國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

利用計算叢集優化 QAOA 的大規模模擬

Optimizing Large-Scale QAOA Simulation using Computing Clusters

邱信瑋

Shin-Wei Chiu

指導教授: 洪士灝 博士

Advisor: Shih-Hao Hung Ph.D.

中華民國 114 年 7 月

July, 2025

doi:10.6342/NTU202501430

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

利用計算叢集優化 QAOA 的大規模模擬

Optimizing Large-Scale QAOA Simulation using Computing Clusters

本論文係邱信瑋君（學號 R12922054）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 114 年 6 月 24 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 24 June 2025 have examined a Master's thesis entitled above presented by CHIU, SHIN-WEI (student ID: R12922054) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

洪士灝		
（指導教授 Advisor）		
	江介宏	余家佐

系主任/所長 Director: 陳祝嵩

Acknowledgements

首先，我要衷心感謝洪士灝教授這兩年來的悉心指導。在您的引領下，我這個初入研究殿堂的學生得以一步步學習並掌握了獨立完成研究的方法，並成功將研究成果投稿至 2025 年 Practice & Experience in Advanced Research Computing Conference。

同時，我也要感謝江介宏教授和涂嘉恒教授，百忙之中撥冗擔任我的口試委員，並為我的論文提供了寶貴的建議與指導。

特別感謝同為量子組的楊卓敏和侯善融同學。在這兩年辛苦的研究生生涯中，我們時常一同挑燈夜戰，為研究付出無數心力。閒暇之餘，我們也一同打籃球、健身，共同維護身心健康，這段時光彌足珍貴。

此外，也要感謝官澔恩同學這兩年來在課業上的陪伴。許多需要分組的課程，能與他同組總讓我倍感安心，他是一位非常優秀的夥伴。

最後，謹向我的父母和女朋友獻上最深的謝意。你們是我最堅實的後盾，在我失落迷茫時，總能溫暖地陪伴在旁；在我無助時，給予我莫大的鼓勵與信心。即使遭遇挫折，你們也毫無保留地接納我。特別是在碩二上學期，面臨論文寫作、修課、實習、求職、刷題、外語學習等多重壓力，每天都身處巨大的考驗之中，感謝你們堅定的支持，讓我得以順利平安地完成學業。

doi:10.6342/NTU202501430

摘要

本研究開發了一款資源高效的量子電路模擬器，專為在大規模組合優化問題上執行量子近似優化算法 (QAOA) 而設計。我們的核心貢獻在於整合了狀態向量轉置 (State Vector Transposition, SVT) 技術。這項技術透過在跨節點或跨裝置傳輸前執行本地端的量子位元（Qubit）重排，能有效集中傳輸所需的狀態向量區塊。此方法不僅顯著提升了傳輸效率，更有效減輕了模擬大型量子電路所固有的記憶體需求，大幅提升了模擬性能。這種方法有助於在多樣化的分散式計算平台（包括多節點 CPU 和多設備 GPU 架構）上實現高效的擴展。實驗驗證突顯了我們提出的模擬器卓越的性能。在大規模 QAOA 電路上的基準測試表明，與現有最先進的模擬器相比，我們的模擬器實現了顯著的加速，在 GPU 平台上性能提升高達 1490 倍，在純 CPU 環境中提升 28 倍。此外，我們進行全面的可擴展性實驗證實了模擬器在分散式計算環境中保持接近最佳效率的能力。這種可擴展性使我們的工作成為模擬深度 QAOA 電路的寶貴工具，有效地彌合了當前經典計算資源的能力與量子算法研究日益增長的需求之間的差距，成功縮短了經典計算資源與量子算法研究需求之間的鴻溝。最後，我們亦針對 SVT 技術進行全面效能分析，確認其在 QAOA 模擬中的最佳化效益。

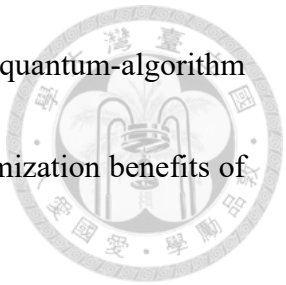
關鍵字：量子近似演算法、量子計算、量子模擬、組合優化問題、平行計算、效能分析

doi:10.6342/NTU202501430

Abstract

This study presents a resource-efficient quantum-circuit simulator purpose-built for running the Quantum Approximate Optimization Algorithm (QAOA) on large-scale combinatorial-optimization problems. Our key contribution is the integration of a State Vector Transposition (SVT) technique. By locally reordering qubits before inter-node or inter-device communication, SVT concentrates the state-vector blocks that actually need to be transferred, dramatically improving data-transfer efficiency while easing the memory footprint inherent to large-circuit simulation. The result is a substantial performance boost and seamless scalability across heterogeneous distributed platforms, including multi-node CPU clusters and multi-device GPU systems. Extensive experiments underscore the superior performance of the simulator. On large QAOA benchmarks, it outperforms state-of-the-art simulators by up to 1,490× on GPUs and 28× on CPUs. Scalability studies further show that the simulator maintains near-optimal parallel efficiency in distributed environments, making it a powerful tool for simulating deep QAOA circuits and closing the gap between

doi:10.6342/NTU202501430



classical computing capabilities and the rapidly growing demands of quantum-algorithm research. Finally, a thorough performance analysis confirms the optimization benefits of SVT within QAOA simulations.

Keywords: quantum approximate optimization algorithm (QAOA), quantum computing, quantum circuit simulation, combinatorial optimization problem, parallel programming, performance analysis

doi:10.6342/NTU202501430

Contents

doi:10.6342/NTU202501430

doi:10.6342/NTU202501430

List of Figures

doi:10.6342/NTU202501430

doi:10.6342/NTU202501430

doi:10.6342/NTU202501430

List of Tables

doi:10.6342/NTU202501430

Chapter 1 Introduction

Combinatorial optimization is a frequently discussed topic in computer science and engineering. Its goal is to find the optimal solution in a limited discrete space by minimizing or maximizing an objective function under given constraints. There are notable examples, including the Max-Cut problem [2], portfolio optimization [6], and various routing challenges, such as the Traveling Salesman Problem [18]. Applications span diverse areas such as logistics, production planning, data analysis, and electronic design automation. Since many combinatorial optimization problems (including Max-Cut) are inherently NP-hard, finding exact solutions for large instances is computationally difficult on classical computers. Therefore, there have been various kinds of approximate algorithms developed to find approximately optimal solutions within polynomial time, such as genetic algorithms [34], Monte Carlo methods [30], and simulated annealing [40]. However, due to the inherent limitations of classical computing architectures, traditional computing strategies are fundamentally limited in terms of scalability and performance.

Quantum computing offers a potential paradigm shift to address these limitations by exploiting principles of quantum mechanics such as *superposition* and *entanglement*. Unlike classical bits, *qubits* possess the ability to exist in a superposition of states, which facilitates a form of inherent parallelism. Entanglement establishes non-classical correlations between qubits, allowing for the exploration of vast computational spaces far

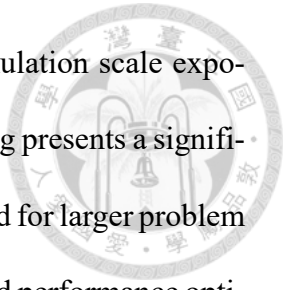
doi:10.6342/NTU202501430

more efficiently than is possible classically. These inherent characteristics underpin the promise of substantial accelerations in the resolution of specific problems, including those in combinatorial optimization and number theory. Pivotal quantum algorithms, such as the Quantum Fourier Transform [11], Shor's algorithm for integer factorization [37], and Grover's algorithm for database search [19], clearly demonstrate the advantages of using quantum computing. Continuous progress in quantum hardware development and theoretical algorithm design has made quantum computing a stunning transformative technology in fields as diverse as cryptography and complex data analysis in computer science and materials science in engineering.

Quantum computing offers several promising methodologies for addressing combinatorial optimization challenges. These include Quantum Annealing [25], the Quantum Adiabatic Algorithm [17], the Variational Quantum Eigensolver (VQE) [32], and the Quantum Approximate Optimization Algorithm (QAOA) [16]. Among these techniques, QAOA has attracted considerable interest, particularly due to its suitability for near-term intermediate-scale quantum (NISQ) devices and its adaptability as a hybrid quantum-classical algorithm. QAOA employs a variational approach to optimize a parameterized quantum circuit, aiming to maximize the anticipated value of a problem's objective function. This iterative process involves executing the quantum circuit with specific parameters, measuring the output, and then using a classical optimizer to update the parameters for the next iteration, progressively refining the solution.

While physical quantum hardware development is ongoing, classical simulation of quantum circuits, especially for algorithms like QAOA, remains essential for algorithm design, testing, and benchmarking. Classical simulation offers the advantage of being free from the noise and decoherence inherent in current physical quantum systems. How-

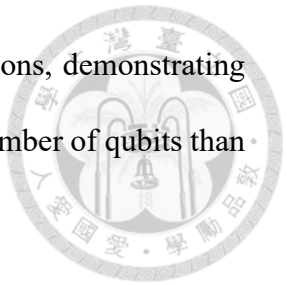
doi:10.6342/NTU202501430



ever, the computational resources necessary for precise classical simulation scale exponentially with the increasing number of qubits. This exponential scaling presents a significant challenge, particularly for simulating deep QAOA circuits required for larger problem instances. Prior research, such as the work by Lin et al. [27], introduced performance optimizations for QAOA simulation on single CPU or GPU nodes, demonstrating superiority over existing methods. Nevertheless, the memory capacity of a single node fundamentally limits these approaches to simulating circuits with a moderate number of qubits (typically below 30-40), precluding simulations for significantly larger problem sizes crucial for demonstrating practical quantum advantage.

To address the limitations of single-node classical simulation and facilitate the investigation of QAOA on more extensive problem instances, this research presents a high-performance, distributed QAOA simulator. A core technical contribution is the development of techniques that exploit the inherent tensor structure of quantum state vectors and gate operations to minimize redundant computations and, critically, reduce costly inter-node and inter-device communication overhead. We conducted extensive experimental evaluations on both CPU-based clusters and GPU-based workstations to assess the performance and scalability of our methodology. On multi-node CPU platforms, our simulator demonstrates superior performance compared to state-of-the-art distributed simulators, including mpiQulacs [23], QuEST [24], and UniQ [43]. Similarly, on multi-device GPU platforms, utilizing configurations up to 64 NVIDIA H100 GPUs, our approach significantly outperforms leading GPU simulators such as Qiskit-Aer [1], cuQuantum [4], and HyQuas [42]. Our experimental results show speedups of up to 28x on multi-node CPU and an impressive 1490x on multi-device GPU platforms compared to existing methods. Furthermore, both strong and weak scaling studies indicate that our distributed simulator

doi:10.6342/NTU202501430



achieves near-optimal efficiency across various hardware configurations, demonstrating its potential for simulating QAOA circuits on a significantly larger number of qubits than previously possible.

The key contributions of this research are outlined below:

1. We present a resource-efficient, distributed QAOA simulator designed to scale to a large number of qubits on multi-node CPU clusters and multi-device GPU platforms.

2. We propose and implement a suite of optimization techniques that leverage the structure of quantum computations to significantly reduce data transfer and computational overhead in large-scale distributed quantum circuit simulations.

3. We provide a comprehensive performance analysis and scalability benchmark of our distributed methodology, comparing it against leading state-of-the-art quantum circuit simulators on both multi-node CPU and multi-device GPU architectures.

doi:10.6342/NTU202501430

Chapter 2 Background

2.1 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) is a hybrid quantum-classical variational method developed for current Noisy Intermediate-Scale Quantum (NISQ) computers. Its fundamental objective is to identify approximate solutions for combinatorial optimization problems. The central concept behind QAOA involves employing a parameterized quantum circuit (referred to as an ansatz) to create a quantum state whose energy expectation value corresponds to the cost function of the original optimization problem.

The QAOA procedure is as follows:

1. **Problem Mapping:** This initial step involves transforming the cost function, denoted as $C(z)$, of the combinatorial optimization problem into a quantum Hamiltonian H_C. This Hamiltonian is typically diagonal in the computational basis, guaranteeing that its ground state aligns with the problem's optimal solution.

2. **Ansatz Preparation:** Prepare a quantum state $|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$ is constructed utilizing a parameterized quantum circuit with p layers, also known as the QAOA depth. This ansatz usually comprises two alternating Hamiltonians: the problem Hamiltonian

doi:10.6342/NTU202501430

H_C and a mixer Hamiltonian H_M (often expressed as $\sum_i X_i$, where X_i represents the Pauli-X operator acting on qubit i). Each layer involves two parameters, γ_k and β_k. The initial state is generally the ground state of the mixer Hamiltonian, such as $|+\rangle^{\otimes n}$.

$$|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = e^{-i\beta_p H_M} e^{-i\gamma_p H_C} \cdots e^{-i\beta_1 H_M} e^{-i\gamma_1 H_C} |+\rangle^{\otimes n}$$

Here, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ are the parameters to be optimized.

3. **Expectation Value Measurement:** The parameterized circuit is executed on a quantum computer, and the expectation value of the problem Hamiltonian H_C is determined with respect to the final state $|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$, resulting in $E(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})| H_C |\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$.

4. **Classical Optimization:** A classical optimization algorithm (e.g., gradient descent, Nelder-Mead) is employed on a classical computer to fine-tune the parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$. The objective is to minimize (or maximize, depending on the specific problem) the measured expectation value $E(\boldsymbol{\gamma}, \boldsymbol{\beta})$.

