QBLAS: A Quantum Basic Linear Algebra and Simulation Library

Xiaopeng Cui[1] and Yu Shi[1,*]

[1] *Department of Physics & State Key Laboratory of Surface Physics, Fudan University, Shanghai 200433, China*

Abstract

An open source quantum basic linear algebra and quantum simulation library: QBLAS is developed based on the newest Q# quantum programming language developed by Microsoft. With the rapid development of quantum computing software and hardware, various quantum algorithms characterized by quantum linear algebra spring up like mushrooms. These algorithms exponentially accelerate various computational and machine learning tasks, but rely heavily on efficient quantum simulation. In addition, as the killer application of quantum computing, quantum simulation has also unique advantages in solving monte carlo symbol problems, strong correlation problems and non-equilibrium dynamics simulation problems. With this trend, it is necessary to systematically develop an algorithm library focus on quantum basic linear algebra and quantum simulation algorithm for welcoming the arrival of the quantum era. QBLAS is such a quantum algorithm library.

* yushi@fudan.edu.cn

I. INTRODUCTION

With the rapid development of quantum computing software and hardware technology, a programmable general-purpose quantum computer is coming out. With this trend, the development of quantum software is imperative[1]. In recent years, various quantum linear algebra algorithms for the purpose of quantum machine learning have surged. With the help of constructing quantum versions of vector and matrix operations, various linear algebra calculation tasks are exponentially accelerated, thereby exponentially accelerating a variety of machine learning. Thus a research boom for quantum machine learning algorithms is emerging[2], which have proposed a variety of exponentially accelerated algorithms, such as quantum K-means unsupervised classification algorithm based on the acceleration of the inner product of quantum vectors[3], quantum support vector machine supervised classification algorithm based on the matrix inversion HHL algorithm[4], quantum principal component analysis based on density matrix quantum simulation[5], and quantum Boltzmann machines and quantum deep learning for deep learning[6, 7] and other quantum acceleration algorithms.

However, the above various quantum linear algebra algorithms rely heavily on efficient quantum simulation of the matrix. In addition, as a killer application of quantum computing, quantum simulation has unique advantages in solving Monte Carlo sign problem, strong correlation problems, non-equilibrium dynamics simulation problems, etc.[8], and will gradually become the mainstream to study the simulation of a physical system. The good news is that quantum simulation algorithms have become more and more mature in recent years. Trotter decomposition simulation method[9–11], quantum walk simulation method for sparse Hamiltonian [12], density matrix exponentiation method for dense Hamiltonian[5, 13, 14], adiabatic quantum evolution method[15–17] etc. laid a good foundation for various Hamiltonian quantum simulations. With such a trend, it is necessary to systematically develop a quantum algorithm and quantum simulation library to welcome the arrival of the quantum era. QBLAS is such an open source quantum library.

The QBLAS library is designed to help you run various quantum basic linear algebra algorithms and quantum simulation tasks as described above on a quantum computer or a quantum simulator. QBLAS is the abbreviation of "quantum basic linear algebra and quantum simulation library", which is developed based on Microsoft Q# and is oriented to

quantum basic linear algebra and quantum simulation tasks. Q# is a high-level programming language focused on quantum programming launched by Microsoft since 2018. It compiles quantum programs written in high-level programming languages into quantum assembly code for specific platforms, and supports quantum simulators based on high-performance computing devices (such as Microsoft Quantum Cloud, QuEST, etc.) and also future real quantum computer platforms (such as IBMQ). For details of Q# language, please refer to its development document https://docs.microsoft.com/en-us/quantum/language.

II. QBLAS OVERVIEW

The overall algorithm structure of QBLAS is shown in Fig.1. QBLAS is divided into two major parts: the quantum linear algebra part and the quantum simulation part.

The quantum linear algebra part is divided into two sub-parts: matrix operation and vector operation. The matrix operation part mainly implements the matrix inversion HHL algorithm [18, 19] , the quantum principal component analysis(QPCA) algorithm based on the HHL algorithm[5], the eigenvalue and singular value decomposition algorithm[13]. In addition, this part also implements the quantum phase estimation algorithm [20] and quantum Fourier transform algorithm [20] that these algorithms rely on. Quantum phase estimation relies on matrix quantum simulation. Matrix quantum simulation is implemented in the quantum simulation part. The quantum vector operation part mainly implements inner product and distance algorithm of vectors and the centers of the vector groups [3] that depend on the quantum swap test[21].

The quantum simulation part implements three types of quantum simulation. (1) Sparse matrix simulation based on quantum walk method[12, 22, 23]. (2) Density matrix simulation and low-rank dense matrix simulation based on density matrix exponentiation method[5, 13, 19]. (3) General matrix quantum simulation based on Trotter Hamiltonian decomposition method[9–11, 24].

Similar to classical computers which requiring memory, the operation of various algorithms of quantum computers requires quantum memory QRAM[25, 26] based on the quantum physical system. In order to demonstrate the algorithm in this library, a virtual QRAM is constructed to help the algorithm run.

The file structure of QBLAS is shown in Fig.2. The entire library uses the principle of

modularization, and each file implements a type of functions. Modules can directly call each other through a fixed quantum function interface. This design will improve the readability and expandability of the library code. Today, with the rapid development of quantum algorithms, scalability is crucial.

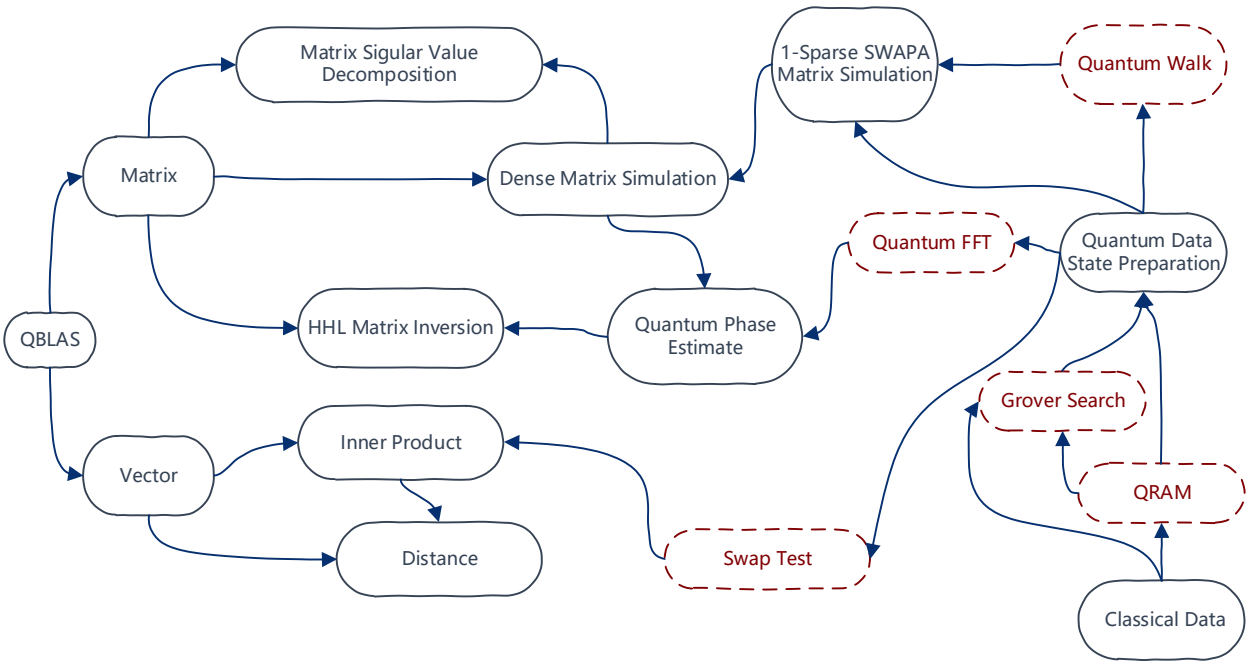

FIG. 1. QBLAS algorithm structure. The dotted box is the core algorithm of quantum exponential or root acceleration.

III. BIT FORMAT OF DATA TYPE

All data types are stored in quantum vector states in little-endian format. As shown below, there are four data types in the current library: Bool, Int, Real, Complex. Except for the Bool type, the last bit is the sign bit, 0 indicates a positive number, and 1 indicates a negative number.

Bool: $|b\rangle = |bool\rangle\,,\ b = 0, 1$

Int: $|b\rangle = |b_0 b_1 ... b_{n-1} b_n\rangle\,,\ b = (b_0 2^0 + b_1 2^1 + ... + b_{n-1} 2^{n-1}) sign(b_n)$

Real: $|b\rangle = |b_0 b_1 ... b_{n-1} b_n\rangle\,,\ b = (b_0 2^{-1} + b_1 2^{-2} + ... + b_n 2^{-(n-1)}) sign(b_n)$

Complex: $|b_n\rangle\,|b_\phi\rangle\,,\ b_n$ is the complex modulus, and b_ϕ is the complex radius.

n changes according to the number of qubits occupied by the data.

