SW_Qsim: A Minimize-Memory Quantum Simulator with High-Performance on a New Sunway Supercomputer

Fang Li
National Supercompuer Center
Wuxi, China
38349735@qq.com

Xin Liu*
National Supercompuer Center
Wuxi, China
yyylx@263.net

Yong Liu
National Supercompuer Center
Wuxi, China
liuy_99@163.com

Pengpeng Zhao
National Supercompuer Center
Wuxi, China
zhaop_1213@163.com

Yuling Yang
National Supercompuer Center
Wuxi, China
yulingyoung@yeah.net

Honghui Shang
SKL of Computer Architecture,
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
shanghonghui@ict.ac.cn

Weizhe Sun
National Supercompuer Center
Wuxi, China
aworker_2010@126.com

Zhen Wang
National Supercompuer Center
Wuxi, China
okcwestbrook@foxmail.com

Enming Dong
National Supercompuer Center
Wuxi, China
dream0617@163.com

Dexun Chen
Tsinghua University
Beijing, China
adch@263.net

ABSTRACT

Classical simulation of quantum computation plays a critical role in numerical studies of quantum algorithms and the validation of quantum devices. Here, we introduce SW_Qsim, a tensor-network-based quantum simulator, which is designed with a two-level parallel structure for efficient implementation on the many-core New Sunway Supercomputer. We propose a minimize-memory contraction path algorithm for rectangular quantum grids to reduce the memory overhead, and provide the memory-limited simulation capacity of SW26010pro. Moreover, tensor operations are carefully optimized on the SW processor to achieve high performance. We design a fault tolerance mechanism to improve the extreme-scale parallel stability. We benchmark SW_Qsim's simulation of RQCs up to 400-qubits, achieving near-linear strong and weak scaling with up to 28.75 million cores, far beyond the previous state of the art. Our work sheds light on the development of efficient quantum algorithms for use in the physical, chemical, and engineering science fields.

*Corresponding authors

SC '21, November 14–19, 2021, St. Louis, MO, USA

ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476161

1 INTRODUCTION

Quantum computation is a new paradigm of computing that takes advantage of the quantum mechanics.Quantum computers can run specialized algorithms exponentially faster than their classical counterparts in some cases. According to the quantum superposition ,simulating a quantum computer using a classical machine is believed to be exponentially costly with respect to the number of qubits. Despite this, classical simulation of quantum computation is vital for the study of new algorithms and architectures. Simulation tools are needed on many different classical architecture, such as ProjrctQ[27] and Qiskit[33].However, as both the time and the memory required to simulate an elementary circuit operation grow exponentially with the number of qubits, it is necessary to design a distributed quantum simulator to take full advantage of high-performance supercomputers,for example QuEST[16], qHipster[25], Quantum++[7] and qflex[30, 31]. Qflex acquired efficient matrix multiplications and tensor transpose kernel on NVIDIA GPUs and developed fast sampling technique to solve million numbers of amplitude on Summit. Here, we propose SW_Qsim, a distributed, many-core accelerated simulator of quantum circuits on the New Sunway Supercomputer. We design a novel parallel contraction path algorithm to reduce memory usage, and carefully optimize tensor operations on SW26010pro. We also apply the fractal recursive slice parallel algorithm to distribute the huge memory and time requirements of the entire tensor network on the whole processors of the New Sunway Supercomputer. To improve the long-term calculation stability of SW_Qsim, we propose a job migration technology

and a step-by-step collective communication method, which enables SW_Qsim to work continuously to get the correct calculation results even if some nodes fail.

The rest of this paper is organized as follows. In Section II, the basic algorithms of the quantum simulator and the new Sunway Supercomputer are introduced. We present our optimization strategies in Section III. Then, the performance is described and analyzed in Section IV. Finally, we conclude in Section V.

2 BACKGROUND

2.1 Quantum Simulation Method of PEPS

Before describing our methods, we introduce some basic notation and definitions that are used in this paper. A qubit is a basic computing unit of a quantum computer and is usually represented by a two-level system: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$[21]. The initial state of an n-qubit quantum circuit simulation is usually presented by the product state[21]:

$$|\psi\rangle = |0\rangle^{\otimes n} = |0\rangle \otimes |0\rangle \otimes \cdots |0\rangle$$

Quantum gates are the basic operations of quantum computers. The gate operations in quantum circuits can always be represented by matrices in a classical simulation. A single-qubit gate can be represented by a rank-2 tensor, that is, a 2×2 matrix $U = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. The two-qubit gates can be represented by a rank-4 tensor(tensors of 4 indexes with dimension 2 each). In an n-qubit system, an entangled n-qubit gate operation can be expressed as a rank-2^n tensor. (tensors of 2^n indexes with dimension 2 each[30]). Some common quantum gates are exampled in TABLE 1.

