Full State Quantum Circuit Simulation Beyond Memory Limit

Yilun Zhao[1,2], Yu Chen[1,2], He Li[3], Ying Wang[1,*], Kaiyan Chang[1,2], Bingmeng Wang[4], Bing Li[4] and Yinhe Han[1]
Research Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences[1]
University of Chinese Academy of Sciences[2], *Southeast University*[3], *Capital Normal University*[4]
Emails: {zhaoyilun22b,chenyu21b,wangying2009,yinhes}@ict.ac.cn, changkaiyan@live.com,
helix@seu.edu.cn,{2210502133,bing.li}@cnu.edu.cn

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323666

***Abstract*—Quantum circuit simulation (QCS) is essential in the noisy intermediate scale quantum (NISQ) era when real quantum computers are scarce. However, fully tracking the states of a quantum system in QCS is highly challenging due to the exponential memory growth that significantly limits the computational reach of classical systems for QCS. Though it is straightforward to leverage secondary storage to extend the scale of QCS, excessive data movement between memory and storage dominates the simulation time, making this solution unrealistic. To tackle this challenge, we identify an intrinsic property of QCS and implement an open-source framework to effectively reduce data movement by >116×. We evaluate the framework on various benchmarks and demonstrate 4× memory reduction with only <20% overhead. On a memory constrained system, we show that it extends the scale of QCS to 32 qubits (64 GB memory requirement) while existing simulators are bounded to 28 qubits (4 GB memory requirement). Our implementation can be accessed via https://github.com/Zhaoyilunnn/qdao.**

***Index Terms*—Quantum Computing, Design Methodology, Simulation**

I. Introduction

The rapid development of real quantum computers [1], [2] has sparked significant interest in both academic and industrial sectors. However, quantum computers are still in their prototype phase and we're still in the early noisy intermediate scale quantum (NISQ) [3] era. At present, access to real quantum computers is limited due to their scarcity. Although there are real machines such as IBM Quantum Service [4] and Azure Quantum [5], publicly available machines often contain only a few number of qubits. Meanwhile, researchers and developers worldwide are queuing up to access these machines, resulting in long wait times [6], [7]. To fuel rapid development and verification, large-scale classical simulation of quantum computing plays a crucial role for many different research purposes [8], [9], including quantum arithmetic [10], quantum machine learning [11], quantum chemistry [12], etc.

A quantum circuit composed of a series of quantum operations describes quantum computing [13]. To simulate quantum computing, we use quantum circuit simulation (QCS), which tracks the *state vector* of a quantum system as operations execute on it [14]. However, the exponential growth of the state vector size with the number of qubits makes it too large to be stored in CPU memory (DRAM). This limits the scale of QCS and prompts us to consider leveraging the secondary storage (SSD/HDD) for QCS. While this idea is plausible, excessive data movement between memory and storage becomes a bottleneck that occupies most of the simulation time (see Sec. II-B), making it far from being a practical solution. This is because a quantum operation requires to update the entire state vector before the next operation can be applied, as previously shown in numerous works [15]–[18]. Consequently, **the entire state vector, which grows exponentially with # qubits, must be traversed between memory and storage for each operation.**

In this paper, we theoretically prove that it is possible to apply *a subset of operations*, which we refer to as a *sub-circuit*, on a portion of the state vector before we apply these operations to the remaining part. Therefore, **each sub-circuit requires a full traversal of the state vector across memory and storage, rather than requiring a full traversal for each operation.** With this principle in mind, we design a lightweight and portable framework that reshapes storage-based QCS. By extensively evaluating our framework on various benchmarks, we show that it effectively reduces data movement. For example, on a readily available laptop, data movement is reduced by more than 116×, leading to successful full-state simulations of arbitrary quantum circuits consuming 4× less memory with <20% data movement overhead. Moreover, we show that our framework extends the scale of QCS to 32 qubits, while existing simulators are all bounded to 28 qubits. To summarize, the main contributions of this paper are as follows.

- **Theorem.** For the first time, we provide a theorem with rigorous proof to reveal that state vector traversal between memory and storage can be done at *sub-circuit-level* rather than *operation-level.*
- **Methodology.** We propose a novel storage-based simulation method based on the above theorem, which is complementary to other methods and can be seamlessly integrated with existing simulators to enhance their simulation capacity.
- **Implementation.** We implement a lightweight open-source framework, extensive evaluation shows that it effectively reduces data movement between memory and storage by >116×. On a memory-limited system, it extends the scale of QCS from 28 qubits to 32 qubits, making it a useful tool that eliminates concerns about system memory constraints for researchers and developers.

This work is supported in part by National Natural Science Foundation of China (NSFC) (62222411, 62204164) and Zhejiang Lab under Grants 2021PC0AC01.

[*] Corresponding author.

II. Background And Motivation

A. Quantum Circuit Simulation

1) State Vector: State vector is the most common and intuitive way to describe a quantum system and are widely used in QCS. A state vector describes the complex vector space (a.k.a., Hilbert space) associated to an isolated physical system [13]. The simplest quantum mechanical system: a *qubit*, has a two-dimensional state space, where $|0\rangle$ and $|1\rangle$ form an orthonormal basis. Then an arbitrary state vector can be written $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$. Generally, the state vector of a system with n qubits is:

$$|\psi\rangle = \alpha_0|0\dots00\rangle + \alpha_1|0\dots01\rangle + \dots + \alpha_{2^n-1}|1\dots11\rangle. \quad (1)$$

Quantum circuit is the model for describing sophisticated quantum computations, which is essentially a sequence of quantum operations, and QCS simulates how these operations manipulate the state vector. Quantum operations are described by unitary matrices. For example, an important single-qubit operation: the Hadamard operation is defined as $\mathrm{H} \equiv \frac{1}{\sqrt{2}}\left[\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right]$. Applying an H-operation on qubit 0 of a two qubit system gives us[1]

$$\begin{bmatrix}\alpha'_{00}\\ \alpha'_{01}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha_{00}\\ \alpha_{01}\end{bmatrix}, \quad (2)$$

$$\begin{bmatrix}\alpha'_{10}\\ \alpha'_{11}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha_{10}\\ \alpha_{11}\end{bmatrix}. \quad (3)$$

For an n-qubit system, applying an H operation on qubit j transforms the state amplitudes as [15]:

$$\begin{bmatrix}\alpha'_{\times\cdots\times 0_j \times\cdots\times}\\ \alpha'_{\times\cdots\times 1_j \times\cdots\times}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha_{\times\cdots\times 0_j \times\cdots\times}\\ \alpha_{\times\cdots\times 1_j \times\cdots\times}\end{bmatrix}, \quad (4)$$

where we use '×'[2] to indicate that the bits on the corresponding positions are the same. It is clear that applying a quantum operation requires an entire traversal of the whole state vector.

