Quantum Supremacy Circuit Simulation on Sunway TaihuLight

Riling Li, Bujiao Wu, Mingsheng Ying, Xiaoming Sun, and Guangwen Yang

Abstract—With the rapid progress made by industry and academia, quantum computers with dozens of qubits or even larger size are being realized. However, the fidelity of existing quantum computers often sharply decreases as the circuit depth increases. Thus, an ideal quantum circuit simulator on classical computers, especially on high-performance computers, is needed for benchmarking and validation. We design a large-scale simulator of universal random quantum circuits, often called "quantum supremacy circuits", and implement it on Sunway TaihuLight. The simulator can be used to accomplish the following two tasks: 1) Computing a complete output state-vector; 2) Calculating one or a few amplitudes. We target the simulation of 49-qubit circuits. For task 1), we successfully simulate such a circuit of depth 39, and for task 2) we reach the 55-depth level. To the best of our knowledge, both of the simulation results reach the largest depth for 49-qubit quantum supremacy circuits.

Index Terms—Quantum computing, quantum circuit simulation, Sunway TaihuLight

1 Introduction

The concept of quantum computer was proposed almost four decades ago [8]. But until recently it had been unknown whether quantum computers can indeed exceed the computing capability of their classical predecessors. Thanks to the progress made by industry and academia in recent years, practical quantum computing might become reality soon. However, before a commercial quantum computer is launched on market, many tests and verification need to be done. One of the most important is to test the fidelity of quantum circuit. One way to accomplish this is to simulate quantum circuits by computing the ideal state amplitudes on a classical computer. A quantum simulator on classical computer could also be used to verify correctness of certain quantum algorithms and help the design of new quantum algorithms. Besides, quantum circuit simulator implemented on supercomputers benchmark the frontier of "quantum supremacy", which is the potential ability of quantum computing devices to solve problems that classical computers practically cannot.

Quite a few implementations of quantum circuit simulators have been developed in last few years [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25]. And [2], [26], [27] discuss complexity problems of random circuit simulation. We design a simulator on Sunway TaihuLight, which aims at 49-qubit circuits. Such circuits are hard to directly simulate on other supercomputers due to the limited memory spaces, and it is widely believed that near-term quantum device will first achieve quantum supremacy at this scale. Our simulator can accomplish the following:

- *Task 1*: Computing a complete output state-vector representing a quantum state output by the simulated quantum circuit;
- *Task 2*: Sampling (i.e., calculating a small amount of) the amplitudes of a quantum state output by the quantum circuit.

For Task 1, we are able to solve the lattice of 7×7 qubits with depth 39[1] in 4.2 hours using 131072 core groups, which is around 80 percent of the computing resource of Sunway. As to Task 2, our method can calculate one amplitude for 49-qubit circuits of depth 55. Moreover, our method for Task 1 can also be directly extended to the lattices of 7×8, 8×8, and 9×8 qubits for calculating a few amplitudes, though the reachable depth of the random circuit would be decreasing with the increasing of the qubits. Fig. 1 shows the maximal depth our simulator can reach for different number of qubits.

1.1 Comparison with Other Quantum Simulators

For Task 1, related works in recent years include [15], [16], [17], [18], [19]; in particular:

- The simulator described in [17] can simulate a random circuit with 5×9 qubits and depth 25 on the

- R. Li and G. Yang are with the Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China. E-mail: {rl-li16}@mails.tsinghua.edu.cn, ygw.
- M. Ying is with the Centre for Quantum Software and Information, University of Technology Sydney, NSW 2007, Sydney, Australia, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, and Department of Computer Science and Technology, Tsinghua University, Beijing, China. E-mail: mingsheng.ying@uts.edu.au.
- B. Wu and X. Sun are with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100864, China, and also with the University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: {wubujiao, sunxiaoming}@ict.ac.cn.

Manuscript received 19 Dec. 2018; revised 17 Sept. 2019; accepted 28 Sept. 2019. Date of publication 15 Oct. 2019; date of current version 10 Jan. 2020. (Corresponding author: Riling Li.)
Recommended for acceptance by M. Kandemir.
Digital Object Identifier no. 10.1109/TPDS.2019.2947511

[1] The first layer of Hadamard gates are not counted as the circuit depth, we treat it as layer 0. So the circuit we simulate is from layer 0 to layer 39. This also holds for Task 2.

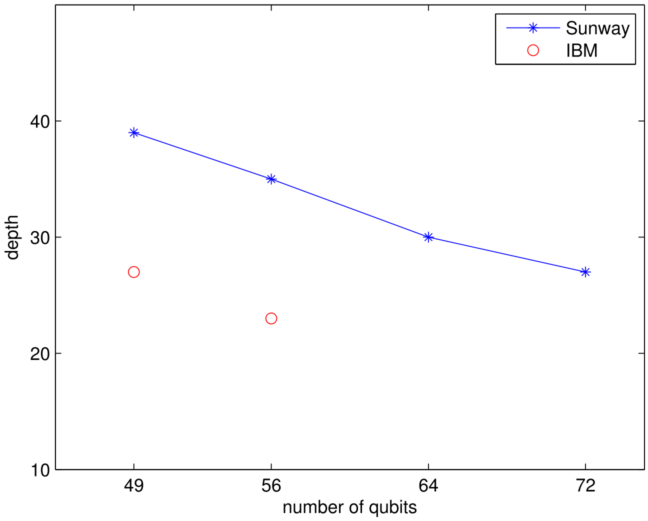


Fig. 1. The maximal circuit depth we can simulate in cases of different number of qubits. Both our simulator and IBM [18] can calculate all 2^{49} amplitudes for 49-qubit case. For 56- or more qubit cases, we only calculate a slice of 2^{32} amplitudes for purpose of demonstration.

Cori II supercomputer in less than 10 minutes, using 0.5 petabytes and 8192 nodes.

- Reference [18] reports a simulation of a 7 × 7 grid of qubits with depth 27, which costs more than one day on IBM Blue Gene/Q.
- The simulator discussed in [19] is also implemented on Sunway. With the adaptive coding technique they can simulate circuits with up to 48 qubits, without loss much accuracy. However, they does not use the computing processing elements (CPEs) of Sunway, hence unable to make most of the computing power of it.

In contrast to these works, our method can simulate a 7×7 grid of qubits with depth 39. It is worth noting that the task of simulating 7×8 grid of qubits with depth 23 was not finished in [18] because 2^{56} amplitudes are too many to calculate. We also calculate 2^{42} amplitudes for 7×8-qubit circuits of depth 35, but only for demonstration. Table 1 compares our implementation and several previous works. Our results of Task 1 reaches the *largest* depth for 49-qubit circuit simulation to our knowledge.

Task 2 of sampling amplitudes can be used to estimate the fidelity by computing the cross entropy, which usually needs $10^3 \sim 10^6$ samples [5]. The sampling target is to calculate an amplitude α_x, where $0 \leq x \leq 2^n - 1$ for an n-qubit circuit. The most popular method for this task is tensor network contraction [20], [21], [22], [23], [24]. In [20], the task of solving one amplitude is finished on a single workstation for 49-qubit circuits with depth 30. In [22], one amplitude is calclulated in Ali Cloud distributed system for 49-qubit circuit with depth 53. Though tensor network contraction technique is very suitable for calculating one amplitude, a drawback is that the performance is impacted by the number of T gate in the circuit. While we have not used the tensor network contraction, our method of calculating the state-vector can also be directly applied to calculate a small number of amplitudes without that drawback. More precisely, we can get one amplitude for 7×7-qubit circuits of depth 55 by calculating the inner product of two 49-qubit state-vector, although this is rather expensive.

TABLE 1
Several Quantum Circuit Simulators Implemented on Supercomputers in Recent Years

Reference	Platform	Qubits	Depth	Time
ETH[17]	Cori II	45	25	10 minutes
IBM[18]	Blue Gene/Q	49	27	2 days
Sunway	Sunway TaihuLight	49	39	4.2 hours

Our simulator reaches the largest number of qubits and also the largest depth for 49-qubit circuits in the case of solving all amplitudes of the final state (Task 1).

1.2 Technical Contributions of this Paper

The first contribution of this paper is that we introduce a novel partition scheme via a technique called implicit decomposition and a dynamic programming algorithm, which enables us to save more time and space than [18]. This partition scheme needs a great amount of network communication, so we also propose an optimization strategy for reducing it when implementing our method.

Our second contribution is some new *local* optimizing techniques on a single node, which take the advantage of the many-core heterogeneous processor of Sunway. Our optimization greatly reduces the amount of memory access while it only increases a small quantity of calculation as shown in Section 3.4. We also apply some standard optimizations such as vectorization. By all of these techniques, we can simulate a 28-qubit circuit on one core group very quickly as shown in Section 3.4, which is significant for improving the performance of 49-qubit circuit simulation.

