Assignment 1: Operation Matrix Simulations

Big Data and Intelligent System Lab @ ECNU

Due date: July 22, 2024, 10:00 AM

1 Introduction

Quantum computing is a promising field that shows capabilities surpassing classical computers in specific applications, such as integer factorization and unstructured search. Simulations are useful for designing and validating classical circuits. The same holds for quantum circuits, especially when the development of physical quantum machines is still in its early stages.

In this assignment, we will implement a fundamental quantum circuit simulation method called the **operation matrix simulation**. It can simulate any given quantum circuit and provides intermediate quantum states during simulations. Unfortunately, we will see that the computation and memory overheads both grow exponentially with the number of qubits increasing, which limits the scale of quantum circuits that can be simulated.

The operation matrix simulation is often referred to as the **full-state quantum circuit simulation** because it provides the complete state vector during and after the simulation. From this perspective, measurement can be omitted as the state vector offers more detailed information.

2 Background

This section briefly reviews the mathematical model of quantum computing. Based on this model, quantum states, operations, and measurements can be simulated on classical computers. The contents most relevant to this programming assignment are highlighted in red.

2.1 Data Structures

The process of quantum computing is described by a quantum circuit consisting of qubits and quantum gates.

Qubits and state vectors. The basic units for storing information in quantum computing are called *qubits*. Mathematically, the state of an n-qubit system can be represented by a vector $|\phi\rangle$ known as the *state vector*, which contains 2^n complex numbers called *amplitudes*. The state vector is represented as

$$|\phi\rangle = \alpha_{00\ldots0}\,|00\ldots0\rangle + \ldots + \alpha_{11\ldots1}\,|11\ldots1\rangle = \begin{bmatrix} \alpha_{00\ldots0} \\ \alpha_{00\ldots1} \\ \vdots \\ \alpha_{11\ldots1} \end{bmatrix},$$

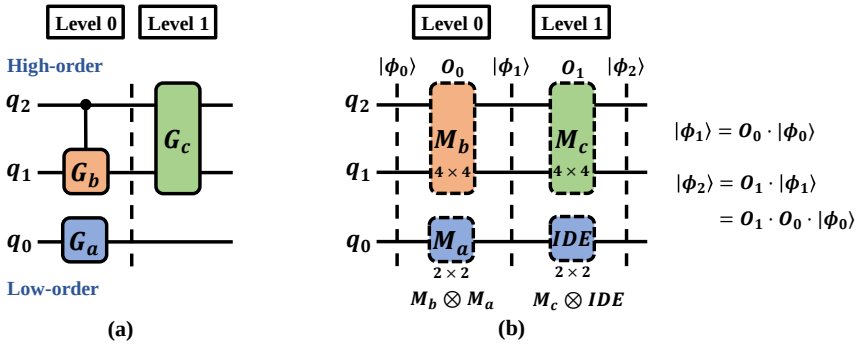

Figure 1: (a) An example of a 3-qubit 2-level quantum circuit. (b) The process of operation matrix simulation (OMSim) for it.

where $\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$, as $|\alpha_x|^2$ represents the probability of the quantum system collapsing into the corresponding basis state $|x\rangle$ after measurement. Suppose the 0th qubit represents the low-order (least significant) qubit. For example, as shown in Fig. 1(a), if initially $q_2 = q_1 = |0\rangle$ and $q_0 = |1\rangle$, the initial state vector of this 3-qubit circuit is $|001\rangle$, with only one amplitude $\alpha_{001} = 1$.

Quantum gates and gate matrices. In a quantum circuit, the operations on qubits are described using *quantum gates*, which can change the state of a quantum system. Mathematically, each quantum gate can be represented by a complex unitary matrix called *gate matrix*.

Fig. 1(a) shows a quantum circuit with two levels of quantum gates, containing a single-qubit gate G_a, a 2-qubit controlled gate controlled-G_b (G_b is applied to q_1 if $q_2 = |1\rangle$), and a 2-qubit non-controlled gate G_c. These three types of gates are commonly used and can form a set of universal quantum gates. Given a quantum gate G_x, if it has i input qubits, mathematically speaking, G_x is a $2^i \times 2^i$ matrix. The following three gates are common examples.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \text{Controlled-}X\ (CNOT,\ CX) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where X is a single-qubit gate, CX is a 2-qubit controlled gate, and $SWAP$ is a 2-qubit non-controlled gate. More quantum gates that are frequently used can be found here.