5. **Iteration:** Steps 3 and 4 are repeated until either the parameters converge or a predefined maximum number of iterations has been completed.

6. **Result Interpretation:** Prepare the final quantum state $|\psi(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*)\rangle$ using the optimal parameters $(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*)$, and perform multiple measurements in the computational basis. The measurement outcome (computational basis state) that appears most frequently typically corresponds to an approximate solution to the original optimization problem.

A significant benefit of QAOA is its comparatively shallow circuit depth, which positions it as a strong contender for demonstrating a quantum advantage on NISQ devices. From a

doi:10.6342/NTU202501430

theoretical standpoint, as the circuit depth p approaches infinity, QAOA holds the potential to identify the optimal solution for the given problem.

2.2 Combinatorial Optimization Problems

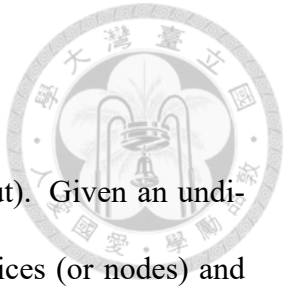
In order to maximize or minimize a specific objective function, combinatorial optimization problems require finding the best solution from a finite (or countably infinite) discrete set of feasible solutions. These problems are widespread in operations research, computer science, engineering, economics, and other fields. The main difficulty is that the size of the space of feasible solutions usually increases exponentially with the size of the problem, making an exhaustive search impractical.

Common instances of combinatorial optimization problems include:

- Knapsack Problem: Choosing the most valuable items to be included in a knapsack without surpassing its weight limit.
- Vehicle Routing Problem (VRP): The purpose of this algorithm is to find the most efficient route for delivering goods or services to customers. The goal is usually to minimize total travel distance, time, or cost while adhering to various restrictions like vehicle capacity or time windows.
- Graph Coloring Problem: This problem involves assigning different colors to the nodes of a graph so that no two adjacent nodes share the same color, with the goal of minimizing the total number of colors used.
- Max-Cut problem: This problem requires splitting the vertices of a given graph into two disjoint sets while maximizing the total number (or sum of weights) of edges

doi:10.6342/NTU202501430

connecting the vertices in the two different sets.



This document centers on the Maximum Cut Problem (Max-Cut). Given an undirected graph $G = (V, E)$, where V represents the collection of vertices (or nodes) and E signifies the set of edges, each edge $(i, j) \in E$ can have an associated weight w_{ij} (if unweighted, $w_{ij} = 1$). The goal of this problem is to partition the vertex set V into two subsets S and $V \setminus S$, such that the cumulative weight of the edges connecting vertices in S and vertices in $V \setminus S$ is maximized.

Formally, we can assign a binary variable $z_i \in \{0, 1\}$ (or $s_i \in \{+1, -1\}$) to each vertex $i \in V$, indicating which set the vertex belongs to. The objective is to maximize the following function:

$$C = \sum_{(i,j)\in E} w_{ij} \frac{1 - s_i s_j}{2}$$

where $s_i = (-1)^{z_i}$. When vertices i and j belong to different sets ($s_i \neq s_j$), then $s_i s_j = -1$, and the edge (i, j) contributes w_{ij} to the objective function. When they belong to the same set ($s_i = s_j$), then $s_i s_j = +1$, and the edge (i, j) contributes 0. The Max-Cut problem is generally NP-hard, meaning that finding an exact solution for large graphs is computationally very difficult.

2.3 Using QAOA to solve Max-Cut Problem

As mentioned before, QAOA is a quantum algorithm designed to solve combinatorial optimization problems. However, it operates directly on quantum states and Hamiltonians, rather than classical graph structures or cost functions. In order to use QAOA to solve the Max-Cut problem, we must first convert the original description of the Max-Cut problem

doi:10.6342/NTU202501430

on a classical computer into a quantum formula suitable for QAOA operations.

Specifically, we need to map the objective function defined on this problem (that is, the sum of the edge weights we want to maximize) to the quantum Hamiltonian H_C. This Hamiltonian should have the following properties: its energy expectation value is related to the value of the objective function, and its ground state (that is, the lowest energy state) corresponds to the optimal solution to this problem (that is, the maximum cut segmentation we require).

Common target formulations for this mapping are two equivalent mathematical models: Quadratic Unconstrained Binary Optimization (QUBO) and the Ising model. Both models help us naturally transition to qubits and quantum Hamiltonians.

2.4 State Vector Simulation

State vector simulation is a method for simulating the process of quantum computation on a classical computer. It is one of the most direct and conceptually simple quantum simulation techniques.

In quantum computing, the state of a system of n qubits is represented by a 2^n-dimensional complex vector, called a quantum state vector, usually expressed as $|\psi\rangle$. This vector resides in a 2^n-dimensional Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$. And, the state vector can be expressed as a linear combination of the computational basis states as below:

$$|\psi\rangle = \sum_{k=0}^{2^n-1} c_k \, |k\rangle$$

where $|k\rangle$ are the computational basis states (e.g., $|00...0\rangle$, $|00...1\rangle$, . . . , $|11...1\rangle$), and $c_k \in$

doi:10.6342/NTU202501430

$\mathbb{C}$ are the complex amplitudes. These amplitudes must satisfy the normalization condition $\sum_{k=0}^{2^n-1} |c_k|^2 = 1$. Among them, the value of $|c_k|^2$ represents the probability of obtaining the result k when the quantum system is measured on the computational basis.

The following is the state vector simulation process:

1. **Representing the State:** Store the 2^n complex amplitudes c_k that represent the quantum state vector $|\psi\rangle$ directly in the memory of a classical computer. This is typically implemented as an array of complex numbers.

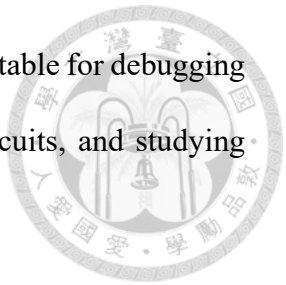
2. **Simulating Evolution:** Operations in quantum computation (quantum gates) correspond to unitary matrices U (of size $2^n \times 2^n$) acting on the state vector. Simulating a quantum gate operation involves multiplying the unitary matrix representing the gate by the state vector stored in memory. For instance, if the system's state is $|\psi\rangle$, applying a gate U results in the new state $|\psi'\rangle = U|\psi\rangle$. In the simulation, this corresponds to computing the new amplitude vector $c' = Uc$. For single-qubit or two-qubit gates, although the corresponding global unitary matrix is large, the update to the state vector amplitudes can often be performed more efficiently due to the sparse or local nature of the operation, without explicitly constructing or storing the full $2^n \times 2^n$ matrix.

3. **Simulating Measurement:** Simulating a measurement of the quantum state $|\psi\rangle$ in the computational basis involves randomly selecting a basis state $|k\rangle$ according to the probabilities $|c_k|^2$. After the measurement, the state vector is updated (collapsed) according to the measurement outcome.

The main advantage of state vector simulation is that it provides complete information about the quantum system's state (all amplitudes c_k) and can simulate quantum evolution

doi:10.6342/NTU202501430

exactly (ignoring numerical precision errors). This makes it highly suitable for debugging quantum algorithms, verifying the correctness of small quantum circuits, and studying quantum phenomena.



However, the primary drawback of state vector simulation is its substantial resource requirement. Storing the state vector for n qubits requires $O(2^n)$ memory space, and simulating a quantum gate operation typically takes $O(2^n)$ computation time (for dense gates) or time related to the number of non-zero amplitudes. This exponential scaling in resource usage limits state vector simulation to systems with a relatively small number of qubits (usually around 30-50, depending on available memory and computational power). For larger quantum systems, other simulation techniques (such as tensor network simulation, stabilizer simulation, etc.) or actual quantum hardware are necessary.

doi:10.6342/NTU202501430

Chapter 3 Related Work

Quantum Approximate Optimization Algorithm (QAOA) has emerged as a leading variational approach for combinatorial optimisation, promising scalable quantum advantage once sufficiently large fault-tolerant hardware becomes available. This chapter reviews five intertwined research threads that underpin large-scale QAOA simulation: (i) algorithmic advances, (ii) classical parameter-search and co-processing, (iii) noise-aware and hybrid variants, (iv) high-performance simulators, and (v) cluster-level systems optimisations.

3.1 Algorithmic Advances in QAOA

Recent algorithmic advancements in QAOA have aimed at improving its performance and scalability. Initial versions of QAOA primarily relied on straightforward implementations with limited levels (P), yet subsequent research has demonstrated that increasing the depth of QAOA circuits enhances solution accuracy. Farhi et al. [16] first showed that raising the depth p improves the Max-Cut approximation ratio. Subsequent works introduced *layer-by-layer* or *p-swap* training strategies that reuse converged shallow parameters as warm-starts for deeper circuits [7, 45]. Adaptive techniques such as parameter transfer across weighted instances [14] and mixer generalisations under the

doi:10.6342/NTU202501430

Quantum Alternating Operator Ansatz [21] further widen the search space. Variants like Recursive-QAOA [3] and Warm-start QAOA [15] reshape the cost landscape or seed parameters with classical heuristics, yielding better convergence on large graphs.

3.2 Classical Parameter-Search and Co-Processing

Parameter optimization remains a critical bottleneck in the implementation of QAOA. Optimising the variational angles γ, β therefore dominates overall runtime. Analytic gradients obtained via the parameter-shift rule [12, 26] or finite-difference methods are widely used, yet they can stall on barren plateaux. Gradient-free meta-heuristics—e.g. Genetic Algorithms [29], Particle-Swarm Optimisation [9], and multi-start local search [36]—explore the rugged landscape more globally. For weighted-Max-Cut instances, an analytic rescaling of unweighted optimal angles has been shown to shrink the search space dramatically and speed convergence [35]. Machine-learning–accelerated co-processing loops that predict promising angle updates on-the-fly further reduce wall-clock time for deep circuits [20].

3.3 Noise-Aware and Hybrid Variants

Considering the practical constraints of current quantum hardware, several noise-aware and hybrid implementations of QAOA have emerged. Noise-aware variants explicitly incorporate noise models into the optimization process or adapt circuits based on hardware characteristics, effectively enhancing robustness under realistic quantum hardware conditions. Noise-robust QAOA seeks to counter hardware decoherence by co-designing parameters with device noise models. Noise-Adaptive VQA [31] and subsequent work

doi:10.6342/NTU202501430

by Bravyi et al. [8] demonstrate fidelity gains by iteratively updating circuit structure or sampling schedule. QuantumNAS [41] generalises this idea to architecture search, offering transferable insight beyond QAOA. These approaches, together with the broader framework of hybrid VQAs [28, 33], underpin practical NISQ-era deployments.

3.4 Quantum Circuit Simulators for QAOA

Reliable simulation of QAOA circuits plays an essential role in algorithmic development, parameter optimization, and benchmarking, particularly given current quantum hardware limitations. Numerous state-of-the-art simulators have been developed to provide efficient and accurate quantum circuit simulation capabilities. Table 3.1 summarizes a comparison of widely used simulators, including QuEST [24], mpiQulacs [23], cuQuantum [4], Qiskit-Aer [1], HyQuas [42], UniQ [43], the GPU-accelerated method by Lin et al. [27], and FOR-QAOA (this work). Simulators such as QuEST, cuQuantum, and Qiskit-Aer leverage distributed computing across multiple nodes and GPU acceleration to enhance computational efficiency. Efficient communication techniques are critical for performance at scale; examples include state vector partitioning with MPI [24], double buffering to overlap communication and computation [23], optimized MPI data transfers combined with computation overlap [1, 4], global-local swap strategies [42, 43], and state vector transposition (used in Our work). These simulators have become indispensable tools for validating QAOA algorithms prior to execution on physical quantum processors.

doi:10.6342/NTU202501430

Table 3.1: Key capabilities of representative state-of-the-art simulators.

Simulator	Distributed	GPU-Accel.	Communication Scheme	Purpose
QuEST [24]	✓	✓	state-vector partition	general purpose
mpiQulacs [23]	✓	✓	double buffering	general purpose
cuQuantum [4]	✓	✓	tensor slicing + overlap	general purpose
Qiskit-Aer [1]	✓	✓	tensor slicing + overlap	general purpose
HyQuas [42]	✓	✓	hybrid partitioner	general purpose
UniQ [43]	✓	✓	hybrid partitioner	general purpose
Lin et al. [27]		✓	N/A	specific for QAOA
Our work	✓	✓	state-vector transposition	specific for QAOA

3.5 Cluster-Level Optimisations for Large-Scale QAOA

Scaling QAOA simulations to large numbers of qubits necessitates careful consideration of cluster-level optimizations. Recent advancements have focused on efficient workload distribution, memory optimization, and communication reduction strategies. Techniques such as domain decomposition (partitioning the state vector or circuit operations), dynamic load balancing, and optimized communication protocols have shown significant potential for improving scalability and performance. Research has explored specialized data structures and memory hierarchy management, such as bit-slicing the Hilbert space representation to reduce memory footprint [39]. Furthermore, parallel computing techniques, including hierarchical parallelism utilizing both inter-node (e.g., MPI) and intra-node (e.g., multi-threading, GPU) parallelism, are crucial for efficiently utilizing cluster resources [10, 13]. Optimizing inter-node communication using high-performance interconnects and protocols like Remote Direct Memory Access (RDMA) has also been investigated to reduce latency and improve bandwidth [22].