1.3 Organisation of the Paper

Section 2 gives a brief introduction to Sunway TaihuLight. Section 3 describe the simulation methodologies and optimization techniques. Section 4 presents the numerical results of our implementation as well as verification of the results. Section 5 discusses portability of our techniques to other supercomputers as well as logical structure of optimizations. Section 6 draws a conclusion and discusses the problems to be solved in the future work.

2 The Sunway TaihuLight Supercomputer

Released in June 2016, the Sunway TaihuLight Supercomputer [9] is so far the largest computing system China has ever developed. The whole system consists of 40960 homegrown CPUs called SW26010, and is able to provide a peak performance of 125 PFlops, a sustainable Linpack performance of 93 PFlops, ranking No.1 for four times in the TOP 500 list, in the year of 2016 and 2017.

Each of the SW26010 processor consists of four core groups(CG) and 32 GB memory. Within one CG, there are 1 management processing elements (MPEs) corresponding to the MPE, and 64 computing processing elements (CPEs) corresponding to the CPEs. The 64 CPE cores are organised as a 8×8 mesh. Each MPE has a 32 KB L1 data cache and a 256 KB L2 cache for both instruction and data, and each CPE has a 64 KB scratch pad memory (SPM). Within a single CG, all the SPMs are able to communicate with each other in a few cycles, resulting in a very fast communication option for fine-grained tunings.' In terms of the programming model, a customised OpenACC is provided for directive-based parallelisation and tuning, while a self-developed Athread is provided for fine-grained tunings.

Since in our implementation, each CG corresponds to a unique MPI process, we regard a CG instead of a SW26010 CPU as a single node in this paper to avoid some messy and confusing description. That is, *each node refers to one CG, containing only 1 MPE and 64 CPEs and corresponding to only one MPI process.*

In the past years, the Sunway TaihuLight has been adopted into various scientific applications, covering major HPC areas such as climate changing[12], earthquake simulations [11], deep learning [10], material [13], etc. Based on the huge computing power as well as a set of sophisticated scaling approaches, a lot of achievements has been achieved, including the works [11], [12] that won the 2016 and 2017 ACM Gordon Bell Prizes, respectively. On the other hand, as quantum computing is becoming a possible and promising computing option, people are eager to see better simulation results using the most powerful supercomputers, such as the Sunway TaihuLight.

3 Methodologies and Optimizations

Before describing our methods and optimizations, let us recall some basic notions and notations.

The basic storage unit in a quantum computer is quantum bit (qubit). Generally, we can use a 2^n-length complex vector to describe an n-qubit state, such as $|\psi\rangle = (\alpha_0, \alpha_1, ...\alpha_{2^n-1})^T$. A practical quantum circuit consists of single-qubit and 2-qubit gates. For example, a single-qubit gate $U^k = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ on qubit k can be treated as an n-qubit gate which acts as identity on the other $n-1$ qubits, as denoted by $U = I^{\otimes n-k-1} \otimes U^k \otimes I^{\otimes k}$, where I is the identity operator on a single qubit. In this paper, a superscript indicates the qubit that the gate is performed on, and a subscript is used as a gate label or an index of amplitude.

To perform U^k on a state $|\psi\rangle = (\alpha_0, \alpha_1, \cdots, \alpha_{2^n-1})^T$, we have:

$$\begin{pmatrix} \alpha'_i \\ \alpha'_{i+2^k} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \alpha_i \\ \alpha_{i+2^k} \end{pmatrix}$$

for every $i = i_{n-1}...i_1 i_0$ where $i_k = 0$. And the resulting state is then $|\psi'\rangle = (\alpha'_0, \alpha'_1, ...\alpha'_{2^n-1})^T = U^k|\psi\rangle$. The 2-qubit gates used in this paper are mainly the controlled-gate: CZ. A controlled-gate $CU^{c,t}$ act on qubits c and t, with the first being the control bit and the second being the target bit. The performance of a 2-qubit gate is similar to that of a single-qubit gate with the only extra consideration of whether the control qubit is in state $|1\rangle$; for more details, we refer to [14].

For a quantum circuit simulation, the initial state is usually the product state:

$$|0\rangle^{\otimes n} = \underbrace{|0\rangle \otimes |0\rangle \otimes ... \otimes |0\rangle}_{n}.$$

If there is no 2-qubit gate in the circuit, the state will persistently remain a product state and only $O(n)$ space is needed to describe the state. However, as the number of two-qubit gates increases, the quantum state may become highly entangled. In this case, the storage of the state will require $O(2^n)$ space. As n increases, the memory requirement becomes insupportable even for the most powerful supercomputers. For example, the maximal qubit number of a state vector that could be stored in the memory of Sunway is 45 (46) using double (float), which requires 0.5 petabytes of memory space. However, for a 2-qubit gate $CU^{c,t}$, we can decompose it into $CU^{c,t} = P_0^c \otimes I^t + P_1^c \otimes U^t$, where $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$, and go along two branching path with respect to P_0 and P_1. deferring the entanglement it brings. Until an appropriate stage we combine the branching paths, finishing the deferred entanglement. Essentially, this idea comes from the notion of "Feynman path integral" and appeared in [2], [18], [20].

A universal random quantum circuit has a 2D grid architecture [5]. The quantum gates used in this type of circuits are $H, X^{1/2}, Y^{1/2}, T$ and CZ. The 2-qubit gate CZ only appears between two adjacent qubits in the grid. Since the positions of CZ gates are fixed in each layer of $n \times m$-grid universal random circuit for arbitrary n, m, we can find a very efficient partition scheme fitting the scale and structure of simulated circuit. Our techniques for finding this partition scheme are described in detail in the following subsection.

3.1 Method for Computing the Complete State-Vector (Task 1)

The 2D grid architecture of universal random ciruits makes circuit partition an appropriate method for simulation. However, as the circuit depth increases, the number of decomposed 2-qubit gates also increases. To increase the depth of circuits that can be simulated, we propose two optimizing techniques, which enable us to simulate 49-qubit circuits of depth 39, computing the complete output state-vector:

- *Technique 1*: We analyze the structure of universal random circuits, exploit the diagonal properties of CZ gates, and propose a technique, called *implicit decomposition*, which can decompose extra 7 CZ gates without requiring too much extra memory space, so as to increasing the depth of simulated circuits by 8.
- *Technique 2*: We propose a dynamic programming (DP) algorithm to find a good partition scheme for a given simulation task. In contrast to the general heuristic search method that usually takes a long time, this DP algorithm is efficient and can find an optimal partition scheme under certain constraints. It also makes the simulation easier to optimize and parallelize and thus improves the performance in time-to-solution.

3.1.1 Implicit Decomposition

Our partition scheme divides the target circuit into three parts A, B and C, with each part requiring less memory than the memory needed to store the entire 49-qubit state vector. To save memory space as far as possible, we need the qubit numbers of part A and part B to be not only close, but also suitable for an efficient partition, so we choose part A with 21 qubits and part B with 28 qubits instead of 24 and 25 qubits[2] respectively. Implicit decomposition further

[2] Part A with 24 qubits and part B with 25 qubits will result in a much more complicated partition scheme and be less memory-efficient because the partition lines will cut more CZ gates, and the implicit decomposition cannot be applied in this case.

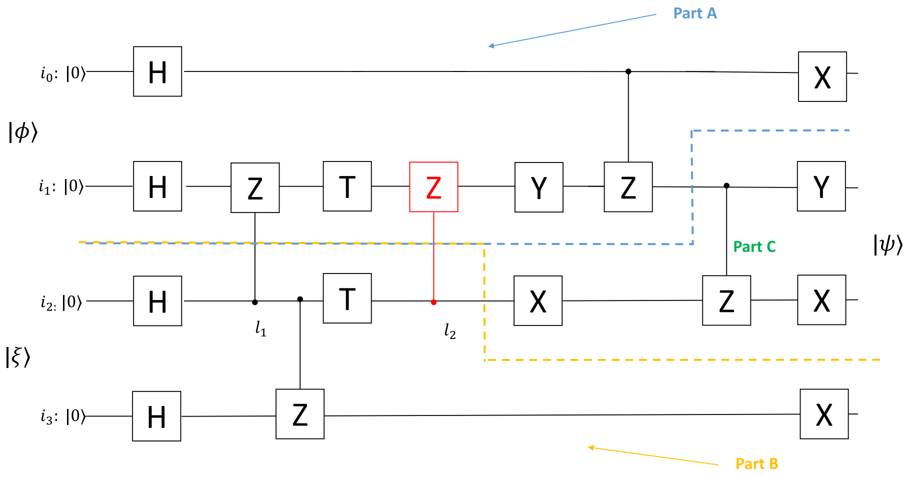


Fig. 2. Example of a 4-qubit circuit. In this example, there are 2 CZ gates being decomposed. Note that the second decomposed CZ gate only doubles the space consumption of part A, because after this CZ gate there is no gate in part B performed on qubit 2, thus we call it implicit decomposition.

balances the memory requirement of these 3 parts, decomposes extra CZ gates and increases the depth of circuits that could be simulated.