Table 1: Tensor representation of specific gates in RQCs

Gate	Tensor
H	$\frac{\sqrt{2}}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
T	$\begin{pmatrix} 1 & 0 \\ 0 & \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i \end{pmatrix}$
$X^{1/2}$	$\frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$
$Y^{1/2}$	$\frac{\sqrt{2i}}{2}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$
CZ	$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

Some problems can be efficiently solved with a quantum computer but not with classical computers, even leading high performance computing(HPC) systems; these problems include Boson sampling[36], random quantum circuit (RQCs) sampling[1], quantum annealing[5], and quantum system simulation[2],et al. Among them, RQCs is proposed to measure the performance of quantum computing devices[3]. The generated rules of the RQCs circuits are as follows: first, apply the Hadamard gate to each qubit; second, repeatedly apply eight different configurations(in Fig. 1) of controlled-Z(CZ) gates until the desired circuit depth is obtained. This model ensures that once performing the cycle of the 8 configurations, each qubit will perform a CZ gate interaction with the adjacent qubit on the lattice. In addition to the CZ gate, a single quantum gate is also applied to the qubits that executed the CZ gate in the previous operation (but not the current operation). Except for the second single quantum gate (T gate) on each qubit (the first is a Hadamard gate), the single-qubit gate is randomly selected from $\{T, X^{1/2}, Y^{1/2}\}$ gates and must be different from the previous single-qubit gate on that qubit. Finally, apply the Hadamard gate to each qubit. Since RQCs comprises specific single-qubit gates and two-qubit gates between the nearest neighbor qubits that can be easily implemented in the current quantum chips, it is considered an ideal problem for current quantum computers[29]. We apply the tensors mentioned above to represent quantum gates in the classical computer simulations. Though the quantum gates are accessible, with hundreds of two-qubit gates, RQCs may have highly entangled quantum states, which are difficult to simulate even with the fastest supercomputers.

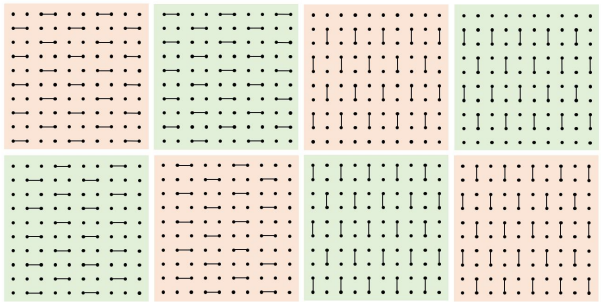


Figure 1: The 8 different CZ configurations of RQCs where the CZ gates are represented by a line between two qubits.

With increasing numbers of qubits, the computational and storage costs of RQCs increase exponentially. In recent years, many algorithms have been developed to simulate large-scale RQCs, including the full-amplitudes models. For example, ETH applied a scheduling algorithm to simulate a 45-qubit circuit with depth 25 on the Cori II supercomputer [13], IBM deferred tensor contractions to calculate all output amplitudes of a 7 × 7-circuit with depth 27 [23], and Tsinghua pushing the depth to 39 employing a novel partition scheme via a dynamic programming algorithm [18]. However, no one simulates full-amplitudes of circuits more than 49 qubits so far, since just the amplitudes' storage reaches the PB level. In terms of simulating more than 50 qubits RQC, there are three mainstream directions which all apply tensor networks and calculate only part of the amplitudes: the first one is based on partitioning a quantum circuit into sub-circuits[6, 18, 23],the second one is based on hypergraphs [8, 20, 35], where quantum gates and quantum circuits are viewed as the edges and vertexes in the hypergraph of the quantum circuits; the third one is inspired by the study of quantum entanglement in quantum information theory, including Density Matrix Renormalization Group [22, 32], matrix product states (MPS) [29] and projected entangled-pair states (PEPS)[9–11, 28]. A broad overview of existing simulators in the field of RQC is listed in TABLE 2.

