorithms with great accuracy. However, quantum circuit simulations have high computational demands, and the data volume increases exponentially as the number of qubits increases.**

In this paper, we propose `SWIFTN`, a quantum circuit simulation optimization framework for HPC systems with scalability. To achieve this, it enhances parallelism by dividing the tensor networks and distributing them across multiple GPUs and nodes. Additionally, it reduces computational costs by bypassing tasks through intermittent tensor contraction. Finally, to mitigate the degradation in accuracy due to intermittent tensor contraction, `SWIFTN` performs amplitude adjustments. We implement and evaluate `SWIFTN` using a Perlmutter supercomputer. Our evaluation results using popular quantum algorithm benchmark (i.e., QAOA) shows that `SWIFTN` can improve the performance by 7.85× with 99.997% accuracy.

***Index Terms*—Quantum Circuit Simulation, High-performance Computing, Tensor Network, Performance Modeling**

I. Introduction

Quantum computers [1], [2] have rapidly advanced, with many studies claiming to achieve quantum supremacy by outperforming classical computers. Unlike classical computers that use bits defined as either 0 or 1, quantum computers rely on qubits. Qubits leverage the principles of superposition [3], where a qubit can exist in a probabilistic combination of both 0 and 1 states simultaneously, and entanglement [4], [5], where different qubits can influence each other. These properties enable quantum computers to perform parallel processing of large amounts of information through a single qubit operation, offering significant advantages over traditional computers in fields such as cryptography [6]–[10], scientific simulations [11]–[16], and machine learning [17]–[20].

Despite their computational advantages, quantum computers have limitations. They require qubits to operate in highly controlled environments, isolated from external noise, and at extremely low temperatures, close to absolute zero [21]. Consequently, managing current-generation quantum computers, known as Noisy-Intermediate-Scale Quantum (NISQ) computers, is costly [22], making them difficult to access. In addition, their vulnerability to noise makes them challenging to apply practical applications in everyday scenarios [23]–[25].

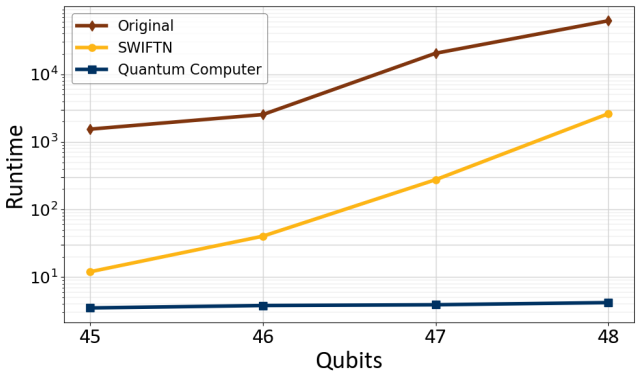


Fig. 1: Runtime of Quantum Computer, Original Quantum Circuit Simulation, and `SWIFTN` Running QAOA Benchmark

To address this, high-performance computing (HPC) systems are utilized to perform quantum circuit simulations [35]. Quantum circuit simulation enables the prediction of quantum computation outcomes in noise-free environments and the direct application of quantum gates and qubits to various scenarios (e.g., quantum machine learning and scientific simulations). To do this, quantum circuit simulation frameworks like cuQuantum [36] and Qiskit [37] represent quantum circuits as the tensor networks, which are commonly used in machine learning. These frameworks utilize GPUs to simulate qubit states after specific operations on qubits. In addition, quantum computing frameworks (i.e., Qiskit, Cirq, and PennyLane [37]–[39]) provide accessible tools for quantum simulation and computing on classical HPC systems. However, as qubits can represent far more information than bits, both computation and memory requirements increase exponentially as the number of simulated qubits grows [40].

Figure 1 shows the runtime of quantum computer (i.e., IBM Quantum), original quantum simulation, and the proposing `SWIFTN` as the number of qubits increases using standard quantum algorithm benchmark (i.e., QAOA). In quantum computers, the runtime maintains nearly constant as qubits increase due to the rapid compuation using qubits. However, they may experience noise due to qubit instability. In contrast, when performing identical benchmark with original quantum circuit

[*]These authors contributed equally.
[†]Corresponding author: Sunggon Kim

TABLE I
CATEGORIES AND COMPARISON WITH PREVIOUS STUDIES

Study	Simulation Type	Circuit Partitioning	Data Compression	Multi-Node
Mingkuan Xu *et al.* [26]	Schrödinger	✓		✓
Burgholzer *et al.* [27]	Schrödinger+Feynman	✓		
Wu *et al.* [28]	Schrödinger		✓	✓
Lykov *et al.* [29]	Tensor Network	✓		✓
Park *et al.* [30]	Schrödinger	✓		
Zhang *et al.* [31]	Schrödinger+Feynman	✓		✓
Fu *et al.* [32]	Tensor Network	✓		✓
Zhao *et al.* [33]	Schrödinger	✓		
Westrick *et al.* [34]	Schrödinger+Feynman	✓		
SWIFTN	Tensor Network	✓	✓	✓

simulation in HPC system, the runtime increases significantly as the number of qubits increases. This is due to the exponential increase in computation requirements in the original quantum circuit simulation. This is further supported by our time analysis (presented later in Figure 12), where computation time accounts for 99.67% of the total runtime, highlighting the significant computational overhead in quantum circuit simulation. Positioned between these two approaches, SWIFTN aims to improve the runtime of quantum circuit simulations while providing more accurate qubit measurements than current quantum computers.

Many previous studies, as shown in Table I, have focused on optimizing quantum circuit simulation using classical computers. For example, prior studies optimize quantum circuit simulation using the Schrödinger method, which fully simulates the quantum state [26], [28], [30], [33]. In contrast, the hybrid method combines complete Schrödinger simulation with partial Feynman path simulation, enhancing efficiency [27], [31], [34]. In contrast, tensor networks, which focus only on a subset of quantum circuits [29], [32], can reduce memory and computation requirements during simulation. SWIFTN distinguishes itself from previous work by focusing on a tensor network-based approach, which enhances scalability by allowing the sub-tensor networks representing sub-circuits to execute in parallel on distributed resources. Additionally, SWIFTN focuses on improving performance through the tensor approximation of large tensor networks derived from quantum circuits.

In this paper, we propose SWIFTN, a quantum circuit simulation framework that optimizes the simulations on large-scale classical computers through the tensor distribution and approximation. SWIFTN expedites (SWIFT) tensor network (TN) contraction for quantum circuit simulation on HPC systems. SWIFTN aims to leverage the trade-off between reducing computational overhead from the tensor contraction bypass and the accuracy drop due to approximation. This trade-off allows SWIFTN to be applicable to many quantum algorithms, such as QAOA [41], that do not require perfect accuracy. To this end, 1) SWIFTN slice the tensor network, which represents quantum circuits, into the sub-tensor networks and distribute them across multiple GPUs and nodes within the cluster to improve scalability. 2) SWIFTN performs intermittent tensor contraction to compress the tensor network and ultimately reduce computations. 3) the final state of the qubits, represented by their amplitudes, is adjusted to account for the effects of intermittent tensor contraction. We evaluate SWIFTN on a Perlmutter supercomputer at NERSC [42] using the QAOA quantum algorithm, the most widely applied quantum algorithm. The evaluation demonstrates that SWIFTN can improve runtime by up to 7.85× compared to the original tensor network-based simulation with 99.997% accuracy.

Our key contributions are as follows:

- We design SWIFTN, which optimizes the tensor network contraction for quantum circuit simulation through slicing and intermittent tensor contraction.
- We observe patterns in the results of the intermittent tensor contraction and adjust the amplitude through trained linear regression models.
- We evaluate SWIFTN using standard quantum algorithm benchmarks such as QAOA using a Perlmutter supercomputer.

