Utilizing GPGPUs for simulating quantum circuit

Yuanfei Xu 1st, 1.a, Lixing Zhang 2nd, 2.b

1School of Mechanical and Electrical Engineering Wuhan University of Technology Wuhan, China
2School of Computer Science and Technology Wuhan University of Technology Wuhan, China
axuyuanfei@whut.edu.cn, b2509957071@whut.edu.cn

Abstract. State vector describing the state of an n-qubit quantum system requires $2^n$ complex numbers, which means the memory usage of simulating a quantum computer grows exponentially with respect to the number of qubits. For the memory of a single hardware is limited, distributed computation is necessary for simulating quantum systems with large number of qubits. In our proposal, we developed a multi-GPU solution. State vector is divided into several chunks and allocated to different GPU device. Making it possible to simulate quantum circuits of any size on appropriate GPU cluster. We analyse the theoretical computation resource and come to the conclusion that memory copy bandwidth between GPUs is the bottleneck of time consumption. Total execution time is $O(\log DN/DN)$ if peer-to-peer memory copy is supported, otherwise time elapsed is $O(\log DN)$ where $DN$ is number of GPU devices. Memory consumption on one GPU is $O(1/DN)$.

1. Introduction
Quantum computing has the potential to make computers far more capable than they are today. However, the field is still in its early stage where simulating quantum system using the classical computers is still vital for the study of new algorithms and architectures.

Unfortunately, state vector describing the state of an n-qubit quantum system requires $2^n$ complex numbers, which means the memory usage of simulating a quantum computer grows exponentially with respect to the number of qubits. In order to mitigate the computational requirements, single-instruction multiple-threaded (SIMT) architecture and distributed computation can be of great help.

Many distributed strategy on CPUs has been developed. OpenMP and MPI hybrid parallelization is used to simulate 33 qubits on 64 nodes in [1]. 45-qubit circuit is simulated on 8192 nodes and 0.5 petabytes of memory in [2]. When the qubit count is above 50, the realm where quantum supremacy reigns, both space and time consumption can be unacceptable even on most advanced supercomputer. Full state vector of 50-qubits takes about 16-PB of memory. Low-depth circuits with more than 49 qubits can be simulated using specially designed method. Geferral of entanglement gates is used to simulate circuit with depth 27 qubits in a 2D lattice of 7×7 in [3]. Feynman path method provides a solution to 7×8 qubits of depth 30 [4].

The operation on state vector is exceedingly separable and parallelizable, which means GPU accelerating can dramatically speed up the process. NVIDIA’s Compute Unified Device Architecture (CUDA) [7] provides high computation and memory bandwidth. We propose a distributed strategy on GPUs and theoretically make it possible to simulate quantum circuits of any size on appropriate GPU cluster.
2. Quantum Compute Theory

2.1. Quantum system

In this section, we briefly introduce basic theory of quantum computing. While classical computers manipulate electrical systems for computational purposes, quantum computers manipulate quantum systems.

In order to manipulate a quantum system, we have to describe its full state first. According to the first fundamental assumption of quantum mechanics, we can describe the state of a quantum system by a complex vector.

\[ c = \begin{pmatrix} a_1 + ib_1 \\ a_2 + ib_2 \\ \vdots \\ a_n + ib_n \end{pmatrix} \]  

(1)

Single quantum bit is the simplest quantum system. Its state vector can be represented as a linear combination of two base vectors.

\[ |ψ⟩ = α|0⟩ + β|1⟩ \]  

(2)

where \( α \) and \( β \) are complex numbers satisfying the normalization condition.

\[ |α|^2 + |β|^2 = αα^* + ββ^* = 1 \]  

(3)

where are base vectors of single qubit state space.

For quantum system with more than 1 qubit, the state of is represented by direct product of each qubit. For example, we can represent the state of quantum system by a single vector size \( 2^3 = 8 \).

\[ \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} \]  

(4)

where \( a, b, \ldots, f \) are all complex numbers. In general, the state space of \( n \) Qubits is

\[ \mathbb{C}^{2^n} = \left\{ \sum_{x \in \{0,1\}^n} a_x |x⟩ : a_x \in \mathbb{C} \right\} \]  

(5)

The general qubit form is linear combination of \( 2^n \) bit-strings

\[ |ψ⟩ = α_0|00 \ldots 00⟩ + α_1|00 \ldots 01⟩ + \ldots + α_{2^n-1}|11 \ldots 11⟩ \]  

(6)

We can easily see that it takes \( 2^n \) complex numbers to represent \( n \) Qubits in a classical computer, that’s why simulating quantum computer is extremely expensive.