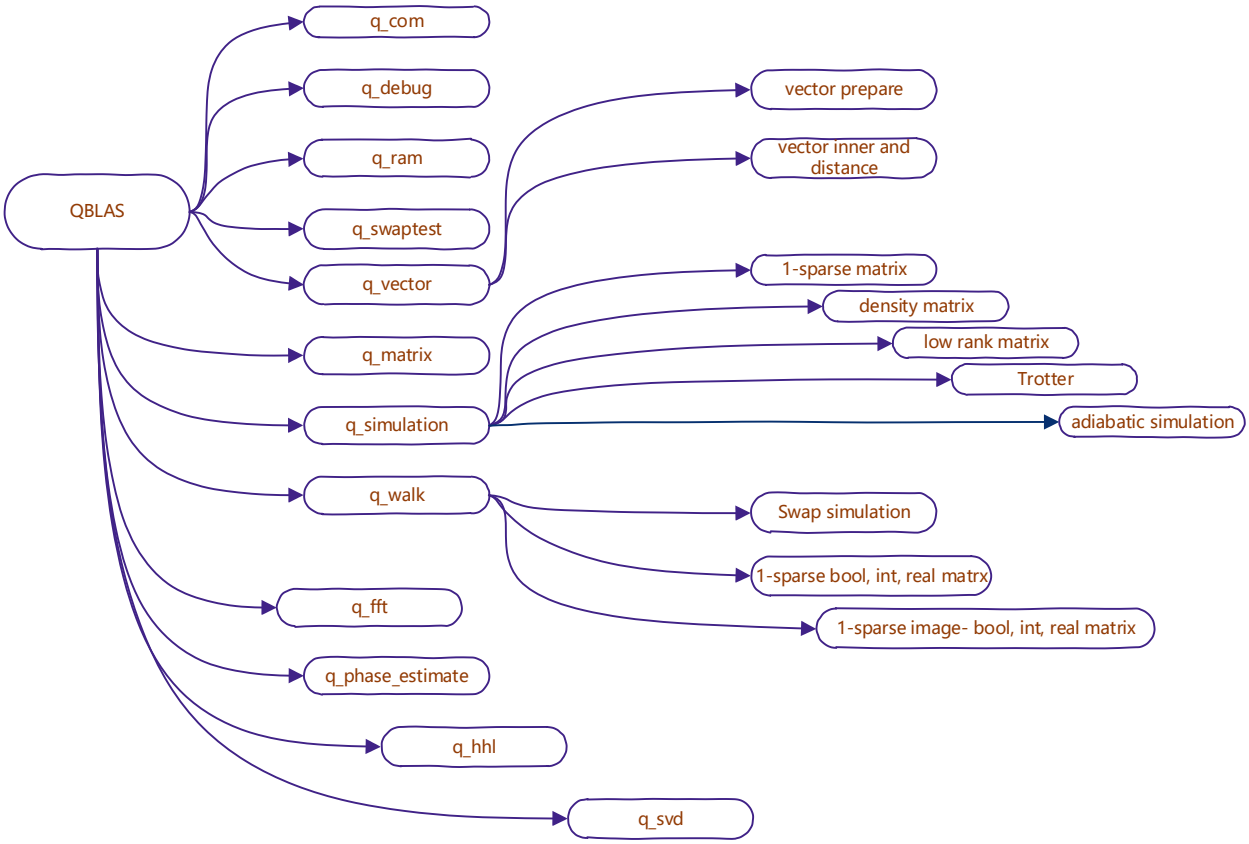

FIG. 2. QBLAS file structure. With modular design, each file contains a separate functional module.

IV. QUANTUM RANDOM MEMORY: QRAM

RAM (random read memory) is the basic construction of a classic computer. For quantum computers it is QRAM. Because the quantum computer needs to access the superposed memory unit, its address is also the superposed address register $\sum_j A_j |a_j\rangle$. After receiving the superposed address, QRAM will return the superposed memory unit content associated with it in the data register d[25]:

$$\sum_{j=0}^{N-1} A_j |a_j\rangle \xrightarrow{QRAM} \sum_{j=0}^{N-1} A_j |a_j\rangle |D_j\rangle \tag{1}$$

a_j is the address register. A_j is the amplitude of the corresponding address. Unlike the classic register, the quantum register can be in the superposition state, and the superposition state can also have the amplitude. D_j is the quantum memory data at the corresponding address.

As discussed in detail in [25, 26], in the QRAM, to complete the transformation from N classic data vectors to a quantum vector requires the physical qubit system with the complexity of $O(log(N))$. Almost all quantum algorithms need to call QRAM to prepare the quantum initial state. In order to facilitate the demonstration of these quantum algorithms, the QBLAS library implements a simulated QRAM in the q_ram module, which can give a calling interface similar to the real QRAM.

V. QUANTUM VECTOR: QVECTOR

Vector is the basic element of linear algebra. Quantum computers use quantum states to represent vectors. A classic vector $\vec{v}$ can be represented by the quantum state $|v\rangle$ plus the modulus parameter $|v|$.

classical vector	quantum vector
$\vec{v} = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{pmatrix}$	$\|v\|,\ \|v\rangle = \sum_{j=0}^{N-1} v_j \|j\rangle$

The transformation of the classic normalized vector $\vec{v}$ to the quantum vector $|v\rangle$ needs to be implemented through the *preparation* process:

$$\vec{v} : \sum_{j=0}^{2^n-1} |j\rangle) \xrightarrow{preparation} \sum_{j=0}^{2^n-1} v_j |j\rangle, n = \log_2(N) \tag{2}$$

n is the number of qubits required. n must be an integer. N that does not meet this condition can be expanded by adding 0 elements.

$$preparation : \sum_{j=0}^{2^n-1} |j\rangle |0\rangle \xrightarrow{QRAM} \sum_{j=0}^{2^n-1} |j\rangle |v_j\rangle \rightarrow \sum_{j=0}^{2^n-1} v_j |j\rangle |v_j\rangle, \xrightarrow{QRAM^\dagger} \sum_{j=0}^{2^n-1} v_j |j\rangle |0\rangle$$

$QRAM^\dagger$ is the Hermitian conjugate of the $QRAM$ operation, which represents the inverse (undo) QRAM operation. It can be seen that through the *preparation* process, the classic

vector element is prepared to the position of the base vector amplitude corresponding to the quantum vector state $v_j \left|j\right\rangle$.

A. Quantum vector preparation: *preparation*

There are many algorithms for preparing quantum vectors from classical vectors. This library provides a measurement-based algorithm to prepare quantum vectors. We take the normalized classic vector $\vec{v}$ as an example.

1. Real vector

$$\vec{v} = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_N \end{pmatrix}$$

preparation process:

$$\begin{aligned}
&\sum_j \left|j\right\rangle \\
\xrightarrow{QRAM} &\sum_j \left|j\right\rangle \left|v_j\right\rangle \\
\xrightarrow{add\ auxiliary\ |0\rangle} &\sum_j \left|j\right\rangle \left|v_j\right\rangle \left|0\right\rangle \\
\xrightarrow{use\ |v_j\rangle\ control-rotate\ CR_y\ auxiliary} &\sum_j \left|j\right\rangle \left|v_j\right\rangle (\sqrt{1-v_j^2}\left|0\right\rangle + v_j \left|1\right\rangle) \\
\xrightarrow{measurement\ auxiliary\ to\ |1\rangle} &\sum_j v_j \left|j\right\rangle \left|v_j\right\rangle \left|1\right\rangle \\
\xrightarrow{QRAM^\dagger (undo\ QRAM)} &\sum_j v_j \left|j\right\rangle
\end{aligned} \tag{3}$$

By measuring the auxiliary qubit, if we get $|1\rangle$, we get the correct state $\sum_i v_j \left|j\right\rangle$, otherwise we get the wrong state (ill state) $\sum_j \frac{\sqrt{1-v_j^2}|j\rangle}{\sqrt{N-1}}$. If we get the ill state, we need to prepare again.