For a better understanding of Technique 1, let us first consider a circuit example shown in Fig. 2.

The partition scheme for simulation is illustrated by the blue dotted line and yellow dotted line. The dotted lines cut two CZ gates, and partition the circuit into 3 parts: A, B *and* C. We use $|\phi\rangle$ and $|\xi\rangle$ to describe the initial states in the two subsystems. Because there are two CZ gates being decomposed (cut by dotted lines), four branching paths in total are generated. Let $|\phi^{out}_{l_1,l_2}\rangle$ be the resulting state after performing the gates in part A, and l_1 and l_2 denote the indices of qubit 2 in two different time. State $|\xi^{out}_{l_1,l_2}\rangle$ is similar to $|\phi^{out}_{l_1,l_2}\rangle$. Then we have:[3]

$$|\phi^{out}_{l_1,l_2}\rangle = X^0 CZ^{0,1} Y^1 CZ^{2,1}_{l_2} T^1 CZ^{2,1}_{l_1} H^0 H^1 |\phi\rangle,$$

where l_1 and l_2 denote the indices of qubit 2 when decomposing the cutting CZ gates, thus $CZ^{2,1}_0 = I^1$ and $CZ^{2,1}_1 = Z^1$. Furthermore, we have:

$$|\xi^{out}_{l_1,l_2}\rangle = X^3 P^2_{l_2} T^2 CZ^{2,3} P^2_{l_1} H^3 H^4 |\xi\rangle,$$

where P is a projection operator: $P_i|i\rangle = |i\rangle$ and $P_i|1-i\rangle = 0$ for $i = 0, 1$. The starting state of part C is:

$$|\psi^{in}\rangle = \sum_{l_1,l_2} |\phi^{out}_{l_1,l_2}\rangle \otimes |\xi^{out}_{l_1,l_2}\rangle. \tag{1}$$

We perform the remaining gates in part C, and the eventually state $|\psi^{out}\rangle$ is:

$$|\psi^{out}\rangle = Y^1 X^2 CZ^{1,2} X^2 |\psi^{in}\rangle. \tag{2}$$

There are $2^4 = 16$ amplitudes to calculate for $|\psi^{in}\rangle$ (or $|\psi^{out}\rangle$). But we do not need to conserve all 16 amplitudes once in memory(not counting memory space for $|\phi\rangle$ and $|\xi\rangle$) to accomplish the calculation. Note that in part C there are gates only performed on qubit 1 and qubit 2, and no gate on qubit 0 and qubit 3. So $|\psi^{in}\rangle$ can be divided into 4 blocks by enumerating the indices of qubit 0 and qubit 3. Let $|\psi^{q_1,q_2}_{i_0,i_3}\rangle$ denote the reduced state of qubit 1 and qubit 2 with qubit 0 at $|i_0\rangle$ and qubit 3 at $|i_3\rangle$. Then $|\psi^{in}\rangle = \bigoplus_{i_0,i_3} |\psi^{q_1,q_2}_{i_0,i_3}\rangle$. Equation (2) turns into:

$$\begin{aligned} |\psi^{out}\rangle &= \bigoplus_{i_0,i_3} |\psi^{out,q_1,q_2}_{i_0,i_3}\rangle \\ &= \bigoplus_{i_0,i_3} (Y^1 X^2 CZ^{1,2} X^2 |\psi^{q_1,q_2}_{i_0,i_3}\rangle). \end{aligned} \tag{3}$$

Now we know that if all the results of part A and B are calculated and stored in memory, there only needs extra space for $2^2 = 4$ amplitudes in part C. From now on we set an amplitude as a basic storage unit ($8 + 8 = 16$ *bytes*), and the total space consumption is $S_A + S_B + 4$ instead of $S_A + S_B + 16$, where S_A and S_B are the space consumption of part A and B, respectively.

From the above analysis, we know $S_A = S_B = 2^{2+2} = 16$. However, S_B can be halved without introducing extra computation. Note that after the second cut CZ (in red) gate being decomposed, there is no any gate performed on qubit 2 in part B, so for any amplitude of $|\xi^{out}_{l_1,l_2}\rangle$, let it be ξ_{i_2,i_3,l_1,l_2}, where i_2 and i_3 are the indices of qubit 2 and qubit 3. We have

$$\xi_{i_2,i_3,l_1,l_2 \neq i_2} = 0, \ \text{for } l_2 = 0, 1.$$

Thus, the index l_2 could be absorbed into index i_2, and we only need to store $|\xi^{out}_{l_1}\rangle$. This means that when we finish the calculation of part B, only $2^{1+2} = 8$ amplitudes need to be stored, while in part A there are still $2^{2+2} = 16$ amplitudes to be stored. Because the second cut CZ gate is decomposed but the space consumption need not be doubled for control part (part B in this example), we call this technique *implicit decomposition*.

The implicit decomposition is useful when the space consumption of part A and B is not balanced, say there need space S_A to store part A and some fewer space S_B to store part B, we could apply implicit decomposition to part A, and only making the space of part A to S'_A while leaving the space of B unchanged, until S'_A and S_B are almost in the same magnitude.

For a universal random quantum circuit of $m \times n$ grid, the implicit decomposition works well when m or n is odd.

[3] The order of performing gates in circuit is from left to right, while the matrix-vector multiplication is from right to left.

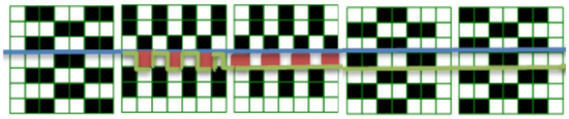


Fig. 3. Implicit decomposition applied to 49-qubit universal random circuits. In this figure and all following figures, two adjacent black blocks represent a CZ gate. And the blocks in red represent the CZ gate that could be implicit decomposed. Because part A has 21 qubits and part B has 28 qubits, without implicit decomposition $S_B = 2^{28+7}, S_A = 2^{21+7}$. After we use implicit decomposition on these seven cut CZ gates at the second and third layer, the ultimate space consumption of part B is $s'_B = S_B/2^7$, and now $S_A = S'_B$.

Our main target is 7 × 7-qubit circuits. At first, the two cutting lines are at the same position and partition the circuit into two parts: part A with 21 qubits, part B with 28 qubits. Then we apply this technique to part B, as shown in Fig. 3.

3.1.2 Dynamic Programming

To reduce the total space consumption and make the simulator easier to optimize, we avoid decomposing CZ-gates between part C and parts A, B. At first two splitting line overlap. When the two splitting lines separate, they start to walk around the CZ gates and will not cut-off any one of them. A feasible partition scheme also requires the total space consumption less than the memory space. To find an optimal partition scheme under these constrains, we design a dynamic algorithm for state compression.

Algorithm 1. Pseudocode of the Dynamic Programming

```
Input: $f(t)$
Output: $f(t+1)$
for $0 \leq i_1, i_2, ..., i_7 \leq 3$ do
    $f(t+1, i_1, i_2, ..., i_7) = \infty$;
    if position $(i_1, i_2, ..., i_7)$ is not illegal then
        continue;
    end
    $cutnum = number\ of\ CZ\ gates\ splitting\ by\ (i_1, ..., i_7)$;
    for $0 \leq j_1 \leq i_1, 0 \leq j_2 \leq i_2, ..., 0 \leq j_7 \leq i_7$ do
        $f(t+1, i_1, i_2, ..., i_7) = min\{f(t, j_1, j_2, ..., j_7) + cutnum, f(t+1, i_1, i_2, ..., i_7)\}$;
    end
end
```

Let $f(t, i_1, i_2, i_3, i_4, i_5, i_6, i_7)$ denote the minimal space consumption (exponential in f) of part A when the partition scheme reaches layer t and the position of upper splitting line is $(i_1, i_2, ..., i_7)$. Here, $(i_1, i_2, ..., i_7)$ is used to indicate the distance between the current position and the initial position. Initially, the upper splitting line is beneath the third row of qubits, and $f(1, 0, ..., 0) = 21$. Note that $f(7, 0, 0, ..., 0) = 21 + 3 = 24$ and $f(8, 0, 0, ..., 0) = 21 + 3 + 4 = 28$.

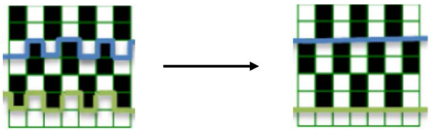


Fig. 4. A illustrative example of legal transition between two adjacent layers. From layer t to $t+1$, the position of upper splitting line (in blue) transforms from (0,1,0,1,0,1,0) to (1,1,...,1). Since (1,1,...,1) does not cut any CZ gate at layer $t+1$, this is a legal transition. So $f(t+1, 1, 1, ..., 1) = min\{f(t+1, 1, 1, ..., 1), f(t, 0, 1, 0, 1, 0, 1, 0)\}$.