2.2. Quantum gate

The second basic assumption of quantum mechanics is that the state of an isolated quantum system in two moments are interrelated by the unitary operator. Operator or gate can be represented as matrix,

Pauli gate

Simplest quantum gate manipulates one qubit, Pauli gate for example is denoted as

\[ X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \]  

(7)

Obviously we have \( X|0⟩ = |1⟩, X|1⟩ = |0⟩ \).

Controlled-Not gate

Controlled gate manipulates more than one Qubits. CNOT gate, for example, is denoted as
C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (8)

Obviously we have

\begin{align*}
C|00⟩ &= |00⟩, C|01⟩ = |01⟩, \\
C|10⟩ &= |11⟩, C|11⟩ = |10⟩,
\end{align*}

where controlQubit is \( q_0 \) and targetQubit is \( q_1 \).

The targetQubit only performs the NOT operation when the controlQubit is \( |1⟩ \), otherwise it remains the same.

### 2.3. Quantum measurement

The third basic hypothesis of quantum mechanics: measurement in state space \( H_n \) can be expressed by operator \( \{M_m\} \), where \( 1, 2, \ldots, m \) indexes all possible measuring results.

The Born rule [6] gives the probability for quantum system to collapse to state \( m \). If system with status vector is measured by \( M_m \), then the probability of results \( m \)

\[ p(m) = |⟨M_m|ψ⟩|^2 = ⟨ψ|M_m^∗ M_m|ψ⟩. \quad (9) \]

We have normalization condition

\[ \sum_m p(m) = 1, \quad (10) \]

which is a constraint condition of the measurement operator.

### 2.4. Quantum circuit

Quantum circuit is a sequence of quantum gates imposed on several qubits. Similar to classical computers, qubits is initialized first (to zero state often). Then designed quantum gates are imposed on these qubits. At last, qubits is measured and the results are stored in classical bits. Taking an example, figure 1 is a quantum circuit designed to calculate \( 1 + 1 \). \( q_0, q_1, q_2, q_3 \) are qubits with initial state \( |0⟩ \). \( c_0, c_1 \) are classical bits to store the measurement results. After Pauli gate, numbers to be operated \( q_0 = 1, q_1 = 1 \) is stored into quantum system. The final simulation result is that all trials get \( c_0 = 0, c_1 = 1 \). Bit-string \( c_1c_0 \) is 2 in binary representation.

**Figure.1** Quantum circuit implementing \( 1 + 1 \).

**Fig. 2a** shows a Pauli gate imposed on qubit \( q_0 \). **Fig. 2b** shows a CNOT gate (controlled-not gate) whose controlQubit is \( q_0 \) and targetQubit is \( q_1 \). **Fig. 2c** shows a Toffoli gate (controlled-controlled-not gate) whose controlQubit is \( q_0, q_1 \) and targetQubit is \( q_2 \).

Quantum gates in a circuit boils down to processing elements in state vector which can be executed parallelly. Each quantum gate corresponds to a kernel function. A kernel function is executed by a grid of thread blocks and thread block is a batch of threads that can cooperate with each other as is shown in Fig. 3. This is the basic CUDA programming model for massively parallel processing.

Mathematical principle and algorithm for QuEST can be found in [8].
Figure 2: Quantum gate in circuit

Figure 3: kernel function.

3. Theoretical Resource Analysis
In this section, we present performance analysis of modified QuEST project (Quantum Exact Simulation Toolkit) [8].

In our multi-GPU solution, state vector is divided into chunks and allocated to different GPU devices. Each device carries out gate implementation concurrently. Multi-GPU solution has two advantages:

- Total time consumption reduces as number of GPU device increases
- Memory consumption on each device reduces as number of GPU device increases

There’s in fact a trade-off between total time consumption and memory usage on a single GPU. If the state vector is divided into more chunks, more quantum gates will have to involve data exchange. In this section, we will analyse time and memory consumption with regard to number of GPUs. Following assumptions are made.

- Number of qubits is QBN, number of GPUs is DN, number of CUDAStream when data exchange is involve is SN, number of quantum gates is QGN.
- Size of state vector is $2^{QBN \cdot AS}$
- All kernel functions have the same bandwidth KFB.
- Memcpy has a fixed bandwidth MCB.
- For all quantum gates in the circuit, value of targetQubit is equally likely to be any possible value \{0,1,...,QBN - 1\}.