To the best knowledge of our knowledge, SWIFTN is the first study to optimize the tensor network contraction for quantum circuit simulation considering trade-off between computation and accuracy through tensor distribution and approximation.

II. BACKGROUND

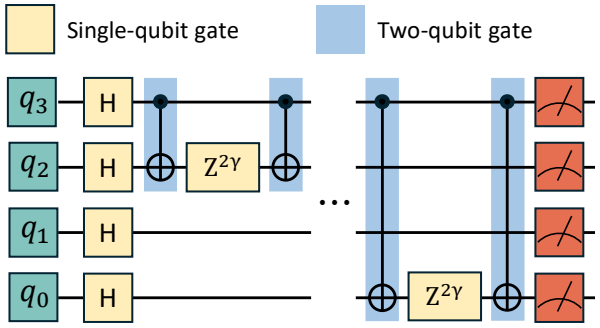
A. Quantum Computing and Quantum Algorithms

Principles of quantum computing: In classical computers, a bit is in one of the two deterministic states: ”0” (i.e., low voltage level) or ”1” (i.e., high voltage level). In contrast, a qubit in a quantum computer can exist a probabilistic state of 0 and 1 which called superposition. This can be represented by two complex numbers, each with real and imaginary components, reflecting the probabilities of states 0 and 1 [43]. These two complex numbers, α_0 and α_1, define the qubit's state ψ, which can be expressed as:

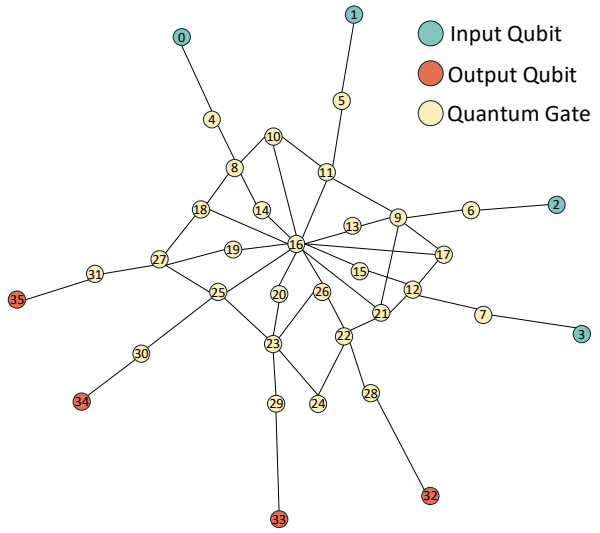
$$\psi = \alpha_0|0\rangle + \alpha_1|1\rangle. \tag{1}$$

where α_0 and α_1 represent the amplitudes for each state. The probabilities of being in the 0-state and 1-state are $|\alpha_0|^2$ and $|\alpha_1|^2$, respectively, and they always satisfy $|\alpha_0|^2 + |\alpha_1|^2 = 1$ which denotes that total probability of being in either the 0-state or 1-state sums to 100 percent.

Quantum gates and algorithms: With this unique property, multiple qubits can become entangled, meaning the state of one qubit directly influences the state of another, regardless of distance. In an entangled state, measuring one qubit immediately provides information about the other, creating an



(a) A Four-qubit Quantum Circuit



(b) Quantum Circuit Represented in Tensor Network

Fig. 2: Example of quantum Circuit and its Representation in Tensor Network

interdependent connection that classical bits cannot achieve. Through superposition within a single qubit and entanglement among multiple qubits, quantum algorithms manipulate qubit states using single-qubit gates (e.g., Pauli-X, Y, Z, Hadamard) or multi-qubit gates (e.g., CNOT, CZ) to compute multiple qubit states simultaneously [43], [44]. Quantum algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) [41], Grover's Search Algorithm [45], and Shor's Algorithm [46] utilize these properties to arrange qubits and gates in specific sequences called quantum circuits, enabling them to solve complex problems by manipulating entangled states. These algorithms demonstrate the potential of quantum computing to solve problems that are intractable for classical computers.

B. Tensor-Based Quantum Circuit Simulator

While quantum computers offer high performance, the inherent instability of qubits makes current quantum computers prone to noise [23]. Despite efforts to control qubits at extremely low temperatures, noise still occurs during the execution of quantum circuits. This makes it challenging to analyze and optimize quantum algorithms, as the resulting

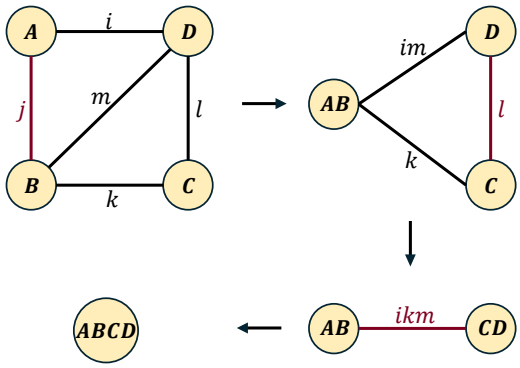


Fig. 3: Tensor Network Contraction (Only the shared indices between nodes are depicted in the figure)

qubit states can be inconsistent across multiple executions of identical quantum circuits, leading to variability in the outcomes.

To address this challenge, tensor network-based quantum circuit simulation is gaining popularity [29], [32], [47]. Tensor networks, widely used in machine learning to tackle complex problems with existing computational hardware such as GPUs, are adept at handling high-dimensional data. In quantum circuit simulations, tensor networks effectively structure circuit architectures, represent qubit states, and model their interactions with gate operations, enabling efficient simulations of large quantum systems using classical computing resources

Figure 2 shows an example of a 4-qubit quantum circuit and its corresponding the tensor network representation. The input and output qubits (qubits 0, 1, 2, and 3) and the quantum gates in the quantum circuit, as shown in figure 2a, are represented in the tensor network as input tensors (index 0, 1, 2, and 3), output tensors (index 32, 33, 34, and 35), and gate tensors (index 4 through 31), as illustrated in figure 2b. A tensor network representing a quantum circuit generally consists of numerous vertices connected by complex edges, encoding high-dimensional data. As the number of qubits grows, the complexity of the quantum circuit increases exponentially, resulting in larger tensor networks that demand extensive computational resources, causing overhead and potential out-of-memory errors that can terminate the process. Thus, efficient processing of these tensors is crucial to ensure scalability and performance in quantum circuit simulations.

C. Tensor Network Contraction

When processing a tensor network derived from a quantum circuit, multiple tensors are contracted into a single tensor (i.e., a one-dimensional state vector). A pathfinder algorithm optimizes this by identifying an efficient sequence for contractions using techniques like diagonal-reduction, column-reduction, rank simplification, and split-simplification. This approach minimizes computational resources while maintaining accuracy. Following the optimized path, the contraction is performed to reduce the high-dimensional tensor network into a single, simplified representation.