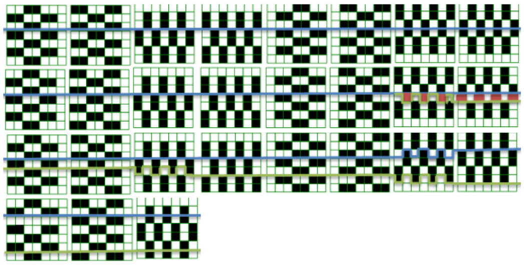


Fig. 5. Partition scheme for 49-qubit circuit of depth 27.

Since we have a restriction that no CZ gate between part A and B could be decomposed, $f(t, i_1, i_2, ..., i_7)$ will be *illegal* when $(i_1, i_2, ..., i_7) \neq (0, 0, ..., 0)$ and $(i_1, i_2, ..., i_7)$ splits some CZ gate at layer t. Fig. 4 is an illustrative example of legal transitions.

Similarly, let $g(t, i_1, ..., i_7)$ denote the target function of part B. Then $g(1, 0, 0, ..., 0) = 28$ and $g(8, 0, 0, ..., 0) = 35$. The recursion from $g(t-1)$ to $g(t)$ is similar to f, and the only difference is that when $(i_1, i_2, ..., i_7)$ first leaves the initial position it has a chance of applying implicit decomposition.

To find a good partition scheme for circuit of depth t, we can traverse $f(t)$ and $g(t)$ to find an optimal combination of $f(t, i_1, i_2, ..., i_7)$ and $g(t, j_1, j_2, ..., j_7)$ which minimizes $S_A + S_B + S_C$, where $S_A = 2^{f(t, i_1, i_2, ..., i_7)}$, $S_B = 2^{g(t, j_1, j_2, ..., j_7)}$ and $S_C = 2^{\sum_{1 \leq k \leq 7}(i_k + j_k)}$. Several partition schemes are illustrated in Figs. 5, 6 and 8. There need space $S_A = S_B = 2^{35}$, $S_C = 2^{28}$ to execute the simulation for depth 27, while $S_A = S_B = 2^{42}$ for depth 35 and 39. The maximal number of qubits that could be simulated on one node is 28. This indicates that from depth 35 to depth 39, there will be a drop in performance, which is shown in section 4.3.

3.1.3 Summary

The two techniques proposed above not only work for universal random circuit. For circuit of an arbitrary 2-D grid structure, our method can find a proper partition scheme to reduce the time and space complexity of simulation. Table 3

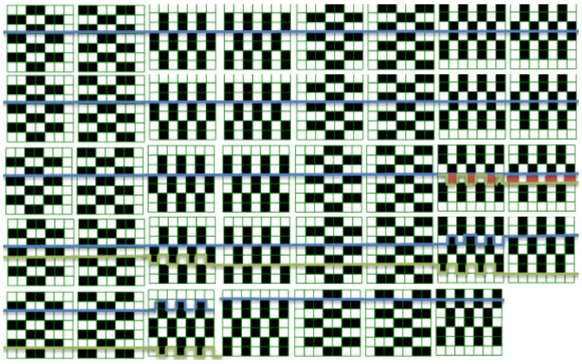


Fig. 6. Partition scheme for 49-qubit circuit of depth 39. The scheme for depth 35 is exactly the front 35 layers of this figure, except for that at layer 35 two splitting lines are still straight as in layer 34 and cut 6 extra CZ gates (See the last layer of partition scheme for 49-circuit of depth 27 in Fig. 5). Note that the CZ gates at the last layer would not impact the probabilities and could be removed because they diagonal. Thus the number of CZ gates being cut (the implicit decomposed CZ gates and CZ gates cut at the last layer are not included) is 14 for both depth 35 and 39, located in layer 7 and layer 14. But in the case of depth 35 $S_C = 2^{28}$ and in the case of depth 39 $S_C = 2^{42}$.

TABLE 2
The Space Consumption of Parts A,B,C for 49-Qubit Circuit of Different Depth

Depth	27	35	39
S_A	2^{35}	2^{42}	2^{42}
S_B	2^{35}	2^{42}	2^{42}
S_C	2^{28}	2^{28}	2^{42}

gives an algorithmic comparison of our partition scheme and the partition scheme in [18].

Obviously, 7×7-qubit circuits of depth 39 and of depth 40 have different difficulties in physical preparation and operation. We provide an evidence showing that our method might reach depth 40 if the target circuit is a "lucky circuit" (i.e., with enough T gates in special positions at layer 40, like in Fig. 7). For example, using the tensor slice technique in [18], if there is at least one T gate in the four single-qubit gates on qubit 0,2,4,6 at layer 40, the size of a slice is at most 2^{45}, which could be directly simulated on Sunway. Though the performance will further drop from depth 39 to depth 40. Because $X^{1/2}, Y^{1/2}, T$ appears randomly at the positions for single-qubit gates, the probability that a 49-qubit and 40-depth universal random circuit could be simulated on Sunway is:

$$p = 1 - (2/3)^4 = 65/81.$$

3.2 Calculating One or a Few Amplitudes (Task 2)

The method in Section 3.1 can be used to compute the complete state-vector for 49-qubit circuits. For circuits of 56 qubits or larger size, it is difficult to calculate all the amplitudes due to limited time and space. However, to test the fidelity of a real quantum circuit, one only needs to sample (i.e., calculate a small number of) amplitudes, usually ranging from 10^3 to 10^6 [5]. Our method can finish this task very efficiently because all the amplitudes of eventually states in part A and B are stored in memory. For example, in the cases of 7×8 qubits with depth 35, or 8×8 qubits with depth 30 it is easy to calculate a large amount of (e.g., $\geq 2^{40}$) amplitudes (in less than 1 hour). Fig. 8 in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2019.2947511 shows the partition scheme for 56-qubit circuits with depth 35. The schemes for 64-qubit and 72-qubit circuit are similar.

When focusing on a 7×7-qubit circuit, we will introduce a special and straight method to calculate an amplitude of the final state. The target is to calculate

$$\alpha_x = \langle x|\mathcal{U}_{circuit} H^{\otimes 49}|00...0\rangle$$

for $0 \leq x \leq 2^{49} - 1$. Since we could calculate the complete state-vector for a 7×7-qubit circuit of depth 27, we can also sample one amplitude for a circuit of depth 55. Let $U_{circuit} = \mathcal{U}_2\mathcal{U}_1$ in which $\mathcal{U}_1$ has 27 layers and $\mathcal{U}_2$ has 28 layers, $|\psi\rangle = \mathcal{U}_1 H^{\otimes 49}|00...0\rangle$ and $|\varphi\rangle = \mathcal{U}_2^{\dagger}|x\rangle$, we have:

$$\alpha_x = \langle\varphi|\psi\rangle.$$

Thus, we simultaneously calculate $|\varphi\rangle$ and $|\psi\rangle$, during the calculation we computing the inner product of every two corresponding blocks of 2^{28} amplitudes of $|\varphi\rangle$ and $|\psi\rangle$. Sum all 2^{21} inner products and we get α_x. Because the least space consumption of simulating a circuit of depth 27 is O(2^{35}), this method can be parallelized to calculate more amplitudes.

TABLE 3
An Algorithmic Comparison of our Partition Scheme and the Partition Scheme in [18] for 49-Qubit Circuits

Reference	Depth	Space	Computation amount
IBM[18]	27	64 TB	$2^{49}(2^{10} + n_C)$
Sunway	27	1 TB	$2^{49}(2^{7} + n_C)$
IBM[18]	39	N/A	N/A
Sunway	39	256 TB	$2^{49}(2^{14} + n_C)$

Space means the least memory needed to execute the simulation. n_C is the number of gates in part C, usually several hundred. The computation amount means the number of float-point operations. No partition scheme for depth 39 is given in [18]. This table shows that the combination of techniques 1 and 2 produces a more efficient algorithm.

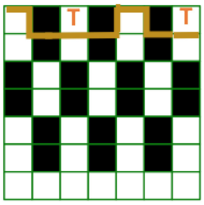


Fig. 7. Tensor slicing technique in [18]. This is a example that two T gates appears in the four positions, thus 5 qubits in total could be sliced.

3.3 Optimization for Reducing Communication Amount

The method presented above mainly concerns the memory space limitation of Sunway. Another issue is network communication, as the network bandwidth of Sunway is limited while the communication amount needed for this method is huge. In this subsection we describe a related method to reduce the communications of a key step in our simulation. We will first explain why such amount of communication is needed and then describe how to optimize it.

