Assignment 3: Distributed Quantum Circuit Simulations

Big Data and Intelligent System Lab @ ECNU

Due date: September 2, 2024, 10:00 AM

1 Introduction

In the previous assignments, we implemented two full-state quantum circuit simulators: the operation matrix simulation (OMSim) and state vector simulation (SVSim). Due to the exponential size of the state vector to the number of qubits, the memory and computation overhead of full-state simulations are exponentially large. The computational resources of a single machine are insufficient. To tackle this issue, some works have utilized distributed clusters or even supercomputers to simulate larger circuits and accelerate simulations.

In this assignment, we will explore two distributed simulation algorithms. **qHiPSTER** [1] proposed a basic implementation of parallel simulations using multiple machines based on SVSim. Despite improving computing and storage power, considerable communication among machines has been introduced. To reduce communication costs, we proposed **QuanPath** [2], which utilizes both SVSim and OMSim. It can significantly eliminate communications when simulating circuits with particular structures.

2 Background

This section briefly introduces qHiPSTER and QuanPath. The simulation task should be partitioned to utilize a distributed cluster with multiple machines. A common method is distributing amplitudes in the state vector evenly across all machines and then performing SVSim to update the state vector.

Assume there are H identical machines, where H is an exponential power of 2, i.e., $H = 2^h$. Given an n-qubit T-level circuit, we divide the state vector of size 2^n into H segments. Let $l = n - h$. Each machine stores $L = N/H = 2^{n-h} = 2^l$ amplitudes. An SVSim operation involves multiple amplitudes. If the stride between amplitudes is too large, the amplitudes involved will be stored on different machines. Then, communications are required.

2.1 qHiPSTER

Fig. 1(a) shows an example circuit and Fig. 1(b) shows four machines. Since each machine stores 2^l amplitudes, from the perspective of qubits, the division of the state vector is equivalent to partitioning n qubits into two sets: high-order qubit set P_0 of size h and low-order qubit set P_1 of size l. For example, when $h = 2$, $P_0 = \{q_{n-1}, q_{n-2}\}$, $P_1 = \{q_{n-3}, \ldots, q_1, q_0\}$.

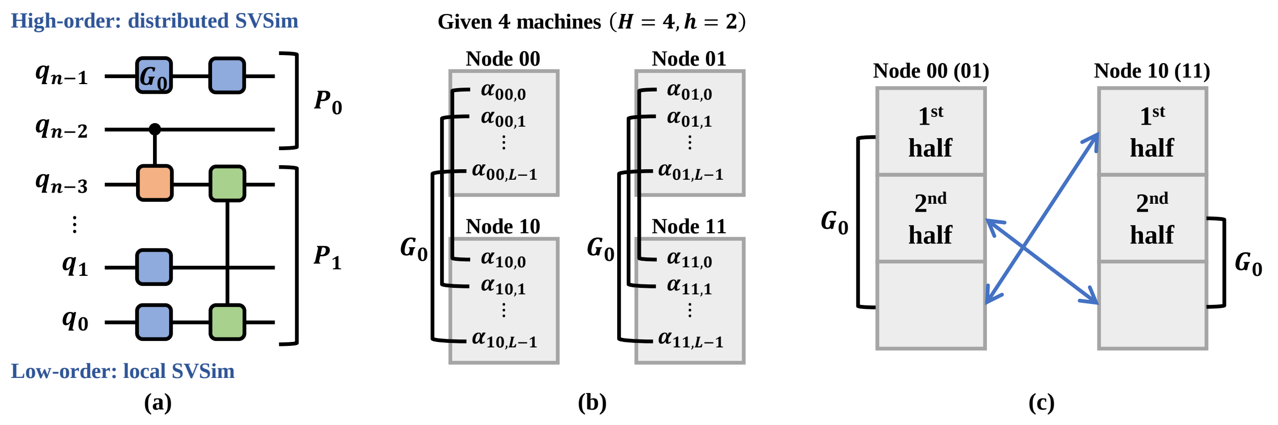

Figure 1: (a) An example of an n-qubit 2-level quantum circuit. (b) Simulating G_0 using distributed SVSim on four machines. (c) Communications between node 00 and node 10 when simulating G_0 in qHiPSTER. The same happens between node 01 and node 11.