2) Graphical Models: Tensor network and decision diagram are two representative graphical models that have been proposed for QCS. Tensor network simulations are efficient in simulating quantum circuits by representing them as a network of tensors, with each tensor representing a specific quantum operation. The goal of tensor network-based QCS is to contract a tensor network into a single rank-0 tensor, i.e., a scalar, which represents a single state amplitude. This method is particularly suited for circuits where only a small number of amplitudes need to be tracked [12], [19]. Decision diagram (DD)-based simulation, on the other hand, is able to perform full state simulation. Consider a quantum system of n qubits $q_{n-1}, \dots, q_1, q_0$. The first 2^{n-1} entries of the state vector represent $q_{n-1} = |0\rangle$ and the remaining 2^{n-1} entries represent $q_{n-1} = |1\rangle$. A decision diagram represents this by a node labeled q_{n-1} connected to two successors q_{n-2}, representing the zero- and one-successors. This process is repeated recursively until sub-vectors of size 1 remain [9]. By exploiting the sparsity and similarity of quantum states, DD-based simulation are shown to be more memory- and time-efficient than state vectors [20]–[22].

3) Stabilizer Formalism: Stabilizer formalism is a mathematical framework for describing and simulating a *restricted class* of quantum circuits, known as stabilizer circuits [14], [23], [24]. A quantum gate is a *stabilizer gate* if it is produced from Clifford group $\{\mathrm{CNOT}, \mathrm{H}, \mathrm{S}\}$. A quantum circuit is called a *stabilizer circuit* if it is made of stabilizer gates applied on input state $|00\dots0\rangle$. The Gottesman–Knill theorem [25] states that there exists classical algorithm that simulates any stabilizer circuits in polynomial time.

4) Summary: In this work, our primary focus is on the full-state simulation of general quantum circuits. As a result, we will not discuss the stabilizer formalism and tensor network as they are beyond the scope of our research. While DD-based simulation and compression-assisted state vector simulation [26] leverage the redundancy of quantum states and quantum operations, they are not universally applicable since redundancy is not a global property of quantum circuit systems. This is mainly due to the presence of arbitrary single or two-qubit rotation gates, which can be defined as Eq. (5) based on OpenQASM specification [27]:

$$\mathrm{U}_3(\theta, \psi, \lambda) = \begin{bmatrix}\cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2}\\ e^{i\psi}\sin\frac{\theta}{2} & e^{i(\psi+\lambda)}\cos\frac{\theta}{2}\end{bmatrix}, \quad (5)$$

where θ, ψ, λ are rotation angles. Irregular distributions of state vectors can result from randomized rotation angles, as illustrated in Fig. 1, making DD/compression based solutions inefficient. In conclusion, simulating general quantum circuits with full-state accuracy is inherently challenging because the size of the state vector grows strictly following an $O(2^n)$ pattern (as shown in Eq. (1)), thus significantly limiting the scale of QCS and the selection of classical systems that can be used.

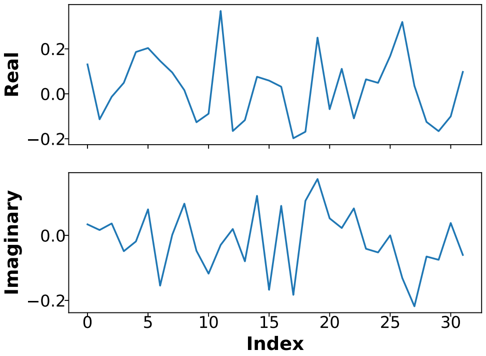


Fig. 1: State amplitudes distribution of a random 5-qubit circuit.

B. Motivation

To break the memory limit of full-state QCS, it is rather straightforward to utilize the large space of secondary storage

[1] Throughout this paper, higher qubit indices are more significant (little endian convention).

[2] For example, $\left[\begin{smallmatrix}\alpha_{\times 0}\\ \alpha_{\times 1}\end{smallmatrix}\right]$ represents either $\left[\begin{smallmatrix}\alpha_{00}\\ \alpha_{01}\end{smallmatrix}\right]$ or $\left[\begin{smallmatrix}\alpha_{10}\\ \alpha_{11}\end{smallmatrix}\right]$.

devices to store the entire state vector. Despite this solution, excessive data movement renders this method inefficient. This is because current simulators [28]–[31] all adhere to a *synchronized simulation paradigm* where **the entire state vector must be updated for each quantum operation before the next one is applied.** As a result, if we have g quantum operations, we must copy the state vector g times between memory and storage. To understand the impact of above principle, we conduct an experiment for a 30-qubit random quantum circuit with 633 quantum operations on system # 2 (See section IV-A2). For such a circuit, the size of state vector is $2^{30} \times 16$ Byte $= 16$ GB. Then we need to copy 633×16 GB ≈ 9.9 TB data between CPU memory and HDDs during simulaiton. After breaking down the simulation time into computation and data movement, we found 98.9% simulation time is spent on data movement and the total simulation time exceeds 2 days, making this simulation method extremely inefficient.

As synchronized simulation introduces significant data movement overhead, we propose to explore an *asynchronous simulation paradigm*. Specifically, we ask, **is it possible to apply the next operation before current operation updates the entire statevector?** If realized, this approach brings significant benefits. Firstly, we divide a state vector into multiple parts with each being able to be stored in memory. Assuming we can apply all operations to a part of state vector, move on to the next one, and continue until all parts updated, then we need to copy the complete state vector between memory and storage only once rather than g times.

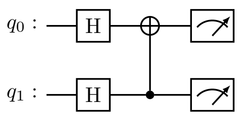


Fig. 2: A sample two-qubit quantum circuit.

Unfortunately, it is intrinsically challenging to implement asynchronous simulation. Consider a simple system as shown in Fig. 2, the state vector is $|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$. Applying an H gate on q_0 follows Eq. (2), (3). After computation of Eq. (2) is finished, can we continue to apply the second H gate to the first part, i.e., $[\alpha_{00}, \alpha_{01}]$? According to Eq. (4), the second H gate on qubit 1 updates the state vector as:

$$\begin{bmatrix}\alpha''_{00}\\ \alpha''_{10}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha'_{00}\\ \alpha'_{10}\end{bmatrix}, \tag{6}$$

$$\begin{bmatrix}\alpha''_{01}\\ \alpha''_{11}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha'_{01}\\ \alpha'_{11}\end{bmatrix}. \tag{7}$$

It is clear that the calculation of α''_{00} depends on both α'_{00} and α'_{10}, and similarly α''_{01} depends on α'_{01} and α'_{11}. Therefore, it is impossible to calculate the correct result of α''_{00} and α''_{01} without knowing α'_{10} and α'_{11}, i.e., we cannot apply the second H gate before the first one updates the entire state vector. While asynchronous approach may bring benefits, we see that it also poses significant challenges, which motivate our research.

III. Quantum Data Access Optimization

A. Sub-circuits Generation

In this paper, we propose a method to implement asynchronous simulation while ensuring the correctness of results. The key insight behind is that, **while each operation typically corresponds to the traversal of an entire state vector, we argue that each *sub-circuit* (see Definition 2) corresponds to a single complete traversal between memory and storage.** This is essentially an intrinsic property supported by the following theorem.

Definition 1. *Given a quantum circuit $\mathcal{Q}$ with n qubits, where the orthonormal computational basis set of the state space is denoted as $\{|i\rangle\}$, with $i \in [0, 2^n - 1]$, the state vector $|\psi\rangle$ of $\mathcal{Q}$ is defined as:*

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \tag{8}$$

where α_i is the state amplitude associated with the basis state $|i\rangle$.