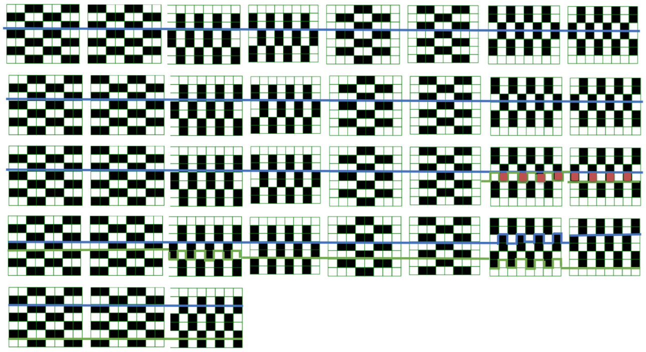


Fig. 8. Partition scheme for 56-qubit circuits of depth 35. The method for solving a part of amplitudes from this circuit is exactly the same as computing a complete state-vector for 49-qubit circuits. Note that $S_A = S_B = 2^{48}$ here, which already exceed the memory limit of Sunway. Thus a little space-time tradeoff [2] is needed. The space-time tradeoff, which is also needed for 64-qubit circuits with depth 30 and 72-qubit circuits with depth 27, can be achieved by simply enumerating the first several decomposed CZ gates [21].

TABLE 4
The Comparison of Network Communication Amounts for Computing Equation (6) with and without Optimization

Optimization	Main work	Overall time	Speedup
without	2^{21} MPI_Gathers	1021.9 min	1
with	2^{14} *MPI_Alltoall*	17.8 min	57.4

Every single MPI_Gather or MPI_Alltoall is within 128 nodes, so they can be executed in parallel. The overall time is under the condition of using 32768 nodes for network communication. For network communication without optimization the overall time is only an estimation since we only executed 2^{15} MPI_Gathers, and multiplies by 64 the execution time, which is 15.96 minutes.

Recall that Equation (3) is the final step to compute the complete state-vector, and this step can be divided into $2^n/S_C$ subtasks which could be paralellized, where n is the number of qubits in the whole circuit and S_C is the space consumption of part C. For 49-qubit circuits of depth 27 and depth 35, Equation (3) can be rewritten as:

$$\begin{aligned}|\psi^{out}\rangle &= \bigoplus_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}} |\psi^{out,q_{14},q_{15},...,q_{41}}_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}}\rangle \\ &= \bigoplus_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}} (\mathcal{U}_C|\psi^{q_{14},q_{15},...,q_{41}}_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}}\rangle),\end{aligned} \tag{4}$$

where $\mathcal{U}_C$ denotes the unitary operation for part C. Note that calculating $|\psi^{out,q_{14},q_{15},...,q_{41}}_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}}\rangle$ in Equation (4) is essentially the same as simulating a 28-qubit circuit, which could be finished in a single node. Now the remaining problem is to efficiently prepare $|\psi^{q_{14},q_{15},...,q_{41}}_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}}\rangle$ for each set of possible values of $(i_0, i_1, ..., i_{13}, i_{42}, i_{43}, ..., i_{48})$, where $i_k \in \{0, 1\}$. Note that

$$\begin{aligned}|\psi^{q_{14},q_{15},...,q_{41}}_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}}\rangle = &\sum_{l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}} \\ &(|\phi^{out,q_{14},q_{15},...,q_{20}}_{i_0,i_1,...,i_13,l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle\otimes \\ &|\xi^{out,q_{21},q_{22},...,q_{41}}_{i_{42},i_{43},...,i_{48},l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle),\end{aligned} \tag{5}$$

where t is the number of decomposed CZ gates, $t = 7$ for depth 27, and $t = 14$ for depth 35. $|\phi^{out,q_{14},q_{15},...,q_{20}}_{i_0,i_1,...,i_6,l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle$ is a 2^7-length state-vector, where the indices of qubit 0-6 are $i_0, i_1, ..., i_6$, the indices of control qubits of t decomposed CZ gates are $l_1, l_2, ..., l_t$, and the indices of control qubits of 7 implicit decomposed CZ gate are $i_{22}, ..., i_{27}$.

Because of implicit decomposition, for each value set of $i_{21}, i_{22}, ..., i_{27}$, $|\xi^{out,q_{21},q_{22},...,q_{41}}_{i_{42},i_{43},...,i_{48},l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle$ has only 2^{14} nonzero amplitudes. Thus, Equation (5) can be rewritten as:

TABLE 5
Frequency of Swaps for 28-Qubit Universal Random Circuits with Different Depth, in the Case that the Number of Local Qubits is 10

Depth	25	30	34	38
Swaps	6	7	8	9

TABLE 6
Performance of Simulating 28-Qubit Circuits with Different Depth on a Single Node

Depth	18	26	34	42	50
Gates	258	375	492	609	726
Swaps	3	4	5	6	7
Time	15.4s	22.6s	29.6s	36.7s	44.3s
Speedup	7.04	6.95	6.96	6.95	6.88

The number of global qubits is 14. Speedup is the speed-up ratio to method of performing gate by gate without any optimization but using the CPEs to accelerate.

$$\begin{aligned}|\psi^{q_{14},q_{15},...,q_{41}}_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}}\rangle = &\sum_{l_1,l_2,...,l_t} (\bigoplus_{i_{21},i_{22},...,i_{27}} \\ &|\phi^{out,q_{14},q_{15},...,q_{20}}_{i_0,i_1,...,i_6,l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle\otimes \\ &|\xi^{out,q_{28},q_{29},...,q_{41}}_{i_{42},...,i_{48},l_1,...,l_t,q_{21}=i_{21},...,q_{27}=i_{27}}\rangle),\end{aligned} \tag{6}$$

where $|\phi^{out,q_{14},q_{15},...,q_{20}}_{i_0,i_1,...,i_6,l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle(|\phi\rangle$ for short) is still of 2^7-length but $|\xi^{out,q_{28},q_{29},...,q_{41}}_{i_{42},...,i_{48},l_1,...,l_t,q_{21}=i_{21},...,q_{27}=i_{27}}\rangle(|\xi\rangle$ for short) is of 2^{14}-length.

Now we consider how to realize Equation (6). The data from part A have 2^{t+7} complex numbers (amplitudes), and the data from part B have 2^{t+14} complex numbers. Thus, for the case of depth 35, the data from part B have $2^{14+21} = 2^{35}$ complex numbers. Directly calculating Equation (6) needs 2^7 nodes to communicate (e.g., an *MPI_Gather* is feasible). This is very inefficient, because there are $2^{49-28} = 2,097,152$ entities of Equation (6) to calculate, and each entity needs an *MPI_Gather* in 128 nodes.

However, we can slightly change the form of Equation (6) to:

$$\begin{aligned}|\psi^{q_{14},q_{15},...,q_{41}}_{i_0,i_1,...,i_{13},i_{42},i_{43},...,i_{48}}\rangle = \bigoplus\nolimits_{i_{21},i_{22},...,i_{27}} \Bigg(&\sum_{l_1,l_2,...,l_t} \\ &|\phi^{out,q_{14},q_{15},...,q_{20}}_{i_0,i_1,...,i_6,l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle\otimes \\ &|\xi^{out,q_{28},q_{29},...,q_{41}}_{i_{42},...,i_{48},l_1,...,l_t,q_{21}=i_{21},...,q_{27}=i_{27}}\rangle\Bigg).\end{aligned} \tag{7}$$

Note that for each element *in the bracket*, the data from part A and B have 2^{21} and 2^{28} complex numbers, respectively. This means the calculation in the bracket can be finished in a single node and the length of result is 2^{21}. We further append $|\phi\rangle$ with qubit 7-13 so that $|\phi\rangle$ also becomes a 2^{14}-length vector:

$$|\phi\rangle = |\phi^{out,q_7,q_9,...,q_{20}}_{i_0,i_1,...,i_6,l_1,l_2,...,l_t,i_{21},i_{22},...,i_{27}}\rangle.$$

Now $|\phi\rangle \otimes |\xi\rangle$ is of 2^{28}-length and can still be calculated in a single node. Actually, it can be regarded as a 28-qubit state:$|\psi^{q_7,...,q_{20},q_{28},...,q_{41}}\rangle$. Our target is $|\psi^{q_{14},...,q_{27},q_{28},...,q_{41}}\rangle$. This can be done by performing an *MPI_Alltoall* on every group of 128 consecutive nodes. Assume again that we have 2^{15} free nodes to work for *MPI_Alltoall*. Then in each group, only $2^{21-15} = 64$ rounds of *MPI_Alltoall* needs to be performed.

Solving Equation (6) for 49-qubit circuit of depth 39 is essentially the same as the case of depth 35. Table 4 shows the improvement in network communication that the optimization brings.