These innovations are essential for enabling high-performance simulations of large-scale QAOA circuits across multi-node computing clusters, paving the way for tackling larger problem instances relevant to real-world applications.

doi:10.6342/NTU202501430

Chapter 4 Methodology

In this section, we demonstrate how our optimization strategies can be integrated into a large-scale QAOA simulation for a Max-Cut problem. First, in section 4.1, we discuss the initial, cost, and mixer layers, along with existing optimization methodologies. Second, section 4.2 describes our technique for enhancing data-transfer efficiency across multi-node or multi-device environments.

4.1 Quantum Approximate Optimization Algorithm Preliminaries

The multi-level QAOA, an extension of the fundamental single-level framework, iteratively refines candidate solutions. This work utilizes the Max-Cut problem as a representative application to illustrate the QAOA methodology. Algorithm 1 outlines the simulation procedure for a P-level QAOA circuit.

A single QAOA level (or layer) is characterized by the sequential application of two unitary operators, derived from a cost Hamiltonian $\hat{H}_C$ and a mixer (or driver) Hamiltonian $\hat{H}_M$. Specifically, a P-level QAOA circuit applies a sequence of P such layers. The evolution under these Hamiltonians is governed by a set of $2P$ variational parameters, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_P)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_P)$. These parameters are classically optimized to

doi:10.6342/NTU202501430

minimize (or maximize) the expectation value of $\hat{H}_C$, thereby steering the quantum state towards an approximate solution. The quality of the approximation we obtain generally improves with increasing P.

Algorithm 1 The typical methodology for the QAOA simulation.

```
stateVector ← initZeroState()
for 0 ≤ i < N do                                   ▷ Initial Layer
    stateVector ← HGate(stateVector, i)
end for
for 0 ≤ p < P do                                   ▷ P-level QAOA
    for 0 ≤ i < N do                               ▷ Cost Layer
        for i < j < N do
            stateVector ← RZZGate(stateVector, i, j, γ_p)
        end for
    end for
    for 0 ≤ i < N do                               ▷ Mixer Layer
        stateVector ← RXGate(stateVector, i, β_p)
    end for
end for
```

The construction of a QAOA circuit involves three primary stages for each level:

Initial State Preparation The algorithm commences by preparing the qubits in an appropriate initial state to facilitate exploration of the solution space. The standard approach is to initialize N qubits to $|0\rangle^{\otimes N}$ and then apply a Hadamard gate [5, 19, 38] to each qubit. This produces an equally weighted superposition of all 2^N computational basis states, $|+\rangle^{\otimes N} = \frac{1}{\sqrt{2^N}} \sum_{z\in\{0,1\}^N} |z\rangle$, providing an unbiased starting point for optimization.

Cost Hamiltonian Layer This layer applies the unitary operator $U_C(\gamma_k) = e^{-i\gamma_k \hat{H}_C}$, where $\hat{H}_C$ represents the encoded objective function of the optimization problem. For the maximum cut problem on a graph $G = (V, E)$, where $N = |V|$ nodes, each node typically corresponds to a qubit. In the computational basis, the cost Hamiltonian is diagonal and can be expressed as $\hat{H}_C = \sum_{(i,j)\in E} w_{ij} \frac{1}{2}(I - \hat{Z}_i \hat{Z}_j)$. Here, w_{ij} signifies the weight of the

doi:10.6342/NTU202501430

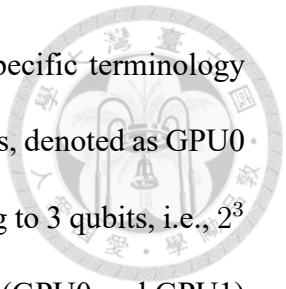
edge between nodes i and j, and $\hat{Z}_k$ is the Pauli-Z operator applied to qubit k. The term $e^{-i\gamma_k w_{ij}\hat{Z}_i\hat{Z}_j/2}$ corresponds to an RZZ gate (or a sequence of CNOT and RZ gates) between qubits i and j, parameterized by γ_k and the edge weight w_{ij}. This encoding directly reflects the Max-Cut objective of maximizing the total weight of edges linking nodes in distinct partitions.

Mixer Hamiltonian Layer Following the cost layer, the mixer layer applies the unitary operator $U_M(\beta_k) = e^{-i\beta_k\hat{H}_M}$. The mixer Hamiltonian $\hat{H}_M$ is chosen to induce transitions between different computational basis states, enabling the algorithm to explore the solution landscape. For $\hat{H}_M$, a common choice is the transverse field Hamiltonian, $\hat{H}_M = \sum_{i=1}^{N}\hat{X}_i$, where $\hat{X}_i$ is the Pauli-X operator on qubit i. And, the corresponding unitary evolution $e^{-i\beta_k\hat{X}_i}$ is an RX gate applied to each qubit. Crucially, $\hat{H}_C$ and $\hat{H}_M$ generally do not commute ($[\hat{H}_C, \hat{H}_M] \neq 0$). This non-commutation is crucial because it ensures that alternating applications of U_C and U_M can explore regions outside the $\hat{H}_C$ eigenstates in the Hilbert space. The classical optimizer then adjusts the parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ to navigate this landscape and find states that produce a good approximate solution.

4.2 State Vector Transposition

In situations where the distance between interacting pairs of quantum states exceeds the memory capacity of a single computational rank, traditional optimization strategies become ineffective. To address this challenge, we introduce *State Vector Transposition (SVT)*, which involves rearranging the state vector within and across computational devices, enabling efficient handling of quantum states beyond the memory constraints of individual ranks.

doi:10.6342/NTU202501430



We utilize Figure 4.1 as an illustrative example to introduce specific terminology used in the proposed algorithm. First, let us assume we have two GPUs, denoted as GPU0 and GPU1. Each GPU's memory can store a statevector corresponding to 3 qubits, i.e., 2^3 statevectors. To simulate a 4-qubit circuit, we need to use two GPUs (GPU0 and GPU1) to store 2^4 statevectors. In this scenario, the index of the fourth qubit (Q3 in the figure) is referred to as a *global qubit*. The reason for this designation is that when operations are performed on this qubit, pairs of statevectors are located on different GPUs rather than locally. It can be observed that the index value of the global qubit directly maps to the GPU index. For instance, GPU0 corresponds to Q3 = 0, and GPU1 corresponds to Q3 = 1. This concept can be further extended. Suppose each GPU's memory can still store a statevector of size corresponding to 3 qubits. If we have 4 GPUs, we can simulate a 5-qubit circuit. In this case, there will be two global qubits (Q4 and Q3). The global qubits corresponding to GPU0 will be Q4 = 0 and Q3 = 0. GPU1 corresponds to Q4 = 0 and Q3 = 1, GPU2 corresponds to Q4 = 1 and Q3 = 0, and GPU3 corresponds to Q4 = 1 and Q3 = 1.

Next, we elaborate on the *State Vector Transposition (SVT)* operation. There are two primary cases for performing SVT, distinguished by the nature of the target qubits involved in the subsequent quantum operation relative to the global qubits and the Local MSB. Further details on these two cases are provided in subsection 4.2.1 and subsection 4.2.2.

4.2.1 Global Qubit Swap with Local MSB

In the first case, at least one target qubit of the subsequent quantum operation is a global qubit, and the remaining target qubits are either not involved in the operation or are not the Local MSB. In this situation, the State Vector Transposition (SVT) is performed

doi:10.6342/NTU202501430

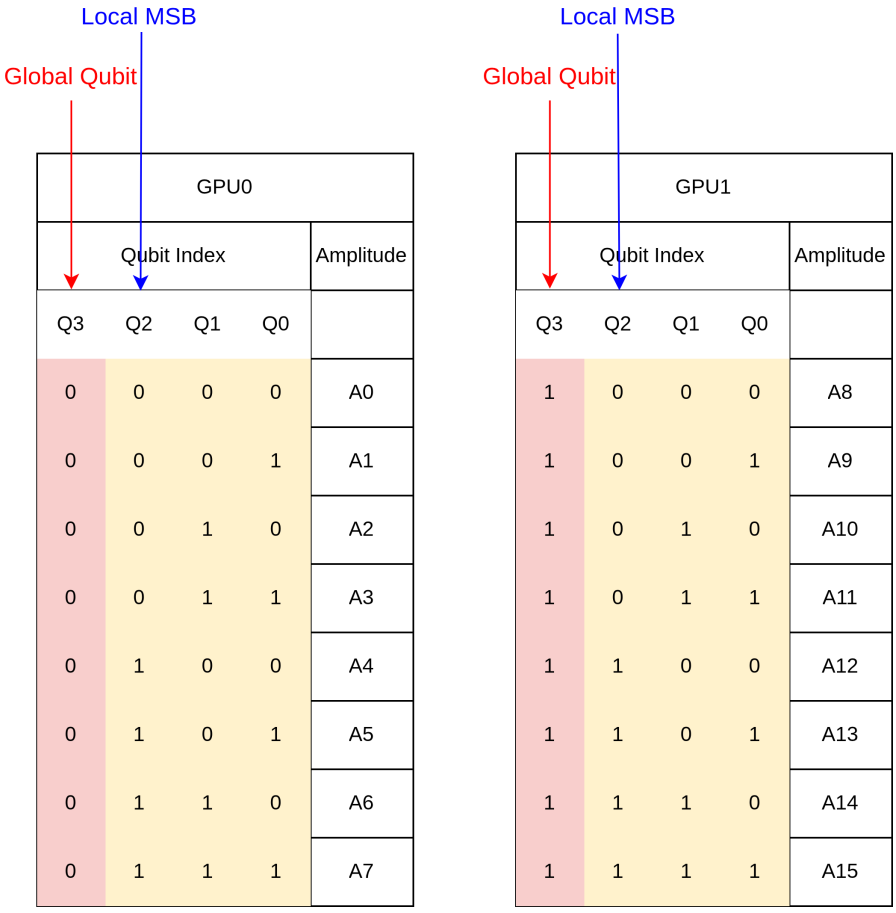


Figure 4.1: Illustration of state vector distribution and global/local qubits on a distributed memory system. A 4-qubit state is distributed across two GPUs, with Q3 designated as the global qubit.

doi:10.6342/NTU202501430

by directly swapping the global target qubit with the Local MSB.

Consider the example of performing a SWAP operation between Q3 (Global) and Q2 (Local MSB) on a 4-qubit state distributed across GPU0 and GPU1, where each GPU holds a 2^3-sized statevector chunk. After the Q3-Q2 SWAP, GPU0 will require amplitudes A8 to A11 from GPU1, while GPU1 will require amplitudes A4 to A7 from GPU0, as conceptually illustrated in Figure 4.2.

GPU0				
Qubit Index				Amplitude
Q2	Q3	Q1	Q0	
0	0	0	0	A0
0	0	0	1	A1
0	0	1	0	A2
0	0	1	1	A3
0	1	0	0	?
0	1	0	1	?
0	1	1	0	?
0	1	1	1	?

GPU1				
Qubit Index				Amplitude
Q2	Q3	Q1	Q0	
1	0	0	0	?
1	0	0	1	?
1	0	1	0	?
1	0	1	1	?
1	1	0	0	A12
1	1	0	1	A13
1	1	1	0	A14
1	1	1	1	A15

Figure 4.2: Conceptual state after Q3-Q2 swap (Case 1). The figure shows the resulting memory layout on GPU0 and GPU1, indicating which amplitudes are missing (marked with '?') and need to be exchanged.

We observe that the amplitudes required for exchange between GPU0 and GPU1 form contiguous blocks, and their relative order is preserved when transferred to the destination GPU. The question then arises: how do we determine which block of amplitudes needs to be exchanged with which GPU? To address this, let's first consider the 2^3 statevector space held locally by each GPU, temporarily ignoring the global qubit distribution. This

doi:10.6342/NTU202501430

2^3 space can be divided into N equal parts, where N is the number of GPUs. For the case of 2 GPUs, this results in two 2^2-sized blocks. Each GPU is responsible for a specific block based on its GPU index. For GPU0 (index 0), the 0-th 2^2 block is local, and the 1st 2^2 block is conceptually associated with GPU1 (index 1). Similarly, for GPU1 (index 1), the 1st 2^2 block is local, and the 0-th 2^2 block is associated with GPU0 (index 0).

When swapping the global qubit (Q3) with the Local MSB (Q2), the data exchange pattern is dictated by the values of Q3. GPU0 holds the data where Q3=0, and GPU1 holds the data where Q3=1. After the swap, the qubit indices are reordered. The amplitudes originally associated with Q3=0, Q2=1 (held by GPU0) need to move to the location where the new index corresponds to Q2=1, Q3=0. This new location is on GPU1. Conversely, amplitudes originally associated with Q3=1, Q2=0 (held by GPU1) need to move to the location where the new index corresponds to Q2=0, Q3=1. This new location is on GPU0. This corresponds precisely to exchanging the 2^2 blocks identified above based on the GPU index/global qubit value. The advantage of performing a qubit swap between the global qubit and the Local MSB is that it significantly reduces the overhead typically associated with arbitrary gather and scatter operations, thereby enhancing data transmission efficiency.

The process is executed step-by-step as follows:

1. **Prepare for Global Data Exchange:** Copy the block of amplitudes to be sent to another GPU into a local buffer. Figure 4.3 illustrates the copying of blocks A4-A7 (from GPU0) and A8-A11 (from GPU1) to the respective GPU buffers.