Definition 2. *A sub-circuit $\mathcal{S}$ of $\mathcal{Q}$ is a series of operations on m qubits of $\mathcal{Q}$: $\{q_{s_0}, q_{s_1}, \ldots, q_{s_{m-1}}\}$, where $\{s_i\}_{i-0}^{m-1}$ denotes the indices of qubits in $\mathcal{Q}$. The overall operator of $\mathcal{S}$ is a $2^m \times 2^m$ unitary matrix denoted as $\mathrm{U}_{\mathcal{S}}$.*

Theorem 1. *$\forall\ \mathcal{S}$, $\exists$ a partition that divides the sets $\{\alpha_i\}$ and $\{|i\rangle\}$ into subsets, denoted by $\{\sigma_k\}_{k=0}^{2^{n-m}-1}$ and $\{v_k\}_{k=0}^{2^{n-m}-1}$ respectively.* ***These subsets are all of size 2^m and are disjoint,*** *such that applying $\mathcal{S}$ on $\mathcal{Q}$ transforms the state vector of $\mathcal{Q}$ by 2^{n-m}* ***independent*** *transformations:*

$$|\psi'\rangle = \sum_{k=0}^{2^{n-m}-1}\left(\mathrm{U}_{\mathcal{S}} \cdot \sum_{j=0}^{2^m-1} \sigma_k(j) v_k(j)\right), \tag{9}$$

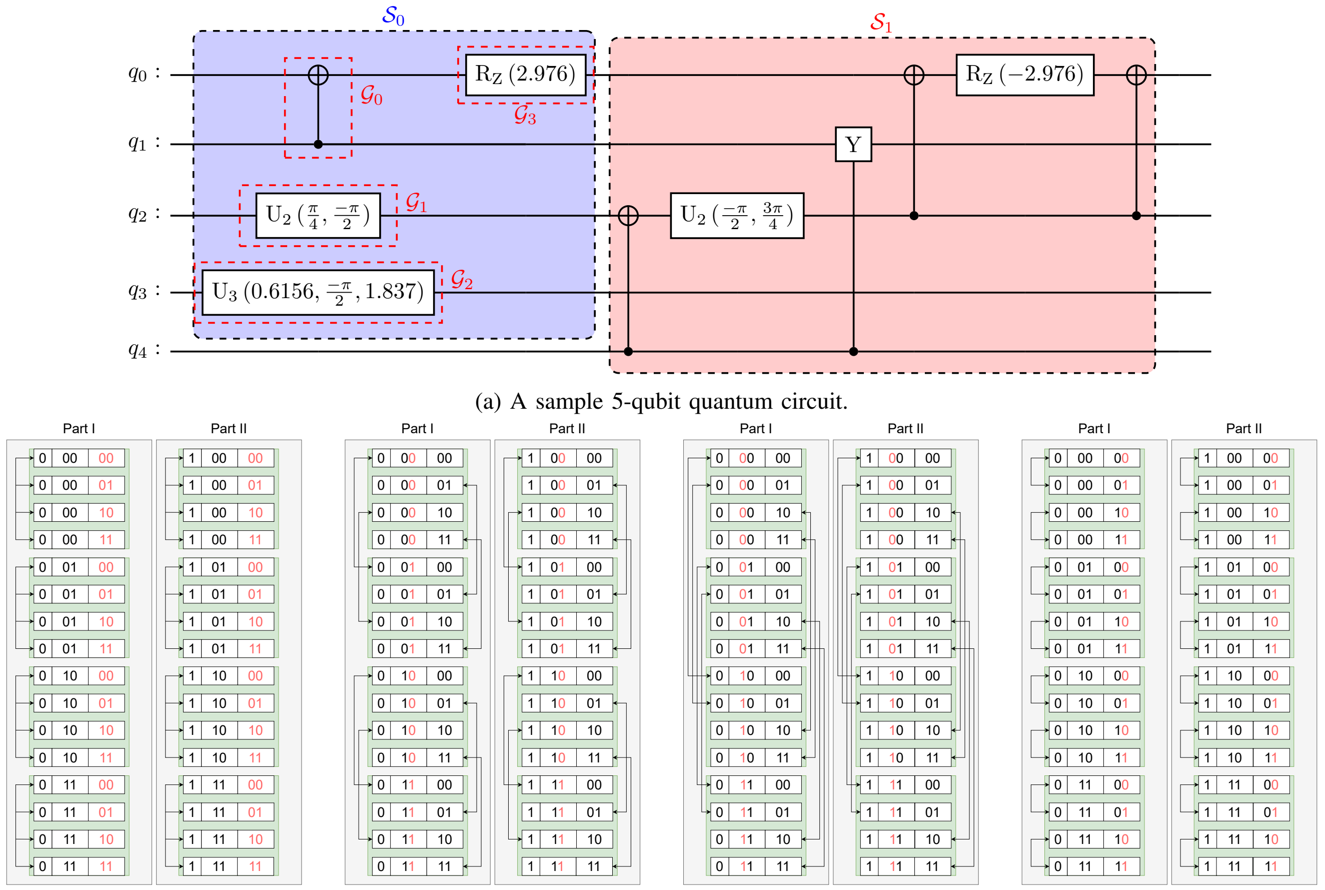
Proof.

For $\mathcal{S}$ operating on m qubits, we denote the orthonormal basis of m-qubit system as $|j\rangle$, where $j \in [0, 2^m - 1]$, and the basis of remaining $(n-m)$-qubits as $|k\rangle$, where $k \in [0, 2^{n-m} - 1]$. Then we have

$$|\psi\rangle = \sum_{k=0}^{2^{n-m}-1} \sum_{j=0}^{2^m-1} \alpha_{j,k} |j\rangle \otimes |k\rangle. \tag{10}$$

Thus the operations of $\mathcal{S}$ applying on $|\psi\rangle$ gives us

$$\begin{aligned}
|\psi'\rangle &= (\mathrm{U}_{\mathcal{S}} \otimes \mathrm{I}) \cdot |\psi\rangle \\
&= (\mathrm{U}_{\mathcal{S}} \otimes \mathrm{I}) \cdot \left(\sum_{k=0}^{2^{n-m}-1} \sum_{j=0}^{2^m-1} \alpha_{j,k} |j\rangle \otimes |k\rangle\right) \\
&= \sum_{k=0}^{2^{n-m}-1}\left((\mathrm{U}_{\mathcal{S}} \otimes \mathrm{I}) \cdot \left(\sum_{j=0}^{2^m-1} \alpha_{j,k} |j\rangle\right) \otimes |k\rangle\right) \\
&= \sum_{k=0}^{2^{n-m}-1}\left(\left(\mathrm{U}_{\mathcal{S}} \cdot \sum_{j=0}^{2^m-1} \alpha_{j,k} |j\rangle\right) \otimes \mathrm{I} \cdot |k\rangle\right) \\
&= \sum_{k=0}^{2^{n-m}-1}\left(\left(\mathrm{U}_{\mathcal{S}} \cdot \sum_{j=0}^{2^m-1} \alpha_{j,k} |j\rangle\right) \otimes |k\rangle\right).
\end{aligned} \tag{11}$$



(a) A sample 5-qubit quantum circuit.