Local SVSim. For gates applied to low-order l qubits, as the strides between the amplitudes involved are less than 2^l, SVSim can be performed on each machine locally, namely *local SVSim*. Local SVSim can be conducted parallelly.

Distributed SVSim. For gates applied to the remaining high-order h qubits, the strides exceed 2^l, so SVSim involves amplitudes stored on different machines. We call it *distributed SVSim*. For example, as shown in Fig. 1(b), when simulating G_0 applied to q_{n-1}, the stride is 2^{n-1}, i.e., the first amplitude on node 00 should be paired with the first amplitude on node 10. The paired data should be transmitted to one machine to perform SVSim.

The distributed implementation of qHiPSTER is straightforward as shown in Fig. 1(c). When simulating G_0, communications occur between node 00 and node 10, as well as node 01 and node 11. Each machine uses temporary storage to hold half of the amplitudes coming from its peer. Then, it performs SVSim on the transmitted half of the amplitudes. Finally, it gives back the updated amplitudes to its peer. Distributed SVSim for T levels of high-order h gates results in significant communication costs.

2.2 QuanPath

QuanPath proposes a low-communication technique to eliminate intermediate multi-level distributed SVSim. As shown in Fig. 2, ① for gates on P_0, instead of distributed SVSim, each machine computes a partial operation matrix of size $H \times H$. ② For gates on P_1, local SVSim is performed as in qHiPSTER. ① OMSim and ② SVSim can be done locally on each machine in parallel. ③ Communications are deferred until the final merge, which uses an "$\odot$" operation to combine the results on all nodes. The "$\odot$" operation can be regarded as performing the distributed SVSim of a high-order h-qubit gate at the last level.

It should be noted that the circuit must satisfy a *separable* condition to use QuanPath. Let $Qub(C) = \{q_{n-1}, \ldots, q_0\}$ be the set of input qubits of circuit C and $Qub(G)$ be the set of input qubits of gate G. We first define a concept related to multi-qubit controlled gates as follows.

Definition 1. *For each control qubit q of a multi-qubit controlled gate G at level j, q is called a*