2. Complex vector

The preparation of complex vectors is similar to real vectors. The difference is that the elements of the complex vector have an angle, which can be prepared to the amplitude angle by rotating the auxiliary qubit. Let's take the normalized complex vector $\vec{v}$ as an example.

$$\vec{v} = \begin{pmatrix} v_0 e^{i\phi_0} \\ v_1 e^{i\phi_1} \\ \vdots \\ v_N e^{i\phi_N} \end{pmatrix}$$

preparation process:

$$\begin{aligned}
&\sum_j |j\rangle \\
\xrightarrow{QRAM} &\sum_j |j\rangle |v_j\rangle |\phi_j\rangle \\
\xrightarrow{add\ auxiliary\ |0\rangle} &\sum_j |j\rangle |v_j\rangle |\phi_j\rangle |0\rangle \\
\xrightarrow{use\ |v_j\rangle\ control-rotate\ CR_y\ auxiliary} &\sum_j |j\rangle |v_j\rangle |\phi_j\rangle (\sqrt{1-v_j^2}|0\rangle + v_j |1\rangle) \\
\xrightarrow{use\ |\phi_j\rangle\ control-rotate\ CR_z\ auxiliary} &\sum_j |j\rangle |v_j\rangle |\phi_j\rangle (\sqrt{1-v_j^2}|0\rangle + v_j e^{i\phi_j} |1\rangle) \\
\xrightarrow{measurement\ auxiliary\ to\ |1\rangle} &\sum_j v_j e^{i\phi_j} |j\rangle |v_j\rangle |1\rangle \\
\xrightarrow{QRAM^\dagger (undo\ QRAM)} &\sum_j v_j e^{i\phi_j} |j\rangle
\end{aligned} \tag{4}$$

B. Vector operation: inner product and distance

The inner product calculation operation is an important operation of vectors. The distance of the vectors can also be obtained by inner product. The QBLAS library implements an exponential acceleration algorithm for vector inner product calculation based on swap test[3]. Then, the distance between the two vectors and the distance between the center vectors of the two vector groups can be obtained by this inner product algorithm.

1. *Swapt test and inner product*

The circuit diagram of the swap test algorithm for two vectors (one qubit per vector) is shown in Fig.3. The circuit estimates the inner product of these two vectors by measuring the auxiliary qubit to obtain the probability of $|1\rangle$: P_{pass}.

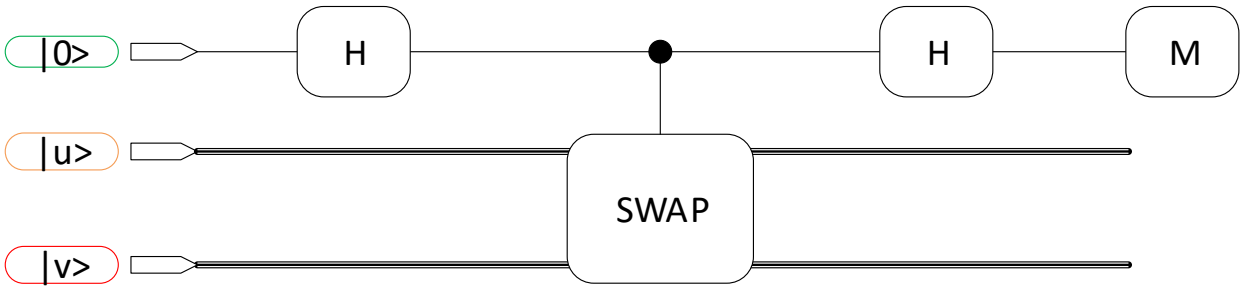

FIG. 3. swap test circuit

Swap test method[21]:

$$|\Phi\rangle = |0\rangle\,|u\rangle\,|v\rangle \xrightarrow{H} \frac{(|0\rangle + |1\rangle)}{\sqrt{2}}\,|u\rangle\,|v\rangle \xrightarrow{CSWAP} \frac{(|0\rangle\,|u\rangle\,|v\rangle + |1\rangle\,|v\rangle\,|u\rangle)}{\sqrt{2}}$$

$$\xrightarrow{H} \frac{|0\rangle\,(|u\rangle\,|v\rangle + |v\rangle\,|u\rangle) + |1\rangle\,(|u\rangle\,|v\rangle - |v\rangle\,|u\rangle)}{2}$$

$$\begin{aligned} P_{pass} &= \frac{(\langle u|\,\langle v| + \langle v|\,\langle u|)(|u\rangle\,|v\rangle + |v\rangle\,|u\rangle)}{4} \\ P_{pass} &= \frac{(1 + |\,\langle u|v\rangle\,|^2)}{2} \\ P_{failure} &= 1 - P_{pass} = \frac{(1 - |\,\langle u|v\rangle\,|^2)}{2} \end{aligned} \tag{5}$$

projection probability $P = |\,\langle u|v\rangle\,|^2 = 2P_{pass} - 1$, projection amplitude $|A_p| = \sqrt{P} = Inner$

$$\text{Inner: } |\,\langle u|v\rangle\,| = \sqrt{P} = \sqrt{2P_{pass} - 1} \tag{6}$$

The swap test algorithm of two vectors (multiple qubits per vector) is different from single bit in that the CSWAP (controlled SWAP) gate acts on each qubit of the vector separately. The distance between these two vectors is D.

$$\begin{aligned} D^2 &= ||\ |u|\,|u\rangle - |v|\,|v\rangle\ ||^2 = |u|^2 + |v|^2 - 2|u||v|\,\langle u|v\rangle \\ &= |u|^2 + |v|^2 - 2|u||v|\sqrt{2P_{pass} - 1} \\ \text{or } D^2 &= |u|^2 + |v|^2 - 2|u||v|\sqrt{1 - 2P_{failure}} \end{aligned} \tag{7}$$

2. Distance between 2 vectors

In addition to the above method, a better swap test method for calculating the inner product of two multi-bit vectors is discussed in detail in [3] and will be used as a general method for calculating the distance between two vectors below. In order to calculate the distance between the two vectors $|u\rangle$ and $|v\rangle$, we need to first construct two quantum states $|\varphi\rangle$ and $|\psi\rangle$. The distance between these two vectors can be estimated by the swap test of the address bits of $|\varphi\rangle$ and $|\psi\rangle$. The advantage of this method is that the swap test part has nothing to do with the number of vector qubits, thus simplifying the operation.

$$\begin{aligned} |\varphi\rangle &= (|0\rangle\,|u\rangle + |1\rangle\,|v\rangle)/\sqrt{2} \\ |\psi\rangle &= (|u|\,|0\rangle - |v|\,|1\rangle)/\sqrt{|u|^2+|v|^2} \end{aligned} \tag{8}$$

The success probability of projection measurement P can be obtained by repeated measurements of the auxiliary qubit.

$$\begin{aligned} P &= |\,\langle\psi|\varphi\rangle\,|^2 = 2P_{pass} - 1 \\ &= |((|u|\,|0\rangle - |v|\,|1\rangle)/\sqrt{|u|^2+|v|^2})^{\dagger}(|0\rangle\,|u\rangle + |1\rangle\,|v\rangle)/\sqrt{2}\;|^2 \\ &= \frac{1}{2}\frac{1}{|u|^2+|v|^2}||\;|u|\,|u\rangle - |v|\,|v\rangle\;\;||^2 \end{aligned} \tag{9}$$

D is the distance between the vectors $|u\rangle$ and $|v\rangle$. and so,

$$\begin{aligned} D^2 &= ||\;|u|\,|u\rangle - |v|\,|v\rangle\;\;||^2 \\ D &= \sqrt{2p(|u|^2+|v|^2)} \end{aligned} \tag{10}$$

3. Distance between 1 vector and the center of 1 vector group

The distance between a vector and the center of a group of vectors is calculated. The center of the M vectors is defined as a center vector as $|v_c\rangle = \sum_{k=0}^{M-1}|v_k\rangle$. Therefore, to find the distance between a vector and the center of a group of vectors is to find the distance between the vector and the center vector $|v_c\rangle$. The method can refer to the above to find the distance between the two vectors. $|\psi\rangle$ is defined as follows:

$$
\begin{aligned}
|\varphi\rangle &= \frac{1}{\sqrt{M+1}}(|0\rangle|u\rangle + \sum_{j=1}^{M}|j\rangle|v_j\rangle) \\
|\psi\rangle &= \frac{1}{\sqrt{|u|^2+\sum_{j=1}^{j=M}|v_m|^2}}(|u||0\rangle - \sum_{j=1}^{M}|v_j||j\rangle)
\end{aligned}
\tag{11}
$$

4. Distance between the centers of 2 vector groups

The distance of two centers of two vector groups is the distance between the center vectors of two vector groups. The method can refer to the above to find the distance between the two vectors. The corresponding $|\varphi\rangle, |\psi\rangle$ is defined as follows:

$$
\begin{aligned}
|\varphi\rangle &= \frac{1}{\sqrt{M}}\sum_{k=0}^{M-1}|k\rangle|v_k\rangle \\
|\psi\rangle &= \frac{1}{\sqrt{Z}}\sum_{k} g_k|v_k||k\rangle, Z = \sum_{k=0}^{M-1}|v_k|^2, g_k = \pm 1
\end{aligned}
\tag{12}
$$

$g_k = \pm 1$ represents the group number 1, 2, M is the number of all vectors in the two groups, and Z is the modular sum of all vectors in the two groups.