(b) Data access pattern of $\mathcal{G}_0$. (c) Data access pattern of $\mathcal{G}_1$. (d) Data access pattern of $\mathcal{G}_2$. (e) Data access pattern of $\mathcal{G}_3$.

Fig. 3: (a) shows a sample quantum circuit and sub-circuits. (b)-(e) depict the data access pattern when applying operations, where the $2^5 = 32$ state amplitudes are denoted by its binary indices and are arranged from top to bottom and from left to right. The corresponding qubit indices of an operation are highlighted. Connected arrows denote a group of amplitudes that must be calculated together.

Defining $V_k \equiv \left(\mathrm{U}_{\mathcal{S}} \cdot \sum_{j=0}^{2^m-1} \alpha_{j,k}|j\rangle\right) \otimes |k\rangle$, we see that $\forall\ p, q \in [0, 2^{n-m}-1], p \neq q$, calculating V_p is independent of calculating V_q. Let $\sigma_k \equiv \{\alpha_{j,k}\}_{j=0}^{2^m}$, $v_k \equiv \{|j\rangle \otimes |k\rangle\}_{j=0}^{2^m}$, we see that the theorem has been proved.

Example 1. *The state vector of a 5-qubit circuit is $|\psi\rangle = \sum_{i=0}^{2^5} \alpha_i |i\rangle$. Given a sub-circuit acting on q_0, q_1, q_3 (indices are bold), for $k = 2$, i.e., $|k\rangle = |10\rangle$, we have*

$$\begin{aligned} v_2 &\equiv \{|000\rangle \otimes |10\rangle, |001\rangle \otimes |10\rangle, \ldots, |111\rangle \otimes |10\rangle \\ &= \{|\mathbf{1}0\mathbf{000}\rangle, |\mathbf{1}0\mathbf{001}\rangle, \ldots, |\mathbf{1}1\mathbf{011}\rangle\}, \end{aligned} \tag{12}$$

$$V_2 \equiv \mathrm{U}_{\mathcal{S}} \cdot (\alpha_{\mathbf{1}0\mathbf{000}}|\mathbf{1}0\mathbf{000}\rangle + \alpha_{\mathbf{1}0\mathbf{001}}|\mathbf{1}0\mathbf{001}\rangle + \cdots + \alpha_{\mathbf{1}1\mathbf{011}}|\mathbf{1}1\mathbf{011}\rangle)\,. \tag{13}$$

Similar to Eq. (4), we use the same symbol '×' to indicate the bits on the corresponding positions are the same. Then we can denote any of $\{V_k\}_{k=0}^{3}$ as:

$$\begin{aligned} V_k \equiv &\mathrm{U}_{\mathcal{S}} \cdot (\alpha_{\times\mathbf{0}\times\mathbf{00}}|\times\mathbf{0}\times\mathbf{00}\rangle + \alpha_{\times\mathbf{0}\times\mathbf{01}}|\times\mathbf{0}\times\mathbf{01}\rangle + \\ &\cdots + \alpha_{\times\mathbf{1}\times\mathbf{11}}|\times\mathbf{1}\times\mathbf{11}\rangle\,). \end{aligned} \tag{14}$$

Corollary 1. *The transformed state amplitudes are denoted as $\{\alpha'_i\}$, we have $\sum_{j=0}^{2^m-1} \alpha'_{j,k}|j\rangle \otimes |k\rangle = \left(\mathrm{U}_{\mathcal{S}} \cdot \sum_{j=0}^{2^m-1} \alpha_{j,k}|j\rangle\right) \otimes |k\rangle$. Based on Eq. (14), we have*

$$\begin{bmatrix} \alpha'_{\times\cdots\times 0_{q_{s_{m-1}}}\times\cdots\times 0_{q_{s_1}}\times\cdots\times 0_{q_{s_0}}\times\cdots\times} \\ \alpha'_{\times\cdots\times 0_{q_{s_{m-1}}}\times\cdots\times 0_{q_{s_1}}\times\cdots\times 1_{q_{s_0}}\times\cdots\times} \\ \vdots \\ \alpha'_{\times\cdots\times 1_{q_{s_{m-1}}}\times\cdots\times 1_{q_{s_1}}\times\cdots\times 1_{q_{s_0}}\times\cdots\times} \end{bmatrix} = \mathrm{U}_{\mathcal{S}} \left[\left.\begin{bmatrix} \alpha_{\times\cdots\times 0_{q_{s_{m-1}}}\times\cdots\times 0_{q_{s_1}}\times\cdots\times 0_{q_{s_0}}\times\cdots\times} \\ \alpha_{\times\cdots\times 0_{q_{s_{m-1}}}\times\cdots\times 0_{q_{s_1}}\times\cdots\times 1_{q_{s_0}}\times\cdots\times} \\ \vdots \\ \underbrace{\alpha_{\times\cdots\times 1_{q_{s_{m-1}}}\times\cdots\times 1_{q_{s_1}}\times\cdots\times 1_{q_{s_0}}\times\cdots\times}}_{n\text{-bit binary indices}} \end{bmatrix}\right\}_{2^m}\right] . \tag{15}$$

Example 2. *Given $\mathcal{Q}$ with $n = 5$ qubits, for $m = 4$, we have $\mathcal{S}_0$ and $\mathcal{S}_1$ (Fig. 3a). Fig. 3b-3e illustrate how we apply each of the operations of $\mathcal{S}_0$ on the state vector, where the elements are arranged from top to bottom and from left to right. Connected arrows means the corresponding state amplitudes are inter-dependent. Clearly, it is shown that there's no dependency between part I and part II when applying any* *of $\{\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$. Therefore, applying $\mathcal{S}_0$ on $\mathcal{Q}$ transforms the state vector by $2^{5-4} = 2$ independent transformations.*

Definition 3. *When applying $\mathcal{S}$ on $\mathcal{Q}$, compute units are the subsets of state amplitudes $\{\sigma_k\}_{k=0}^{2^{n-m}-1}$ (see Theorem 1).*

State amplitudes within a compute unit (Definition 3) are inter-dependent, while there's no dependency between different compute units. Consequently, we are able to store one compute unit in memory, while the entire state vector persists in secondary storage. For each sub-circuit, we sequentially simulate it 2^{n-m} times on all compute units, and thus the entire state vector is copied only once between memory and secondary storage. Therefore, we have the following corollary.