The PEPS method of simulating quantum many-body systems is built on the framework of tensor network contraction. Considering a 2D rectangular lattice, $L_v \times L_h$, where L_v and L_h represent the vertical and horizontal dimensions of the lattice, respectively, a

Table 2: Simulators in the field of RQC

Year	Reference	Platform	Qubits	Depth
2018	[18]	Sunway	7*7	55
2018	[4]	ACQDP	9*9	40
2019	[35]	ACQDP	Bri-70	40
2019	[9]	Tianhe-2	8*8	40
2019	[30]	Pleiades and Electra	7*7	40
2021	Our work	New Sunway	8×8	40
			10×10	32
			20×20	16

qubit is depicted as a rank-5(rank implicates the number of indexes of tensor) tensor $A^{\sigma_n}_{l_n,r_n,u_n,d_n} \triangleq A^{\sigma_n}_n$, where $\sigma = 0, 1$ represents the physical dimension, and l,r,u,d represent four auxiliary dimensions connected to its left, right, up, and down neighbor qubits in PEPS theory. We use $N = L_v \times L_h$ to represent the number of qubits, then the quantum state on this lattice can be expressed as

$$|\psi\rangle_{PEPS} = \sum_{\sigma_1,\cdots,\sigma_N} Tr(A_1^{\sigma_1} A_2^{\sigma_2} \cdots A_N^{\sigma_N}) |\sigma_1, \cdots, \sigma_N\rangle$$

The main operation in the PEPS method is the tensor contraction, which contracts all the auxiliary dimensions of the tensor. In order to implement a single-amplitude simulation, we need to determine the overlap between the evolution of the initial state and the target state. As the numbers of qubits and circuit layers increase, the size of the tensor grows exponentially. When performing a single-amplitude simulation in the $m \times n \times (1+L+1)$, $(m \geq n)$ rectangular grid RQCs, the largest tensor network slice will reach a size of $2^{\lceil L/8 \rceil (n+1)} \times 16$ bytes if the tensor network is contracted in the normal way, where $m \times n$ is mn-qubit rectangular grid and L is the circuit depth. For example, for $L = 32$, $m = 10$, $n = 10$, the largest tensor slice in the progress of tensor network construction reaches 0.25 PB in size, which cannot be saved by any node on any computer. To break the bottleneck of memory overhead of single node, we managed to slice the tensor network to many sub-tasks, descend the dimension of the intermediate tensor networks in tensor construction and implement the simulation on the distributed New Sunway Supercomputer with its extreme-scale nodes.

2.2 The New Sunway Supercomputer

The New Sunway Supercomputer is the successor of the Sunway TaihuLight, which adopts a new generation of high-performance heterogeneous many-core processor SW26016pro and interconnection network chips. SW26016pro is designed for massive thread and data parallelism and to deliver high performance on parallel workloads. Its architecture is shown in Fig. 2. The processor chip is composed of 6 core-groups (CGs), connected via a network on chip(NoC). Each core group includes a management processing element (MPE) and 64 computing processing elements (CPEs) arranged in an 8 by 8 grid, a total of 390 cores.

Similar to its predecessor, SW26016pro is relatively modest in terms of the memory capacity, with 96 GB memory and 16 GB memory in each CG. The MPE and the CPEs within the same CG share the same memory which is controlled by the MC. Each CPE has a 32 KB L1 instruction cache, and a 256KB scratch pad memory (SPM, also called the Local Data Memory (LDM)). Part of LDM can also be configured as a local data cache (LDCache) which is automatically managed by the hardware to optimize the performance of memory access. Data transfer between LDM and main memory can be realized by direct memory access (DMA). The data exchange between each two CPEs in the same CPE cluster is achieved through the Remote Memory Access (RMA) interface. The CPE adopts the SW64 instructions and provide 512-bit SIMD support.

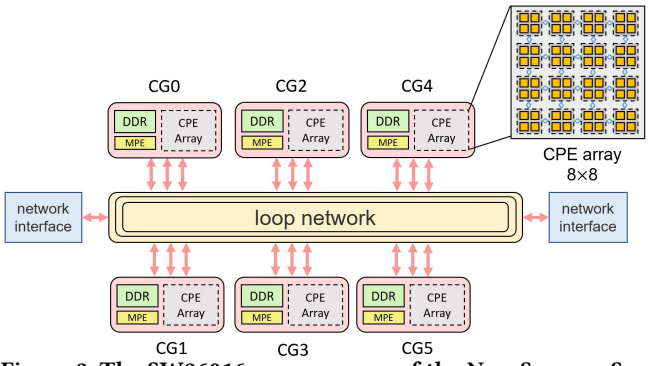


Figure 2: The SW26016pro processor of the New Sunway Supercomputer.