2.2 Operation Matrix Simulations (OMSim)

2.2.1 Constructing an Operation Matrix for Each Level

The state vector of an n-qubit circuit has a size of 2^n. To change its state, the *operation matrix simulation* (OMSim) constructs an operation matrix of size $2^n \times 2^n$ at each level of the circuit.

Matrix tensor product. For the jth level, use M_i^j to denote the ith quantum gate matrix from high-order to low-order. The operation matrix O_j of level j can be constructed by

$$O_j = M_0^j \otimes M_1^j \otimes M_2^j \otimes \ldots,$$

where $\otimes$ refers to the tensor product of matrices. The tensor product of matrices $A_{a\times b}$ and $B_{c\times d}$ has a size of $ac \times bd$. Please note that if no gate is applied to a qubit, it can be supplemented with a 2×2 identity matrix IDE. For example, in Fig. 1(b), the operation matrices of level 0 and 1 are $M_b \otimes M_a$ and $M_c \otimes IDE$, respectively. Suppose $G_a = X$, controlled-$G_b = CX$, and $G_c = SWAP$. Then, we have

$$O_0 = CX \otimes X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

$$O_1 = SWAP \otimes IDE = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Multi-qubit gates with nonadjacent inputs. It should be noted that when a multi-qubit gate G_x is applied to a quantum circuit, its input qubits may not be adjacent, thus covering more qubits. We use M_x to denote the *complete gate matrix* of G_x in the circuit. If G_x **covers** j qubits, M_x has a size of $2^j \times 2^j$. For example, in Fig. 2(a), controlled-G_b covers three qubits q_2, q_1 and q_0, where q_2 and q_0 are its input qubits, so its complete gate matrix M_b has a size of 8×8, as shown in Fig. 2(b). Similarly, G_c covers three qubits, so M_c is also 8×8 matrices. We have $O_0 = M_b \otimes M_a$ and $O_1 = M_c \otimes IDE$.

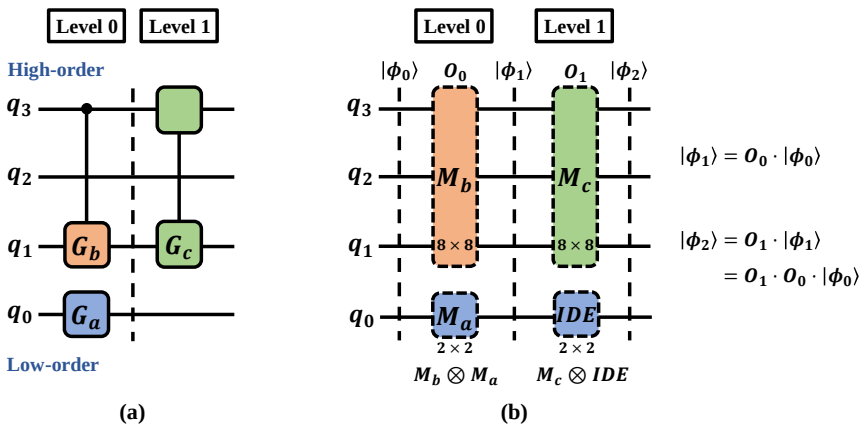

Figure 2: (a) An example of a 3-qubit 2-level quantum circuit with multi-qubit gates applied to nonadjacent qubits. (b) The process of operation matrix simulation (OMSim) for it.