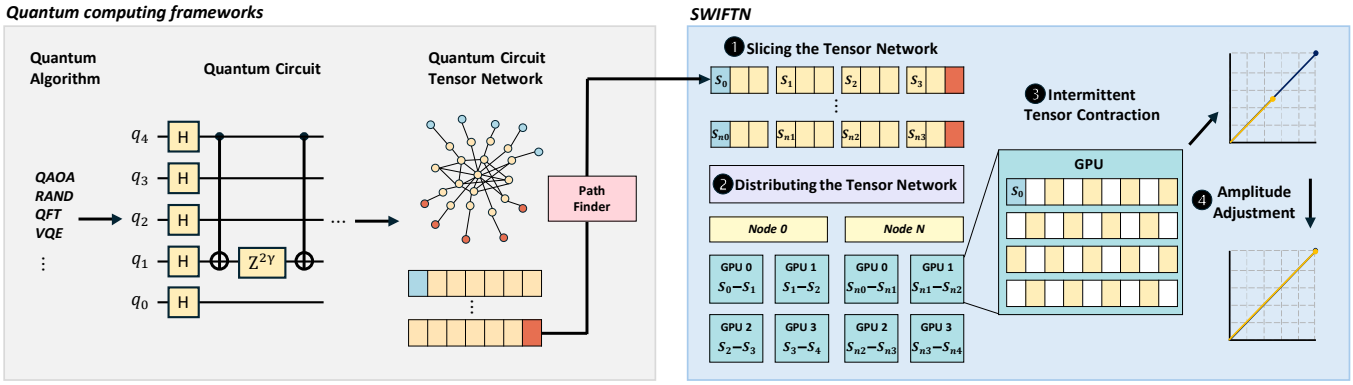


Fig. 4: Overall Procedure of SWIFTN

Figure 3 illustrates an example of tensor network contraction. As shown in the figure, four tensor nodes (A, B, C, and D) exist in the network which are connected by edges (i, j, k, l, and m). Thus, the contraction can be written as follows:

$$ABCD_{ijklm} = \sum_{index_{shared}} A_{ij}B_{jkm}C_{kl}D_{ilm} \quad (2)$$

To perform the contraction, first, node A and B are contracted. As they share the index j, they are contracted into a single node AB over index j. Similarly, nodes C and D are contracted over the shared index l. Finally, the representation of the four nodes expressed in Eq. 2 has been reduced to a single tensor through a single computation involving the shared indices i, k, m of the two nodes AB and CD, that is,

$$ABCD_{ijklm} = \sum_{ikm} AB_{ijkm}CD_{iklm} \quad (3)$$

During the contraction process, contraction of node AB and node BC does not affect each other and the execution order between the two does not affect the final result. Thus, the contraction can be executed independently and in parallel for different computations. By leveraging this characteristic of tensor network contraction, it is possible to efficiently distribute the indices of the tensor network across multiple computation nodes for parallel execution, thereby improving the performance of computation.

III. Design

In this section, we propose SWIFTN, a quantum circuit simulation optimization framework that focuses on tensor network contraction optimization through tensor distribution and tensor approximation.

SWIFTN aims to achieve the following research goals:

- SWIFTN improves parallelism by slicing the tensor network and executing the slices concurrently. (Section III-B)
- SWIFTN improves scalability through inter-node (Nodes) and intra-node (GPUs) tensor distribution.(Section III-B)
- SWIFTN reduces computation through tensor approximation using intermittent tensor contraction.(Section III-C)
- SWIFTN adjusts the amplitude values of the resulting qubits from intermittent tensor contraction to ensure the consistency of the simulation output.(Section III-D)

A. Overall Architecture

Figure 4 illustrates the overall procedure of quantum circuit simulation through tensor network contraction in SWIFTN. Identical to the original quantum circuit simulation, first, a quantum circuit is constructed from the selected quantum algorithm. This circuit is then transformed into a tensor network representation(as described in Section II-B). Subsequently, pathfinder identifies an optimized path for the contraction process.

In SWIFTN, after receiving the tensor and the contraction path, SWIFTN needs to contract the tensor network into a one-dimensional single tensor that contains the amplitude information of each target qubit. To do this, SWIFTN optimizes this tensor network contraction using four major components: Slicing the paths of the tensor network (❶), Multi-node distribution (❷), Intermittent tensor contraction (❸), and Amplitude adjustment (❹).

First, to handle the large data size of the tensor network and the computationally intensive contraction process, SWIFTN slices the tensor network into multiple slice groups (❶). Then, the sliced tensor network is distributed across multiple GPUs and nodes (❷). SWIFTN employs two distribution strategies: inter-node and intra-node. In the inter-node approach, the sliced sub-tensor networks are distributed across multiple nodes via MPI (Message Passing Interface). In the intra-node approach, sub-tensor networks are further sliced and distributed across GPUs within a single node using CUDA P2P (Point-to-Point) communication, enabling independent contractions within each slice group between adjacent nodes. This parallelism across GPUs and nodes enhances scalability for large-scale systems.

After distributing the tensor network and performing contractions on each GPU and node, SWIFTN applies an intermittent tensor contraction, consistently skipping certain tensors during contraction (❸). While this approach reduces simulation accuracy, it significantly accelerates computation and optimizes memory efficiency, which grows exponentially as

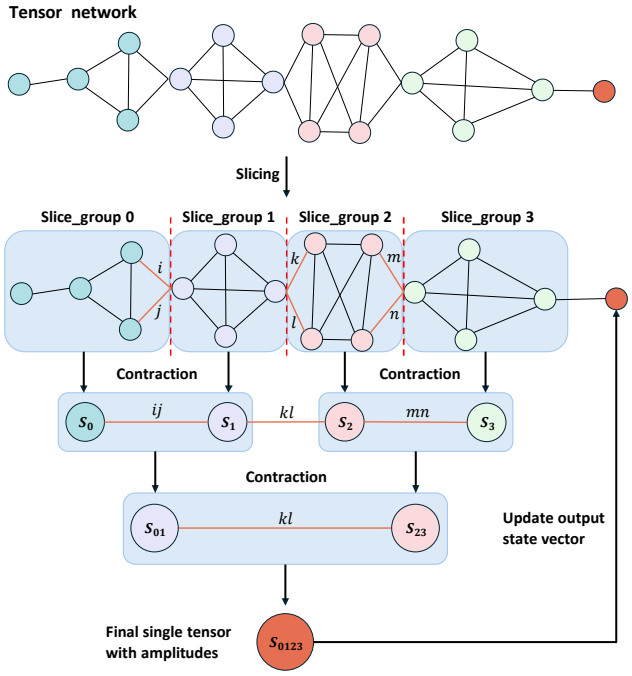


Fig. 5: Tensor Network Slicing

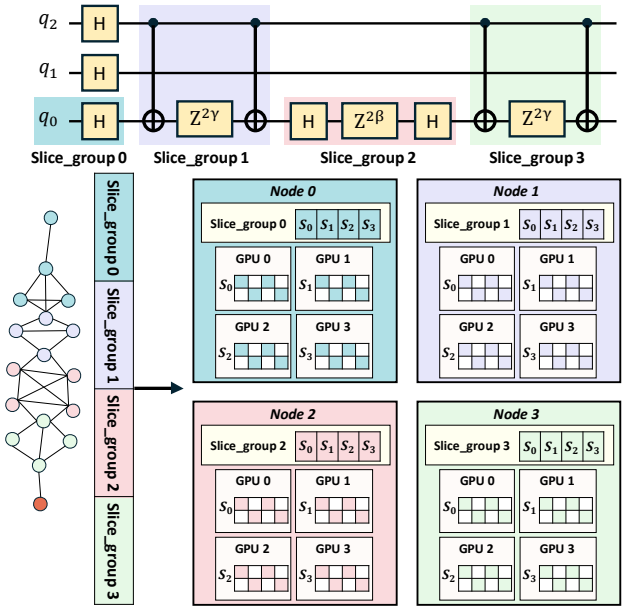


Fig. 6: Two-phase Distribution in SWIFTN

qubits increase, enabling large-scale qubit simulations. Finally, to mitigate the effects of the intermittent tensor contraction, SWIFTN performs amplitude adjustments using linear regression (❹). Thus, by adjusting the resulting amplitudes, SWIFTN reduces computational overhead while maintaining similar accuracy.