2. **Global Qubit Swap (Qubit Reordering):** Conceptually (or by index calculation), swap the Global qubit index and the Local MSB index within the local memory

doi:10.6342/NTU202501430

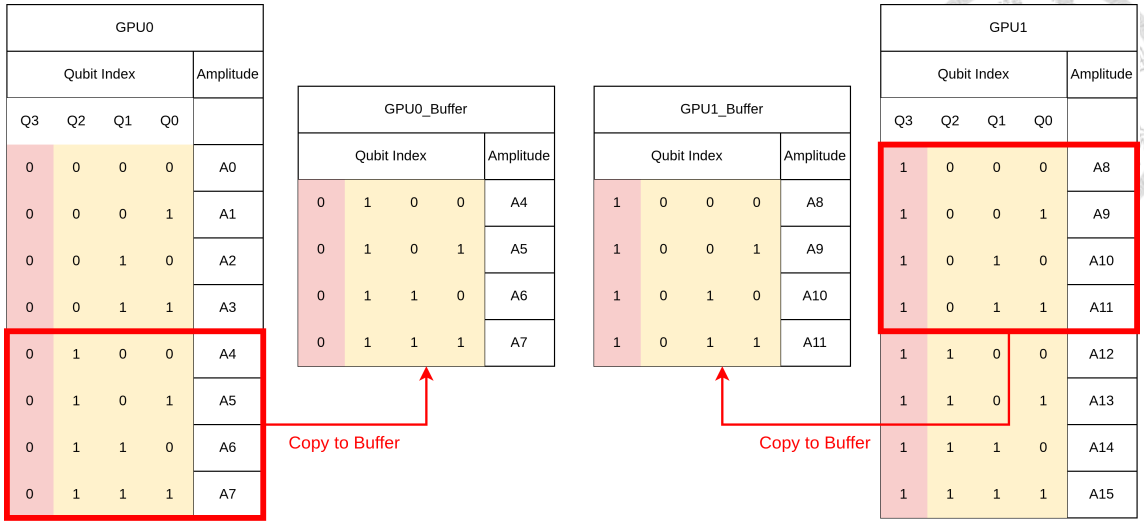


Figure 4.3: Case 1, Step 1: Prepare for Global Data Exchange. The contiguous blocks of amplitudes required by the other GPU (A4-A7 for GPU1, A8-A11 for GPU0) are copied to local buffers.

structure. This operation prepares the memory layout to receive the data from other GPUs. At this stage, each GPU's memory will have a section requiring amplitude blocks from other GPUs, as shown in Figure 4.4. Concurrently, data exchange occurs between the GPUs, with each GPU sending its buffered block to the corresponding destination GPU.

3. **Data Transmission:** Each GPU receives the amplitude block sent from the other GPU into its buffer. The amplitudes from the buffer are then directly written into the corresponding block within the main statevector memory, completing the transposition for this GPU. Figure 4.5 depicts the buffers indicating data being sent and received, and the final statevector arrangement after the received data has been written to the correct locations.

4.2.2 Case 2: Global Qubit Swap with Non-Local MSB

In the second case, at least one target qubit of the subsequent quantum operation is a global qubit, and the remaining target qubits include the Local MSB. Here, we cannot di-

doi:10.6342/NTU202501430

GPU0				
Qubit Index				Amplitude
Q2	Q3	Q1	Q0	
0	0	0	0	A0
0	0	0	1	A1
0	0	1	0	A2
0	0	1	1	A3
0	1	0	0	?(A8)
0	1	0	1	?(A9)
0	1	1	0	?(A10)
0	1	1	1	?(A11)

GPU1				
Qubit Index				Amplitude
Q2	Q3	Q1	Q0	
1	0	0	0	?(A4)
1	0	0	1	?(A5)
1	0	1	0	?(A6)
1	0	1	1	?(A7)
1	1	0	0	A12
1	1	0	1	A13
1	1	1	0	A14
1	1	1	1	A15

Figure 4.4: Case 1, Step 2: Global Qubit Swap (Qubit Reordering). The memory layout is updated according to the new qubit order (Q2, Q3, Q1, Q0), showing the locations where external data (marked with '?') will be written.

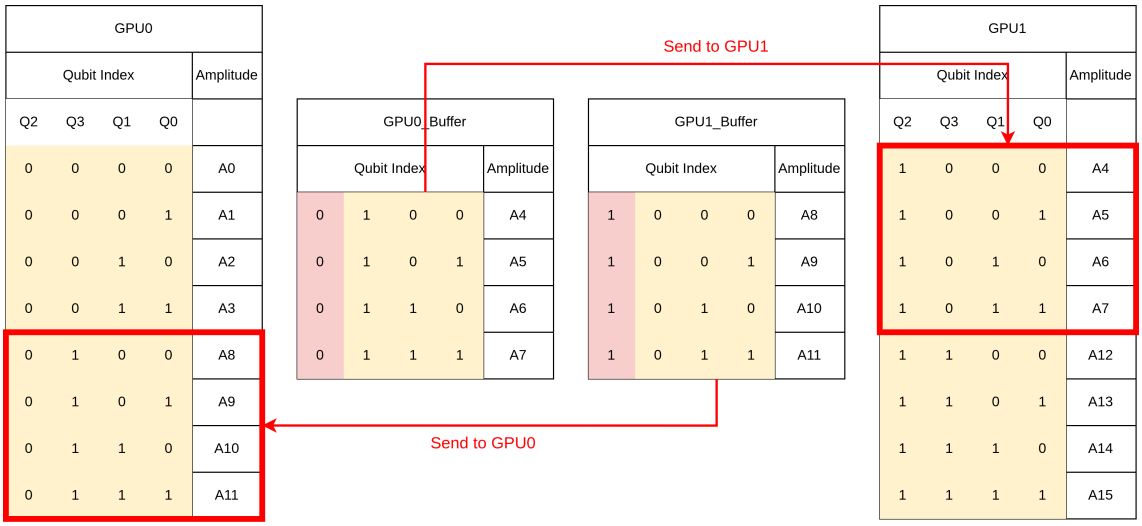


Figure 4.5: Case 1, Step 3: Data Transmission. Data is transmitted between GPUs and written from the buffer into the main statevector memory, completing the SVT for the Q3-Q2 swap.

doi:10.6342/NTU202501430

rectly swap the global target qubit with the Local MSB, as a direct swap between a global qubit (e.g., Q3) and a non-Local MSB local qubit (e.g., Q0 in a Q3-Q0 swap scenario) would result in the statevector amplitudes requiring transmission being scattered, as illustrated in Figure 4.6. Unlike the contiguous data blocks in subsection 4.2.1, this scattered distribution would necessitate frequent calculation and storage of offsets for the data on the local GPU during software implementation, leading to excessive overhead.

GPU0				
Qubit Index				Amplitude
Q0	Q2	Q1	Q3	
0	0	0	0	A0
0	0	0	1	?(A8)
0	0	1	0	A2
0	0	1	1	?(A10)
0	1	0	0	A4
0	1	0	1	?(A12)
0	1	1	0	A6
0	1	1	1	?(A14)

GPU1				
Qubit Index				Amplitude
Q0	Q2	Q1	Q3	
1	0	0	0	?(A1)
1	0	0	1	A9
1	0	1	0	?(A3)
1	0	1	1	A11
1	1	0	0	?(A5)
1	1	0	1	A13
1	1	1	0	?(A7)
1	1	1	1	A15

Figure 4.6: Illustration of scattered data transfer in a direct Global-Non-Local MSB swap (Q3-Q0). The figure shows the state after a direct swap, indicating the non-contiguous amplitudes (marked with '?') that would need to be exchanged between GPUs.

Therefore, to perform a Global Qubit Swap with Non-Local MSB efficiently, the following sequence of steps is executed:

1. **Initial Local Swap:** Perform a local SWAP operation between the local qubit that is a target for the global exchange (the non-Local MSB target qubit, e.g., Q0) and the Local MSB (e.g., Q2). This swap also involves exchanging the corresponding

doi:10.6342/NTU202501430

amplitudes. Figure 4.7 shows the state after performing a local swap between Q0 and Q2 for a target Q3-Q0 global swap. After this step, the amplitudes that would have been scattered in a direct global swap (Figure 4.6) become contiguous and ordered within the local memory blocks when viewed in the context of the subsequent global exchange, as depicted in Figure 4.8.

GPU0				
Qubit Index				Amplitude
Q3	Q0	Q1	Q2	
0	0	0	0	A0
0	0	0	1	A4
0	0	1	0	A2
0	0	1	1	A6
0	1	0	0	A1
0	1	0	1	A5
0	1	1	0	A3
0	1	1	1	A7

GPU1				
Qubit Index				Amplitude
Q3	Q0	Q1	Q2	
1	0	0	0	A8
1	0	0	1	A12
1	0	1	0	A10
1	0	1	1	A14
1	1	0	0	A9
1	1	0	1	A13
1	1	1	0	A11
1	1	1	1	A15

Figure 4.7: Case 2, Step 1: Initial Local Swap. A local swap is performed between the non-Local MSB target qubit (Q0) and the Local MSB (Q2) to reorder the local state vector, making the data for the upcoming global exchange contiguous.

2. **Prepare for Global Data Exchange:** Copy the contiguous block of amplitudes destined for other GPUs (based on the state after the initial local swap) into a local buffer. Figure 4.9 shows the relevant amplitude blocks being copied to buffers on GPU0 and GPU1.

3. **Global Qubit Swap:** Perform the SWAP operation between the Global qubit (Q3) and the qubit currently occupying the Local MSB position (Q0 after Step 1). This

doi:10.6342/NTU202501430

GPU0				
Qubit Index				Amplitude
Q0	Q3	Q1	Q2	
0	0	0	0	A0
0	0	0	1	A4
0	0	1	0	A2
0	0	1	1	A6
0	1	0	0	?
0	1	0	1	?
0	1	1	0	?
0	1	1	1	?

GPU1				
Qubit Index				Amplitude
Q0	Q3	Q1	Q2	
1	0	0	0	?
1	0	0	1	?
1	0	1	0	?
1	0	1	1	?
1	1	0	0	A9
1	1	0	1	A13
1	1	1	0	A11
1	1	1	1	A15

Figure 4.8: Case 2, State after Initial Local Swap. The memory layout after the Q0-Q2 local swap, viewed with the global qubit (Q3) conceptually swapped with the new Local MSB (Q0), showing that the required external data (marked with '?') is now contiguous.

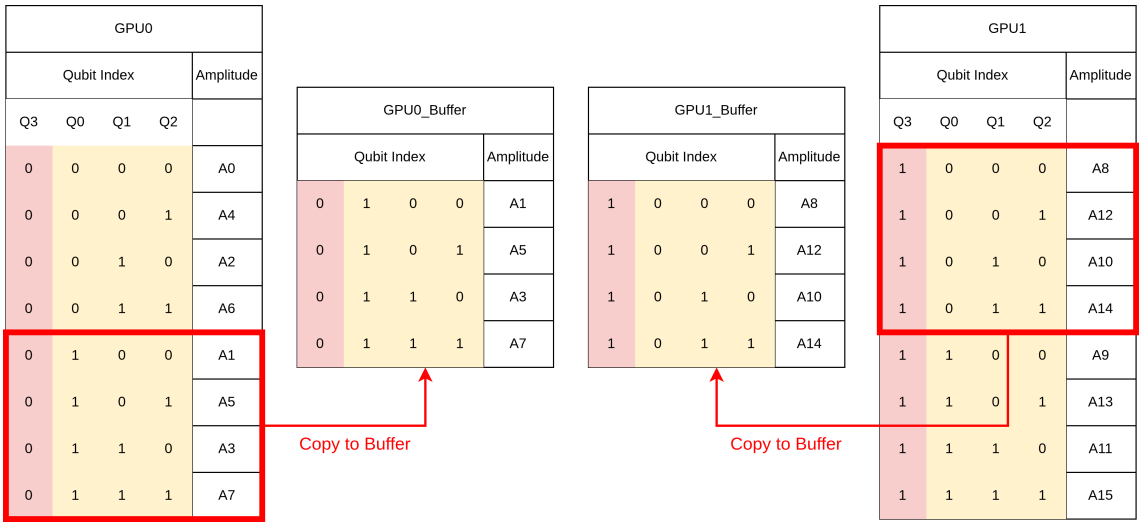


Figure 4.9: Case 2, Step 2: Prepare for Global Data Exchange. Contiguous blocks of amplitudes (resulting from the initial local swap) are copied to local buffers for inter-GPU transmission.

doi:10.6342/NTU202501430

involves transmitting the buffered amplitude blocks between the GPUs. Following this swap, each GPU's statevector memory will have sections that require amplitude blocks from other GPUs, as illustrated in Figure 4.10. Data transmission of the buffered blocks occurs between the corresponding GPUs.

GPU0				
Qubit Index				Amplitude
Q0	Q3	Q1	Q2	
0	0	0	0	A0
0	0	0	1	A4
0	0	1	0	A2
0	0	1	1	A6
0	1	0	0	?(A8)
0	1	0	1	?(A12)
0	1	1	0	?(A10)
0	1	1	1	?(A14)

GPU1				
Qubit Index				Amplitude
Q0	Q3	Q1	Q2	
1	0	0	0	?(A1)
1	0	0	1	?(A5)
1	0	1	0	?(A3)
1	0	1	1	?(A7)
1	1	0	0	A9
1	1	0	1	A13
1	1	1	0	A11
1	1	1	1	A15

Figure 4.10: Case 2, Step 3: Global Qubit Swap. The global swap (Q3-Q0 conceptual swap) is performed, leading to a memory state where external data (marked with '?') is needed. Data is transmitted via buffers.