3.4 Single Node Optimizations

In this subsection, we introduce our optimization for the quantum circuit simulation at the single node (single core group) scale. The optimization for every single node is very important for improving the overall performance of our method, which needs a huge amount of 28-qubit circuit simulation according to the analysis in Sections 3.1 and 3.3.

As agreed in Section 1, one node represents a core group in Sunway, and it has 1 MPE and 64 CPEs. Each node has 8 GB main memory, shared by both MPE and CPEs. The maximum qubit number that could be simulated on one node is 28 if we use two doubles to represent an amplitude, since $2^{28} \times 16\ \mathrm{B} = 4\ \mathrm{GB}$. To obtain full power of Sunway TaihuLight, one must allocate most of the computing tasks to CPEs. Each CPE has a 64 kB private and separate high-bandwidth storage unit called *local data memory (LDM)* (also known as *scratch pad memory* in Section 2). To fullfill high-speed calculations a CPE must fetch data from the main memory to its own LDM and keeps it in LDM for calculation as long as possible (like cache). This fetching-data behavior is usually called *direct memory access* (DMA). To get high DMA bandwidth, it usually requires the data fetched consecutive.

If LDMs fetch data from the main memory for every gate performed, and put the data back to main memory when the calculation is finished, the simulation will be inefficient due to low flop-to-Byte ratio.[4] In our experiments, the average execution time is 0.32s per gate in such way, thus the performance is bounded by DMA speed.[5] However, we can take advantage of the data locality in a better way to reduce the DMA amount. For example, if each CPE has fetched from the main memory 16KB data in its LDM, that is, $2^{14-4} = 2^{10}$ amplitudes. For any gate performed on those qubits with their ranks lower than 10 (0 is the lowest rank), the calculation can be executed in this 16 KB data, i.e., α_i and α_{i+2^k} are in the same LDM. Thus we can perform a bunch of gates acting on low-rank qubits once instead of performing the circuit gate by gate. We call these 10 qubits with lowest ranks *local qubits*, and other 18 qubits are *global qubits*.[6]

This idea is similar to the gate fusion techniques in [16], which deals with the gates on low-rank qubits in cache. However, gate fusion is one-off, which can only be applied at the start of the circuit.

To make the above procedure repeatable, we also adopt the qubit reordering method. Qubit reordering are used in [15] and [17] to reduce the network communications. Here, our aim is to maintain the data locality and reduce the amount of memory access. That is, only diagonal gates, or non-diagonal gates on local qubits are calculated. To accomplish this, we need to execute two types of qubit rank swaps:

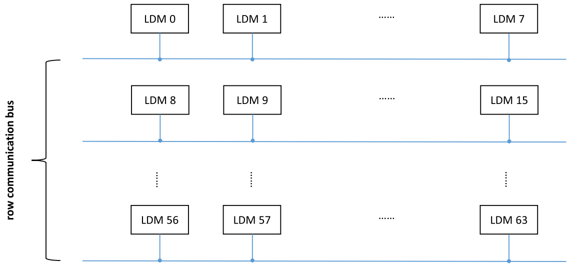


Fig. 9. Register communication. There are 8 row communication buses and 8 column communication buses in a core group. In this figure only row communication buses are plotted, since in our experiments only row communication is used. In fact, the row communication is used to achieve the swap between qubits of rank 8-10 and qubits of rank 11-13. Register communication has very high bandwidth, more than 200GB/s in total for 8 row communication buses. So the cost of this step is very small, and we can treat 8 LDMs in a row as new a composite LDM. Thus the number of local qubits turns into 14.

- Swap the qubits of rank 0-9 and qubits of rank 11-20
- Swap the qubits of rank 14-20 and qubits of rank 21-27

With a gate scheduling preprocessing program, which also utilizes the diagonal properties of gate T and CZ, we get the amount of swaps for 28-qubit quantum supremacy circuit:

We achieve fast swaps of qubit rank with the help of CPEs. Since a vector with dimension 2^n can be denote as a $2^{n/2} \times 2^{n/2}$ matrix, swapping the qubits of rank 0-9 and qubits of rank 10-19 is essentially a transpose of the corresponding complex matrix, where the dimension of this matrix is $2^{10} \times 2^{10}$, and $2^{28-20} = 256$ matrices in total need to be transposed. Swapping the qubits of rank 14-20 and qubits of rank 20-27 is similar, which is equivalent to a transpose of a $2^8 \times 2^8$-dimensional matrix, but each element of this matrix contains 2^{12} amplitudes.

Register communication is a unique function in Sunway CPU, designed for fast data transmission between LDMs. The qubit reordering can be further optimized using this feature. As shown in Fig. 9, the CPEs in a row can send/receive messages in a communication bus, we can treat a row of 8 separate LDMs as a composite LDM, while the data exchange between these 8 LDMs is fulfilled by register communication. If each CPE fetches 32 kB consecutive data from the main memory to its LDM, there will be 11 local qubits. But when considering a composite LDM formed by a row of LDMs, the number of local qubits turns into 14. Thus we only need to execute one type of qubit swap:

- Swap the qubits of rank 0-13 and qubits of rank 14-27

This is simply a transpose of a $2^{14} \times 2^{14}$-dimensional complex matrix, which can be quickly accomplished by CPEs.

Because of the high bandwidth of register communication, the time consumed on register communication is very small, so as to improving the overall performance of single node case.

3.5 Other Standard Optimizations

In this subsection we briefly introduce some other standard optimizations provided by Sunway TaihuLight, which can

[4] A 2×2 complex matrix multiplying a 1×2 complex vector needs 14 float-point operations. For a 28-qubit circuit, each non-diagonal gate requires $2^{28-1} \times 14$ float-point operations and 4G DMA *get* and *put*

[5] The DMA *get* bandwidth and *put* bandwidth are both less than 25 GB/s in our experiments

[6] This is different to [15], [17]. Their distinction of 'global' and 'local' is at the multi-node and single-node level. While our 'global' corresponds to main memory, and 'local' corresponds to LDM

TABLE 7
The Comparison of 28-Qubit Simulation with/without Standard Optimizing Techniques, While Keeping All Other Single Node Optimizations Applied

Depth	18	26	34	42	50
Before	30.7s	46.4s	61.2s	75.9s	89.9s
After	15.4s	22.6s	29.6s	36.7s	44.3s
Speedup	1.99	2.05	2.07	2.07	2.03

The results in this table show that about a 2x speed up is achieved by these standard optimizing techniques such as vectorization and instruction Reordering.

be exploited for our simulation of quantum circuits. Table 7 gives an evaluation of these techniques.

3.5.1 Vectorization

Sunway TaihuLight provides many 256-bit data types. In our simulation the type $doublev4$ is adopted for vectorization. The data stored in LDM is a complex number array with one double as the real part of a complex number and another double as its imaginary part. To vectorize the double-precise float-point calculation, we put four amplitudes into two $doublev4$ registers v_r, v_i once. However, the real parts of these amplitudes are not consecutive, the imaginary parts neither. We use the instruction *vshuffle* to solve this problem.

3.5.2 Instruction Reordering

Another optional optimization is instruction pipeline. The put and $store$ operations, together with the multiply-add operations, can form a pipeline to further reduce the calculation time, especially when the data dependency between adjacent instructions is little. This optimization further improves the computational efficiency.

4 Numerical Experiments and Results

4.1 Setup of Experiments

Sunway TaihuLight is one of the most powerful supercomputer with over 100 Pflops computing capacity [1]. To test the limitation of our simulator on Sunway, we used 131072 nodes (32768 cpu processors), which is around 80 percent of computing resource of the whole machine with nearly 1PB main memory in total. To avoid misunderstanding, we recall that in this paper 1 node = 1 CG (and in section 2 one SW26010 processor consists of 4 nodes). We implemented our simulator in C++ for MPE managing programs and C for CPE computing programs. We use MPI for inter-node communications. To facilitate the most of computing capacity of Sunway we have used the *athread* library [9].

According to previous analysis, a simulation task in our implementation has two stages:

- stage 1: computing the results for part A and B and store them in the memory;
- stage 2: using the the results in stage 1 to generate the input and compute the results for part C, so as to solving all the amplitudes;

Stage 1 can be finished in 10 minutes if there is enough space to store the results for part A and B. Stage 2 thus is the bottleneck of the whole task. As illustrated in Section 3.3, stage 2 could be evenly divided into 64 rounds, making it convenient to parallelize and providing good strong scalability.

Stage 2 has two computing kernels: the first one is generating the input for part C, more precisely, computing the entities of Equation (6); the second one is simulating a 28-qubit circuit on single nodes. We call the first kernel *tensor* because it calculates the tensor product of two complex vectors and sums them. We call the second kernel *sim*.