3.1. Total Elapsed Time
Intuitively, when task is distributed into more device, the total time consumption should decrease. But that’s not always the case. State vector is operated by block, the size of which depends on targetQubit,

$$\text{blockSize} = 2^{\text{targetQubit}} + 1.$$  \hspace{1cm} (11)
As long as
\[
\text{blockSize} \leq \text{numAmpsPerChunkor},
\]
or
\[
\text{targetQubit} \leq QBN - \log_2^{DN} - 1,
\]
data on local GPU is enough to implement the quantum gate. But when
\[
\text{targetQubit} > QBN - \log_2^{DN} - 1,
\]
there must be data exchange between GPU devices which takes more time than implementing kernel function. To reduce the impact of data exchange on execution time, we use CUDAStream to enable memory copy to execute concurrently with quantum gate.

If quantum system is simulated in single GPU device, execution time of one quantum gate is
\[
QGT_{\text{local}} = \frac{2^{QBNA\text{S}}}{DN \cdot KFB}.
\]

![CUDAStream timeline with peer-to-peer memcpy.](image)

CUDA stream [9] is a runtime mechanism allowing CUDA kernels and memory copy to run concurrently. For gate involving data exchange, CUDAStream helps to speed up. In ideal condition, bi-directional memcpy between a pair of GPUs starts and ends simultaneously as shown in Fig. 4. So time consumed is
\[
QGT_{\text{memcpy}} = \frac{2^{QBNA\text{S}}}{DN} \left( \frac{1}{MCB} + \frac{1}{KFB \cdot SN} \right).
\]

On most platforms, memory copy tend to have much smaller bandwidth than kernel function $MCB \ll KFB$,
therefore,
\[
QGT_{\text{memcpy}} \approx \begin{cases} \frac{2^{QBNA\text{S}}}{DN \cdot MCB}, & P2P = 1, \\ \frac{2^{QBNA\text{S}}}{MCB}, & P2P = 0. \end{cases}
\]

where $P2P = 1$ means peer-to-peer memcpy [10] between GPUs is supported. If $P2P = 0$, GPU device can only exchange data directly with the host(CPU memory), which means data exchange between each pair of GPUs is serial. If $P2P = 1$, each pair of GPUs can exchange data concurrently. In our platform, peer-to-peer memcpy is not supported. Fig. 7 shows timeline of CUDAStreams. We can see that data exchange between GPU2 and GPU3 did not start until GPU0 and GPU1 finished data exchange.
In each gate, every qubits have same probability to be target qubit, therefore, the expectations of execution time is

\[ QGT = \left(1 - \frac{\log_{2}^{QBN}}{QBN}\right) QG_{\text{local}} + \frac{\log_{2}^{DN}}{QBN} QG_{\text{memcpy}}. \]  

(14)

total elapsed time \( TET = QGN \cdot QGT \) is

\[ TET \approx \begin{cases} 
2^{QBN} \frac{AS}{QBN} \left( \frac{\log_{2}^{DN}}{MCB \cdot DN} + \frac{QBN}{KFB \cdot DN} \right), & P2P = 1, \\
2^{QBN} \frac{AS}{QBN} \left( \frac{\log_{2}^{DN}}{MCB} + \frac{QBN}{KFB \cdot DN} \right), & P2P = 0.
\end{cases} \]

(15)

With the big O notation,

\[ TET(DN) = O(\log DN), \]
which means elapsed time of simulating the circuit grows as the logarithm of DN. But if peer-to-peer memcpy is supported, as is shown in Fig. 9,

\[ TET(DN) = O\left(\log \frac{DN}{DN}\right), \]

which means more GPU devices do provide a speed-up effect. In our platform, memcpy and kernel function bandwidth ratio

\[ MKR = \frac{MCB}{KFB} \]

is roughly between 0.1 and 0.01.

Theoretical time consumption for workload random.c with respect to number of GPUs is shown in Fig. 8 shows execution time without peer-to-peer memcpy.