B. Slicing and Distributing the Tensor Network

Slicing the tensor network: To improve the parallelism of tensor contraction of quantum circuits, SWIFTN leverages the inherent characteristics of the tensor network contraction, where subsets of tensors can be contracted independently and simultaneously. By explicitly slicing the tensor network based on the contraction path determined by the pathfinder, SWIFTN optimizes computational resources through parallelization.

Figure 5 illustrates an example of tensor network slicing in SWIFTN. The original tensor network is divided into four slice groups (Slice_group 0–3) to balance the computational load, ensuring each group contains an equal number of tensors. After slicing, each group is independently contracted, producing a single tensor network per group. As each slice group can be contracted concurrently, SWIFTN distributes the contraction of independent slices across multiple GPUs and nodes. For example, as shown in the figure, slice group 0 and 1 is contracted into a single tensor network of S_0 and S_1, respectively. Further contraction then combines adjacent groups (e.g., S_0 and S_1, sharing indices i and j), forming S_{01}. This process iterates until a final single tensor is produced, which updates the output state vector node, completing the slicing procedure. Therefore, by slicing the tensor network and concurrently executing independent slice groups, SWIFTN enhances the parallelism of quantum circuit simulations.

After dividing the tensor network into multiple slice groups, SWIFTN distributes these groups across multiple nodes and GPUs in the HPC system to improve scalability. As depicted on figure 6, SWIFTN introduces a two-phase distribution, consisting of inter-node (Node) and intra-node (GPU) distribution.

Inter-node distribution: Sliced sub-tensor networks are distributed across nodes with balanced load allocation. To do this, as shown in the figure, SWIFTN creates slice groups so that the number of slice groups matches the available number of nodes. When the tensor network is sliced, all sub-tensor networks are initially stored in the memory of the first node (node 0). To distribute them, node 0 performs MPI communications to send sub-tensor networks to other nodes. For example, as shown in the figure, four nodes (nodes 0–3) exist in the system. Thus, SWIFTN creates four slice groups (i.e., slice_group 0-3) Then, node 0 distributes sub-tensors by sending slice_group 1 to node 1, slice_group 2 to node 2 and slice_group 3 to node 3. After the sub-tensor networks are distributed across nodes, there is no further communication between the nodes and additional slicing is performed within each node.

Intra-node distribution: The sub-tensor network in main memory of each node is further divided according to the number of available GPUs within the node, and then distributed to each GPU. For example, as shown in the figure, each slice_group within the node is divided into slices (i.e., s0-3). To distribute the slices, each node performs CUDA P2P communication to the GPU, allowing for efficient parallel execution of independent contraction tasks. Once the tasks of the inter-node distribution are completed, the output tensor networks from each GPU are contracted to generate a single tensor network per node. Finally, the intra-node tasks are completed and the single tensor network from each node are contracted to produce a final tensor network with amplitude information. While cuQuantum SDK provides features for

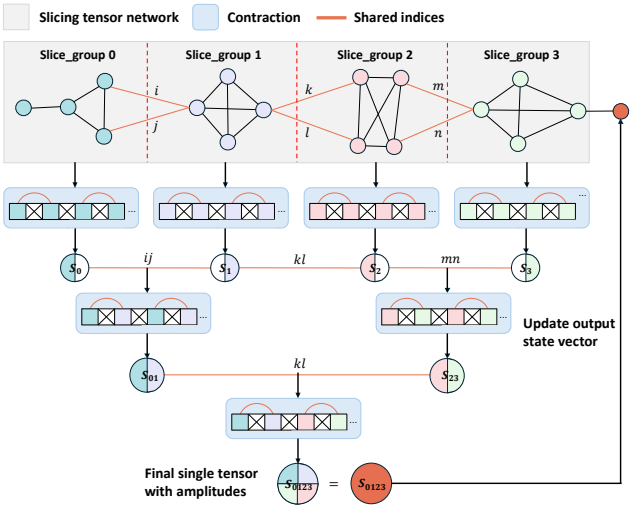


Fig. 7: Tensor Approximation through Intermittent Tensor Contraction

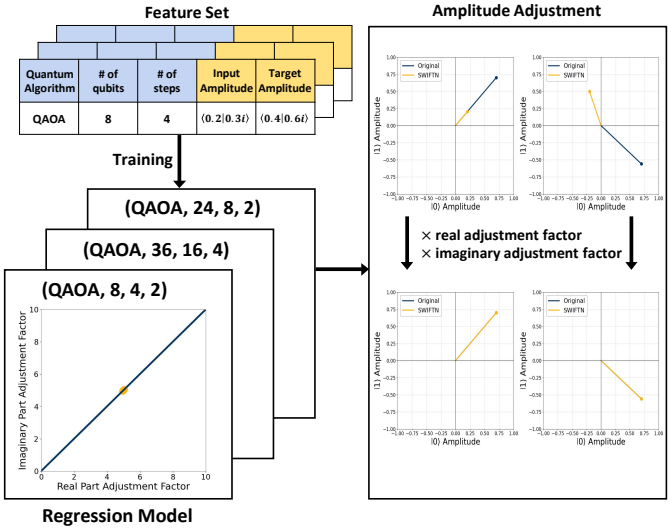


Fig. 8: Amplitude Adjustment through Linear Regression

tensor network slicing, and distribution, these are closed-source and dose not allow fine-grained control and modification of the logic. Therefore, SWIFTN implemented explicit methods for distributing slices using CUDA P2P for intra-node communication and MPI for inter-node communication, refining these processes to enhance scalability and optimize resource utilization in large-scale HPC system.

Even with slicing and distributing the tensor network, the divided tensor segments must still be executed on each GPU and node. While parallel execution can reduce total runtime, it cannot counteract the absolute increase in computation required, as qubits inherently possess complex states, and quantum circuits with numerous gates add further complexity by significantly altering the qubits. To overcome this, SWIFTN employs tensor approximation [48], [49] which is widely used in machine learning which also utilizes the tensor network contraction. To do this, SWIFTN bypasses certain number of tensor contraction steps which eliminates the computational and memory overhead for the tensor.

C. Intermittent Tensor Contraction

Figure 7 illustrates the overall procedure of tensor approximation through intermittent path operations in SWIFTN. As shown in the figure, SWIFTN slices the entire tensor network into sub-networks (i.e., Slice_group 0-4) that are contracted into tensors (i.e., S0-3). This contraction is performed independently and in parallel across each GPU. During tensor contraction, certain contractions are skipped based on a configurable step size. For example, if the step size is set to 2, as shown in the figure, contraction occurs in the first and third tensors, while the contraction of the second tensor is skipped, representing intermittent tensor contraction. Although this process results in approximate tensors, depicted by half-colored circles in the contracted tensors (i.e., S0-3), SWIFTN adjusts the final contraction result once all tensor contractions are complete. As subsequent tensor contractions are performed (e.g., S01, S23, and S1234), similar intermittent tensor contractions are applied to reduce computational overhead.

Given the trade-off between increased computational efficiency and decreased accuracy, SWIFTN can apply intermittent tensor contraction to certain quantum algorithms where exact precision is not essential. For example, optimization algorithms such as the Quantum Approximate Optimization Algorithm (QAOA) are often designed to find near-optimal solutions rather than exact ones. Moreover, real quantum computer hardware supports only a limited set of quantum gates, simplifying their overall implementation. For example, Finnish quantum computer (i.e., Helmi) uses only two types of quantum gates, with the Phased-RX gate as a single-qubit gate and the Controlled-Z gate as a two-qubit gate [50]. This simplicity in the circuit gate structure and the resulting qubits makes approximation methods more practical, enabling SWIFTN to effectively identify patterns in the discrepancies between inaccurate and accurate results, thereby facilitating efficient quantum circuit simulation.