The projected measurement probability p is obtained through repeated measurements:

$$
p = |\langle\psi|\varphi\rangle|^2 = \frac{1}{M}\frac{1}{Z}||g_k|v_k||v_k\rangle||^2
$$

$$
p = 2P_{pass} - 1
$$

Corresponding projection amplitude:

$$
A_p = |\langle\psi|\varphi\rangle| = \sqrt{p}
$$

Distance between two center vectors D:

$$
\begin{aligned}
D^2 &= ||g_k|v_k||v_k\rangle||^2 \\
D &= \sqrt{MZp} = \sqrt{MZ(2P_{pass}-1)}
\end{aligned}
\tag{13}
$$

VI. MATRIX: QUANTUM SIMULATION

The quantum simulation of the matrix refers to the given matrix A, and the initial state $|\psi(0)\rangle$, find the final state $|\psi(t)\rangle$ after a given time t :

$$|\psi(t)\rangle = U_A |\psi(0)\rangle = e^{-iAt} |\psi(0)\rangle \tag{14}$$

U_A is the unitary operation corresponding to the simulation. For quantum mechanics, A is required to be Hermitian matrix. For general quantum algorithms, A may be any matrix. In order to fit into the framework of quantum mechanics, it is common to use A to construct a Hermitian matrix $\tilde{A}$, indirectly by simulating $\tilde{A}$ simulate A. So the following discussion is the case of Hermite matrix. If the size of a matrix is $N * N$, and the relationship between the time complexity of the simulation algorithm and N is $O(Log(N)$, the simulation is said to be efficient. The QBLAS library implements several efficient matrix simulation methods that have been discovered.

Controlled simulation refers to the implementation of the CU_A operation which is the U_A operation controlled by a control qubit $|r\rangle$,

$$CU_A(a|0\rangle + b|1\rangle)_r |\psi(0)\rangle = a|0\rangle_r |\psi(0)\rangle + b|1\rangle_r |\psi(t)\rangle \tag{15}$$

Under normal circumstances, the U_A operation can be efficiently implemented, which does not mean that CU_A can also be efficiently implemented in the same way.

A. General matrix quantum simulation based on Trotter Hamiltonian decomposition

If we can simulate the matrix $H_1, H_2, ..., H_q$, then we can simulate $H = H_1 + H_2 + ... + H_q$ approximately. The simulation error is

$$\varepsilon = ||e^{-i(H_1+...+H_q)t} - (e^{-iH_1t/n}...e^{-iH_qt/n})^n|| = O(\frac{qlt^2}{n}) \tag{16}$$

l is the maximum noncommutative modular of any two matrices. t is the simulation time, and n is the number of simulation steps. This decomposition is called Trotter decomposition [9], and there are many variations to improve accuracy such as the following symmetric Trotter[10, 27].

If a Hamilton can be written as $H = A + B$, and A, B can be simulated, then H can be simulated approximately by the symmetric Trotter , the error is

$$\begin{aligned} \varepsilon &= ||e^{-iHt} - (e^{-iA\frac{t}{2n}} e^{-iB\frac{t}{n}} e^{-iA\frac{t}{2n}})^n|| \\ &= ||(\frac{1}{24}[[A,B],A] + \frac{1}{12}[[A,B],B])\frac{t^3}{n^2}|| = O(\frac{lt^3}{n^2}) \end{aligned} \tag{17}$$

It can be seen that the symmetric Trotter pushes the error to the third order and thus improves the accuracy. There are more higher-order Trotter variants along this idea to further improve accuracy[11].

B. Sparse matrix simulation based on quantum walk method

A sparse matrix refers to a matrix with a large number of elements but only a few non-zero elements. A matrix with a sparsity of k means that each row of the matrix has at most k non-zero elements. The sparse matrix with sparsity of 1 can be accurately and efficiently simulated by quantum walk method[12, 22, 23] through the following general simulation techniques.

$$\text{if } H = U^{\dagger}VU, \quad \text{then } e^{-iHt} = e^{-iU^{\dagger}VUt} = U^{\dagger}e^{-iVt}U \tag{18}$$

The simulation of the matrix with sparsity of 1 can be converted to the simulation of the T matrix. The T matrix is the matrix corresponding to the swap gate $SWAP$. A sparse matrix with a sparsity of k can be approximated by Trroter decomposition into k matrices with a sparsity of 1[28].

1. *T matrix simulation*

$|a\rangle, |b\rangle$ is the m qubits state. Swap gate $SWAP\,|a\rangle\,|b\rangle = |b\rangle\,|a\rangle, T = SWAP$, The quantum simulation circuit of the T matrix is shown in Fig.4

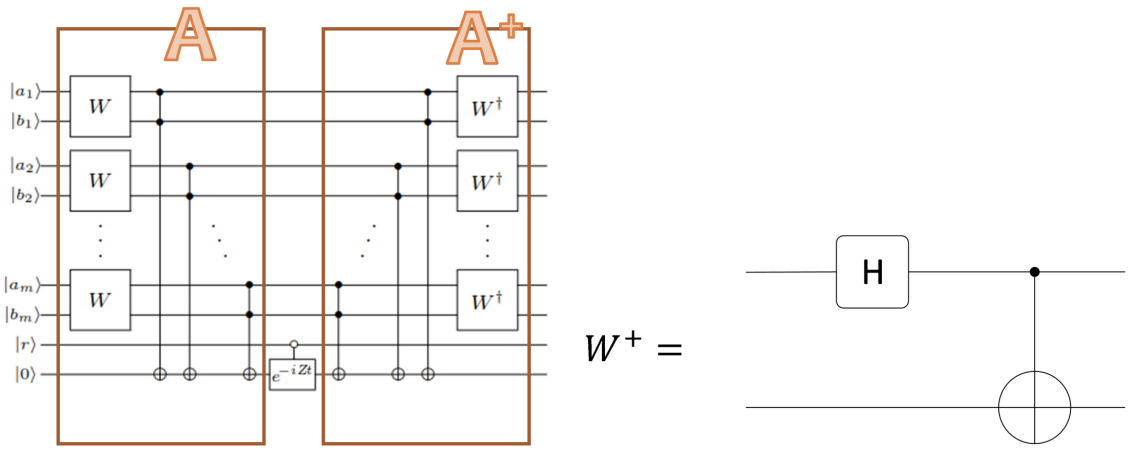

FIG. 4. The quantum simulation circuit of the T matrix

$$
\begin{aligned}
&T\left|a\right\rangle\left|b\right\rangle\left|s\right\rangle = A^{\dagger}ZA\left|a\right\rangle\left|b\right\rangle\left|s\right\rangle, Z = \sigma_z \\
&U_T = e^{-iTt} = e^{-iA^{\dagger}ZAt} = A^{\dagger}e^{-iZt}A, \quad e^{-iZt} = e^{-i\sigma_z t} = R_z(-2t)
\end{aligned}
\tag{19}
$$

$|s\rangle$ is the auxiliary qubit. It is initially set to $|0\rangle$. Z only affects the auxiliary qubit $|s\rangle$. It can be seen that the $SWAP$ matrix T can be accurately simulated, which is converted into a R_z rotation of the single auxiliary qubit $|s\rangle$, that is, a rotating quantum gate operation around the σ_z axis. The time simulation of multi-qubit is transformed to rotate spatial angle of the auxiliary qubit. Among them, $W^{\dagger} = CNOT(H \bigotimes I)$

$$
\begin{aligned}
&W^{\dagger}\left|00\right\rangle = \frac{1}{\sqrt{2}}(\left|00\right\rangle + \left|11\right\rangle), \quad W^{\dagger}\left|01\right\rangle = \frac{1}{\sqrt{2}}(\left|01\right\rangle + \left|10\right\rangle), \\
&W^{\dagger}\left|10\right\rangle = \frac{1}{\sqrt{2}}(\left|00\right\rangle - \left|11\right\rangle), \quad W^{\dagger}\left|11\right\rangle = \frac{1}{\sqrt{2}}(\left|01\right\rangle - \left|10\right\rangle),
\end{aligned}
\tag{20}
$$

It can be seen that W is responsible for the conversion between a Bell basis and its corresponding calculation basis.