Corollary 2. *Given a quantum circuit $\mathcal{Q}$ with n-qubit and g operations with the ith operation applying on x_i qubits, where $i \in [0, g-1]$. $\forall\ m \in [\max(x_i), n]$, $\exists\ l \in [1, g]$ such that $\mathcal{Q}$ can be decomposed into l sub-circuits $\{\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{l-1}\}$. Then we can achieve deterministic memory reduction and data movement reduction:*

$$\textit{Memory Reduction} = \frac{\textit{sizeof(Original State Vector)}}{\textit{sizeof(Compute Unit)}} = 2^{n-m}, \tag{16}$$

$$\textit{Data Movement Reduction} = \frac{\textit{\# Operations}}{\textit{\# Sub-circuits}} = \frac{g}{l}. \tag{17}$$

Example 3. *For the sample circuit depicted in Fig. 3a, we have $n = 5$, $g = 10$. Given $m = 4$, it is possible to decompose the circuit into $l = 2$ sub-circuits, i.e., the blue ($\mathcal{S}_0$) and red ($\mathcal{S}_1$) blocks. Then we are able to simulate the circuit using $2^{5-4} = 2\times$ less memory with $10/2 = 5\times$ data movement reduction.*

B. Compute Unit Aggregation and Decomposition

While the effectiveness of our solution has been proven in principle, obstacles still exist in its implementation. From Eq. (15), we observe that the state amplitudes within a compute unit may be discretely distributed in the original state vector. However, we need to store the state amplitudes densely in a 2^m-dimensional continuous vector space.

Example 4. *$\mathcal{S}_0$ and $\mathcal{S}_1$ are respectively depicted in Fig. 4a and Fig. 4b. According to Eq. (15), the inter-dependent state amplitudes, i.e., the compute units of $\mathcal{S}_0$ are $[\alpha_{\times 0000}, \alpha_{\times 0001}, \ldots, \alpha_{\times 1111}]$. Each compute unit corresponds to one continuous sub-state vector, i.e.,*

$$\begin{aligned}\textit{Compute Unit}_0 &\leftarrow [\alpha_{00000}, \alpha_{00001}, \ldots, \alpha_{01111}]\\ \textit{Compute Unit}_1 &\leftarrow [\alpha_{10000}, \alpha_{10001}, \ldots, \alpha_{11111}].\end{aligned} \tag{18}$$

In contrast, compute units of $\mathcal{S}_1$ are $[\alpha_{0\times 000}, \alpha_{0\times 001}, \ldots,$ $\alpha_{1\times 111}]$. A little thought shows that each compute unit corresponds to two separate continuous sub-state vectors, i.e.,

$$\begin{aligned}\textit{Compute Unit}_0 &\leftarrow [\alpha_{00000}, \ldots, \alpha_{00111}] + [\alpha_{10000}, \ldots, \alpha_{10111}]\\ \textit{Compute Unit}_1 &\leftarrow [\alpha_{01000}, \ldots, \alpha_{01111}] + [\alpha_{11000}, \ldots, \alpha_{11111}].\end{aligned} \tag{19}$$

For $\mathcal{S}_1$, we need to aggregate and decompose discrete sub-vectors into a contiguous vector. Besides, the original indices are re-mapped to $[\times 0000', \times 0001', \ldots, \times 1111']$ in compute units. Thus we use the virtual indices (highlighted in Fig. 4b) $\{q'_0, q'_1, q'_2, q'_3\}$ when simulating $\mathcal{S}_1$ on compute units.

To realize efficient aggregation and decomposition, we introduce *storage unit*, which is the primitive sub-state vector that will be copied between compute devices and storage devices.

Definition 4. *$\forall\ t \in [0, m)$, the ith storage unit is a continuous sub-state vector $[\alpha_{i\cdot 2^t}, \alpha_{i\cdot 2^t+1}, \ldots, \alpha_{(i+1)\cdot 2^t-1}]$. The number of storage units is 2^{n-t}.*

Example 5. *Again, we take Fig. 3a as an example. The aggregation and decomposition processes of $\mathcal{S}_0$ and $\mathcal{S}_1$ are illustrated in Fig. 4c-4d. We set $t = 2$ and thus we have $2^{5-2} = 8$ storage units. For $\mathcal{S}_0$, the first and second 4 storage units are aggregated as compute units. And for $\mathcal{S}_1$, storage units $\{0, 1, 4, 5\}$ and $\{2, 3, 6, 7\}$ are aggregated as compute units. It can be seen that Eq. (18) and Eq. (19) are respected.*

C. Implementation

Algorithm 1: Sub-circuits generation.

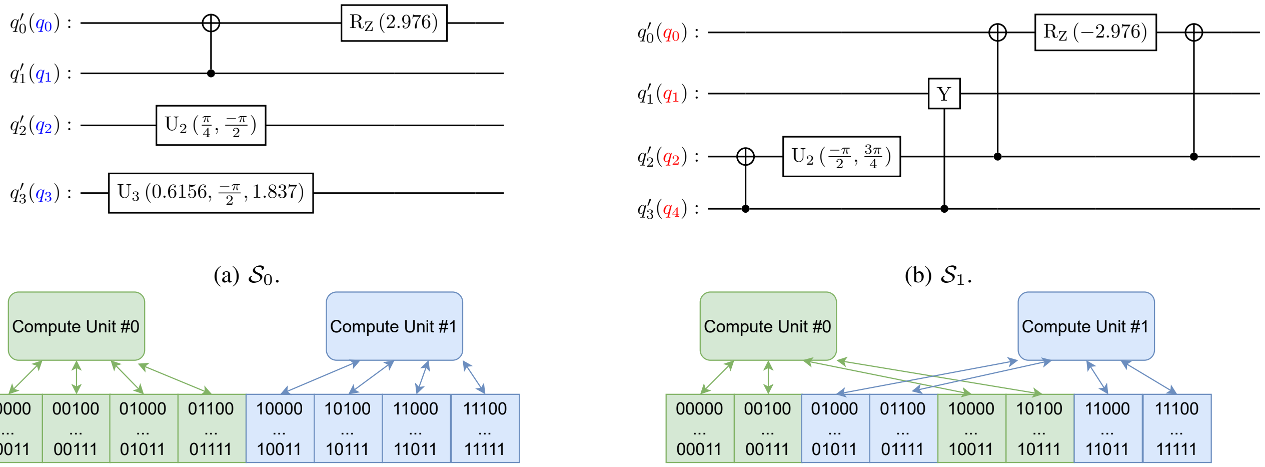
```
   Input  : $\mathcal{Q}$ // Original Quantum Circuit
   Input  : $t$ // # Qubits in a storage unit
   Input  : $m$ // # Qubits in a sub-circuit
   Output: $circ\_list$ // List of generated sub-circuits.
   // List of operations in current sub-circuit.
1  $op\_list = []$
   // Set of qubits that current sub-circuit acts on.
2  $q\_set = \{\}$
3  for $op$ in $\mathcal{Q}$.operations() do
       // Set of qubits that current operation acts on.
4      $op\_q\_set = \{\}$
5      for $q$ in $op$.qubits() do
6          if $q$.index() $\geq t$ then
7              $q\_set$.add($q$)
8      if len($q\_set \cup op\_q\_set$) $\leq (m - t)$ then
9          $q\_set = q\_set \cup op\_q\_set$
10         $op\_list$.append($op$)
11     else
12         $\mathcal{S}$ = gen_sub_circ($op\_list$)
13         $circ\_list$.append($\mathcal{S}$)
14         $op\_list = [op]$
15         $q\_set = op\_q\_set$
16 if $op\_list$ then
17     $\mathcal{S}$ = gen_sub_circ($op\_list$)
18     $circ\_list$.append($\mathcal{S}$)
19 return $circ\_list$
```

To this end, we leverage Corollary 2 to implement a **Q**uantum **D**ata **A**ccess **O**ptimization framework, named QDAO (Fig. 5). Our framework consists of two stages.