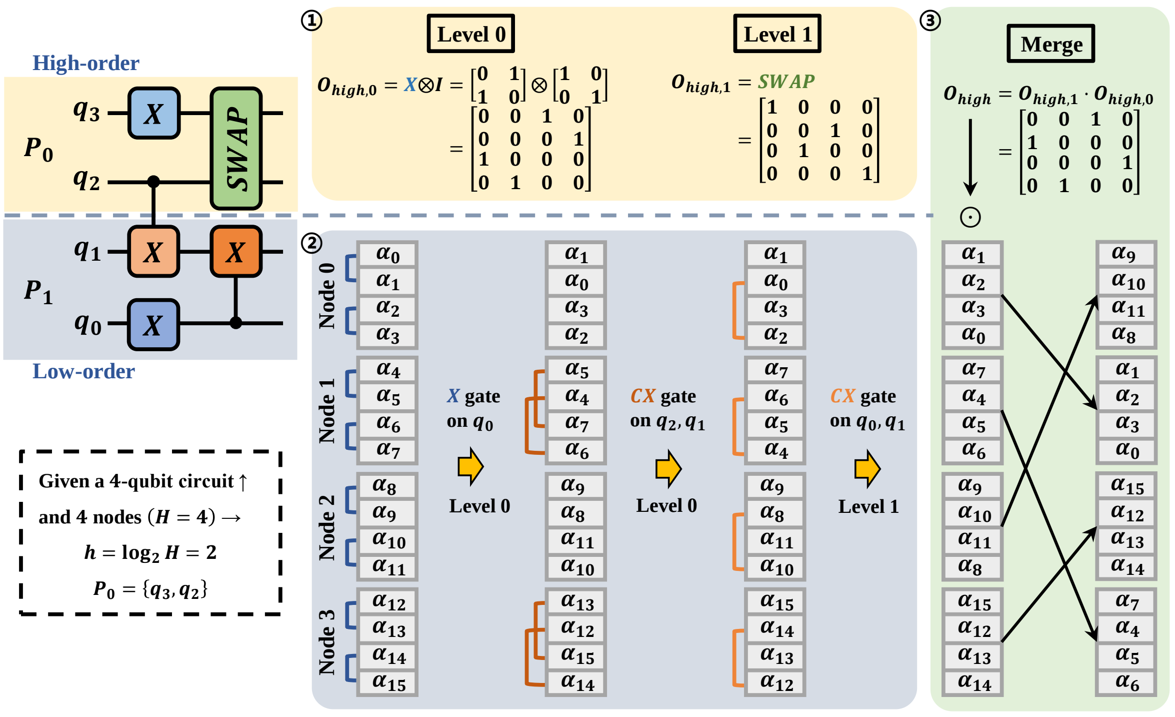
High-order
Low-order
P_0
P_1
q_3 q_2 q_1 q_0
X
SWAP
Given a 4-qubit circuit ↑ and 4 nodes ($H = 4$) → $h = \log_2 H = 2$ $P_0 = \{q_3, q_2\}$
① Level 0
$O_{high,0} = X \otimes I$
Level 1
$O_{high,1} = SWAP$
③ Merge
$O_{high} = O_{high,1} \cdot O_{high,0}$
② Node 0 Node 1 Node 2 Node 3
X gate on q_0 Level 0
CX gate on q_2, q_1 Level 0
CX gate on q_0, q_1 Level 1

Figure 2: The simulation process of QuanPath for a 4-qubit 2-level separable quantum circuit.

clean-control qubit of G if level j is the first level or at any level before level j, q either passes through identity gates or acts as a control qubit. If so, G is called clean-controlled by q.

On this basis, separable circuits can be defined as follows.

Definition 2. *Given a quantum circuit C and the number of high-order qubits h. Let $Qub(C)$ be partitioned into high-order qubit set P_0 ($|P_0| = h$) and low-order qubit set $P_1 = Qub(C) - P_0$. Then, C is "separable" by P_0 if the following condition is satisfied:*

For each multi-qubit gate G in C such that $Qub(G) \cap P_0 \neq \emptyset$ and $Qub(G) \cap P_1 \neq \emptyset$, each qubit in $Qub(G) \cap P_0$ must be a clean-control qubit of G.

If a circuit is separable, the high-order gate operations can be merged and postponed until the end.

3 Getting Started

Please clone the branch `a3-disqsim` of the repository QSimLab from GitHub. It contains the necessary files to start this assignment, including the following contents.

disqsim/ Two files in this folder need to be modified, including `qhipster.cpp` and `quanpath.cpp`. They are where you write your code.

qsim/ Files in this folder provide basic data structures and functions necessary for quantum circuit simulations. Descriptions can be seen in `qsim/README.md`. Please merge your previous work on OMSim and SVSim into this assignment as they are necessary.

main/ This contains the main function of this project.

py/ This folder includes a Python file that uses a Qiskit simulator. 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 Linux.

4 Implementation

This section outlines the functions required to implement in your solution (in `qhipster.cpp` and `quanpath.cpp` only). In this assignment, we use MPI (Message Passing Interface) to conduct distributed simulations. The installation process of MPI on Linux can be found here.

4.1 `qhipster.cpp`

For qHiPSTER, the circuit is simulated level by level. At each level, qubits are processed from low-order to high-order. For low-order qubits, `localSVSimForOneLevel` is called. For high-order qubits, `distributedSVSimForOneLevel` is called.