4. **Data Transmission:** Upon receiving the amplitude block from the other GPU into the local buffer, write these amplitudes directly into the corresponding block within the main statevector memory. This completes the global exchange part of the process. Figure 4.11 shows the data exchange via buffers and the state after writing the received data.

5. **Final Local Swap:** Perform a final local SWAP between the qubit currently at the Local MSB position (Q0) and the qubit that needs to be moved back to the posi-

doi:10.6342/NTU202501430

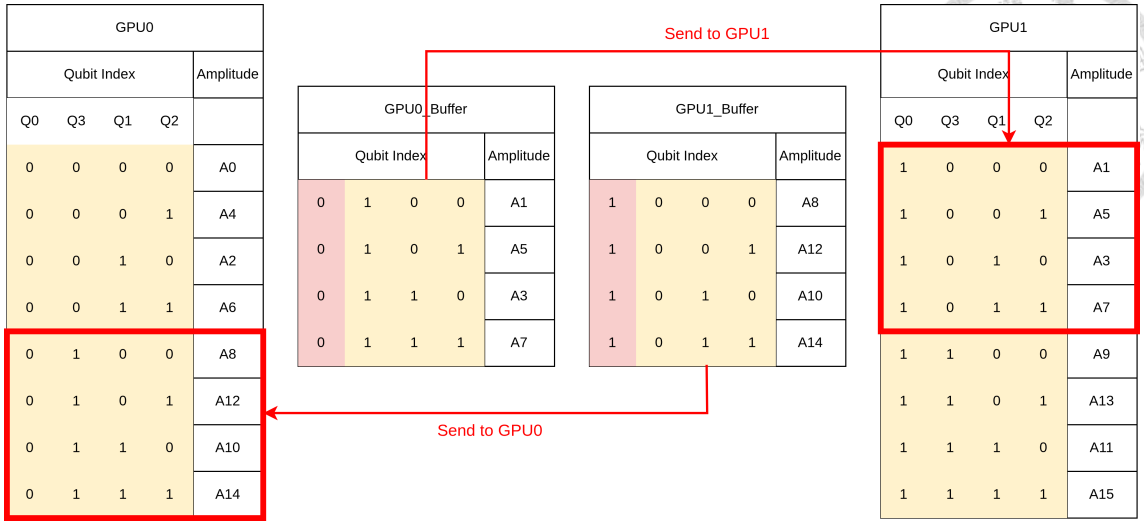


Figure 4.11: Case 2, Step 4: Data Transmission. Data is written from the buffers into the main statevector memory, completing the SVT up to the final local reordering.

tion originally occupied by the local qubit targeted for the global swap (Q2). This final swap restores the correct local qubit ordering relative to the now-transposed global qubit (Q0). Figure 4.12 shows the final state after this concluding local swap, achieving the desired qubit ordering (Q0, Q2, Q1, Q3) for the completed global swap.

doi:10.6342/NTU202501430

GPU0				
Qubit Index				Amplitude
Q0	Q2	Q1	Q3	
0	0	0	0	A0
0	0	0	1	A8
0	0	1	0	A2
0	0	1	1	A10
0	1	0	0	A4
0	1	0	1	A12
0	1	1	0	A6
0	1	1	1	A14

GPU1				
Qubit Index				Amplitude
Q0	Q2	Q1	Q3	
1	0	0	0	A1
1	0	0	1	A9
1	0	1	0	A3
1	0	1	1	A11
1	1	0	0	A5
1	1	0	1	A13
1	1	1	0	A7
1	1	1	1	A15

Figure 4.12: Case 2, Step 5: Final Local Swap. A concluding local swap (Q0-Q2) is performed to achieve the final desired qubit ordering (Q0, Q2, Q1, Q3) after the global Q3-Q0 transposition.

doi:10.6342/NTU202501430

Chapter 5 Evaluation

This chapter presents a performance evaluation of our simulator by comparing it against existing state-of-the-art simulators. We begin in Section 5.1 by outlining the experimental setup for both CPU-based and GPU-based implementations of our work. Subsequently, Section 5.2 details a comprehensive performance evaluation and comparison with other leading simulators. Next, Section 5.3 offers an in-depth analysis of the scalability of our approach in relation to several well-known simulators. In addition, Section 5.4 presents a strong-scaling study of the proposed SVT technique, verifying that our communication efficiency approaches its theoretical optimum. Finally, Section 5.5 analyzes the impact of buffer configurations on transmission performance, leading to further improvements in overall simulation efficiency.

5.1 Experiment Setup

The hardware and software configurations employed for our work are detailed in Table 5.1, encompassing both CPU-only and GPU-accelerated platforms provided by a supercomputing center. The CPU platform is a multi-node cluster interconnected via RDMA, while the GPU platform leverages NVSwitch and InfiniBand for high-bandwidth inter-GPU communication. Both platforms are equipped with contemporary processors,

doi:10.6342/NTU202501430

ample memory resources, and optimized software stacks, including MPI and CUDA, specifically tailored to enhance quantum circuit simulation performance. QAOA benchmark measurements are based on the average execution time of ten independent runs, utilizing double-precision floating-point arithmetic. We selected the five-level QAOA due to its demonstrated superior approximation ratios compared to classical algorithms [44]. Given the repetitive nature of circuits across QAOA levels, performance trends observed for the five-level implementation are representative of those for six or more levels [27].

Table 5.1: The hardware and software configurations of CPU-Only and GPU platforms

Category	CPU-Only Platform	GPU Platform
Cluster Configuration	8 nodes	8 DGX-H100 servers
CPU Model	2 × Intel® Xeon® Platinum 8352V per node	2 × Intel® Xeon® Platinum 8480+ per server
CPU Cores	36 cores per CPU (72 per node)	56 cores per CPU (112 per server)
Memory	768 GB per node	80 GB per GPU, total 64 GPUs
Interconnect	RDMA switch (100 Gbps)	NVSwitch (900 GBps) intra-server, InfiniBand NIC (400 Gbps) inter-server
Operating System	Ubuntu 24.04, Kernel 6.8.0	Red Hat 8.5.0, Kernel 4.18.0
MPI Library	OpenMPI 4.1.7	OpenMPI 4.1.6
CUDA Toolkit	N/A	CUDA 12.2
NCCL Version	N/A	NCCL 2.18.5

5.2 Experimental Results

In this section, we present experimental results demonstrating the performance of our simulator on both CPU and GPU platforms, focusing on the advantages brought by our State Vector Transposition (SVT) strategy for distributed computation across multiple devices and nodes, as introduced in section 4.2.

CPU-Only Platform For the CPU-only platform, we evaluated performance on both single-node and multi-node configurations against several established simulators, including QuEST, mpiQulacs, UniQ, and Lin et al. Figure 5.1 illustrates the simulation time

doi:10.6342/NTU202501430

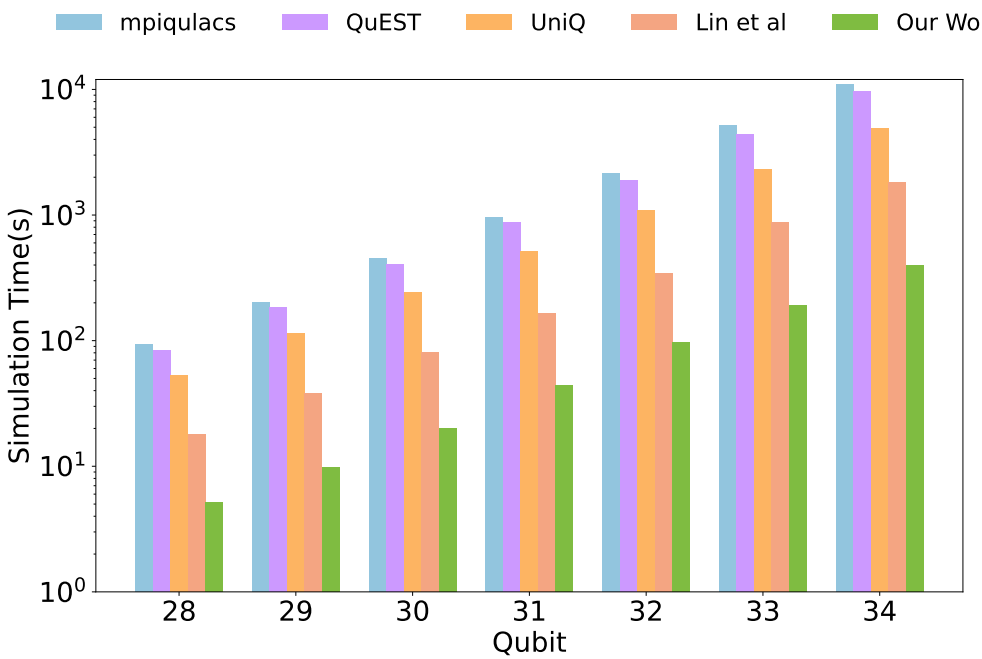


Figure 5.1: Comparison of simulation time on the single-node CPU-only platform.

Table 5.2: The overall speedup for 34-qubit QAOA simulation on the single-node CPU-only platform.

Simulator	Time (sec)	Speedup
QuEST [24]	9666.67	1.2x
mpiqulacs [23]	10983.6	–
UniQ [43]	4861.64	2.3x
Lin et al [27]	1823.83	6.0x
Our Work	**397.59**	**27.6x**

Table 5.3: The overall speedup for 36-qubit QAOA simulation on the multi-node CPU-only platform.

Simulator	Time (sec)	Speedup
QuEST [24]	7181.82	1.6x
mpiqulacs [23]	11800.43	–
UniQ [43]	4875.32	2.4x
Our Work	**415.18**	**28.4x**

doi:10.6342/NTU202501430

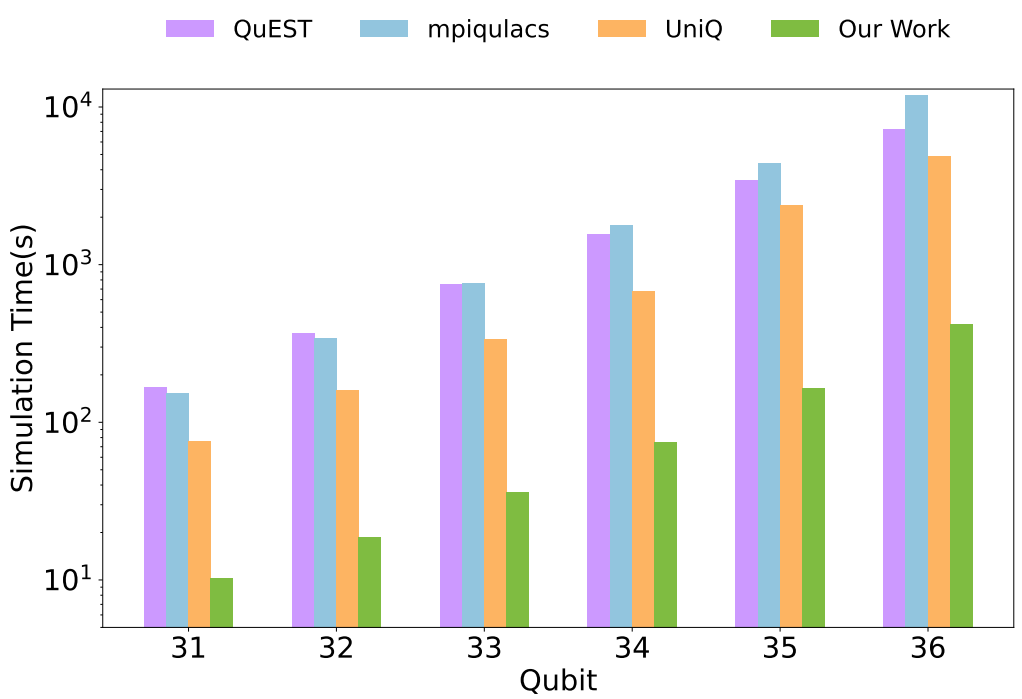


Figure 5.2: Comparison of simulation time on the multi-node CPU-only platform.

on a single computational node for a 5-level QAOA circuit with 28 to 34 qubits. These results establish the strong performance baseline of our simulator on a single CPU node, which is based on prior single-node optimization efforts developed by members of our research team. Table 5.2 further quantifies this, showing the overall speedup for a 34-qubit QAOA simulation where our work achieves a significant 27.6x speedup compared to mpiQulacs [23]. The primary demonstration of our distributed approach, however, is shown on a cluster platform comprising 8 computational nodes for the same 5-level QAOA circuit with 31 to 36 qubits. As depicted in Figure 5.2, our simulator continues to exhibit exceptional performance in this multi-node environment. As detailed in Table 5.3 for the 36-qubit QAOA simulation, the significant speedups achieved by our simulator (up to 28.4x over mpiQulacs, 17.3x over QuEST, and 11.7x over UniQ for 34 qubits) highlight the effectiveness of our SVT strategy in optimizing data transfers and computation across multiple CPU nodes. Lin et al. was not included in multi-node tests as it lacks support for this configuration.