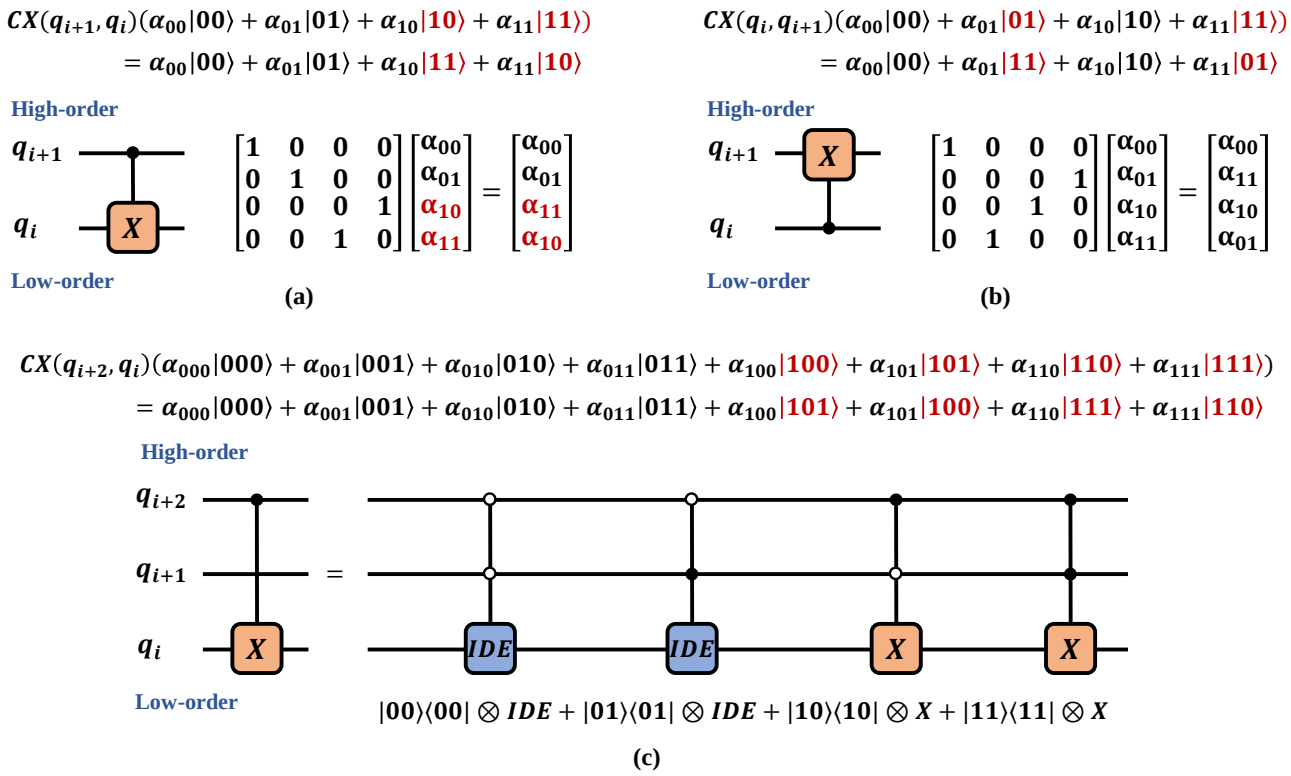

Figure 3: (a) A CX gate with q_{i+1} controlling q_i. (b) A CX gate with q_i controlling q_{i+1}. (c) A CX gate with q_{i+2} controlling q_i.

How to construct such a complete gate matrix M_x? Take the CX gate as an example. Its gate matrix usually appears in the following 4×4 form, representing the situation where q_{i+1} controls q_i, as shown in Fig. 3(a). This matrix cannot be directly decomposed into two 2×2 matrices by tensor product (you may verify it by yourself), but it can be regarded as the sum of two matrices that can be decomposed as follows.

$$CX(q_{i+1}, q_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = |0\rangle\langle 0| \otimes IDE + |1\rangle\langle 1| \otimes X,$$

where the non-control state $|0\rangle$ of q_{i+1} is associated with an identity gate applied to q_i, while the control state $|1\rangle$ of q_{i+1} is associated with an X gate applied to q_i.

As shown in Fig. 3(b), when q_i controls q_{i+1}, the complete gate matrix becomes

$$CX(q_i, q_{i+1}) = IDE \otimes |0\rangle\langle 0| + X \otimes |1\rangle\langle 1| = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

As shown in Fig. 3(c), when q_{i+2} controls q_i, CX covers q_{i+2}, q_{i+1}, and q_i. The control patterns can be regarded as $q_{i+2}q_{i+1} = |10\rangle$ and $|11\rangle$. As a result, the complete gate matrix can be constructed

according to

$$CX(q_{i+2}, q_i) = |00\rangle\langle 00| \otimes IDE + |01\rangle\langle 01| \otimes IDE + |10\rangle\langle 10| \otimes X + |11\rangle\langle 11| \otimes X.$$

Following this procedure, we can obtain the complete gate matrix for a controlled gate with any number of control qubits and one target qubit.