Intermittent tensor contraction can significantly accelerate computation by bypassing tensor contracts. Additionally, in quantum machine learning applications, such as quantum-enhanced classifiers and quantum neural networks, approximate results from SWIFTN are often sufficient for model training. Thus, SWIFTN is adaptable to a variety of practical scenarios in quantum computing, enabling larger and more complex simulations that would otherwise be computationally infeasible.

D. Amplitude Adjustment

While intermittent tensor contraction reduces computational overhead and the runtime, it can result in lower accuracy of the tensor contraction outcomes which is the amplitude of qubit in quantum circuit simulation. To mitigate the drop in accuracy, SWIFTN performs a machine learning-based adjustment that identifies recurring patterns in the reduced

accuracy of intermittent tensor contraction and adjusts the amplitude accordingly.

Figure 8 shows the overall procedure of amplitute adjustment in SWIFTN. As shown in the figure, after the execution of quantum algorithms, an entry is created in the feature set table. In the table, features describing the quantum algorithm (e.g., the specific algorithm and number of qubits), SWIFTN parameters (e.g., number of slice groups and steps), and input/output data (e.g., input and output amplitudes) are collected. We collect these features because changes in them can uniquely affect the output qubit amplitude, helping SWIFTN identify and understand patterns in the impact on accuracy. Using the feature set, we train a regression model with linear regression. The model can be built per quantum algorithm, increasing the number of training data (i.e., executions) but adding pattern complexity. In contrast, the model can also be built for each combination of algorithm, qubit count, and steps which reduces training data but simplifies pattern complexity. For the evaluation, we create the model for each combination of algorithm, qubit count, and steps.

Utilizing the trained model, SWIFTN calculates the amplification factor. Since the resulting amplitudes of a qubit are combinations of real and imaginary numbers, the regression is performed by plotting each execution history with output amplitudes in a two-dimensional vector space, where the x-axis represents real values and the y-axis represents imaginary values. The model then trains to produce two amplification adjustment factors—one for the real and one for the imaginary part—that adjust the degraded accuracy of the approximated qubit state from intermittent tensor operations to match the original state achieved with full tensor contraction. While this requires multiple executions of quantum circuit simulations using both the original and SWIFTN with intermittent tensor contraction, our heuristic evaluation shows that the pattern is simple enough for linear regression to quickly learn with only a few repeated executions. Additionally, this pattern-oriented adjustment allows SWIFTN to recognize patterns in both scenarios: where the slope of the degraded amplitude aligns with the original (as shown in the left graph of Figure 8) and where it is not aligned but still exhibits a similar pattern (as shown in the right graph). Finally, the amplitude is adjusted by multiplying the adjustment factors for the real and imaginary components. Thus, while SWIFTN offers a trade-off between computational efficiency and simulated amplitude accuracy, it aims to mitigate reduced accuracy by identifying patterns in the degradation and adjusting accordingly.

IV. Evaluation

In this section, we evaluate SWIFTN using various quantum algorithm benchmarks on a Perlmutter supercomputer.
Our evaluation of SWIFTN focuses on answering these questions:

- Can SWIFTN provide the scalability within the multiple GPUs in a single node (Section IV-B) and multiple nodes (Section IV-C)?
- What is the impact of the intermittent tensor contraction in SWIFTN on performance (Section IV-B and Section IV-C)?
- Can amplitutue adjustment in SWIFTN restore the original amplitude(Section IV-D)?
- What are the communication overhead of distributing sub-tensor network to multiple GPUs and Nodes (Section IV-E)?

A. Evaluation Setup

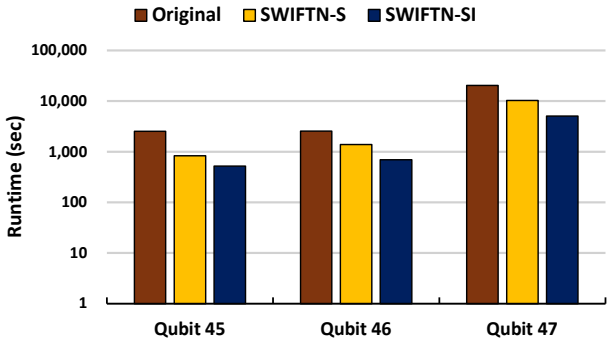
We evaluate SWIFTN on a Perlmutter supercomputer at NERSC. The system consists of 3,072 CPU nodes and 1,792 GPU-accelerated nodes. For this evaluation, only the GPU nodes are used for the tensor network contraction in quantum circuit simulations. Each GPU node has 256GB of DDR4 DRAM as main memory and a single AMD EPYC 7764 (Milan) CPU, along with four NVIDIA A100 (Ampere) GPUs connected via PCIe 4.0. Each GPU features 40GB of HBM2e memory with 1,552 GB/s bandwidth. The four GPUs are interconnected through third-generation NVLink, each providing 25GB/s per direction.

For quantum algorithm benchmark, we use the Quantum Approximate Optimization Algorithm (QAOA). QAOA is optimized for the tensor network computations and is widely used to evaluate quantum circuit simulation performance. We used Cirq as the quantum computing framework to construct the quantum circuits and implemented optimizations of SWIFTN on top of cuTensorNet from the cuQuantum SDK for the tensor network operations. We denote the original cuTensorNet as the tensor computation backend as 'Original' and the SWIFTN-optimized cuTensorNet as 'SWIFTN'.

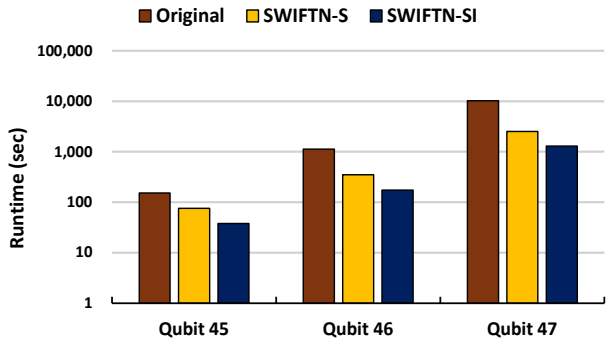
B. Single-node Multi-GPU Scalability

We first evaluated the runtime performance of SWIFTN on a small scale by conducting evaluations on a single node. This is to assess how SWIFTN scales as the number of GPUs increases (2 and 4), without introducing inter-node communication and processing overhead. We use QAOA quantum algorithm benchmark with different qubit counts. To evaluate SWIFTN and analyze the impact of its individual components, we define SWIFTN-S to represent SWIFTN with slicing operations only, while SWIFTN-SI incorporates both slicing and intermittent tensor contraction. This approach allows us to focus not only on overall performance of SWIFTN but also on how specific subcomponents contribute to its effectiveness.

Figure 9 shows the performance of SWIFTN using two GPUs in a single node. As shown in the figure, SWIFTN-S and SWIFTN-SI achieve significantly higher performance with lower runtime compared to the original scheme. When performing QAOA with 45, 46, and 47 qubits, SWIFTN-S achieves runtime improvements of 3.02×, 1.82×, and 2.00×, respectively, compared to the original. Similarly, SWIFTN-SI achieves even greater improvements, with runtime gains of 4.86×, 3.65×, and 4.06× over the original. This is because, in the original scheme, although the contraction of sub-tensor networks consists of independent tasks, many tasks are



(a) Single-node with Two GPUs



(b) Single-node with Four GPUs

Fig. 9: Runtime Performance of Slicing and Intermittent Tensor Contraction on a Single Node Using QAOA (Runtime is in log-scale)

performed sequentially and require frequent synchronization of the entire tensor network. For example, original performs path optimization before and after slicing the tensor network. In addition, even though path optimizations are performed in parallel, frequent synchronization is performed between GPUs to adjust path after the slicing. While this approach aims to capture computational differences before and after slicing, the overhead from frequent GPU communication outweighs the benefits of reduced computation costs achieved through additional path optimization and synchronization. In contrast, `SWIFTN` leverages the independence of the tensor network contractions by slicing the tensor network to match the number of GPUs and explicitly allocating each sub-tensor network to a GPU without additional communication after the initial slicing and distribution. This approach enables parallel execution and reduces synchronization overhead, significantly enhancing runtime performance over the original scheme. Additionally, `SWIFTN-SI` outperforms `SWIFTN-S` by further reducing computational overhead on each GPU through intermittent tensor contraction following slicing.