Controlled version CU_T, add control bit $|r\rangle$ as shown in Fig.4, replace Z with CZ controlled by $|r\rangle$,

$$
CU_T = A^{\dagger}CR_z(-2t)A \tag{21}
$$

2. Quantum simulation of 1-sparse matrix

A matrix with a sparsity of 1 means that the matrix has at most one non-zero element in each row. If the sparsity of the $N * N$ Hermimatrix H is 1, then only $H_{x,m(x)} \neq 0$,$m(x)$ is the non-zero element of the x row where the column is $m(x)$. This kind of matrixs are divided into 4 sub-categories according to the type of non-zero elements: non-zero elements are bool (1), real number, imaginary bool (i,-i), imaginary number.

The storage of such a matrix in memory only needs to store the position and value of its non-zero elements, each non-zero element is represented by a triplet $(x, m(x), w(x))$, $w(x)$ is a non-zero element value. Define the QRAM memory fetch operation M, and to undo the memory fetch operation is $M^{\dagger}$.

$$
M\left|x\right\rangle\left|0\right\rangle\left|0\right\rangle = \left|x\right\rangle\left|m(x)\right\rangle\left|w(x)\right\rangle, \quad M^{\dagger}\left|x\right\rangle\left|m(x)\right\rangle\left|w(x)\right\rangle = \left|x\right\rangle\left|0\right\rangle\left|0\right\rangle \tag{22}
$$

The three registers used by M are row register, column register, and element value register. The M function is to read out column values and element values from QRAM

based on the row register value. For bool type, because $w(x) \equiv 1$, it is not necessary to store its element values, $M(x)$ is simplified to

$$M|x\rangle|0\rangle = |x\rangle|m(x)\rangle, M^{\dagger}|x\rangle|m(x)\rangle = |x\rangle|0\rangle$$

1) Bool type:

$$H_{x,m(x)} = 1, H|x\rangle = |m(x)\rangle \tag{23}$$

$$\begin{aligned} M^{\dagger}TM|x\rangle|0\rangle = M^{\dagger}T|x\rangle|0\rangle = M^{\dagger}|m(x)\rangle|x\rangle = |m(x)\rangle|0\rangle = H|x\rangle|0\rangle \\ \text{So, } H = M^{\dagger}TM \end{aligned} \tag{24}$$

Therefore, the simulation of the Boolean H can be converted to the simulation of T through the M operation, and then into the rotation of R_z for the single auxiliary qubit $|s\rangle$.

$$e^{-iHt} = e^{-iM^{\dagger}TMt} = M^{\dagger}e^{-iTt}M = M^{\dagger}A^{\dagger}R_z(-2t)AM \tag{25}$$

R_z only works on auxiliary qubit $|s\rangle$, $|s\rangle$ is initially $|0\rangle$.

The controlled version is to add the control qubit $|r\rangle$, replacing R_z with CR_z controlled by $|r\rangle$.

2) Real type:

$$H_{x,m(x)} = w(x), H|x\rangle = w(x)|m(x)\rangle, w(x)\text{is real number} \tag{26}$$

$$\begin{aligned} M^{\dagger}(T\otimes F)M|x\rangle|0\rangle|0\rangle &= M^{\dagger}(T\otimes F)|x\rangle|m(x)\rangle|w(x)\rangle|s\rangle \\ &= M^{\dagger}|m(x)\rangle|x\rangle w(x)|w(x)\rangle \\ &= w(x)|m(x)\rangle|0\rangle|0\rangle \\ &= H|x\rangle|0\rangle|0\rangle \\ \text{So, } H &= M^{\dagger}(T\otimes F)M \end{aligned} \tag{27}$$

The simulation of $T \otimes F$ can be converted into the R_z operation on the auxiliary qubit $|s\rangle$ through the following conversion. $|s\rangle$ is initially $|0\rangle$.

$$\begin{aligned} F|w(x)\rangle &= w(x)|w(x)\rangle \\ e^{-iFt}|w(x)\rangle &= e^{-iw(x)t}|w(x)\rangle \end{aligned} \tag{28}$$

$$
\begin{aligned}
&e^{-i(T\otimes F)t}\left|x\right\rangle\left|m(x)\right\rangle\left|w(x)\right\rangle\left|s\right\rangle \\
&\qquad = A^{\dagger}e^{-i(Z\otimes F)t}A\left|x\right\rangle\left|m(x)\right\rangle\left|w(x)\right\rangle\left|s\right\rangle \\
&\qquad = A^{\dagger}e^{-iZw(x)t}A\left|x\right\rangle\left|x\right\rangle\left|m(x)\right\rangle\left|w(x)\right\rangle\left|s\right\rangle \\
&\qquad = A^{\dagger}R_z(-2w(x)t)A\left|x\right\rangle\left|x\right\rangle\left|m(x)\right\rangle\left|w(x)\right\rangle\left|s\right\rangle
\end{aligned}
\tag{29}
$$

Therefore, the simulation of the real type H can be converted to the simulation of $Z\otimes F$ through the operation of M, and then into the rotation of R_z for the single auxiliary qubit $|s\rangle$.

$$
e^{-iHt} = e^{-iM^{\dagger}A^{\dagger}(Z\otimes F)AMt} = M^{\dagger}A^{\dagger}e^{-iZ\otimes Ft}AM = M^{\dagger}A^{\dagger}R_z(-2w(x)t)AM \tag{30}
$$

The controlled version is to add the control qubit $|r\rangle$, replacing R_z with CR_z controlled by $|r\rangle$.

3) Imaginary bool type:

$$
H_{x,m(x)} = \pm i, H\left|x\right\rangle = \pm i\left|m(x)\right\rangle \tag{31}
$$

Unlike the bool type, the element value stored in QRAM need to add an imaginary digit mark $|sbit\rangle$, $|w(x)\rangle = |sbit\rangle$. When the element value is $+i, sbit = 0$, when the element value is $-i, sbit = 1$

It is close to the bool type, the difference is that the M read operation needs to read $|sbit\rangle$ into the auxiliary qubit $|s\rangle$, and the rotation operation of the auxiliary qubit $|s\rangle$ should be changed to R_y around the σ_y axis. correspondingly

$$
\begin{aligned}
T_y &= A^{\dagger}YA, Y = \sigma_y \\
e^{-iT_yt} &= A^{\dagger}e^{-iYt}A = A^{\dagger}R_y(-2t)A
\end{aligned}
\tag{32}
$$

The simulation of the imaginary bool H can be converted to the simulation of T_y through the above M operation.

$$
e^{-iHt} = e^{-iM^{\dagger}T_yMt} = M^{\dagger}e^{-iT_yt}M = M^{\dagger}A^{\dagger}R_y(-2t)AM \tag{33}
$$

The controlled version is to add the control qubit $|r\rangle$, replacing R_y with CR_y controlled by $|r\rangle$.

2) Imaginary number type:

$$
H_{x,m(x)} = w(x), w(x)\text{is imaginary number} \tag{34}
$$

The format for storing imaginary $w(x)$ in QRAM is $|w(x)\rangle = ||w(x)|\rangle\,|sbit\rangle$, $|sbit\rangle$ is added imaginary digit positive and negative signs, when the imaginary number is $+$, $sbit = 0$, when the imaginary number is $-$, $sbit = 1$. Refer to the relationship between imaginary bool and bool type, the imaginary number type is close to the real type, the difference is M read operation should read $|sbit\rangle$ into the auxiliary qubit $|s\rangle$, and the rotation operation of the auxiliary qubit $|s\rangle$ should be replaced with R_y around the σ_y axis.

Therefore, the simulation of the imaginary number H can be converted to the simulation of $Y \otimes F$ through the M operation, and then into the rotation of R_y for the single auxiliary qubit $|s\rangle$.

$$e^{-iHt} = e^{-iM^\dagger A^\dagger (Y\otimes F)AMt} = M^\dagger A^\dagger e^{-iY\otimes Ft} AM = M^\dagger A^\dagger R_y(-2w(x)t)AM \tag{35}$$

The controlled version is to add the control qubit $|r\rangle$, replacing R_y with CR_y controlled by $|r\rangle$.

3. Quantum simulation of k-sparsity matrix

Generally, the matrix H with a sparsity of k can always be reduced to the sum of a series of the above four types of matrices with a sparsity of 1, $H_1, H_2, ..., H_q$, $H = H_1 + H_2 + ... + H_q$, which can be approximated by the Trotter decomposition method. A complex matrix with a sparsity of 1 can also be reduced to the sum of a real and a imaginary matrices with a sparsity of 1, then it can be simulation approximately by Trotter decomposition.

C. Quantum simulation of density matrix

The sparse matrix can be efficiently simulated by the above quantum walk method. If the general dense matrix of $N * N$ is decomposed into a sparse matrix according to Trotter, it is not always possible to obtain efficient simulation, because in some cases the number of decomposed matrices And N are of the order of magnitude, and N increases exponentially with the number of qubits, which means that Trotter complexity increases with qubit exponentially, which is difficult to apply. But the dense density matrix can be efficiently simulated by an algorithm, which was first introduced in [5].