1) Stage I: Given an n-qubit quantum circuit, we decompose it into a list of m-qubit sub-circuits, which we describe in Alg. 1. We track the contained operations (*op_list*, line 1) and involved qubits (*q_set*, line 2) of each sub-circuit. While traversing the operations of $\mathcal{Q}$, we greedily add operations to *op_list* until the involved qubits exceed m and then we move forward to next sub-circuit. Note that, we only track the qubits outside the storage unit (line 5-7).

Example 6. *For $m = 5, t = 2$, if we find a sequence of operations on $\{q_0, q_2, q_3, q_5\}$, we consider it as a 5-qubit sub-circuit even if it actually acts on 4 qubits.*

We observe that m and t have an impact on the generation of a sub-circuit. It is left to the users to control these parameters based on their specific requirements. A detailed analysis can be found in Sec. IV-C.



(a) $\mathcal{S}_0$.

(b) $\mathcal{S}_1$.

(c) Compute unit aggregation and decomposition of $\mathcal{S}_0$.

(d) Compute unit aggregation and decomposition of $\mathcal{S}_1$

Fig. 4: (a),(b) depict $\mathcal{S}_0$ and $\mathcal{S}_1$. The labels highlighted in distinct colors represent the positions of the qubits within the original circuit. (c),(d) depict the compute unit aggregation and decomposition processes of $\mathcal{S}_0$ and $\mathcal{S}_1$. The state vector is partitioned into 8 storage units (represented by squares). Compute units are represented by rounded rectangles. The corresponding storage units of a compute unit are depicted using the same color.

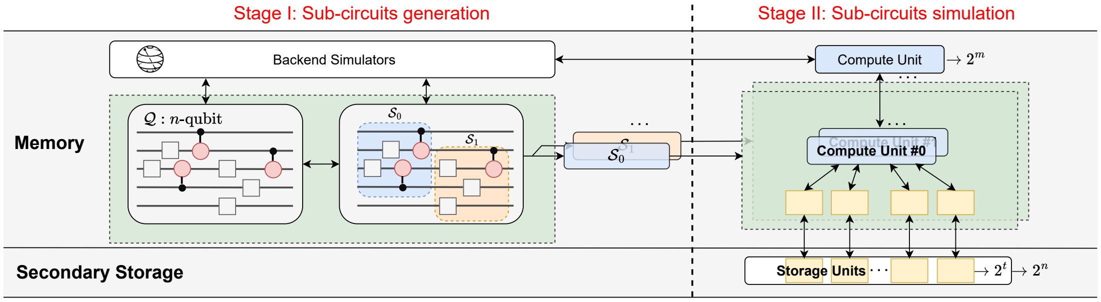


Fig. 5: An overview of QDAO framework.

Algorithm 2: Sub-circuits simulation.

```
  Input : $circ\_list$ // List of sub-circuits from stage I
  Input : $t$ // # Qubits in a storage unit
  Input : $m$ // # Qubits in a sub-circuit
  Input : $n$ // # Qubits in original circuit
1 for $\mathcal{S}$ in $circ\_list$ do
2     for $cu\_id$ in range($2^{n-m}$) do
          // Aggregate storage units into a compute unit
             (cu) from storage
3         $cu$ = aggregate($\mathcal{S}, t, m, cu\_id$)
          // Simulate a $\mathcal{S}$ on cu
4         $cu$ = simulate($\mathcal{S}, input = cu$)
          // Decompose cu into storage units and save
             back to storage
5         decompose($\mathcal{S}, sv, t, m, cu\_id$)
```

2) Stage II: We repeatedly simulate each sub-circuit for 2^{n-m} times as described in Alg. 2. In each simulation, we aggregate storage units into a compute unit, simulate the sub-circuit on the compute unit, decompose the compute unit into storage units, and save them back to secondary storage. Both aggregation and decomposition utilize the same algorithm as calculating the indexes of involved state amplitudes when applying an operation, which can be found in existing simulators.[3]

Example 7. *As indicated in example 5, given $\mathcal{S}_1$ on qubits $\{q_0, q_1, q_2, q_4\}$, $m = 4, t = 2$, and $cu\_id = 0$ (line 2 in Alg. 2), the involved storage unit indexes are $\{0, 1, 4, 5\}$, which is exactly the first group of indexes when applying a two-qubit operation on $\{q_0, q_2\}$.*

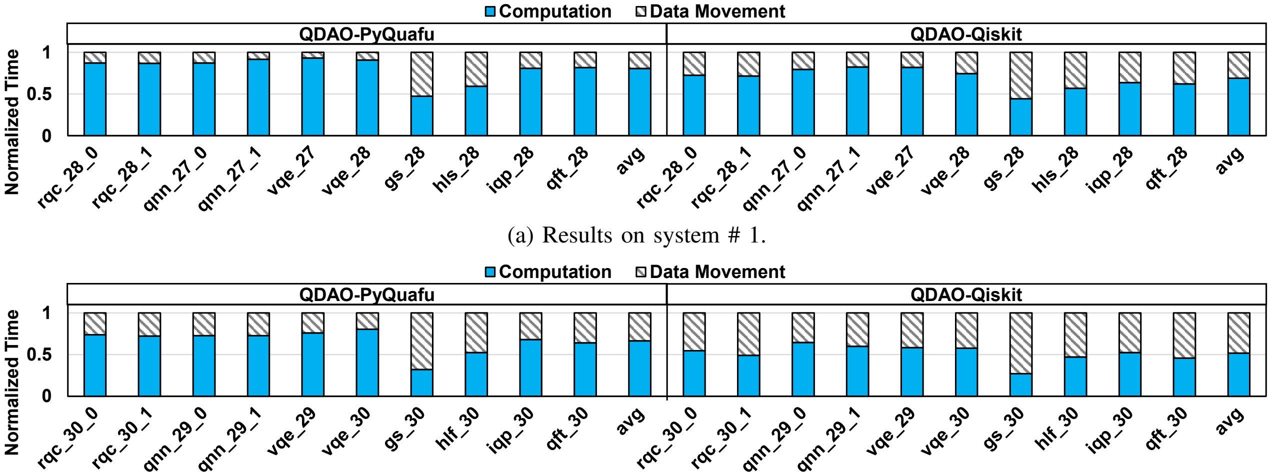
Moreover, it is worthy to mention that the functions (`simulate()`, `operations()`, `gen_sub_circ()`, etc.) in Alg. 1, 2 can be easily implemented on top of most of existing simulators, making QDAO a portable tool for users to enhance the simulation capacity on memory-limited systems.

IV. Evaluation

A. Experiment Setups

1) Benchmarks: In this work, we primarily focus on two types of quantum circuits: *random quantum circuit* (RQC) [32], [33] and *variational quantum circuit* (VQC) [34], [35] to demonstrate the universal effectiveness of QDAO, both are widely studied and play important roles in NISQ era. RQC

[3] https://github.com/Qiskit/qiskit-aer/blob/main/src/simulators/statevector/indexes.hpp



(a) Results on system # 1.