When performing identical evaluation with four GPUs, as shown in Figure 9b, `SWIFTN-S` achieves performance improvements of 2.03×, 3.22×, and 4.03×. Similar to the evaluation results with two GPUs, `SWIFTN-SI` further reduces runtime by 4.02×, 6.47×, and 7.85× compared to the original. Compared to the runtime with two GPUs, when utilizing four GPUs, both `SWIFTN-S` and `SWIFTN-SI` reduce runtime by

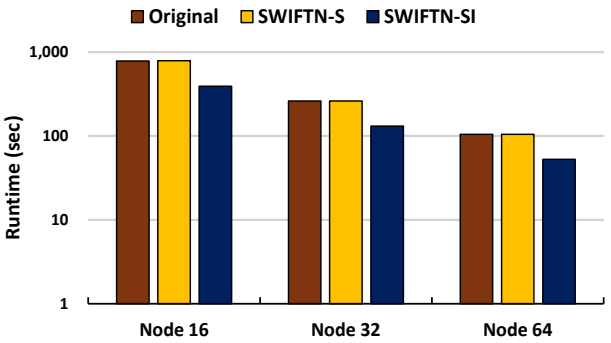


Fig. 10: Multi-node Runtime Performance

up to 1.98×, 2.01×, and 1.95× respectively. The close to linear scalability observed with multiple GPUs on a single node suggests that by explicitly slicing the tensor network to match the number of GPUs, `SWIFTN` effectively reduces concurrency and synchronization overhead during quantum circuit simulation.

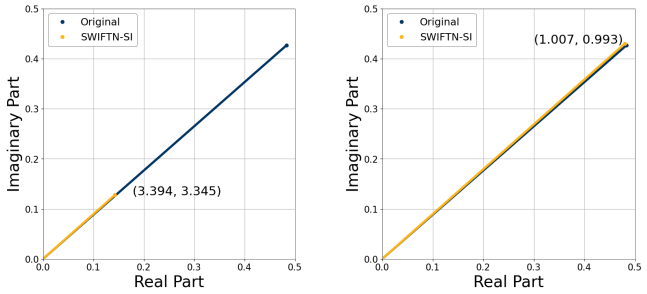
C. Multi-node Runtime Performance

To evaluate the scalability of `SWIFTN` beyond a single node, we perform evaluations using multiple nodes. We use the same QAOA benchmark with 47 qubits, increasing the number of nodes to perform strong scaling evaluation with a fixed problem size. Each node is equipped with four GPUs, and we utilize all available GPUs per node (e.g., 256 GPUs for the 64-node evaluation).

Figure 10 presents a comparison between original, `SWIFTN-S`, and `SWIFTN-SI`. As shown in the figure, the runtime of original and `SWIFTN-S` implementations is similar, with original sometimes performing better in multi-node configurations. This is because as the number of slices increases with the number of nodes and GPUs, the complexity of the tensor network slicing also increases. Unlike `SWIFTN-S`, which simply creates slices with an equal number of tensors and distributes them to the GPUs, original continuously recalculates the optimal path and performs the tensor contraction. In this complex scenario, the benefit of `SWIFTN-S`'s explicit slicing and distribution is comparable to, or even less than, the computational advantage of optimal pathfinding through frequent communication in original. In contrast, `SWIFTN-SI` improves performance by 2.01×, 2.00×, and 1.99× for nodes 16, 32 and 64, respectively, while original shows 13.82% higher performance compared to `SWIFTN-SI`. This demonstrates that `SWIFTN-SI` achieves higher performance as the intermittent tensor contraction reduces computation overhead, while proving its effectiveness at large scales as well.

D. Amplitude Adjustment

As shown in Figure 9, `SWIFTN-SI`, which performs both slicing and intermittent tensor contraction, achieves substantial performance improvements compared to `SWIFTN-S`, which only performs slicing. However, intermittent tensor contraction involve a trade-off between computation time and data consistency, as certain tensors are skipped during the tensor



(a) SWIFTN-SI (b) SWIFTN-SI after Amplitude Adjustment

Fig. 11: Pattern of Intermittent Tensor Contraction

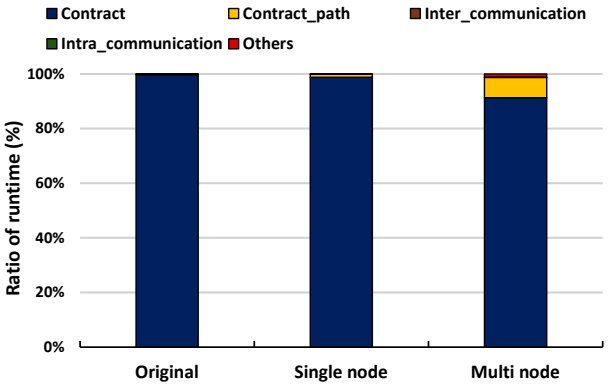


Fig. 12: Runtime Performance of QAOA on 47 Qubits

contraction. In SWIFTN, we aim to adjust the amplitude of the resulting qubit through adjustments. To evaluate the effect of amplitude adjustment, we conduct tests using a quantum circuit benchmark provided by the cuQuantum SDK for amplitude accuracy measurements, represented in the Matrix Product State (MPS) [51] format, which efficiently models quantum circuits using tensor networks.

Figure 11a shows the amplitudes and multiplier factors of output qubit using original and SWIFTN-SI. In the figure, we present the execution results from five runs for both original and SWIFTN-SI. As shown in the figure, while there are some magnitude differences in both the real and imaginary components, the graph maintains the similar slope. Given the similar slope, it confirms that while tensor approximation through intermittent tensor contraction reduces the accuracy, the drop in accuracy is not significant and can be adjusted. Additionally, the execution results from five different runs produce extremely similar vectors, noted by a single overlapping line. This confirms a recurring accuracy pattern even when performing tensor approximation, indicating that adjustments can be made accordingly.

With this accuracy pattern in mind, Figure 11b shows the adjusted amplitude, using the results from Figure 11a and a linear regression model based on the five runs. As shown in the figure, the adjusted amplitude is closely aligned with the original amplitude. By utilizing cosine similarity, an accuracy of 99.997% was achieved. This evaluation result show that SWIFTN can detect patterns within the intermittent tensor contraction, allowing us to compress data effectively, thereby achieving significant improvements in runtime performance while preserving data consistency.

E. Time Analysis

To analyze the time consumption of each subcomponent in SWIFTN, we present the proportion of each subcomponent relative to the total runtime for original, SWIFTN on a single node with four GPUs, and SWIFTN on 32 nodes, as shown in Figure 12. We use QAOA benchmark with 47 qubits.

In the case of original, we only present the time components for contract, contract_path, and others, as the original cuTensorNet implementation is closed-source and does not allow for independent measurement of communication time. As shown in the figure, contraction computation accounted for 99.67% of the total runtime, consuming nearly all processing time. The next most time-intensive task was the contraction path computation, which took 31.50 seconds. In comparison, the contraction itself required 10,205.46 seconds, making path computation a relatively low-intensity task. These findings indicate that contraction computation is the principal component of runtime in quantum circuit simulation, highlighting the need to prioritize contraction time optimization.