For the density matrix $\rho = |\psi\rangle\langle\psi|$, to simulate $U_\rho = e^{-i\rho t}$, the simulated state is $|\phi\rangle$, the density matrix is $\sigma = |\phi\rangle\langle\phi|$, so the task to be realized is $U_\rho \sigma U_\rho^\dagger = e^{-i\rho t}\sigma e^{i\rho t}$.

It can be simulated by the following approximate method

$$\begin{aligned} Tr_\rho(e^{-iT\delta t}\rho\otimes\sigma e^{iT\delta t}) &= (cos^2\delta t)\sigma + (sin^2\delta t)\rho - isin\delta t cos\delta t[\rho,\sigma] \\ &\approx \sigma - i\delta t[\rho,\sigma] + O(\delta t^2) \\ &\approx e^{-i\rho\delta t}\sigma e^{i\rho\delta t} \\ e^{-i\rho t}\sigma e^{i\rho t} &= (e^{-i\rho\delta t}\sigma e^{i\rho\delta t})^n, \delta t = \frac{t}{n} \end{aligned} \tag{36}$$

It can be seen that the idea is close to the simulation of the density matrix, firtly prepare a new swap matrix T_A, then perform T_A simulation on $|\psi\rangle$ and $|\phi\rangle$, and lastly find the trace dropping $|\psi\rangle$ allows the dense matrix $\frac{A}{N}$ to be simulated. Number of simulation steps $n = O(\frac{t^2}{\varepsilon}||A||^2_{max})$. Typically, $||A||_{max} = O(Rank(A))$, so $n = O(\frac{t^2}{\varepsilon}Rank(A))$. For σ is not pure state, the above is still valid.

Controlled version CU_ρ, replace U_T with controlled version CU_T.

D. Quantum simulation of low-rank dense matrix

With the help of dense density matrix which can be efficiently simulated, another type of dense matrix can also be simulated efficiently, that is, low-rank dense matrix. This method was first proposed in [13]. The low-rank matrix of $N * N$ matrix A means that its rank $Rank(A)$ is much lower than N.

Simulation task $U_{\frac{A}{N}}\sigma U^\dagger_{\frac{A}{N}} = e^{-i\frac{A}{N}t}\sigma e^{i\frac{A}{N}t}$,$\sigma = |\phi\rangle\langle\phi|$ is the simulated state.

First, fill in the elements of A into the swap matrix of $N^2 * N^2$ at the position where the vaule is 1 in T to construct a new swap matrix

$$T_A = \sum_{j,k=0}^{N-1} A_{jk}|k\rangle|j\rangle\otimes|j\rangle|k\rangle, \tag{37}$$

Such T_A is also a matrix with sparsity of 1, which can be accurately and efficiently simulated by quantum walk. Then borrow the method of density matrix simulation, but take $\rho = |\psi\rangle\langle\psi|, |\psi\rangle = \sum_{i=0}^{N-1}|i\rangle$. The $|\psi\rangle$ is an equal weight superposition state, which is efficiently

can be prepared.

$$\begin{aligned} Tr_\rho(e^{-iT_A\delta t}\rho\otimes\sigma e^{iT_A\delta t}) &= \sigma - iTr_\rho\{T_A\rho\otimes\sigma\}\delta t + iTr_\rho\{\rho\otimes\sigma T_A\}\delta t + O(\delta t^2) \\ &= \sigma - i\frac{\delta t}{N}[A,\sigma] + O(\delta t^2) \\ &\approx e^{-i\frac{A}{N}\delta t}\sigma e^{i\frac{A}{N}\delta} \\ e^{-i\frac{A}{N}t}\sigma e^{i\frac{A}{N}t} &= (e^{-i\frac{A}{N}\delta}\sigma e^{i\frac{A}{N}\delta})^n, \delta t = \frac{t}{n} \end{aligned} \tag{38}$$

It can be seen that the idea is close to the simulation of the density matrix, firtly prepare a new swap matrix T_A, and then perform T_A simulation on $|\psi\rangle$ and $|\phi\rangle$, and last find the trace Dropping $|\psi\rangle$ to allow the dense matrix $\frac{A}{N}$ to be simulated. Number of simulation steps $n = O(\frac{t^2}{\varepsilon}||A||^2_{max})$. Typically, $||A||_{max} = O(Rank(A))$, so $n = O(\frac{t^2}{\varepsilon}Rank(A))$. For σ is not pure state, the above method is still valid.

Controlled version $CU_{\frac{A}{N}}$, replace U_{T_A} with CU_{T_A} controlled by control qubit.

VII. MATRIX OPERATION: INVERSION AND DECOMPOSITION

The matrix inversion HHL algorithm and various matrix decomposition algorithms rely on matrix simulation and quantum phase estimation, and quantum phase estimation is an inverse Fourier transform process.

A. Quantum Fourier transformation (QFT)

$$|m\rangle \xrightarrow{QFT} \frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1} e^{im\frac{2\pi}{2^n}y}|y\rangle,\ m\frac{2\pi}{2^n} = \phi \tag{39}$$

B. Quantum phase estimation (QPE)

Quantum phase estimation is divided into two parts: phase preparation and inverse Fourier transform[20].

1) Phase preparation:

If $U = e^{iA}, U|u\rangle = e^{i\phi}$, ϕ is the phase to be estimated, a total of n control qubits, CU^{2^j} is the evolution of U^{2^j} controlled by the jth control qubit, and phase preparation can be done through the following process:

$$CU^{2^j}(\sum_{x=0}^{2^n-1}|x\rangle)|u\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i2^j\phi}|1\rangle)|u\rangle = \frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1} e^{i\phi y}|y\rangle|u\rangle \tag{40}$$

2) Inverse Fourier transform:

$$\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1} e^{im\frac{2\pi}{2^n}y}|y\rangle|u\rangle \to |m\rangle|u\rangle, \ m\frac{2\pi}{2^n} = \phi \tag{41}$$

the phase ϕ value ($[-\pi, \pi)$) can be calculated by m, where the format of m is:

$$\begin{aligned} &|m\rangle = |m_0 m_1 ... m_{n-1} m_n\rangle, n+1 \text{ qubits} \\ &m_n = 0, \ \phi = \frac{\pi}{2^n}(m_0 2^0 + m_1 2^1 + ... + m_{n-1}2^{n-1}) \\ &m_n = 1, \ \phi = \frac{\pi}{2^n}(m_0 2^0 + m_1 2^1 + ... + m_{n-1}2^{n-1} - 2^n) \end{aligned} \tag{42}$$

C. Matrix inversion HHL algorithm

Matrix inversion operation is the key operation of linear algebra. Given a $N * N$ invertible matrix A, $Ax = b$, x, b are vectors, and solve $x = A^{-1}b$. The time complexity of the best classic algorithm is $O(N)$. The quantum corresponding version is $A|x\rangle = |b\rangle$, solve the quantum vector $|x\rangle = A^{-1}|b\rangle$, the article[18] with the help of matrix Simulation, quantum phase estimation and quantum measurement give a quantum exponentially accelerated (time complexity $O(log(N))$) matrix inversion algorithm, called HHL algorithm [2]. The implementation process is as follows:

In the eigenvector space of A, A is the diagonal matrix:

$$A = \begin{pmatrix} \lambda_0 & ... & 0 & 0 \\ 0 & \lambda_0 & ... & 0 \\ ... & .... & ... & ... \\ 0 & ... & 0 & \lambda_{N-1} \end{pmatrix}, A^{-1} = \begin{pmatrix} \lambda_0^{-1} & ... & 0 & 0 \\ 0 & \lambda_0^{-1} & ... & 0 \\ ... & .... & ... & ... \\ 0 & ... & 0 & \lambda_{N-1}^{-1} \end{pmatrix}, \tag{43}$$

And $|b\rangle, |x\rangle$ are:

$$|b\rangle = \sum_{j=0}^{N-1}\beta_j|u_j\rangle, \quad |x\rangle = A^{-1}|b\rangle = \sum_{j=0}^{N-1}\beta_j\lambda_j^{-1}|u_j\rangle \tag{44}$$

$|u_j\rangle$ is the eigenvector of A, and it can be seen that to solve $|x\rangle$ is to preparare the state

$\sum_{j=0}^{N-1} \beta_j \lambda_j^{-1} |u_j\rangle$, the preparation steps are as follows:

$$\begin{aligned}
e^{iAt} |b\rangle &= \sum_{j=0}^{N-1} e^{i\lambda_j t} \beta_j |u_j\rangle \\
&\xrightarrow{QPE} = \sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle |u_j\rangle \\
&\xrightarrow{\text{add auxiliary qubit}|0\rangle} = \sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle |u_j\rangle |0\rangle \\
&\xrightarrow{C_Ry|\lambda\rangle|0\rangle} = \sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle |u_j\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda} |1\rangle\right) \\
&\xrightarrow{\text{measure auxiliary qubit to } |1\rangle} = \sum_{j=0}^{N-1} \beta_j \lambda_j^{-1} |\lambda_j\rangle |u_j\rangle \\
&\xrightarrow{\text{undo QPE: } QPE^\dagger} = \sum_{j=0}^{N-1} \beta_j \lambda_j^{-1} |u_j\rangle
\end{aligned} \tag{45}$$

C_R_y means the rotation of the auxiliary qubit $|0\rangle$ around the σ_y axis controlled by $|\lambda\rangle$. In the auxiliary qubit measurement step, as with the measurement-based quantum vector preparation, if the measurement result of the auxiliary qubit is$|0\rangle$, the preparation fails, and it needs to be re-prepare until $|1\rangle$ is measured. The time complexity of each preparation operation is $O(Log(N))$, and the success probability is related to the matrix eigenvalue λ distribution. For some matrices [18], a high success probability can be obtained. Therefore, compared with the classical matrix inversion algorithm, the exponential speed-up is realized.

