A Quantum Computing Simulator

Zhikai Hu, Junyi Mei

March 26, 2026

1 Summary

We are going to implement a quantum computing simulator on GPU and CPU platforms, and perform a detailed analysis of the performance on both the systems.

2 Background: Quantum Computing Simulation

2.1 Qubit, Statevector and Entanglement

A simple quantum circuit is shown in figure 1. There are two qubits q_0 and q_1 in this circuit, H and X in the rectangles represent one-qubit quantum gates that apply to the qubits. The dot, vertical line, and $\oplus$ represents a two-qubit quantum gates (the controlled NOT gate, or CNOT) that apply to both the qubits. The last meter-like symbol means doing the measurement on the corresponding qubit and it will output 0 or 1 by some probabilities according to the state of qubits.

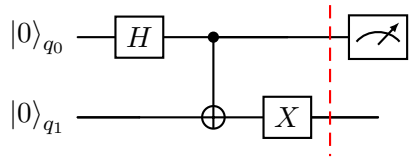

Figure 1: An example of quantum circuit

A qubit represents a piece of state in a quantum circuit, and is represented with a unit vector in $\mathbb{C}^2$, usually written using the coefficient (or amplitude, $\alpha, \beta \in \mathbb{C}$) and basis ($|0\rangle, |1\rangle$).

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle, \text{where } |\alpha|^2 + |\beta|^2 = 1$$

The state of a n-qubit circuit is represented using a dense statevector of 2^n complex numbers.

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

For example, the state of a two qubt circuit can be represented using 4 coefficients.

$$|\psi\rangle = a |00\rangle + b |01\rangle + c |10\rangle + d |11\rangle$$

While some multiple qubit states can be written as the tensor product ($\otimes$) of individual qubits, entangled states cannot. For example, the Bell state cannot be factorized into $(a\,|0\rangle + b\,|1\rangle) \otimes (c\,|0\rangle + d\,|1\rangle)$

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

The statevector can also be denoted in the form of vectors, for example:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix},\ |1\rangle = \begin{bmatrix}0\\1\end{bmatrix},\ \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$$

Also for the two-qubit states, for example:

$$\begin{aligned}(a\,|0\rangle + b\,|1\rangle) \otimes (c\,|0\rangle + d\,|1\rangle) &= \begin{bmatrix}a\\b\end{bmatrix} \otimes \begin{bmatrix}c\\d\end{bmatrix} = \begin{bmatrix}ac\\ad\\bc\\bd\end{bmatrix} = ac\begin{bmatrix}1\\0\\0\\0\end{bmatrix} + ad\begin{bmatrix}0\\1\\0\\0\end{bmatrix} + bc\begin{bmatrix}0\\0\\1\\0\end{bmatrix} + bd\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \\ &= ac\,|00\rangle + ad\,|01\rangle + bc\,|10\rangle + bd\,|11\rangle\end{aligned}$$

The size of the statevector grows exponentially by the number of qubits.

2.2 Quantum Gates

Quantum gates are operators on the vector states. A one-qubit gate is essentially a 2×2 unitary matrix and a two-qubit gate is a 4×4 unitary matrix. An example of one-qubit gate is Pauli-X gate, which inverts the state of a qubit:

$$X\,|0\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$$

Applying gates separately on individual qubits is equivalent to applying the tensor product of these gates to the whole statevector. When only one gate is applied, it is equivalent to applying identity gate (which is an identity matrix) on all other qubits, and the tensor product of all the identity gates and this gate can be applied to the statevector. For example, the three circuits shown in figure 2 below are equivalent. Note that in this example, $|\psi_0\rangle$ and $|\psi_1\rangle$ might be entangled, so we cannot apply U_0 and U_1 separately on the corresponding qubits and do tensor product on the result states to obtain the new statevector. Instead, we must apply them on the whole statevector. However, we also cannot naïvely do tensor product on all the gates (U_0 and U_1 in this example) and then apply the result on the statevector, because the memory required to store the result of this tensor product is $2^n \times 2^n$, which is not realistic when the number of qubits is large. So, we need to apply them gate-by-gate, but also on the whole statevector.

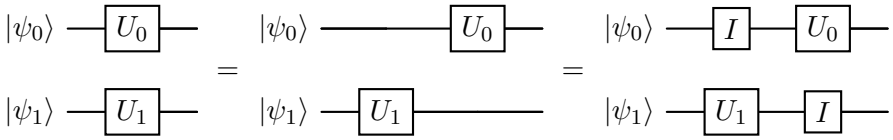

Figure 2: Example of equivalent quantum circuits

The application of a gate is illustrated as follows. Let the state of a n-qubit circuit be $|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$. x is actually the binary representation of the index of the coefficients (or amplitude). Suppose $x = \sum_{r=0}^{n-1} b_r 2^r$ where $b_r \in \{0,1\}$, and the gate to be applied is:

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}$$

Suppose it applies on qubit k, then new amplitudes after application is

$$\alpha'_{b_{n-1}\cdots b_{k+1} j\, b_{k-1}\cdots b_0} = \sum_{i\in\{0,1\}} U_{ji} \alpha_{b_{n-1}\cdots b_{k+1} i\, b_{k-1}\cdots b_0}, \text{where } j \in \{0,1\}$$

This also means that the simulation must visit the entire statevector. If i_0 is an index whose k-th bit is 0 under binary representation, and $i_1 = i_0 + 2^k$, then the corresponding amplitudes needs to be updated by:

$$\begin{bmatrix} \alpha'_{i_0} \\ \alpha'_{i_1} \end{bmatrix} = U \begin{bmatrix} \alpha_{i_0} \\ \alpha_{i_1} \end{bmatrix}$$

This needs to be done for all such pairs of i_0 and i_1. The application on two-qubit gates is similar to this. All amplitude coefficients need to be modified to update.