In the cases of SWIFTN with single-node and multi-node, inter-node communication time refers to data transfer between nodes via MPI communication, while intra-node communication time denotes data transfer within a node across its GPUs through CUDA P2P communication. As shown in the figure, in the single-node, the contract computation time still dominated the total runtime, similar to original, accounting for 98.74% of the total runtime at 2,608.43 seconds. Additionally, intra-node communication time in the single-node is negligible, indicating that data transfer overhead is not a significant concern in the single-node. In contrast, while contraction still accounts for 91.29% of the total runtime in the multi-node setup, its dominance is reduced. Consequently, the contract path time occupies a more substantial portion of the total runtime at 7.41%, in contrast to the original and single-node. This highlights the need to optimize contract path computation in multi-node environments, where contraction tasks have been highly optimized via SWIFTN. Further details on this topic are discussed in Section VI.

V. Related Works

A. Optimizing Quantum Circuit Simulation

There have been many studies that optimize the quantum circuit simulation to enhance performance. Previous studies [26], [30], [32], [33] focused on partitioning quantum circuits into sub-circuits to manage Schrödinger style full-state simulations enabling the execution of larger quantum circuits through memory offloading, while distributing computation across multiple GPUs. Additionally, other studies [27], [31], [34] targeted quantum circuit simulation by dividing it into sub-circuits, performing parallel processing using independent Feynman-style computations, and synchronizing the results between sub-circuits using the Schrödinger style. The approach of dividing the entire quantum circuit into multiple

sub-circuits is aligned with SWIFTN. However, SWIFTN not only optimizes the slicing of sub-circuits through a two-phase distribution to better utilize computational units, but also approximates the result of the tensor through intermittent tensor contraction. Our approach eliminates the need for additional overhead associated with dynamically tracking large-scale circuits and partitioning sub-circuits, as well as the data movement overhead from continuously modifying and relocating sub-circuits.

B. Utilizing Tensor Network Contraction for Quantum Circuit Simulation

Several studies have conducted to optimize quantum circuit simulation through the tensor network contraction. Previous studies [52]–[57] focused on finding the optimal contraction paths using machine learning(i.e., reinforcement learning) or leveraging various algorithms (e.g., greedy algorithms, polynomial algorithms). In addition, other studies [29], [47], [58] focused on enhancing the parallelization of the tensor network contraction by slicing the tensor network based on indices and partitioning it into sub-tasks. SWIFTN is consistent with these studies in terms of dividing the tensor network into sub-tensor networks through slicing. However, SWIFTN focuses on employing a two-phase distribution to optimize for multi-node GPU clusters in HPC systems through two rounds of the tensor network slicing, while also focusing on amplitude adjustment by correcting the trend identified in intermittent tensor contraction.

VI. Future Works

SWIFTN optimizes large-scale tensor network contraction computation by slicing, two-phase distribution, and tensor approximation, thereby maximizing the runtime performance of quantum circuit simulations. However, SWIFTN poses two primary limitations: contract path computation time and data consistency.

Contract path computation time: While SWIFTN significantly reduces contract computation time compared to the original scheme, as shown in Figure 12, this reduction causes contract path computation time to occupy a larger proportion of the total runtime. Our plan for future work is to reduce contract path runtime by using CPU multi-nodes as contract path computations are more complex and performed on the CPU. To achieve this, we will identify independent computational sections within the contract path process, optimizing paths to improve runtime performance. Additionally, as contract path computations are memory-intensive, we plan to distribute them across the main memory of multiple nodes, enabling scalability for quantum algorithm benchmarks with larger qubit counts efficiently.

Data consistency: SWIFTN-SI achieves improved runtime performance over SWIFTN-S by approximating the tensor needed for contraction through intermittent tensor contraction. Although intervals are set to adjust amplitude patterns, some discrepancies inevitably occur when compared to the original results. Our plan for future work is to improve adjustment accuracy by training an RNN-based machine learning model on these patterns to enable more precise adjustments. A trained RNN model will allow us to predict patterns even when running quantum algorithm benchmarks with a greater number of qubits in the future. By incorporating an RNN model, we aim to enhance runtime performance through data approximation while achieving results closely aligned with the original output.

VII. Conclusion

In this paper, we proposed SWIFTN, a quantum circuit simulation framework for large-scale HPC system, focusing of tensor network distribution and contraction. To this end, we proposed following schemes in SWIFTN: (1) **Slicing and distributing the tensor network**: The tensor network is divided into multiple groups along the contraction path. Also, when deploying SWIFTN on a HPC system, sliced groups are efficiently distributed and processed in parallel through a two-phase distribution method involving both inter-node and intra-node distribution. (2) **Intermittent tensor contraction**: Through tensor approximation, certain tensors are bypassed during contraction to reduce the computation overhead. (3) **Amplitude adjustment**: While intermittent tensor contraction degrades the accuracy of the resulting qubit's amplitude, the absolute values are heuristically adjusted to maintain consistency with the original amplitude. Our evaluation using a real large-scale HPC system shows that SWIFTN improves runtime performance by up to 7.85× compared to the original tensor network contraction result with 99.997% accuracy. We believe that SWIFTN optimizes quantum circuit simulation, enhancing computational performance and expanding scalability.

VIII. Acknowledgment

This research was supported by Seoul National University of Science & Technology. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and also used resources of the National Energy Research Scientific Computing Center (NERSC). We acknowledge the use of LLM to refine the academic language.

References

[1] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien, "Quantum computers," *nature*, vol. 464, no. 7285, pp. 45–53, 2010.

[2] J. L. O'brien, "Optical quantum computing," *Science*, vol. 318, no. 5856, pp. 1567–1570, 2007.

[3] L. M. Procopio, A. Moqanaki, M. Araújo, F. Costa, I. Alonso Calafell, E. G. Dowd, D. R. Hamel, L. A. Rozema, Č. Brukner, and P. Walther, "Experimental superposition of orders of quantum gates," *Nature communications*, vol. 6, no. 1, p. 7913, 2015.

[4] G. J. Mooney, C. D. Hill, and L. C. Hollenberg, "Entanglement in a 20-qubit superconducting quantum computer," *Scientific reports*, vol. 9, no. 1, p. 13465, 2019.

[5] T. Graham, Y. Song, J. Scott, C. Poole, L. Phuttitarn, K. Jooya, P. Eichler, X. Jiang, A. Marra, B. Grinkemeyer *et al.*, "Multi-qubit entanglement and algorithms on a neutral-atom quantum computer," *Nature*, vol. 604, no. 7906, pp. 457–462, 2022.

[6] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Theoretical computer science*, vol. 560, pp. 7–11, 2014.

[7] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani *et al.*, "Advances in quantum cryptography," *Advances in optics and photonics*, vol. 12, no. 4, pp. 1012–1236, 2020.

[8] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental quantum cryptography," *Journal of cryptology*, vol. 5, pp. 3–28, 1992.

[9] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," *Reviews of modern physics*, vol. 74, no. 1, p. 145, 2002.

[10] D. J. Bernstein and T. Lange, "Post-quantum cryptography," *Nature*, vol. 549, no. 7671, pp. 188–194, 2017.

[11] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri *et al.*, "Towards quantum chemistry on a quantum computer," *Nature chemistry*, vol. 2, no. 2, pp. 106–111, 2010.

[12] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. Sawaya *et al.*, "Quantum chemistry in the age of quantum computing," *Chemical reviews*, vol. 119, no. 19, pp. 10 856–10 915, 2019.

[13] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, "Simulated quantum computation of molecular energies," *Science*, vol. 309, no. 5741, pp. 1704–1707, 2005.