(b) Results on system # 2.

Fig. 6: Execution time breakdown of QDAO.

Benchmark	# Sub-circuits (l)	# Operations (g)	Data Movement Reduction
System# 1			
rqc_28_0	8	528	66.0×
rqc_28_1	10	573	57.3×
qnn_27_0	2	161	80.5×
qnn_27_1	2	161	80.5×
vqe_27	2	618	309.0×
vqe_28	2	641	320.5×
gs_28	2	55	27.5×
hls_28	2	84	42.0×
iqp_28	4	377	94.3×
qft_28	5	420	84.0×
Average	3.9	361.8	116.2×
System# 2			
rqc_30_0	8	589	73.6×
rqc_30_1	10	633	63.3×
qnn_29_0	2	174	87.0×
qnn_29_1	2	174	87.0×
vqe_29	2	664	332.0×
vqe_30	2	687	343.5×
gs_30	2	60	30.0×
hls_30	2	90	45.0×
iqp_30	3	396	132.0×
qft_30	5	480	96.0×
Average	3.8	394.7	128.9×

TABLE I: Results of sub-circuit generation.

is the key component of the random circuit sampling task designed by Google to demonstrate quantum supremacy [32]. And VQC serves as the quantum part of the most promising hybrid quantum-classical algorithms in NISQ era [36]. In our evaluation, we select two representative VQCs: quantum neural network (QNN) [11] circuits and variational quantum eigensolver (VQE) circuits [35][4]. More importantly, since RQCs are randomly generated and rotation angles are trainable parameters in VQCs, both of them contain amounts of randomized rotation gates. Therefore, these benchmarks are most suitable to demonstrate the effectiveness of our solution (see Sec. II-A). In addition, we select several commonly used subroutine benchmarks[5] including quantum Fourier transform (QFT), graph state (GS) circuit, hidden linear function (HLF) and instantaneous quantum polynomial (IQP) circuit to show the robustness of QDAO.

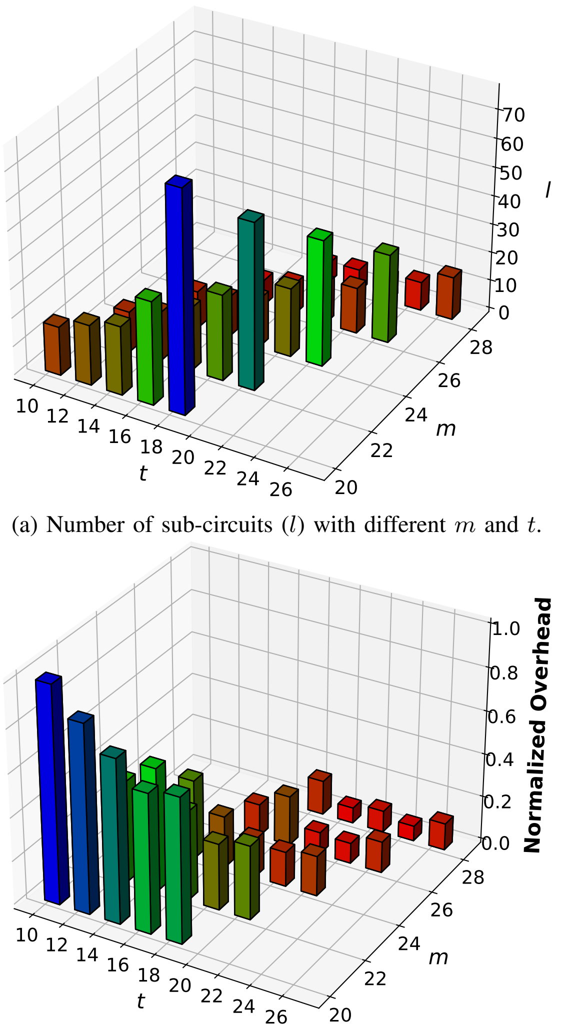
2) System Configurations: Our evaluation is done on two different classical computer systems. System # 1 is a laptop MacBook with a 2.6 GHz 6-Core Intel Core i7 processor and a 512 GB SSD. System # 2 is a Linux server with two 2.2 GHz 10-core Intel Xeon E5-2630 processors and 51 TB HDDs. We evaluate QDAO on two pilot supported backends: IBM Qiskit [28] and PyQuafu [31], as they can easily implement required APIs (see Sec. III-C2). More backend simulators will be supported in the future.

B. Overall Performance

In this section, we set $m = n - 2$, (i.e., we use $4\times$ less memory, see Corollary 1 in Sec. III-A), and $t = n - 8$. The impact of m and t will be analyzed later in Sec. IV-C. Firstly, we run sub-circuits generation for all benchmarks. Table I summarizes # operations (g), # sub-circuits (l), and the corresponding data movement reduction ratios (Eq. (17)). For benchmarks used on system # 1 and # 2, the average reduction is $116.2\times$ and $128.9\times$ respectively. Then we run simulations for all benchmarks. Fig. 6 breaks down the normalized simulation time into 1) computation on CPU and 2) data movement between CPU memory and secondary storage. The results can be concluded in three folds: 1) Data movement is no longer the performance bottleneck. On average, the data movement overhead of QDAO-PyQuafu is 19.3% and 33.5% on system # 1 and # 2, respectively, making secondary storage-based simulation paradigm a practical solution. 2) The proportion of computation and data movement depends on the performance of backend simulators and storage devices. For QDAO-Qiskit, computation occupies 60.3% of simulation on average across two systems, while computation of QDAO-PyQuafu occupies 73.5%. Also, we observe that the averaged proportion of data movement (25.2%) on system # 1 is lower than system # 2 (40.9%). 3) QDAO is highly effective for RQCs and VQCs. For RQCs and VQCs, the averaged data movement overhead is 25.3%. As a comparison, the averaged overhead of subroutines is 44.7%.

[4] VQC benchmarks are gathered from QASMBench [37].

[5] https://github.com/Qiskit/qiskit-terra/tree/main/qiskit/circuit/library



(a) Number of sub-circuits (l) with different m and t.

(b) Normalized data movement overhead (i.e., the shaded part in Fig. 6) with different m and t.

Fig. 7: Impact of m and t on # sub-circuits (l) and data movement overhead.

C. Impact of Parameter Settings on Performance

As indicated in Sec. III-C: Alg. 1, l depends on m and t. To understand the relationship between l and (m, t), we conduct experiments using a 30-qubit RQC. As shown in Fig. 7a, l can be minimized by setting larger m and smaller t. The reason behind can be intuitively illustrated as follows. Assume we set $m = n$, then we only have one sub-circuit which is equivalent to the original one. Moreover, setting smaller t leaves less restrictions (see Alg. 1) on generating a sub-circuit and thus leads to smaller l. According to Corollary 2, smaller l leads to larger data movement reduction, however, it does not necessarily result in less overhead and thus better performance. As depicted in Fig. 7b, We can see that although smaller t corresponds to smaller l, it can result in higher data movement overhead. For example, given $m = 26$, the best performance is achieved by setting $t = 22$ rather than $t = 18$. This is because smaller t results in smaller size of a storage unit. For example, when setting $t = 10$, there are $2^{n-t} = 2^{30-10} = 1048576$ storage units, each with a size of 16 KB. Transmitting such a huge amount of small files between secondary storage and memory incurs significant performance degradation, which can be avoided by setting $t \geq 20$ (16 MB storage unit) according to our evaluation. Moreover, the selection of m depends on the requirement of users, setting smaller m reduces memory requirement but sacrifices performance.