2.3 GEMM and Tensor contraction optimizations

The matrix-matrix multiplication usually plays a critical role in many scientific and engineering applications, like weather forecastquantum physics, hydromechanics, etc. GEMM (General Matrix Multiply) develops fast and achieve many mature research results in order to fully exploits the parallelism and near-processor memory of the powerful high-performance computing systems, especially with the help of modern heterogeneous many-core processors. For example, in Sunway TaihuLight supercomputer, [15] proposed five surrogate algorithms and a machine learning–based algorithm selector to cope with batched matrix multiplicationsreaching around 84.8% of the performance upper bound. [34] presented a runtime adaptive matrix multiplication methodology, RTAMM, to further adapt to various matrix shapes in the SW26010 architecture. [14] presented the double-precision general format matrix-matrix multiplication (DGEMM) kernel on the SW26010 processor. Although the New Sunway supercomputer inherits from the Sunway TaihuLight supercomputer, the architecture details of the new generation many-core processor (SW26010pro) are different from the previous one (SW26010). More importantly, there are various shapes of matrices in the field of RQC simulation, especially a large number of narrow matrices with relatively large aspect ratio, but the previous research mainly focused on square matrix multiplication. Therefore, redesigning the algorithm of matrix multiplication, especially narrow matrix multiplication to fully exploit the computing capability of SW26010pro is a necessity.

In recent years, tensor contraction have been widely used in emerging fields such as deep learning and quantum computing. Tensor-tensor multiplication is an extension of matrix-matrix multiplication, which has gradually attracted scientists' attention and become a promising research topic. For example, [11] developed

TNSPackage, which includes all kinds of basic tensor operations designed for tensor network states (TNS) algorithms by transforming the tensors to dense matrices. NVIDIA provided cuTENSOR[26], a high-performance GEMM-like Tensor-Tensor Multiplication. [12] presented "GEMM-like Tensor-Tensor multiplication" (GETT), which attained up to 91.3% of peak floating-point performance for compute-bound tensor contractions. However, there is currently no similar library on the New Sunway Supercomputer, so we need to develop related algorithms and make it a tensor library for Sunway architecture in the future.

3 INNOVATIONS

SW_Qsim is based on PEPS, which is especially suitable for massive parallelization. We use two-level hardware-friendly parallelism on the many-core New Sunway Supercomputer, which enables us to perform simulations with high numbers of qubits and deep depth of quantum circuits.

3.1 Process-level parallelism

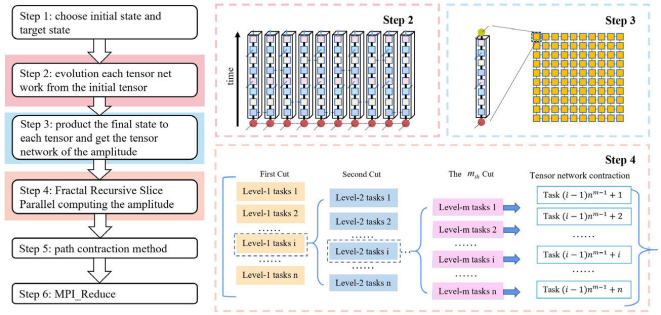


Figure 3: Execution steps of SW_Qsim.

SW_Qsim uses PEPS to calculate the probability amplitude of a single final state of a quantum grid. The execution steps are shown in Fig. 3. First, we start from the initial state of each qubit which is represented by a vector $(0, 1)$, and perform tensor network evolution according to a given RQC, evaluating each qubit to a tensor. Second, we multiply the final state of each qubit $(0, 1)$ or $(1, 0)$ to their evaluated tensor, and obtain the target tensor network which equals to the scalar Z that represents the amplitude of the final state. Then, we decompose the tensor network into numbers of independent sub-tensor network using the "fractal recursive slice parallel" method, and each sub-tensor network contracts, resulting in a scalar. Finally, we collect the contraction results obtained on these parallel tasks, and the probability of transitioning the quantum grid from the initial state to a certain final state after the evolution of the random circuit is obtained.

3.1.1 Fractal recursive slice parallel algorithm. Tensor contraction produces massive intermediate tensor networks, which cannot be stored on a single computing node. In order to avoid memory bottlenecks, a fine-grained "cut" technology has been reported to decompose the contraction of the squared tensor network into independent contraction of several easy-to-calculate sub-tensor networks, so that the memory requirements are balanced and the parallel calculations are done independently[19].

On this basis, we further developed the "cut" technology and proposed a method of fractal recursive slice parallel. Fractal recursive slice comes from fractal theory, and its typical feature is recursion. The branches obtained by one division continue to be divided according to the same rules, and each branch in different levels is self-similar.