[14] H. R. Grimsley, S. E. Economou, E. Barnes, and N. J. Mayhall, "An adaptive variational algorithm for exact molecular simulations on a quantum computer," *Nature communications*, vol. 10, no. 1, p. 3007, 2019.

[15] H. Shang, L. Shen, Y. Fan, Z. Xu, C. Guo, J. Liu, W. Zhou, H. Ma, R. Lin, Y. Yang *et al.*, "Large-scale simulation of quantum computational chemistry on a new sunway supercomputer," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–14.

[16] N. Lambert, Y.-N. Chen, Y.-C. Cheng, C.-M. Li, G.-Y. Chen, and F. Nori, "Quantum biology," *Nature Physics*, vol. 9, no. 1, pp. 10–18, 2013.

[17] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[18] M. Cerezo, G. Verdon, H.-Y. Huang, L. Cincio, and P. J. Coles, "Challenges and opportunities in quantum machine learning," *Nature Computational Science*, vol. 2, no. 9, pp. 567–576, 2022.

[19] W. Huggins, P. Patil, B. Mitchell, K. B. Whaley, and E. M. Stoudenmire, "Towards quantum machine learning with tensor networks," *Quantum Science and technology*, vol. 4, no. 2, p. 024001, 2019.

[20] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa *et al.*, "Tensorflow quantum: A software framework for quantum machine learning," *arXiv preprint arXiv:2003.02989*, 2020.

[21] K. Bader, D. Dengler, S. Lenz, B. Endeward, S.-D. Jiang, P. Neugebauer, and J. Van Slageren, "Room temperature quantum coherence in a potential molecular qubit," *Nature communications*, vol. 5, no. 1, p. 5304, 2014.

[22] M. J. Martin, C. Hughes, G. Moreno, E. B. Jones, D. Sickinger, S. Narumanchi, and R. Grout, "Energy use in quantum data centers: Scaling the impact of computer architecture, qubit performance, size, and thermal parameters," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 4, pp. 864–874, 2022.

[23] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[24] A. A. Saki, M. Alam, and S. Ghosh, "Study of decoherence in quantum computers: A circuit-design perspective," *arXiv preprint arXiv:1904.04323*, 2019.

[25] T. Grurl, J. Fuß, and R. Wille, "Considering decoherence errors in the simulation of quantum circuits using decision diagrams," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–7.

[26] M. Xu, S. Cao, X. Miao, U. A. Acar, and Z. Jia, "Atlas: Hierarchical partitioning for quantum circuit simulation on gpus (extended version)," *arXiv preprint arXiv:2408.09055*, 2024.

[27] L. Burgholzer, H. Bauer, and R. Wille, "Hybrid schrödinger-feynman simulation of quantum circuits with decision diagrams," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2021, pp. 199–206.

[28] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–24.

[29] D. Lykov, R. Schutski, A. Galda, V. Vinokur, and Y. Alexeev, "Tensor network quantum simulator with step-dependent parallelization," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 582–593.

[30] D. Park, H. Kim, J. Kim, T. Kim, and J. Lee, "Snuqs: scaling quantum circuit simulation using storage devices," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–13.

[31] C. Zhang, Z. Song, H. Wang, K. Rong, and J. Zhai, "Hyquas: hybrid partitioner based quantum circuit simulation system on gpu," in *Proceedings of the ACM International Conference on Supercomputing*, 2021, pp. 443–454.

[32] R. Fu, Z. Su, H.-S. Zhong, X. Zhao, J. Zhang, F. Pan, P. Zhang, X. Zhao, M.-C. Chen, C.-Y. Lu *et al.*, "Achieving energetic superiority through system-level quantum circuit simulation," *arXiv preprint arXiv:2407.00769*, 2024.

[33] Y. Zhao, Y. Chen, H. Li, Y. Wang, K. Chang, B. Wang, B. Li, and Y. Han, "Full state quantum circuit simulation beyond memory limit," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[34] S. Westrick, P. Liu, B. Kang, C. McDonald, M. Rainey, M. Xu, J. Arora, Y. Ding, and U. A. Acar, "Grafeyn: Efficient parallel sparse simulation of quantum circuits," in *Proceedings of the IEEE International Conference on Quantum Computing and Engineering*, 2024.

[35] C. Zalka, "Simulating quantum systems on a quantum computer," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1969, pp. 313–322, 1998.

[36] H. Bayraktar, A. Charara, D. Clark, S. Cohen, T. Costa, Y.-L. L. Fang, Y. Gao, J. Guan, J. Gunnels, A. Haidar *et al.*, "cuquantum sdk: A high-performance library for accelerating quantum science," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2023, pp. 1050–1061.

[37] 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 *et al.*, "Quantum computing with qiskit," *arXiv preprint arXiv:2405.08810*, 2024.

[38] S. V. Isakov, D. Kafri, O. Martin, C. V. Heidweiller, W. Mruczkiewicz, M. P. Harrigan, N. C. Rubin, R. Thomson, M. Broughton, K. Kissell *et al.*, "Simulations of quantum circuits with approximate noise using qsim and cirq," *arXiv preprint arXiv:2111.02396*, 2021.

[39] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[40] T. Peng, A. W. Harrow, M. Ozols, and X. Wu, "Simulating large quantum circuits on a small quantum computer," *Physical review letters*, vol. 125, no. 15, p. 150504, 2020.

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

[42] "Nersc perlmutter," https://www.nersc.gov/systems/perlmutter/, accessed: 2024-08-15.

[43] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[44] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[45] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[46] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[47] C. Huang, F. Zhang, M. Newman, X. Ni, D. Ding, J. Cai, X. Gao, T. Wang, F. Wu, G. Zhang *et al.*, "Efficient parallelization of tensor network contraction for simulating quantum computation," *Nature Computational Science*, vol. 1, no. 9, pp. 578–587, 2021.

[48] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.
[49] H. Wang and N. Ahuja, "A tensor approximation approach to dimensionality reduction," *International Journal of Computer Vision*, vol. 76, pp. 217–229, 2008.
[50] "Helmi quantum computer architecture." [Online]. Available: https://vttresearch.github.io/quantum-computer-documentation/helmi/?utm_source=chatgpt.com
[51] G. Vidal, "Efficient simulation of one-dimensional quantum many-body systems," *Physical review letters*, vol. 93, no. 4, p. 040502, 2004.
[52] E. Meirom, H. Maron, S. Mannor, and G. Chechik, "Optimizing tensor network contraction using reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 15 278–15 292.
[53] S. Orgler and M. Blacher, "Optimizing tensor contraction paths: A greedy algorithm approach with improved cost functions," *arXiv preprint arXiv:2405.09644*, 2024.
[54] J. Gray and S. Kourtis, "Hyper-optimized tensor network contraction," *Quantum*, vol. 5, p. 410, 2021.
[55] F. Schindler and A. S. Jermyn, "Algorithms for tensor network contraction ordering," *Machine Learning: Science and Technology*, vol. 1, no. 3, p. 035001, 2020.
[56] J. Xu, L. Liang, L. Deng, C. Wen, Y. Xie, and G. Li, "Towards a polynomial algorithm for optimal contraction sequence of tensor networks from trees," *Physical Review E*, vol. 100, no. 4, p. 043309, 2019.
[57] C. Ibrahim, D. Lykov, Z. He, Y. Alexeev, and I. Safro, "Constructing optimal contraction trees for tensor network quantum circuit simulation," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–8.
[58] T. Vincent, L. J. O'Riordan, M. Andrenkov, J. Brown, N. Killoran, H. Qi, and I. Dhand, "Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction," *Quantum*, vol. 6, p. 709, 2022.