doi:10.6342/NTU202501430

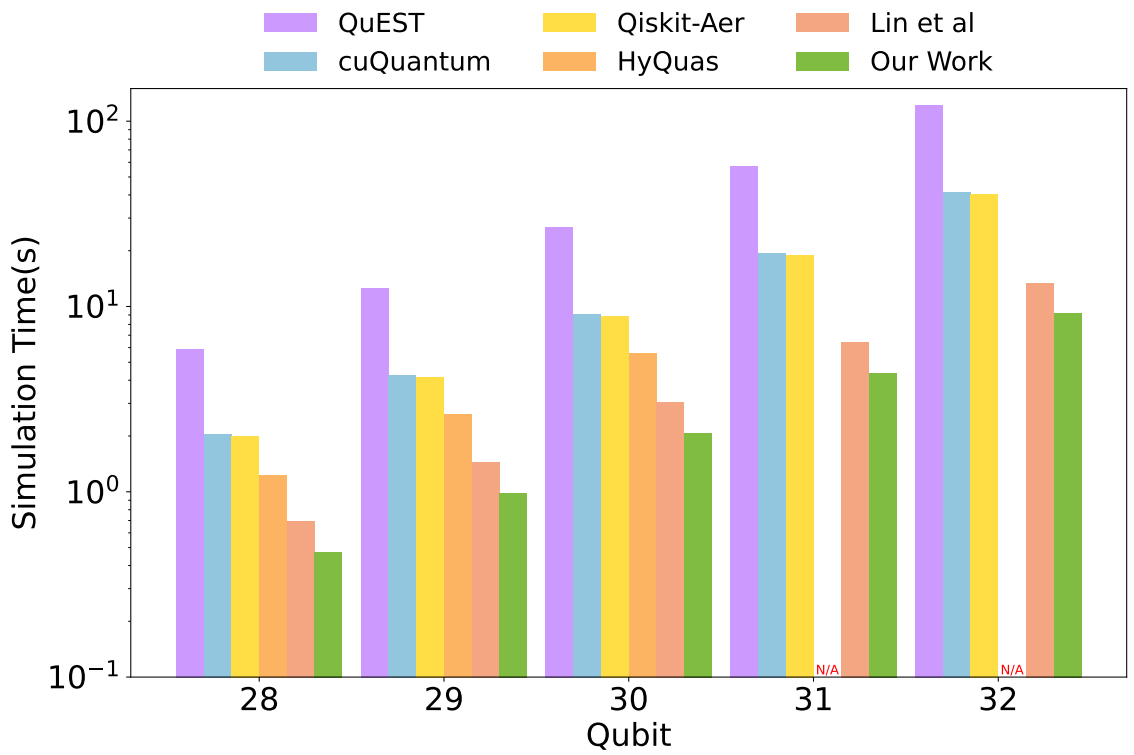


Figure 5.3: Comparison of simulation time on the single-device GPU platform.

Table 5.4: The overall speedup for 32-qubit QAOA simulation on the single-device GPU platform.

Simulator	Time (sec)	Speedup
QuEST [24]	120.94	–
cuQuantum [4]	41.42	2.9x
Qiskit-Aer [1]	40.30	3.0x
HyQuas [42]	OOM	–
Lin et al [27]	13.24	9.1x
Our Work	**9.15**	**13.2x**

Table 5.5: The overall speedup for 30-qubit QAOA simulation on the single-device GPU platform.

Simulator	Time (sec)	Speedup
QuEST [24]	26.68	–
cuQuantum [4]	9.09	2.9x
Qiskit-Aer [1]	8.86	3.0x
HyQuas [42]	5.57	4.8x
Lin et al [27]	3.03	8.8x
Our Work	**2.06**	**13.0x**

doi:10.6342/NTU202501430

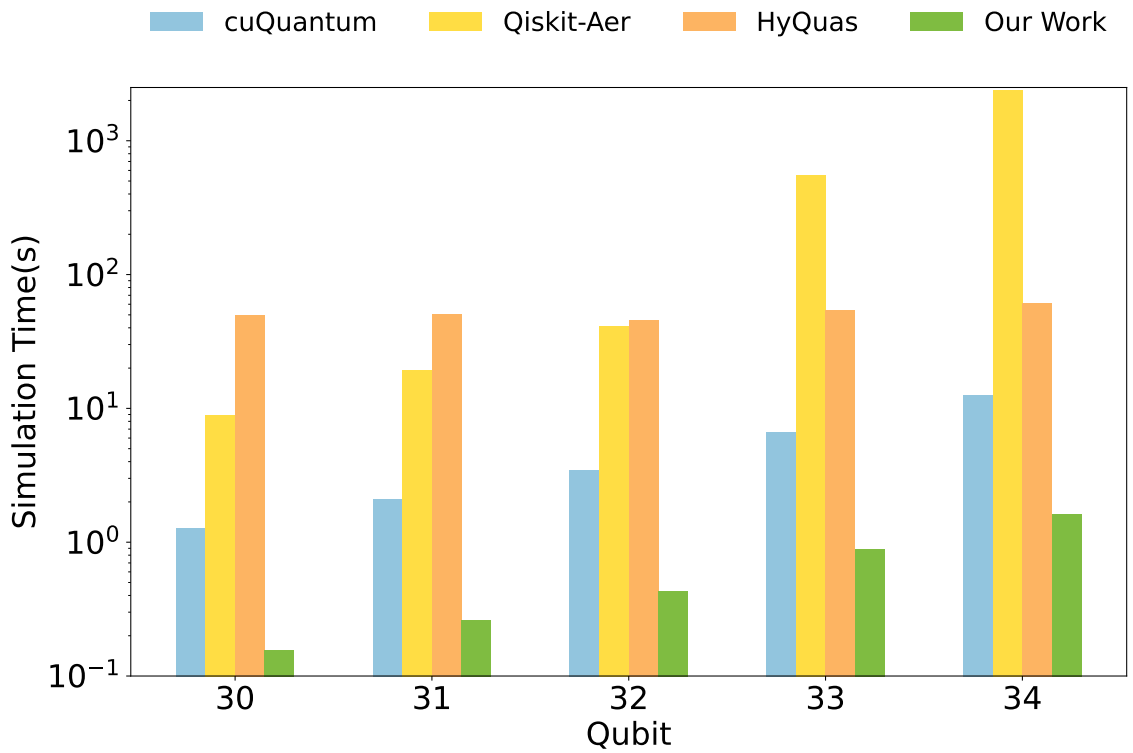
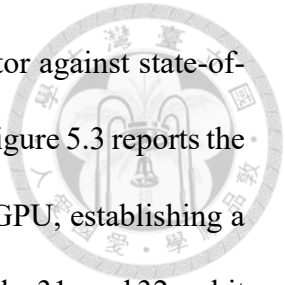


Figure 5.4: Comparison of simulation time on the multi-device GPU platform.

Table 5.6: The overall speedup for 34-qubit QAOA simulation on the multi-device GPU platform.

Simulator	Time (sec)	Speedup
cuQuantum [4]	12.81	187.6x
Qiskit-Aer [1]	2402.86	–
HyQuas [42]	61.49	39.1x
Our Work	**1.61**	**1490.6x**

doi:10.6342/NTU202501430



GPU Platform On the GPU platform we benchmarked our simulator against state-of-the-art frameworks—QuEST, cuQuantum, Qiskit-Aer, and HyQuas. Figure 5.3 reports the runtime of a five-layer QAOA circuit with 28–32 qubits on a single GPU, establishing a strong single-device baseline for our method. HyQuas fails to execute the 31- and 32-qubit cases because its simulation strategy requires excessive memory.

Table 5.4 lists the detailed runtimes for the 32-qubit experiment; our simulator delivers the best performance, achieving a **13.2**× speed-up over the baseline QuEST. Because HyQuas cannot handle 32 qubits, we additionally present 30-qubit results in Table 5.5, where our approach still attains the leading **13.0**× speed-up.

While single-GPU benchmarks already demonstrate clear advantages, the primary benefit of **SVT** lies in distributed simulations. Figure 5.4, therefore, shows the runtime on up to 64 GPUs for circuits with 30–34 qubits. Because HyQuas can simulate at most 34 qubits with eight GPUs, the plot is limited to this size. Table 5.6 summarizes the 34-qubit results: our simulator achieves a dramatic **1490.6**× speed-up relative to the baseline Qiskit-Aer and surpasses cuQuantum (**8.0**×) and HyQuas (**38.2**×). These findings confirm that our optimized data-transfer and computation strategy scales efficiently across multiple GPU devices. QuEST and the method of Lin *et al.* were excluded from the multi-GPU comparison because they do not support this platform configuration.

5.3 Scalability Analysis

To comprehensively evaluate the performance and scalability of our work on distributed systems, we conducted both weak and strong scaling analyses, two standard metrics in high-performance computing. Our experiments spanned multi-node CPU clusters

doi:10.6342/NTU202501430

(analyzed across 1 to 8 nodes) and multi-device GPU platforms (assessed across 1 to 64 devices), covering exponentially increasing resource configurations.

Weak Scaling Weak scaling assesses how effectively a system maintains performance efficiency as the problem size is increased proportionally to the available computational resources. In the context of simulating quantum circuits, this evaluates the simulator's ability to handle larger state vectors by utilizing more nodes or devices while ideally keeping the execution time constant.

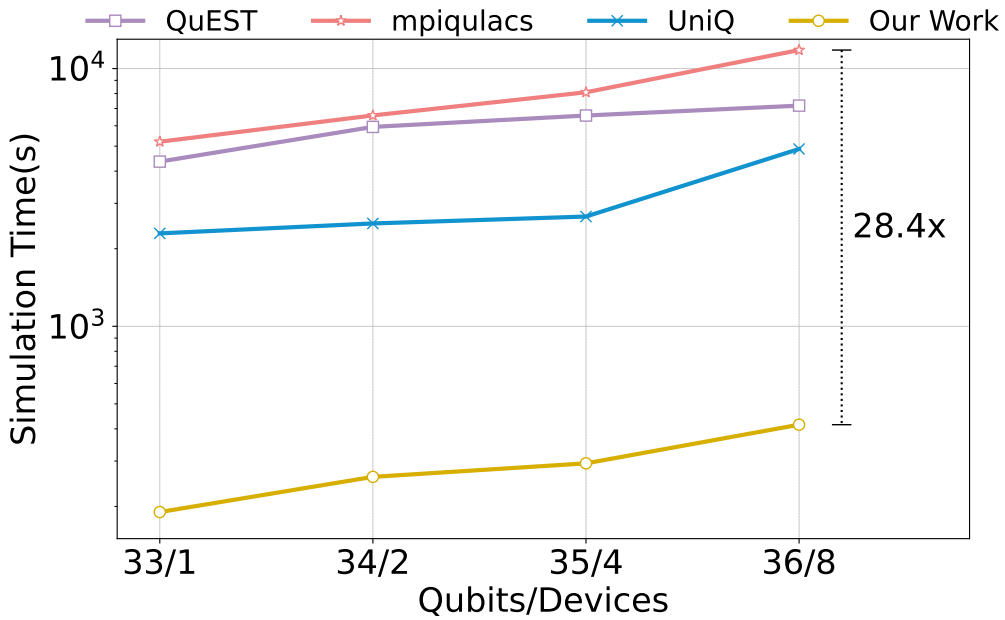


Figure 5.5: Weak-scaling results for proportionally larger 33-qubit−per-node QAOA circuits on a multi-node CPU cluster (1 −8 nodes).

For the CPU-based implementation, weak scaling experiments were conducted on a cluster, starting with a 33-qubit simulation on a single node and scaling up the problem size proportionally with the number of nodes (up to 8 nodes). As shown in Figure 5.5, our approach demonstrates consistently stable efficiency throughout the scaling range, comparable to that of QuEST, mpiQulacs, and UniQ. Importantly, our work achieves a significant performance advantage, maintaining up to a 28.4x speedup over these other

doi:10.6342/NTU202501430

frameworks across all tested configurations.

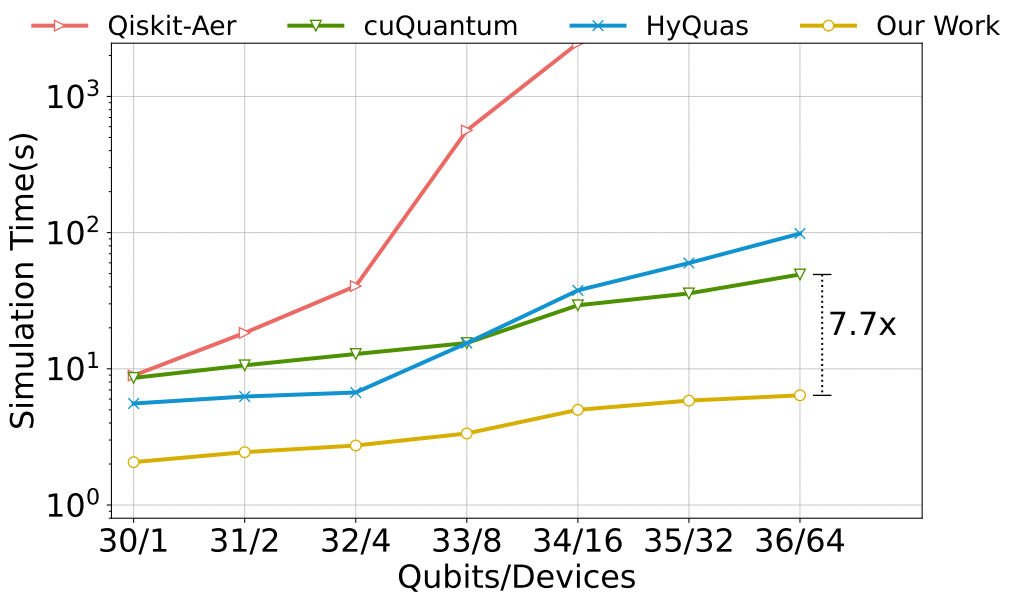


Figure 5.6: Weak-scaling results for proportionally larger 30-qubit−per-GPU QAOA circuits on a multi-GPU platform (1 −64 GPUs).