4.2 Performance Measurement

The performance is usually computed in two ways:

- Manually counting all double-precision arithmetic instructions in the assembly code;
- Using the hardware performance monitor of Sunway, PERF, to get the amount of double-precision arithmetic instructions retired on the CPE cluster.

Both ways provide similar results of counting the arithmetic operations. We employ the second way (PERF) in our study. And we obtain the sustained performance of two kernels: Kernel *tensor* achieves 92.8 GFlops per core group; kernel *sim* achieves 37.1 GFlops per core group. The performance of these two kernels fits the overall performance when simulating a 49-qubit circuit of depth 39.

Table 6 shows the performance of kernel *sim* and speedup under cases of 28-qubit circuits with different depth. The number of global qubits is 14. Speedup is the speed-up ratio to the method of performing gate by gate without any optimization but using the CPEs to accelerate.

4.3 Time-to-Solution

For the task of simulating a 49-qubit circuit of depth 35 and computing the complete state-vector, it takes around 3.7 hours. The bottleneck is Equation (6), because solving 2^{21} entities of Equation (6) needs $2^{21+7+35} = 2^{63}$ times of complex number multiplication. This step occupied around 90 percent of the run-time in the simulation of 35-depth circuit. For the task of simulating a 49-qubit circuit of depth 39, it takes around 4.2 hours. Fig. 11 shows the log-transformed distributions of case depth 35 and case depth 39. The reason for causing this drop in performance is that part C has 42 qubits in the case of depth 39, while part C only has 28 qubits in the case of depth 35. Thus in the case of depth 35, the calculation in part C are all within single nodes. While calculating a block of part C in the case of depth 39 is essentially simulating a 42-qubit circuit of depth 15, which needs one all-to-all communication on 2^{14} nodes [17]. Because there are $2^{49-42} = 128$ blocks to calculate, the amount of communication increases a lot.

The sustained performance is 4.92 PFlops for the case of depth 35 and 4.3 PFlops for the case of depth 39. 4.3 PFlops is around 3.44 percent of the peak performance of Sunway. There are two reasons for this low efficiency: 1) we only use 131072 nodes, which is just around 80 percent of the whole computing resource of Sunway; 2) for the cases of depth 35 and depth 39, the kernel *tensor* is the bottleneck. However, during the simulation only half of the nodes are used for kernel *tensor* due to the limited memory space of each node.[7] If we can find a better method for memory allocation

[7] In our implementation at current stage, 1/4 of the nodes need to store the results for part A, another 1/4 of the nodes need to store the entities of Equation (6), they can not participate in the computation of kernel *tensor*.

TABLE 8
The Performance Comparison between our Work and IBM Simulator [18]

Work	Qubits	Depth	Rmax (TFlop/s)	Time-to-solution	Speedup
IBM	49	27	17,173.2	$\geq$ 24 hrs	1
Sunway	49	27	9,524.7 (16384 nodes)	1.49 hrs	$\geq$ 29

RMAX *means the maximal sustained performance [1].* 10.24% *of Sunway has the Rmax of 9,524.7 TFlop/s. The speedup is calculated by:* $speedup = \frac{IBM\_time \times IBM\_Rmax}{Sunway\_time \times Sunway\_Rmax}$. *The main reason for such speedup is that in [18], 10 CZ gates are decomposed in the case of 49 qubits with depth 27. While in our method there are only 7 decomposed CZ gates benefiting from implicit decomposition and dynamic programming techniques.*

we might let all 131072 nodes work for kernel *tensor*, doubling the overall performance roughly.

4.4 Improvement over Previous Works

To show the improvement that our method brings, we first compare the overall performance between our work and the IBM's work [18], in the case of 49 qubits with depth 27. In principle, simulating the circuit with such a depth only needs 1024 nodes using our method. Increasing the nodes will decrease the time-to-solution. For a fair comparison, the performance of machine should be taken into consideration. We choose the result using 16384 node for comparison. It is 10.24 percent of Sunway. And the improvement is shown in Table 8. The main reason for such improvement is also given.

Another reason for the speedup in Table 8 is our single node optimizations, which make better use of the machine performance. To make this claim more convincing, we compare our single node optimizations with the ETH's work [17], in which their single node case is highly optimized too. Again, to make comparison fair, we should consider a rather similar benchmark. Because the bottleneck of quantum circuit simulation in single node case is *memory access*, we consider the memory bandwidth of one node in either machine. See Table 9.

The comparison with the ETH's work is not absolutely fair, but at least demonstrates that our single node optimizations are also very efficient to deal with the large amount of memory access even in nodes without high memory bandwidth.

4.5 Scalability

Fig. 10 shows the strong scaling behavior for circuits of different depth. As the stage 1 of computing results for part A and B usually takes a few minutes, this causes the drop of parallel efficiency especially when each node executes few rounds of stage 2. Note that the time consumption of stage 1 for the case of depth 27 is much less than that for the case of depth 35 and depth 39, so the parallel efficiency for the case of depth 27 is slightly better the other two cases. Moreover, simulating a circuit of depth 35 or depth 39 requires at least 65536 nodes.

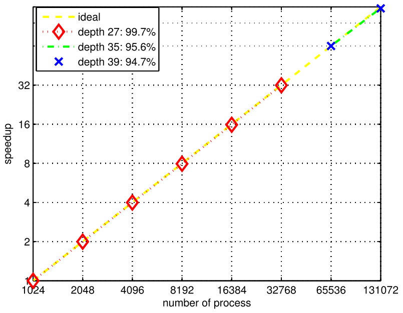


Fig. 10. Strong scaling behaviour for the cases of depth 27, 35 and 39. The parallel efficiency under these three cases is also illustrated in the figure.

5 Discussions

In this section, we further discuss several issues about how our techniques presented in the previous sections can be applied or generalized to other simulators.

5.1 Portability to Other Supercomputers

As our simulator implemented on Sunway TaihuLight which has a many-core heterogeneous architecture, one might ask whether the optimizing techniques in this paper are portable to other supercomputers. The answer is yes. The implicit decomposition and dynamic programming in Section 3.1 are used for reducing the memory consumption, and the optimizing techniques in Section 3.3 are used for reducing the network communication time. They can be directly applied on other supercomputers with different architectures such as x86 architecture. Actually, the only one requirement is over 0.5 PB memory for a supercomputer to simulate a 49-qubit universal random circuit with depth 35 or 39 when these optimizing techiniques are applied.

Applying the single node optimization on other supercomputers is a little different. Nevertheless, it is still feasible with appropriate modification. The core idea of single node optimization is making the most of data locality. In our simulator, the LDM plays a substituted role for cache. As other supercomputers usually have cache instead of LDM, we could use cache to store a length of amplitudes and then handle a bunch of low-level gates in one step. But to

TABLE 9
Analysis of Two Highly Optimized Simulators in the Single Node Cases

Platform	Local qubits	Depth	Gates	Time-to-solution	Time per gate	Memory access per gate	Single node bandwidth	Speedup
Cori II	30	25	369	9.58 s	0.026 s	16 GB	460 GB/s	1
Sunway	28	26	375	22.6 s	0.060 s	4 GB	27 GB/s	1.84

We compare their average performance per unit memory bandwidth (say 1 GB/s). We reemphasize that in this article, one single node only means one core group of Sunway. While a SW26010 CPU has 4 core groups, its memory bandwidth quadruples, which is more than 100GB/s. Counting in the memory access and average time per gate, we get an approximate speedup: $speedup = \frac{(ETH\_time\_per\_gate \times ETH\_MA\_per\_gate)/Cori\_single\_node\_bandwidth}{(Sunway\_time\_per\_gate \times Sunway\_MA\_per\_gate)/Sunway\_single\_node\_bandwidth} = 1.84$

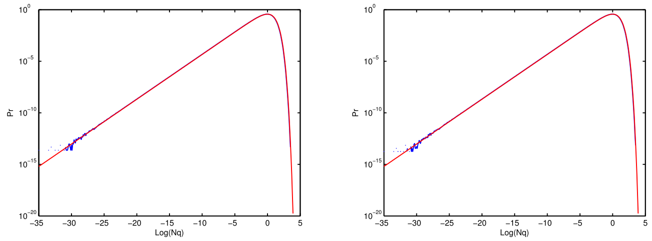


Fig. 11. Histograms of log-transformed outcome probabilities for 49-qubit circuits, compared to theoretical Porter-Thomas distribution [5]. The left side is the result of simulating a circuit of depth 35. The right side is the result of circuit of depth 39. Red lines mean the theoretical Porter-Thomas distribution, and blue lines represent the distribution of our experimental results. Both results fit the theoretical distribution well.

maximize the speedup effect we must realize the corresponding cache structure such as cache size or how many ways(4-way or 8-way) it has, and perhaps practical single nodes experiments are needed.