D. Matrix: eigenvalue decomposition algorithm

Given a $N*N$ matrix A, its eigenvalue $\lambda_j, j = 0...N-1$ and the corresponding eigenvector $|u_j\rangle$, A can be decomposed into the following form

$$A = \sum_{j=0}^{N-1} |u_j\rangle \langle u_j| \tag{46}$$

This is the eigenvalue decomposition of A. It can be seen that the steps to solve this problem have already appeared in the above HHL algorithm:

$$e^{iAt} |b\rangle = \sum_{j=0}^{N-1} e^{i\lambda_j t} \beta_j |u_j\rangle \xrightarrow{QPE} = \sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle |u_j\rangle \tag{47}$$

Properly select $|b\rangle$ to obtain the required λ_j, and because of the entanglement between $|\lambda_j\rangle$ and $|u_j\rangle$, the corresponding eigenvector $|u_j\rangle$ can be obtained by measurement at the same time.

E. Matrix: singular value decomposition(SVD) algorithm

Given a $M * N$ matrix A, its singular value $\sigma_j, j = 0...R-1$, corresponding to the right eigenvector $|u_j\rangle$, corresponding to the left eigenvector $ketv_j$, $R = Min(M, N)$, A can be decomposed into the following form

$$A = \sum_{j=0}^{R-1} \sigma_j |u_j\rangle \langle v_j| \tag{48}$$

This is called the singular value decomposition of A. The eigenvalue decomposition algorithm can be used for reference, but A is not necessarily a square matrix or Hermite. To facilitate quantum simulation, you need to construct a $(M+N)*(M+N)$ Hermite square matrix$\tilde{A}$

$$\tilde{A} = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix}$$

The eigenvalue of $\tilde{A}$ is the singular value of A, $\sigma_j, j = 0...R-1$, corresponding to the eigenvector

$$|\tilde{u_j}\rangle = |u_j\rangle |v_j\rangle , \ \tilde{A} = \sum_{j=0}^{R-1} \sigma_j |\tilde{u}\rangle \langle \tilde{u}| \tag{49}$$

Then follow the eigenvalue decomposition to decompose $\tilde{A}$

$$e^{i\tilde{A}t} |b\rangle = \sum_{j=0}^{R-1} e^{i\sigma_j t} \beta_j |\tilde{u_j}\rangle \xrightarrow{QPE} = \sum_{j=0}^{R-1} \beta_j |\sigma_j\rangle |\tilde{u_j}\rangle \tag{50}$$

Properly select $|b\rangle$ to get the required σ_j, and because of the entanglement between $|\sigma_j\rangle$ and $|\tilde{u_j}\rangle$, by measurement you can simultaneously get the eigenvalue and corresponding eigenvector $|\tilde{u_j}\rangle = |u_j\rangle |v_j\rangle$, you can see the right eigenvector $|u_j\rangle$ and its associated left eigenvector $|v_j\rangle$ was also obtained. This algorithm was introduced in [13].

F. Quantum principal component analysis(PCA) algorithm

A quantum vector

$$|X\rangle = \sum_{i=0}^{M-1} w_i |x_i\rangle \tag{51}$$

$|x_i\rangle$ is N dimension component vectors, w_i is component weight.

Construct the density matrix of $|X\rangle$

$$\rho = |X\rangle \langle X| = \sum_{i=0,j=0}^{M-1} w_i w_j^\dagger |x_i\rangle \langle x_j| \tag{52}$$

If there is a new set of N dimension vector groups $|u_j\rangle$ which is consisting of $|x_i\rangle$ linear combination, $j = 0...R-1, R \ll M$, such that

$$|X\rangle = \sum_{j=0}^{R-1} \sqrt{\lambda_j} |u_j\rangle, \quad \rho = \sum_{j=0}^{R-1} \lambda_j |u_j\rangle \langle u_j| \tag{53}$$

Then $|u_j\rangle$ constitutes a new set of component vectors for $|X\rangle$, where the $|u_j\rangle$ corresponding to the larger λ_j is the principal component of $|X\rangle$. The the best classic algorithm complexity of principal component analysis is $O(MN)$.

Perform eigenvalue decomposition of density matrix ρ with $|X\rangle$

$$|X\rangle = \sum_{j=0}^{R-1} \sqrt{\lambda_j} |u_j\rangle, \quad e^{i\rho t} |X\rangle = \sum_{j=0}^{R-1} e^{i\lambda_j t} \sqrt{\lambda_j} |u_j\rangle \xrightarrow{QPE} = \sum_{j=0}^{R-1} \sqrt{\lambda_j} |\lambda_j\rangle |u_j\rangle \tag{54}$$

It can be seen that λ_j is located at the amplitude position. Due to its entanglement with $|u_j\rangle$, the larger the *lambda*, the greater the probability of obtaining the corresponding principal component $|u_j\rangle$. when $R \ll M$, it can be obtained with only a few measurements. The time complexity of each quantum operation is $O(Log(MN))$, which is exponentially faster than the best classic algorithm. This algorithm first introduced in the reference[5].

VIII. TEST

In order to demonstrate the correctness and usability of the library, some simple test tasks are done with a few qubits. The test tasks in the test code are as follows.

A. Vector part

1. *Quantum Vector preparation*

real vector preparation test: $\vec{v} = [0.3, 0.3, 0.6, 0.9]$

complex vector preparation test: $\vec{v} = [0.3, 0.3, 0.6e^{i\pi}, 0.9e^{i2\pi}]$

2. Swap test and inner product algorithm

Task: two unit vector $\vec{v_1} = [1, 0], \vec{v_2} = [0, 1]$

Expected result: inner product is 0,distance is $\sqrt{2}$

3. distance between 2 center vector of vector groups

Task: Two vector groups, group 1 $\vec{v_{11}} = [e^{i\pi}, 0], \vec{v_{12}} = [1, 0]$, group 2 $\vec{v_{21}} = [0, 1], \vec{v_{21}} = [0, 1]$

Expected result: $A_p = 0.5,\ D = 2\sqrt{2}$

B. Matrix part

1. QPE

Task: $|u\rangle = |0\rangle\,,\ A = \sigma_z,\ t = -1$

$$e^{-iAt}\,|u\rangle = e^{-i\sigma_z t}\,|0\rangle = e^{-iR_z 2t}\,|u\rangle = e^{i\phi}\,|0\rangle$$

Expected result: $R_z = \frac{\sigma_z}{2}, \phi = 1$

2. matrix inversion HHL algorithm

Task: $|b\rangle = |1\rangle\,, A = \sigma_x, A\,|x\rangle = |b\rangle$

Expected result: $|x\rangle = \sigma_x^{-1}\,|1\rangle \to |0\rangle$ —

3. quantum simulation of density matrix

Task: take a density matrix $\rho = |+\rangle\,\langle +| = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $\sigma = |0\rangle\,\langle 0|$, simulation time $t = \frac{\pi}{3}$. simulate $e^{-i\rho t}\sigma e^{i\rho t}$.

$$e^{-i\rho t} = \sum_{k=0}^{\infty} \frac{(-1)^{2k}}{(2k)!}(\rho t)^{2k} + \sum_{k=0}^{\infty} \frac{(-1)^{2k+1}}{(2k+1)!}(\rho t)^{2k+1} = [1 + (cos(t) - 1)\rho] - i\,sin(t)\rho$$

$$= 1 + (e^{-it} - 1)\rho$$

$\rho|0\rangle = \frac{1}{2}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{2}(|0\rangle + |1\rangle)$