Please note that `svsimForGate` called by `localSVSimForOneLevel` has been modified since `a2-svsim` in terms of checking a legal control pattern. The function `isLegalControlPattern` has been changed to `isDistributedLegalControlPattern`.

4.1.1 `bool isDistributedLegalControlPattern(ll ampidx, QGate& gate, int numLowQubits, int myRank)`

For single-machine SVSim implemented in `a2-svsim`, we only need to check if the binary index of an amplitude forms a legal control pattern. In this assignment, to simulate a controlled gate in distributed SVSim whose target gate(s) is (are) applied to low-order qubit(s), if a control qubit is low-order, we should check the binary index of the local amplitude `ampidx`. If a control qubit is high-order, we should check the binary index of `myRank`.

4.1.2 `MPI_Comm getPeerComm(int numLowQubits, int numHighQubits, QGate& gate, int myRank)`

This function is called by `distributedSVSimForOneLevel` to find a group of MPI processes that need to communicate with each other when conducting distributed SVSim. This group of processes will perform `MPI_Alltoall` to exchange data.

4.1.3 `pair<vector<ll>, vector<ll>> compactLocalAmp(vector<DTYPE>& sendbuf, QGate& gate)`

This function is called by `distributedSVSimForOneLevel` to construct a send buffer for communications. For a multi-target gate spanning across low-order and high-order qubits, the low-order target may involve discontinuous amplitudes in the local state vector. As a result, after deciding the communication targets by `getPeerComm`, we should reorder the amplitudes in the local state vector, i.e., making the discontinuous amplitudes continuous.

After that, we can use `MPI_Alltoall` to transmit amplitudes, `svsimOnCompactedAmp` to perform SVSim on the compacted receive buffer, `MPI_Alltoall` to give back updated amplitudes, and `recoverLocalAmp` to recover the order of amplitudes. These steps have already been implemented.

4.2 `quanpath.cpp`

The given testing circuit is separable. For QuanPath, we use a thread to calculate the high-order partial operation matrix on each machine. Meanwhile, local SVSim is conducted in parallel. After these two parts are both finished, the merge operation is conducted.

4.2.1 `void* highOMSim(void* arg)`

This is a thread function called by `QuanPath` to calculate a partial operation matrix for high-order qubits. There is one difference from the function `OMSim` implemented in `a1-omsim`. If a controlled gate has a high-order control qubit and a low-order target qubit, the high-order control position filled with a `MARK` gate should be regarded as an identity matrix. Please note that we have already modified `getCompleteMatrix` to return an identity matrix for a `MARK` gate.

4.2.2 `void merge(Matrix<DTYPE>& localSv, Matrix<DTYPE>* ptrOpmat, int numWorkers, int myRank)`

This function is called by `QuanPath` to conduct the merge operation. It can be regarded as performing the distributed SVSim of an h-qubit gate. Therefore, the implementation of the merge operation can be similar to the distributed SVSim implemented for qHiPSTER. The function `MPI_Alltoall` is useful. Details can also be found around Fig. 13 in [2].

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 uploaded to the shared folder on August 26 at 10:00 AM. Feel free to refer to it if needed. The submission format is as follows.

1. If you only modify `qhipster.cpp` and `quanpath.cpp`, please compress them into a zip file, name it as "`a3-`*`your_name`*`.zip`", 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 "`a3-`*`your_name`*`.zip`", and submit it. You can add a README briefly explaining the modifications.

References

[1] M. Smelyanskiy, N. P. Sawaya, and A. Aspuru-Guzik, “qHiPSTER: The quantum high performance software testing environment,” 2016, *arXiv:1601.07195*.

[2] Y. Song, E. H.-M. Sha, Q. Zhuge, W. Xiao, Q. Dai, and L. Xu, “QuanPath: achieving one-step communication for distributed quantum circuit simulation,” *Quantum Inf. Process.*, vol. 23, no. 1, p. 1, 2023.