### 3.2. Memory Consumption

Memory on each GPU is mainly consumed by deviceStateVec and devicePairStateVec. deviceStateVec is local state vector with size

\[ SVS = \frac{2^{QBNA}}{SN}. \tag{16} \]

When data exchange is involved, devicePairStateVec is used to cache the received state vector and in each CUDAStream, only 1/SN the size of state vector is received. Therefore, size of devicePairStateVec is only

\[ PVS = \frac{2^{QBNA}}{DN\cdot SN}. \tag{17} \]

Only when DN > 1, devicePairStateVec is needed and

\[ SGM = SVS + PVS. \]

\[ SGM = \begin{cases} 2^{QBNA}, DN = 1, \\ (1 + \frac{1}{SN})\frac{2^{QBNA}}{DN}, DB = 2, 4, 8, \ldots \end{cases} \tag{18} \]
In our workloads, \( \text{QBN} = 30 \) and \( 2^{\text{QBN}} \text{ AS} = 16597 \) approximatively. Fig. 10 shows single GPU memory consumption. With relatively large SN,\[
SGM \approx \frac{2^{\text{QBN}\text{ AS}}}{DN}.
\] (19)

4. Experiment

4.1. Task
Our workloads are quantum random circuit (random.c) and the quantum Fourier transform circuits (GHZ_QFT.c).

4.2. Result
In our computing platforms, memcpy bandwidth is not specially optimized, only 8.32 GB/s on average. In GHZ_QFT.c, many quantum gate has targetQubit = 28. When the whole state vector is divided into 2 part instead of 4, they don’t involve data exchange. That explains why it takes much more time executing GHZ_QFT.c on 4 Tesla T4. Our multiGPU solution provides a correct result and actual resource consumption TABLE 1, 2 accords with the theoretical analysis equation 20, 21.

| Workload name | Memory Consumption on each GPU/MiB |
|---------------|-----------------------------------|
|               | 2*Tesla T4                        | 4*Tesla T4                     |
| GHZ_QFT.c     | 10363                             | 5227                            |
| random.c      | 10363                             | 5227                            |

| Workload name | Elapsed time/s |
|---------------|----------------|
|               | 2*Tesla T4     | 4*Tesla T4     |
| GHZ_QFT.c     | 24.773         | 65.052          |
| random.c      | 53.009         | 60.037          |

5. Conclusion
In conclusion, our proposed multiGPU solution make it possible to simulate quantum circuits of any size on appropriate GPU cluster. Memory consumption on each GPU(SGM) is guaranteed to scale down as device number increases.

\[
SGM(DN) = O\left(\frac{1}{DN}\right)
\] (20)

Memory consumption is only relevant with number of qubits. It scales down as number of GPU device increases. For total memory on one GPU is limited, this feature is highly helpful when simulating large quantum circuit.
On platforms where GPU devices can exchange data directly instead of using CPU memory as intermediary cache, multiGPU solution speeds up the simulation. But without peer-to-peer memcpy support, there is a trade-off between total time consumption and memory usage.

\[
TET(DN) = \begin{cases} 
O(\log DN), & P2P = 0 \\
O\left(\frac{\log DN}{DN}\right), & P2P = 1. 
\end{cases}
\] (21)

References

[1] Larose R. Distributed Memory Techniques for Classical Simulation of Quantum Circuits[J]. 2018.
[2] Hner, Thomas, and D. S. Steiger. "0.5 petabyte simulation of a 45-qubit quantum circuit." the International Conference for High Performance Computing, Networking, Storage and Analysis 2017.
[3] Pednault E, Gunnels J A, Nannicini G, et al. Breaking the 49 Qubit Barrier in the Simulation of Quantum Circuits[J]. 2017.
[4] Pednault, Edwin, et al. "Breaking the 49-Qubit Barrier in the Simulation of Quantum Circuits." 2017.
[5] Chen Z Y, Zhou Q, Xue C, et al. “64-Qubit Quantum Circuit Simulation”[J]. Science Bulletin, 2018, v.63(15):22-29.
[6] F. Logiurato and A. Smerzi, “Born Rule and Noncontextual Probability,” Journal of Modern Physics, Vol. 3 No. 11, 2012, pp. 1802-1812. doi: 10.4236/jmp.2012.311225.
[7] Kirk D. “NVIDIA CUDA software and GPU parallel computing architecture”[C] ISMM. 2007, 7: 103-104.
[8] Jones T, Brown A, Bush I, et al. “QuEST and High Performance Simulation of Quantum Computers”[J]. Scientific Reports.
[9] H. Li, D. Yu, A. Kumar and Y. Tu, "Performance modeling in CUDA streams — A means for high-throughput data processing," 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, 2014, pp. 301-310, doi: 10.1109/BigData.2014.7004245.
[10] Schroeder T C. “Peer-to-peer & unified virtual addressing”[C] GPU Technology Conference, NVIDIA. 2011.