The most commonly used gates with multiple target input qubits are $SWAP$ gates. They will not trouble you since we have already implemented the generation of the gate matrix for an arbitrary $SWAP$ gate in this assignment by swapping rows of an identity matrix. Apart from this method, a $SWAP$ gate can be decomposed into three CX gates as described here.

2.2.2 Updating the State Vector using Operation Matrices

Matrix-vector multiplication. Once the operation matrices are computed, we can use matrix-vector multiplication to update the state vector. The state vector after T levels can be described as

$$|\phi_T\rangle = O_{T-1} \cdot O_{T-2} \cdot \ldots \cdot O_0 \cdot |\phi_0\rangle\,,$$

where $|\phi_0\rangle$ is the initial state vector.

This matrix-vector mathematical model, OMSim, serves as a universal representation of quantum computing, but the overhead is extremely high. For an n-qubit circuit, the time complexity of the multiplication of the $2^n \times 2^n$ operation matrix and 2^n state vector at each level is $O(2^{2n})$. Storing operation matrices also takes huge memory footprints. Simulating large-scale quantum circuits using OMSim is difficult, but it is a very intuitive simulation method for beginners.

3 Getting Started

Please clone the branch `a1-omsim` of the repository QSimLab from GitHub. It contains the necessary files to start this assignment, including the following contents.

qsim/omsim.cpp This is the only file that needs to be modified and is where you write your code. It is unnecessary to make changes to other files.

qsim/ The other files in this folder provide some basic data structures and functions necessary for quantum circuit simulation. Descriptions can be seen in `qsim/README.md`.

main/ This contains the main function of this project.

py/ This folder includes a Python file that uses an OMSim simulator provided by Qiskit. It can be used to verify whether the simulator that you implement works correctly.

Makefile This is the Makefile indicating how to compile this project on Windows.

4 Implementation

This section outlines the functions required to implement in your solution (in `omsim.cpp` only). We will explain them in a top-down approach.

4.1 `Matrix<DTYPE> OMSim(Matrix<DTYPE>& sv, QCircuit& qc)`

This function is called in the main function to simulate a given quantum circuit `qc` using OMSim. The initial state vector is recorded in `sv`, which should be updated during the simulation. It returns the operation matrix O of the entire circuit.

4.2 `Matrix<DTYPE> getCompleteMatrix(QGate& gate)`

This function is called by `OMSim` to get the complete gate matrix of a quantum gate `gate`. In this assignment, **only single-qubit gates** and **2-qubit gates** are considered for simplicity. For any single-qubit gate, the complete gate matrix is exactly its gate matrix. For 2-qubit gates, the testing circuit only contains 2-qubit controlled gates (such as CX) and $SWAP$ gates that may have nonadjacent input qubits. Getting the complete matrix for $SWAP$ is already implemented in `genSwapGateMatrix`.

4.3 `Matrix<DTYPE> genControlledGateMatrix(QGate& gate)`

This function is called by `getCompleteMatrix` to calculate the complete gate matrix of a 2-qubit controlled gate that may have nonadjacent input qubits. The implementation method is described in Section 2.2.1. You can also refer to this article.

5 Assignment Submission

Your assignment should be submitted via https://yunbiz.wps.cn/c/collect/cBtAOXFPmT9 before the due date. For your convenience, a feasible solution will be pushed to the GitHub branch `a1-omsim-ans` on July 19 at 10:00 AM. Feel free to refer to it if needed. The submission format is as follows.

1. If you only modify `omsim.cpp`, please rename it as "`a1-`*`your_name`*`.cpp`" and submit it.
2. If other files have been modified for some reason, please compress the entire project into a zip file, name it as "`a1-`*`your_name`*`.zip`", and upload it. It is strongly recommended to add a README briefly explaining the modifications.