Similarly, weak scaling of the GPU-based implementation was evaluated on a multi-device GPU environment, starting with a 30-qubit simulation on a single device and increasing the problem size proportionally up to 64 devices. Figure 5.6 illustrates that our work exhibits better scalability and maintains more stable efficiency compared to Qiskit-Aer, cuQuantum, and HyQuas. Notably, Qiskit-Aer encountered out-of-memory errors when scaling to 32 and 64 devices, highlighting a limitation in its ability to handle larger problem sizes in this distributed setting. Across all tested GPU configurations, our simulator consistently delivers a speedup exceeding 7.7x compared to the other simulators.

Strong Scaling Strong scaling measures the reduction in execution time for a fixed-size problem as the number of computational resources increases. This metric assesses how efficiently a system can utilize additional resources to solve a constant problem faster, ideally showing a linear reduction in execution time with a linear increase in resources.

doi:10.6342/NTU202501430

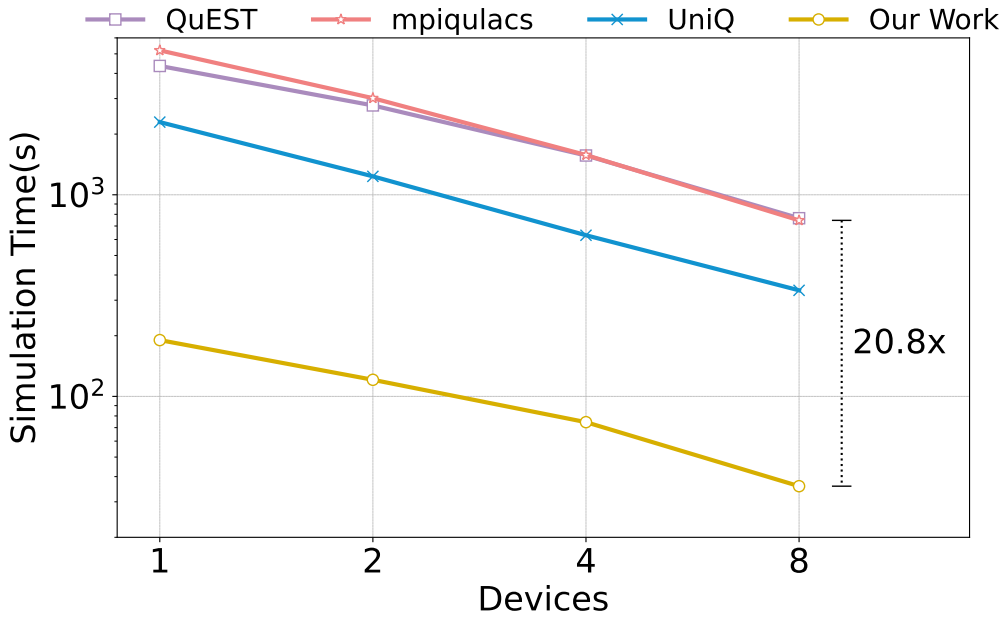


Figure 5.7: Strong-scaling results for a fixed 33-qubit QAOA circuit on a multi-node CPU cluster.

The strong scaling performance for the CPU-based implementation, shown in Figure 5.7, evaluates a fixed 33-qubit QAOA simulation on the cluster. The results demonstrate remarkable efficiency: our simulator, utilizing a single node, achieves an execution time that is only one-quarter of the time required by QuEST and mpiQulacs using eight nodes. Furthermore, comparing performance at the maximum tested scale, our approach with eight nodes shows a substantial 20.8x speedup compared to QuEST, highlighting its superior efficiency in solving a fixed problem size with increasing resources.

For the GPU-based implementation, strong scaling tests evaluated a fixed 30-qubit QAOA simulation across multiple GPU devices (up to 64 devices), as depicted in Figure 5.8. Our simulator sustains consistently low runtimes as resources scale, delivering an 11.3 × speed-up over cuQuantum, the most stable competitor in our evaluation, when scaling from a single GPU up to 64. By contrast, HyQuas benefits from multi-GPU configurations only within a single node; once the simulation spans multiple nodes, its runtime increases, indicating bottlenecks introduced by inter-node communication. Although

doi:10.6342/NTU202501430

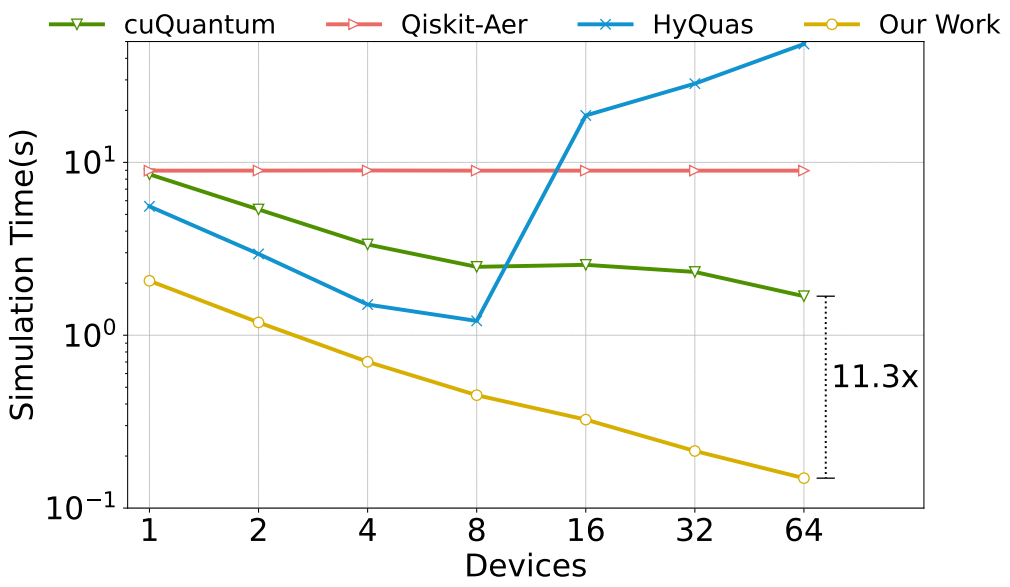


Figure 5.8: Strong-scaling results for a fixed 30-qubit QAOA circuit on a multi-GPU platform (1–64 GPUs).

Qiskit-Aer supports MPI-based parallelism, its runtime decreases only marginally with additional GPUs, suggesting that the current MPI backend does not fully exploit multi-GPU resources for this workload.

5.4 State Vector Transposition Analysis

In the preceding sections, we demonstrated the performance gains achieved by integrating our SVT technique. Across both **CPU** and **GPU** clusters, our simulator outperforms other well-known simulators in benchmark tests. Moreover, **weak** and **strong** scaling analyses confirm that the optimized simulator exhibits superior scalability. However, these results represent the overall simulation time and do not isolate the contribution of SVT itself. Consequently, this section revisits the **strong scaling** experiments on CPU and GPU clusters to investigate the intrinsic performance and scalability characteristics of SVT.

doi:10.6342/NTU202501430

In a strong scaling scenario, the ideal simulation time should decrease inversely with the number of devices:

$$\textbf{Simulation Time} = \frac{\textbf{Single-Device Simulation Time}}{\textbf{Number of Devices}}.$$

However, as communication overhead is introduced on the simulator during the communication process, it is essential to evaluate whether the sum of the expected simulation time and the SVT execution time aligns with the actually measured strong scaling simulation time. In other words:

$$\textbf{Actual Measured Simulation Time} = \textbf{Ideal Simulation Time} + \textbf{SVT Overhead}.$$

Next, we analyze the performance of SVT under a strong scaling scenario on CPU and GPU clusters separately.

SVT Analysis on CPU Clusters We performed a strong scaling study of a 5-level, 33-qubit system on a CPU cluster with SVT enabled; the results are summarized in Table 5.7. Two key observations emerge:

1. **SVT overhead decreases with node count.** Although the total data exchanged per node remains constant, the *SVT overhead* gradually declines as the number of nodes increases because:
    - a reduced amount of data destined for each peer (due to even partitioning), and
    - the ability to transfer these smaller chunks in parallel, which amortizes communication costs. Consequently, the additional time introduced by SVT diminishes with higher parallelism.

doi:10.6342/NTU202501430

2. **Close agreement between predicted and measured times.** The measured simulation times match the sum of the ideal times and SVT overhead, with similarity ratios ranging from 0.93 to 0.98, indicating that SVT imposes virtually no additional cost beyond its own overhead.

Table 5.7: Strong scaling analysis for 33-qubit systems on CPU clusters.

Qubits/Nodes	Ideal Time (s)	SVT Overhead (s)	Measured Time (s)	Similarity (Ideal + SVT vs. Measured)
33/1	N/A	N/A	190.22	N/A
33/2	95.11	22.89	120.89	0.98
33/4	47.56	22.06	74.51	0.93
33/8	23.78	21.37	45.87	0.98

SVT Analysis on GPU Clusters A *strong-scaling* study of a 5-level, 32-qubit system was carried out on a GPU cluster with **SVT** enabled; the results are presented in Table 5.8. Two key findings emerge:

1. **Inter-node communication substantially increases SVT overhead.** When data exchange crosses node boundaries, the *SVT overhead* rises markedly, reducing the acceleration gained in the measured simulation time. A detailed breakdown of intra-node versus inter-node communication performance is provided in subsequent sections.

2. **High agreement between predicted and measured times is maintained.** Although the similarity between the predicted time (Ideal + SVT) and the measured time is lower than that on the CPU cluster, it never falls below 0.87. The lower ratio is attributed to the extremely short compute time on GPUs, which magnifies any additional overhead; nevertheless, SVT introduces no noticeable extra simulation cost.

doi:10.6342/NTU202501430

Table 5.8: Strong scaling analysis for 32-qubit systems on GPU clusters.

Qubits/Nodes	Ideal Time (s)	SVT Overhead (s)	Measured Time (s)	Similarity (Ideal + SVT vs. Measured)
32/1	N/A	N/A	8.87	N/A
32/2	4.43	0.43	5.06	0.96
32/4	2.22	0.29	2.66	0.94
32/8	1.11	0.19	1.50	0.87
32/4	0.56	0.59	1.29	0.88
32/8	0.28	0.49	0.86	0.89

5.5 Buffer Tuning for State Vector Transposition

Achieving optimal performance for distributed quantum simulations on high-performance computing (HPC) systems requires meticulous tuning of data-intensive operations such as *State Vector Transposition (SVT)* described in section 4.2. In this context, *buffer tuning* is pivotal to maximizing data-transfer efficiency.

Buffer tuning selects the optimal size of the data chunk (number of subvectors) transmitted per communication call. Transfer performance is dictated by the trade-off between latency and throughput:

- **Buffer too small—latency dominated.** Tiny chunks trigger many transfer initiations, incurring high per-byte overhead and frequent stalls, which degrade overall performance.
- **Buffer too large—*bufferbloat.*** Oversized chunks can saturate the communication pipeline, lowering the effective throughput.

Accordingly, this section investigates buffer tuning on both **CPU** and **GPU** clusters to identify well-balanced buffer sizes, thereby boosting transfer efficiency and reducing total simulation time.

doi:10.6342/NTU202501430

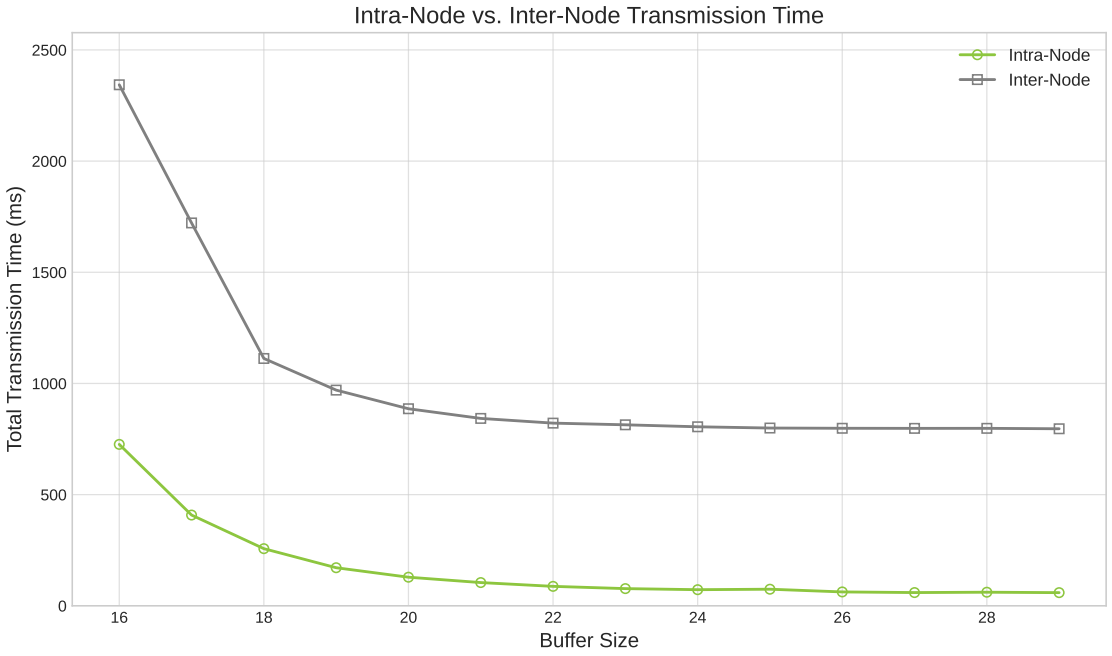


Figure 5.9: SVT transfer time versus buffer size on an H100 GPU cluster. Intra-node transfers use NVSwitch (900 Gbps); inter-node transfers use InfiniBand NICs (400 Gbps).