5.2 Logical Structure of Optimizations

When applying these techniques on a supercomputer (not only on Sunway), the order of applying them extremely matters. For a given universal random circuit, we first apply the circuit partition techniques in Section 3.1 to minimize memory consumption. Next, we apply network communication optimization. The communication alternate with the tensor kernel but they cannot overlap, as detailedly explained in Section 3.3. Finally, we apply single node optimizations in section 3.4 to accelarate the calculation of part C. The structure is briefly shown in Fig. 12.

6 Conclusion and Future Works

This paper describes our method and implementation of quantum circuit simulator on Sunway TaihuLight. The results indicate that for current universal random circuits, 49 qubits with depth 39 is reachable. To find a proper bound of quantum supremacy in terms of universal random circuits, one might 1) increase the depth or qubits of the circuits; or 2) modify the structure of current universal random circuits. Whatever, classical computers have their limits on simulating quantum circuits. We believe there will be one day that quantum computer can solve certain problems which classical computers cannot. Before that day comes, simulating quantum circuits on classical computers, especially on supercomputers, is crucial to understand the power and limit of quantum computers. Even after that day, a simulator of quantum circuits on a classical computer will still be helpful for design, synthesis, testing and verification of quantum circuits.

Follow-up work of this paper includes further optimizations of our simulator and adding some new functions to it, e.g., (1) quantum circuit testing and verification; (2) simulation of real circuits with quantum noise, and (3) simulation, debugging and verification of more sophisticated quantum algorithms and quantum programs (with control flows) [30]. Another line of research is to consider how to extend to our work when EB-scale supercomputers come out, as more powerful supercomputers will help more in testing, verification and simulation of large-scale quantum circuits.

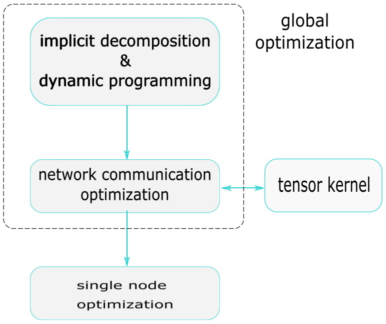


Fig. 12. Logical structure of different part of optimizing techniques.

Acknowledgments

The authors are very grateful to Haining Yu, Wei Zhang, Shupeng Shi, Hongsong Meng, Hongkun Yu, Wenlai Zhao and the whole team at the National Super Computing Center in Wuxi for their kind helps. Special thanks go to Zhao Liu and Lin Gan, who have given us a lot of useful suggestions and assistance. This work was partially supported by National Key R&D Program of China (Grant No. 2018YFA0306701), the National Supercomputing Center in Wuxi, the National Natural Science Foundation of China (Grant No. 61832015), and the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB28000000).

References

[1] “top500 list june 2017.” [Online]. Available: https://www.top500.org/lists/2017/06/

[2] S. Aaronson and L. Chen, “Complexity-theoretic foundations of quantum supremacy experiments,” in *Proc. 32nd Comput. Complexity Conf.*, 2017.

[3] I. L. Markov and Y. Shi, “Simulating quantum computation by contracting tensor networks,” *SIAM J. Comput.*, vol. 38, no. 3, pp. 963–981, 2008.

[4] G. F. Viamontes, I. L. Markov, and J. P. Hayes, *Quantum Circuit Simulation*. New York, NY, USA: Springer, 2009

[5] S. Boixo et al., “Characterizing quantum supremacy in near-term devices,” *Nature Phys.*, vol. 14, no. 6, p. 595, 2018.

[6] D. Castelvecchi, “IBM’s quantum cloud computer goes commercial,” *Nature News*, vol. 543, no. 7644, p. 159, Mar. 9, 2017.

[7] C. Song et al., “10-qubit entanglement and parallel logic operations with a superconducting circuit,” *Phys. Rev. Lett.*, vol. 119, Nov. 2017, Art. no. 180511.

[8] R. P. Feynman, “Simulating physics with computers,” *Int. J. Theor. Phys.*, vol. 21, pp. 467–488, 1982.

[9] H. Fu et al., “The sunway taihulight supercomputer: System and applications,” *Sci. China Inf. Sci.*, vol. 59, no. 7, Jun. 2016, Art. no. 072001.

[10] J. Fang, H. Fu, W. Zhao, B. Chen, W. Zheng, and G. Yang, “swDNN: A library for accelerating deep learning applications on Sunway TaihuLight,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2017, pp. 615–624.

[11] H. Fu et al., “18.9-Pflops nonlinear earthquake simulation on Sunway Taihulight: Enabling depiction of 18-hz and 8-meter scenarios,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2017, Art. no. 2.

[12] C. Yang et al., “10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2016, pp. 57–68.

[13] J. Zhang et al., “Extreme-scale phase field simulations of coarsening dynamics on the Sunway Taihulight supercomputer,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2016, Art. no. 4.

[14] M. A Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[15] K. De Raedt et al., "Massively parallel quantum computer simulator," *Comput. Phys. Commun.*, vol. 176, no. 2, pp. 121–136, 2007.

[16] M. Smelyanskiy, N. Sawaya, and A. Aspuruguzik, "qhipster: The quantum high performance software testing environment," 2016, *arXiv:1601.07195*.

[17] T. Häner and D. S. Steiger, "0.5 petabyte simulation of a 45-qubit quantum circuit," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Art. no. 33, 2017.

[18] E. Pednault et al., "Breaking the 49-qubit barrier in the simulation of quantum circuits," 2017, *arXiv:1710.05867*.

[19] H. De Raedt, F. Jin, and D. Willsch, "Massively parallel quantum computer simulator, eleven years later," *Comput. Phys. Commun.*, vol. 237, pp. 47–61, 2019.

[20] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, and H. Neven, "Simulation of low-depth quantum circuits as complex undirected graphical models, 2017, *arXiv:1712.05384*.

[21] Z. Chen, Q. Zhou, and C. Xue, "64-qubit quantum circuit simulation," *Chinese Sci. Bull.*, vol. 63, pp. 964–971, 2018.

[22] J. Chen et al., "Classical simulation of intermediate-size quantum circuits," 2018, *arXiv:1805.01450*.

[23] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, "Quantum supremacy is both closer and farther than it appears," 2018, *arXiv:1807.10749*.

[24] B. Villalonga et al., "A flexible high-performance simulator for the verification and benchmarking of quantum circuits implemented on real hardware," 2018, *arXiv:1811.09599*.

[25] M. C. Chen et al., "Quantum teleportation-inspired algorithm for sampling large random quantum circuits," 2019, *arXiv:1901.05003*.

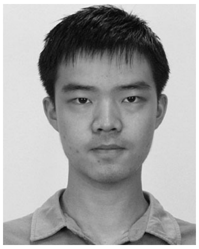
[26] J. Biamonte et al., "Quantum supremacy lower bounds by entanglement scaling," 2018, *arXiv:1808.00460*.

[27] A. Bouland, B. Fefferman, and C. Nirkhe, "On the complexity and verification of quantum random circuit sampling," *Nature Phys.*, vol. 15, no. 2, pp. 159–163, 2019.

[28] K. M. Svore et al., "Q#: Enabling scalable quantum computing and development with a high-level DSL," in *Proc. Real World Domain Specific Lang. Workshop*, 2018, Art. no. 7.

[29] D. Wecker and K. M. Svore, "LIQUI|⟩: A software design architecture and domain-specific language for quantum computing," 2014, *arXiv:1402.4467*.

[30] M. Ying, *Foundations of Quantum Programming*. San Mateo, CA, USA: Morgan Kaufmann, 2016.



Riling Li is currently working toward the PhD degree in the Department of Computer Science and Technology, Tsinghua University. His research interests include quantum programs and quantum circuits.

Bujiao Wu is currently working toward the PhD degree in the CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include quantum computing and algorithm complexity.

Mingsheng Ying received the bachelor's degree from Fuzhou Teachers College, Jiangxi, China, in 1981. He is a distinguished professor with the Centre for Quantum Software and Information, University of Technology Sydney, Australia, Cheung Kong professor with the Department of Computer Science and Technology, Tsinghua University, China, and a research professor with the Institute of Software, Chinese Academy of Science. His research interests include quantum computation, programming theory and logical foundations of artificial intelligence. He is the author of the books Topology in Process Calculus: Approximate Correctness and Infinite Evolution of Concurrent Programs (SpringerVerlag, 2001) and Foundations of quantum programming (Elsevier, 2016).

Xiaoming Sun received the PhD degree from Tsinghua University, Beijing, China. He is a professor with the CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences. His research interests include theoretical computer science, algorithms and quantum computing.

Guangwen Yang received the PhD degree from the Harbin Institute of Technology, Heilongjia, China. He is a professor with the Department of Computer Science and Technology, Tsinghua University, and the director of the Centre of High performance Computing in Tsinghua. His research interests include parallel computing and earth system simulation.