2.3 Schrödinger-style Statevector Simulation

The algorithm of simulation on a circuit is shown below.

Algorithm 1 ApplyOneQubitGate

Require: statevector of amplitude $\psi \in \mathbb{C}^{2^n}$
Require: target qubit index k.
Require: gate $U \in \mathbb{C}^{2\times 2}$
1: $\mathcal{P}_k \leftarrow \{(i_0, i_1) \mid i_0 \text{ and } i_1 \text{ only differ on bit } k\}$
2: **for all** $(i_0, i_1) \in \mathcal{P}_k$ **do**
3: $\begin{bmatrix} \psi_{i_0} \\ \psi_{i_1} \end{bmatrix} \leftarrow U \begin{bmatrix} \psi_{i_0} \\ \psi_{i_1} \end{bmatrix}$
4: **end for**
5: **return** ψ

Algorithm 2 ApplyTwoQubitGate

Require: statevector of amplitude $\psi \in \mathbb{C}^{2^n}$
Require: target qubit index m, n.
Require: gate $U \in \mathbb{C}^{4\times 4}$
1: $\mathcal{P}_{m,n} \leftarrow \{(i_{00}, i_{01}, i_{10}, i_{11}) \mid i_{00}, i_{01}, i_{10} \text{ and } i_{11} \text{ only differ on bit } m, n\}$
2: **for all** $(i_{00}, i_{01}, i_{10}, i_{11}) \in \mathcal{P}_{m,n}$ **do**
3: $\begin{bmatrix} \psi_{i_{00}} \\ \psi_{i_{01}} \\ \psi_{i_{10}} \\ \psi_{i_{11}} \end{bmatrix} \leftarrow U \begin{bmatrix} \psi_{i_{00}} \\ \psi_{i_{01}} \\ \psi_{i_{10}} \\ \psi_{i_{11}} \end{bmatrix}$
4: **end for**
5: **return** ψ

Algorithm 3 SimulateCircuit

Require: initial statevector $\psi_0 \in \mathbb{C}^{2^n}$
Require: circuit $\mathcal{C} = (G_1, G_2, \cdots, G_\ell)$
$\psi \leftarrow \psi_0$
for all $G \in \mathcal{C}$ **do**
    if $G = (k, U)$ is a one-qubit gate **then**
        $\psi \leftarrow \textsc{ApplyOneQubitGate}(\psi, k, U)$
    else if $G = (m, n, U)$ is a two-qubit gate **then**
        $\psi \leftarrow \textsc{ApplyTwoQubitGate}(\psi, m, n, U)$
    end if
end for
return ψ

3 The Challenge

As the algorithm of simulation shows above, within a single gate, it is highly data-parallel and straightforward to implement. However, it is difficult to optimize globally across gates because all the gates are updating on the same global statevector, and there can be dependencies between different gates (e.g., The two-qubit and one-qubit gates in figure 1).

The performance is also highly dependent on the input quantum circuit. When applying one gate, the locality of memory access depends on the index of the target qubit. When applying on low-indexed qubits, the distance of (i_0, i_1) is smaller (as mentioned above, $i_1 = i_0 + 2^k$), and the spatial locality is better, but on high-indexed qubits, the locality will get worse.

To ensure correctness, on GPU, the kernel must be re-launched on a per-gate basis to synchronize all threads (across blocks) after the statevector is updated. This will introduce a large overhead. In addition, the qubit-relevant memory access pattern mentioned above makes it harder to utilize the memory coalescing feature in CUDA.

4 Resources

We will make use of the standard description language of quantum circuits, OpenQASM, as the input, but we will not write the parser by ourselves. Instead, we are going to use qasmtools to parse the programs. For benchmarks, we will use the programs in QASMBench.

We will also refer to the methods proposed by Fatima and Markov [1].

5 Goals and Deliverables

5.1 Plan to Achieve

We plan to implement the within-gate parallelism on CPU and GPU. The simulator will accept QASM 2.0 program and output the statevector or the measurement results. It should be able to

handle the small and medium sets of benchmarks in QASMBench. We are not planning to run the QASM programs with mid-circuit measurement.

We also plan to implement at least one other parallelization method in addition to straightforward within-gate parallelism, potentially an across-gate approach which should run a bunch of gates in parallel.

We plan to carry out a detailed performance analysis on the result simulator and compare the bottlenecks of it on CPU and GPU, and deliver a detailed performance comparison.

5.2 Hope to Achieve

We hope to implement the mid-circuit measurement in the simulator and examine how it will affect the parallelism and the performance. We also hope to implement some gate fusion techniques or try to implement dedicate kernels for some gates.

6 Platform Choice

We choose to implement the simulator in C++ and CUDA. On CPU, we are going to try to use OpenMP for the within-gate parallelism.

7 Schedule

Weeks	Target	Time Period
Week 1	Project planning	2026-03-23 - 2026-03-29
Week 2	Exam. QSAM Parser. CPU within-gates	2026-03-30 - 2026-04-05
Week 3	GPU within-gate. CPU/GPU Across Gates Trial	2026-04-06 - 2026-04-12
Week 4	CPU/GPU Gates Fusion	2026-04-13 - 2026-04-19
Week 5	Poster & Report	2026-04-20 - 2026-04-26

A Project Webpage

15418 Project webpage

References

[1] Aneeqa Fatima and Igor L. Markov. Faster schrödinger-style simulation of quantum circuits. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 194–207, 2021.