D. Comparison with Other Simulators

QDAO can be integrated in existing simulators to extend the scale of QCS on memory-constrained systems. We conduct experiments using VQE benchmarks on system # 1 (8 GB memory) and compare QDAO with existing simulators including IBM Qiskit [28], Google Cirq [29] and PyQuafu [31]. As shown in Fig. (8), existing simulators are bounded to 28 qubits (4 GB memory), while QDAO scales to 32 qubits (64 GB memory). Note that 29 qubits actually consumes >8 GB memory as we need to store additional parameters (e.g., operation matrices). As a result, out-of-memory (OOM) issue occurs from 29 qubits.

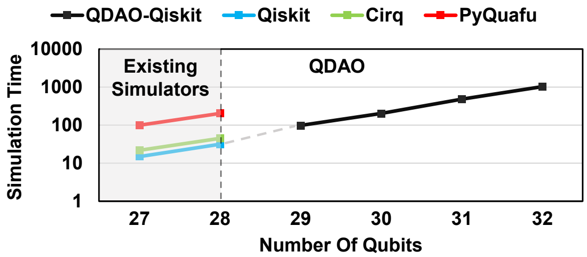


Fig. 8: Comparison of QDAO with existing simulators.

Besides, we assess DDSIM[6] [20]. We find that it simulates a 32-qubit VQE circuit for over 48 hours with growing memory consumption and eventually encounters OOM, confirming that exploiting redundancy is not effective for arbitrary circuits (as explained in Sec. II-A4).

V. Conclusion and Future Directions

In this work, we present QDAO, an open-source framework that effectively utilizes secondary storage to alleviate the memory bottleneck of QCS. Our innovation lies in demonstrating that state vector movement can be performed at the sub-circuit level rather than the operation-level, and we provide a theorem to support this claim. By evaluating QDAO on RQCs, VQCs, and subroutine circuits, we observe a remarkable reduction in data movement between memory and storage (>116×) compared to vanilla operation-level implementation. Additionally, we compare QDAO with state-of-the-art simulators and demonstrate its effectiveness to extend the scale of QCS on memory-limited systems. In future work, we aim to optimize QDAO for large-scale supercomputing environments and assess its potential for breaking classical computers' maximum simulation capacity.

[6] https://github.com/cda-tum/ddsim

References

[1] IBM. (2022) Expanding the IBM quantum roadmap to anticipate the future of quantum-centric supercomputing. [Online]. Available: https://research.ibm.com/blog/ibm-quantum-roadmap-2025

[2] ——. (2022) IBM unveils 400 qubit-plus quantum processor and next-generation IBM quantum system two. [Online]. Available: https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two

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

[4] IBM, "IBM quantum." [Online]. Available: https://quantum-computing.ibm.com/

[5] Microsoft, "Azure quantum cloud service." [Online]. Available: https://azure.microsoft.com/en-us/products/quantum

[6] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, "A case for multi-programming quantum computers," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 291–303.

[7] L. Liu and X. Dou, "Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 167–178.

[8] T. Häner and D. S. Steiger, "0.5 petabyte simulation of a 45-qubit quantum circuit," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126947

[9] R. Wille, L. Burgholzer, S. Hillmich, T. Grurl, A. Ploier, and T. Peham, "The basis of design tools for quantum computing: arrays, decision diagrams, tensor networks, and zx-calculus," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1367–1370.

[10] H. Li, J. Liang, H. Fan, and Y. Tang, "Design space exploration for efficient quantum most-significant digit-first arithmetic," *IEEE Transactions on Computers*, 2022.

[11] Z. Hu, P. Dong, Z. Wang, Y. Lin, Y. Wang, and W. Jiang, "Quantum neural network compression," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[12] 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 *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2022, pp. 175–188.

[13] M. A. Nielsen and I. L. Chuang, "Quantum computation and quantum information," 2010.

[14] Y. Ding and F. T. Chong, "Quantum computer systems: Research for noisy intermediate-scale quantum computers," *Synthesis Lectures on Computer Architecture*, vol. 15, no. 2, pp. 1–227, 2020.

[15] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, T. Lippert, H. Watanabe, and N. Ito, "Massively parallel quantum computer simulator," *Computer Physics Communications*, vol. 176, no. 2, pp. 121–136, 2007.

[16] A. Fatima and I. L. Markov, "Faster schrödinger-style simulation of quantum circuits," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 194–207.

[17] J. Doi, H. Takahashi, R. Raymond, T. Imamichi, and H. Horii, "Quantum computing simulator on a heterogenous hpc system," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 85–93.

[18] Y. Zhao, Y. Guo, Y. Yao, A. Dumi, D. M. Mulvey, S. Upadhyay, Y. Zhang, K. D. Jordan, J. Yang, and X. Tang, "Q-gpu: A recipe of optimizations for quantum circuit simulation using gpus," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 726–740.

[19] Y. Liu, X. Liu, F. Li, H. Fu, Y. Yang, J. Song, P. Zhao, Z. Wang, D. Peng, H. Chen *et al.*, "Closing the" quantum supremacy" gap: achieving real-time simulation of a random quantum circuit using a new sunway supercomputer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–12.

[20] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2018.

[21] S. Hillmich, R. Kueng, I. L. Markov, and R. Wille, "As accurate as needed, as efficient as possible: Approximations in dd-based quantum circuit simulation," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 188–193.

[22] 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.

[23] C. Gidney, "Stim: a fast stabilizer circuit simulator," *Quantum*, vol. 5, p. 497, 2021.

[24] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, no. 5, p. 052328, 2004.

[25] D. Gottesman, "The heisenberg representation of quantum computers," *arXiv preprint quant-ph/9807006*, 1998.

[26] 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.

[27] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017.

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

[29] C. Developers, "Cirq," Dec. 2022, See full list of authors on Github: https://github .com/quantumlib/Cirq/graphs/contributors. [Online]. Available: https://doi.org/10.5281/zenodo.7465577

[30] M. Developer, "Mindquantum, version 0.6.0," March 2021. [Online]. Available: https://gitee.com/mindspore/mindquantum

[31] B. A. of Quantum Information Sciences., "PyQuafu." [Online]. Available: https://github.com/ScQ-Cloud/pyquafu

[32] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[33] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, pp. 595–600, 2018.

[34] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, p. 4213, Sep. 2014. [Online]. Available: http://www.nature.com/articles/ncomms5213

[35] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, Sep. 2017. [Online]. Available: http://www.nature.com/articles/nature23879

[36] K. Bharti, A. Cervera-Lierta, T. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. Kottmann, T. Menke *et al.*, "Noisy intermediate-scale quantum (nisq) algorithms (2021)," *arXiv preprint arXiv:2101.08448*, 2021.

[37] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–26, 2023.