Buffer size (2^X) \ Transmission Type	16	18	20	22	24	26	28
Intra-Node	725.95 (3.2x)	256.99 (4.3x)	128.63 (6.9x)	87.46 (9.4x)	72.49 (11.1x)	62.43 (12.8x)	61.11 (13.0x)
Inter-Node	2343.29	1112.39	886.28	821.57	805.10	798.44	798.16

Figure 5.10: Detailed comparison of intra-node and inter-node SVT transfer times and the resulting speed-up factor (inter-node time divided by intra-node time). The speed-up approaches the theoretical 18× limit implied by the NVSwitch-to-InfiniBand bandwidth ratio..

doi:10.6342/NTU202501430

Buffer Tuning on GPU Clusters We benchmarked a five-layer, 30-qubit QAOA circuit on H100 GPU clusters to study how buffer size affects the data-transfer time of *State Vector Transposition (SVT)*. Two scenarios were considered—**intra-node** transfers over NVSwitch (900 Gbps) and **inter-node** transfers over InfiniBand (400 Gbps)—as summarized in Fig. 5.9. Intra-node communication is markedly faster than inter-node communication. For both modes, transfer time decreases with larger buffers, albeit with diminishing returns. Figure 5.10 presents the detailed results, showing that the inter-/intra-node time ratio (i.e., the intra-node speed-up) grows monotonically with buffer size and asymptotically approaches the 18-fold difference predicted by the link bandwidths.

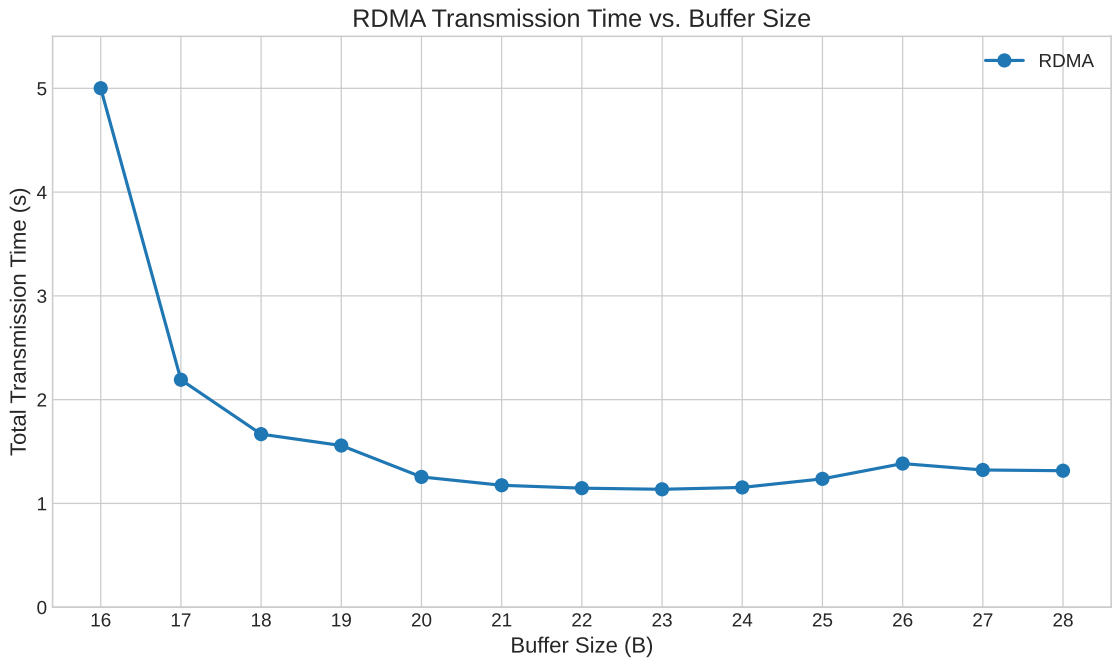


Figure 5.11: RDMA transfer time versus buffer size for a single-layer, 31-qubit QAOA simulation on a multi-node CPU cluster. The minimum (1.14 s) occurs at 2^{23} B; larger buffers provide diminishing returns and eventually incur extra latency due to incipient *bufferbloat*.

Buffer Tuning on CPU Clusters On the CPU cluster we benchmarked a single-layer, 31-qubit QAOA circuit and swept the RDMA buffer size (Fig. 5.11). Transfer time drops steeply from 5.00 s at 2^{16} B to 1.14 s at 2^{23} B; beyond that point the curve flattens and rises

doi:10.6342/NTU202501430

Buffer size (2x) \ Transmission Type	16	17	18	19	20	21	22	23	24	25	26	27
RDMA	5.00	2.20	1.67	1.56	1.26	1.17	1.15	1.14	1.15	1.24	1.39	1.32

Figure 5.12: Detailed RDMA transfer times (seconds) for buffer sizes 2^{16} – 2^{27} B. The optimal value (1.14 s at 2^{23} B) is highlighted in red.

slightly, signalling the onset of *bufferbloat*. Figure 5.12 lists the detailed measurements, showing that a buffer size of 2^{23} B offers the best latency–throughput trade-off, yielding a 4.6 × reduction relative to the smallest buffer.

doi:10.6342/NTU202501430

Chapter 6 Conclusion

In this work, we presented a resource-efficient quantum circuit simulator specifically optimized for executing the Quantum Approximate Optimization Algorithm (QAOA) on large-scale combinatorial optimization problems. Our key contribution lies in the integration of State Vector Transposition (SVT), a technique that effectively mitigates the inherent exponential computational and memory demands associated with simulating large quantum circuits. This approach facilitates efficient scaling across diverse distributed computing platforms, including multi-node CPU and multi-device GPU architectures.

Experimental validation highlights the superior performance of our proposed simulator. Benchmarking on large-scale QAOA circuits demonstrates substantial speedups compared to existing state-of-the-art simulators, achieving performance improvements of up to 1490x on GPU platforms and 28x on CPU-only environments. Furthermore, comprehensive scalability experiments confirm the simulator's ability to maintain near-optimal efficiency in distributed computing environments. This scalability positions our work as a valuable tool for simulating deep QAOA circuits, effectively bridging the gap between the capabilities of current classical computational resources and the growing demands of quantum algorithm research.

doi:10.6342/NTU202501430

References

[1] Qiskit aer: High-performance quantum circuit simulation, 2024.

[2] G. Ausiello, A. Marchetti-Spaccamela, P. Crescenzi, G. Gambosi, M. Protasi, and V. Kann. *Complexity and Approximation*. Springer, 1999.

[3] E. Bae and S. Lee. Recursive qaoa outperforms the original qaoa for the max-cut problem on complete graphs. *Quantum Information Processing*, 23(3):78, 2024.

[4] H. Bayraktar, A. Charara, and D. e. Clark. cuquantum sdk: A high-performance library for accelerating quantum science, 2023.

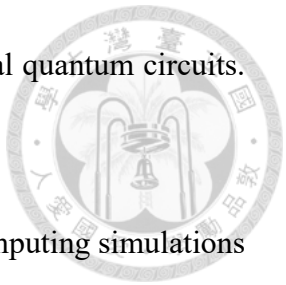
[5] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[6] F. Black and R. Litterman. Global portfolio optimization. *Financial Analysts Journal*, 48(5):28–43, 1992.

[7] A. Brady and S. van Frank. Layerwise learning for the quantum approximate optimization algorithm. In *IEEE Quantum Week 2021*, 2021.

[8] S. Bravyi and N. Klco. Mitigating device noise in qaoa via adaptive circuit compilation. *Physical Review Research*, 2022.

doi:10.6342/NTU202501430



[9] K. Bui and M. Pham. Particle swarm optimisation of variational quantum circuits. *Quantum Science and Technology*, 2024.

[10] D. Claudino, D. Lyakh, and A. McCaskey. Parallel quantum computing simulations via quantum accelerator platform virtualization. *arXiv preprint*, 2024.

[11] D. Coppersmith. An approximate fourier transform useful in quantum factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 536–545. IEEE, 1994.

[12] G. Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. *Physical Review A*, 2019.

[13] A. Doi and D. Takahashi. Quantum computing simulator on a heterogeneous hpc system. In *Proceedings of HPC Asia 2019*, pages 1–9, 2019.

[14] Y. Dong, C. Ho, Y. Wang, L. Cincio, and P. Coles. Parameter transfer for quantum approximate optimization of weighted max-cut. *arXiv preprint*, 2022.

[15] D. Egger, J. Mareček, and S. Woerner. Warm-starting quantum optimization. *Quantum*, 5:479, 2021.

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

[17] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475, 2001.

[18] M. M. Flood. The traveling-salesman problem. *Operations research*, 4(1):61–75, 1956.

doi:10.6342/NTU202501430

[19] L. Grover. A fast quantum mechanical algorithm for database search, 1996.

[20] G. Guerreschi and A. Matsuura. Accelerating variational quantum algorithms using machine learning. *Quantum Science and Technology*, 4(4):045012, 2019.

[21] S. Hadfield, Z. Wang, B. O'Gorman, E. Rieffel, D. Santoro, and S. Pakin. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Quantum Science and Technology*, 4(1):014004, 2019.

[22] C. Hsu, C. Wang, N. Hsu, C. Tu, and S. Hung. Towards scalable quantum circuit simulation via rdma. In *Proceedings of RACS' 23*, 2023.

[23] S. Imamura, M. Yamazaki, T. Honda, A. Kasagi, A. Tabuchi, H. Nakao, N. Fukumoto, and K. Nakashima. mpiqulacs: A distributed quantum computer simulator for a64fx-based cluster systems, 2022.

[24] T. Jones, A. Brown, I. Bush, and S. Benjamin. Quest and high-performance simulation of quantum computers. *Scientific Reports*, 9(1), 2019.

[25] T. Kadowaki and H. Nishimori. Quantum annealing: A new method for minimizing multidimensional functions. *Chemical Physics Letters*, 219(5):343–348, 1994.

[26] W. Lavrijsen, A. Tudor, J. Müller, C. Iancu, and W. de Jong. Classical optimizers for noisy intermediate-scale quantum devices. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 267–277, 2020.

[27] Y. Lin, C. Wang, C. Tu, and S. Hung. Towards optimizations of quantum circuit simulation for solving max-cut problems with qaoa. In *Proceedings of SAC' 24*, 2024.

doi:10.6342/NTU202501430

[28] J. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[29] L. Mitchell and R. Shaydulin. Genetic-algorithm assisted parameter optimisation for qaoa. In *HPEC 2023*, 2023.

[30] C. Z. Mooney. *Monte carlo simulation*. Number 116. Sage, 1997.

[31] T. O'Brien and B. Tarasinski. Noise-adaptive variational quantum algorithms with reduced measurement requirements. *npj Quantum Information*, 2022.

[32] A. Peruzzo, J. McClean, P. Shadbolt, M. Yung, X. Zhou, P. Love, A. Aspuru-Guzik, and J. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5, 2014.

[33] J. Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

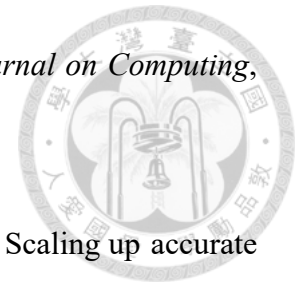
[34] C. Reeves. *Genetic algorithms*. Springer, 2003.

[35] R. Shaydulin, A. F. Izmaylov, Y.-H. Chen, and S. Bianco. Parameter setting in quantum approximate optimization of weighted maxcut. *Quantum Science and Technology*, 7(2):025024, 2022.

[36] R. Shaydulin, I. Safro, and J. Larson. Multistart methods for quantum approximate optimization. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, 2019.

[37] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

doi:10.6342/NTU202501430



[38] D. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[39] Y. Tsai, J. Jiang, and C. Jhang. Bit-slicing the hilbert space: Scaling up accurate quantum circuit simulation to a new level. *arXiv preprint*, 2020.

[40] P. J. Van Laarhoven, E. H. Aarts, P. J. van Laarhoven, and E. H. Aarts. *Simulated annealing*. Springer, 1987.

[41] H. Wang, Y. Ding, J. Gu, Y. Lin, D. Pan, F. Chong, and S. Han. Quantumnas: Noise-adaptive search for robust quantum circuits. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 692–708, 2022.

[42] 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*, pages 443–454, 2021.

[43] C. Zhang, H. Wang, Y. Li, Q. Liu, and Z. Chen. Uniq: A unified programming model for efficient quantum circuit simulation. In *Proceedings of SC 2022*, 2022.

[44] L. Zhou, S. Wang, S. Choi, H. Pichler, and M. Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 10(2):021067, 2020.

[45] S. Zhu, X. Luo, and Z. Sun. p-swap: Efficient depth expansion for qaoa. *Quantum*, 7:1104, 2023.

doi:10.6342/NTU202501430