$e^{-i\rho t}|0\rangle = 1 + (e^{-it}-1)\rho|0\rangle = \frac{e^{-it}+1}{2}|0\rangle + \frac{e^{-it}-1}{2}|1\rangle = A|0\rangle + B|1\rangle$

Expected result: when $t = \frac{\pi}{3}$, $\quad |A|^2 = 0.75$

4. quantum simulation of T(Swap) matrix

Task: Given $|a\rangle = |11\rangle$, $|b\rangle = |00\rangle$, perform T(Swap) matrix simulaiton, simulaiton time $t = \frac{\pi}{2}$.

$$e^{-iTt} = cos(t)I - isin(t)T = -iT$$

Expected result: $t = \frac{\pi}{2}, e^{-iTt}|a\rangle|b\rangle = -i|b\rangle|a\rangle$

5. quantum simulation of 1-sparsity matrix

$$\sigma_x = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}, \sigma_y = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix}$$

Task: The initial state is $|0\rangle$

1)bool matrix σ_x, simulation time $t = \frac{\pi}{4}$

$$e^{-i\sigma_x\frac{\pi}{4}}|0\rangle = e^{-iR_x\frac{\pi}{2}}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$$

2)imaginary bool matrix σ_x, simulation time $t = \frac{\pi}{4}$

$$e^{-i\sigma_y\frac{\pi}{4}}|0\rangle = e^{-iR_y\frac{\pi}{2}}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

3)real matrix $2\sigma_x$, simulation time $t = \frac{\pi}{8}$

$$e^{-i2\sigma_x\frac{\pi}{8}}|0\rangle = e^{-iR_x\frac{\pi}{2}}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$$

4)imaginary matrix $2\sigma_x$, simulation time $t = \frac{\pi}{8}$

$$e^{-i2\sigma_y\frac{\pi}{8}}|0\rangle = e^{-iR_x\frac{\pi}{2}}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

6. quantum simulation of low-rank dense matrix

Task: take a small low-rank dense matrix $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, simulation time $t = \frac{\pi}{6}$. because of $e^{-iA\frac{\pi}{6}} |0\rangle = e^{-i\rho\frac{\pi}{3}} |0\rangle, \rho = |+\rangle \langle +|$

Expected result: refer to the density matrix simulation test.

IX. CODE INSTRUCTIONS

The code of QBLAS organized in the form of a Git repository is released on the famous open source website GitHub with the GPLv3 open-source license, website link https://github.com/xpclove/qblas. Git is an excellent code version management software emerging in recent years, which gives open source software convenient development mode and continuous vitality. Given the rapid growth in the field of quantum algorithms, an open code management paradigm is beneficial.

Code format and function names are in the Linux underline style, which try to let you know the module and its function by name. Also, usually we use lowercase function names and variable names, but use uppercase in some parts according to the naming conventions of the Q# language.

X. CONCLUSION

We use Microsoft's latest Q# quantum programming language to complete a quantum basic linear algebra and quantum simulation library: QBLAS. The library realizes some major quantum linear algebra acceleration algorithms and quantum simulation algorithms developed in recent years through quantum code. The quantum linear algebra part implements quantum acceleration algorithms such as vector operation: inner product, distance and matrix operation: inversion, eigenvalue decomposition, singular value decomposition, principal component analysis. Quantum simulation part realizes quantum walk, density matrix exponentiation simulation, low-rank dense matrix simulation, Trotter decomposition and so on. Through a few simple examples, the main algorithm in the library is tested, and the result is consistent with the theory, which proves the correctness and availability of the library.

With the advance of quantum computer and the rapid development of quantum machine learning, quantum software and quantum program will become an important subject. The library provides exploration for the paradigm and application of quantum programming. As an important application of quantum computation in the field of physics, it is imperative to develop a universal and flexible quantum simulation software library. QBLAS is expected to promote the development of open source quantum simulation software.

QBLAS is published on the well-known GitHub open source website with GPLv3 open source license, featuring open source, modularity and extensibility, which facilitates the subsequent continuous development and software community cooperation. As a new revolution of information technology, quantum computing and quantum programming will surely lead to a new era. Reviewing all kinds of software communities and companies created by the development of classical computer in the past decades, strengthening software layout is the most urgent task for the development of quantum computer and the pre-dawn action of quantum computing.

---

[1] Frederic T. Chong, Diana Franklin, and Margaret Martonosi, "Programming languages and compiler design for realistic quantum hardware," Nature **549**, 180 (2017).

[2] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd, "Quantum machine learning," Nature **549**, 195–202 (2017), arXiv:1611.09347 [quant-ph].

[3] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost, "Quantum algorithms for supervised and unsupervised machine learning," arXiv e-prints , arXiv:1307.0411 (2013), arXiv:1307.0411 [quant-ph].

[4] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd, "Quantum support vector machine for big data classification," Physical Review Letters **113**, 130503 (2014).

[5] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum principal component analysis," Nature Physics **10**, 631–633 (2014).

[6] Mohammad H. Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko, "Quantum boltzmann machine," Physical Review X **8**, 021050 (2018).

[7] Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore, "Quantum Deep Learning," arXiv

e-prints , arXiv:1412.3489 (2014), arXiv:1412.3489 [quant-ph].

[8] J. Ignacio Cirac and Peter Zoller, "Goals and opportunities in quantum simulation," Nature Physics **8**, 264–266 (2012).

[9] Seth Lloyd, "Universal quantum simulators," Science **273**, 1073 (1996).

[10] Andrew M. Childs, Yuan Su, Minh C. Tran, Nathan Wiebe, and Shuchen Zhu, "A Theory of Trotter Error," arXiv e-prints , arXiv:1912.08854 (2019), arXiv:1912.08854 [quant-ph].

[11] Naomichi Hatano and Masuo Suzuki, "Finding exponential product formulas of higher orders," in *Quantum Annealing and Other Optimization Methods* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005) pp. 37–68.

[12] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman, "Exponential algorithmic speedup by quantum walk," eprint arXiv:quant-ph/0209131 , quant–ph/0209131 (2002).

[13] Patrick Rebentrost, Adrian Steffens, Iman Marvian, and Seth Lloyd, "Quantum singular-value decomposition of nonsparse low-rank matrices," Physical Review A **97**, 012327 (2018).

[14] L. Wossnig, Z. K. Zhao, and A. Prakash, "Quantum linear system algorithm for dense matrices," Physical Review Letters **120**, 5 (2018).

[15] A. Hamma and D. A. Lidar, "Adiabatic preparation of topological order," Physical Review Letters **100**, 4 (2008).

[16] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser, "Quantum Computation by Adiabatic Evolution," arXiv e-prints , quant-ph/0001106 (2000), arXiv:quant-ph/0001106 [quant-ph].

[17] S. Bachmann, W. De Roeck, and M. Fraas, "Adiabatic theorem for quantum spin systems," Physical Review Letters **119**, 060201 (2017).

[18] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd, "Quantum algorithm for linear systems of equations," Physical Review Letters **103**, 150502 (2009).

[19] Leonard Wossnig, Zhikuan Zhao, and Anupam Prakash, "Quantum Linear System Algorithm for Dense Matrices," Physical Review Letters **120**, 050502 (2018), arXiv:1704.06174 [quant-ph].

[20] Giuliano Benenti, Giulio Casati, and Giuliano Strini, *Principles Of Quantum Computation And Information - Volume I: Basic Concepts* (World Scientific Publishing Co., Inc., 2004).

[21] Juan Carlos Garcia-Escartin and Pedro Chamorro-Posada, "swap test and Hong-Ou-Mandel

effect are equivalent," Physical Review A **87**, 052330 (2013), arXiv:1303.6814 [quant-ph].

[22] Andrew M. Childs, Edward Farhi, and Sam Gutmann, "An example of the difference between quantum and classical random walks," Quantum Information Processing **1**, 35–43 (2002).

[23] Graeme Ahokas, "Improved algorithms for approximate quantum fourier transforms and sparse hamiltonian simulations," (2004).

[24] H. F. Trotter, "On the product of semi-groups of operators," Proceedings of the American Mathematical Society **10**, 545–551 (1959).

[25] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone, "Quantum random access memory," Physical Review Letters **100**, 160501 (2008).

[26] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone, "Architectures for a quantum random access memory," Physical Review A **78**, 052310 (2008).

[27] A. D. Kennedy, M. A. Clark, and P. J. Silva, "Force-gradient integrators," in *Symposium on Lattice Field Theory* (2009) p. 21, arXiv:0910.2950 [hep-lat].

[28] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sand ers, "Efficient Quantum Algorithms for Simulating Sparse Hamiltonians," Communications in Mathematical Physics **270**, 359–371 (2007), arXiv:quant-ph/0508139 [quant-ph].