The layer number L of RQCs is a multiple of 8 in our simulation to implement the eight different CZ configurations(Fig. 1) the same times, making the dimension of each index the same. As examples, the fractal recursive slice parallel of 8-layer and 16-layer RQCs are shown in Fig. 4. In Fig. 4(a) one slice produces two sub-tasks, two slices produce four sub-tasks, three slices produce sixteen sub-tasks and so on. In Fig. 4(b) one slice produces four sub-tasks, two slices produce sixteen sub-tasks, three slices produce sixty-four sub-tasks and so on. Generally for an L-layer RQCs, assuming that the dimension of each rank in all tensors is $d(d = 2^{\lceil L/8 \rceil})$($\lceil\rceil$ is the ceiling function), after s fractal recursive slicings, the computation of each amplitude of the quantum circuit can be divided into $d^s = 2^{\lceil L/8 \rceil s}$ tasks. Each task obtains its tensor branch by a vector projection algorithm and contracts from the first bit to the last bit along the contraction path, resulting in a scalar at last.

We slice the tensor network recursively level by level as the contraction advances, so that all of the sub-tasks have the same sub-tensor network structure, achieving a good load balance. For every level of the fractal recursive slice parallel, the tensor rank of the two cut qubits are reduced by one. The number of slices is adjusted according to the size of the memory of the computing node, which is how trade time for space. Moreover, tensor network cutting is a communication-avoiding algorithm, meaning that our simulator has high scalability.

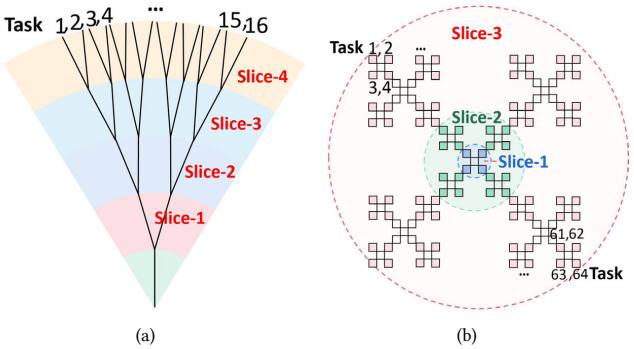


Figure 4: (a) 8-layer RQCs, 4 slices and16 sub-tasks (b) 16-layer RQCs, 3slices and 64 sub-tasks

3.1.2 Minimize-memory contraction path of tensor network. Using the fractal recursive slice method described in the previous section, we divide the tensor into several sub-tensor networks. These branches are recursively contracted in parallel on the supercomputer system. The contraction path is extremely important for each sub-tensor network, since new intermediate tensors will be produced, and the rank of them is decided by the contraction path. Searching for the optimized contracting path is an NP-hard problem[30]. However, we find a minimize-memory contraction path for the s level continuous slice in the same direction for the rectangular tensor network.

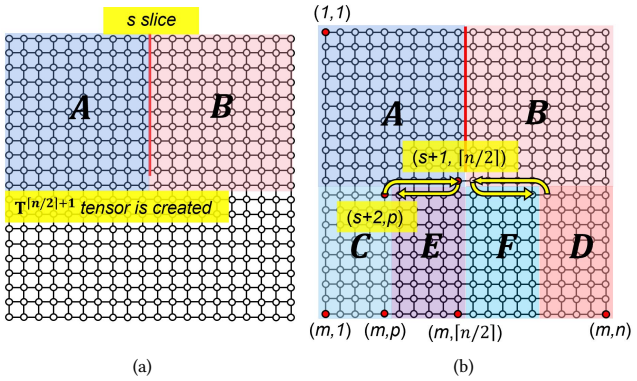


Figure 5: (a) contract A and B, tensor $T^{\lceil n/2\rceil+1}$ is created;(b) Minimize-memory contraction path

Take the $m \times n (m \geq n)$ bits rectangular tensor network shown in Fig. 5(a) as an example. After s continuous slices in the same direction, two partitions A and D have been generated. After they are contracted respectively, tensors of rank $\lceil n/2 \rceil + 1$ will be created. In the sequential contraction, the minimize-memory contraction path is found when the rank of a intermediate tensor does not exceed $\lceil n/2 \rceil + 1$ in it.

Fortunately, we find this kind of minimize-memory contraction path method shown in Fig. 5(b). The rectangular tensor network is divided into six partitions of left-right symmetry for contraction. We take the left partitions as an example to illustrate our contraction path.

- Part A: contract from the upper left corner position $A(1,1)$ to the lower right corner position $A(s+1, \lceil n/2 \rceil)$; the maximum rank of the tensor in this process is $\lceil n/2 \rceil + 1$.
- Part C: contract from the lower left corner $C(n,1)$ to the upper right corner $C(s+2, p)$; the maximum rank of the tensor in this process is $m - s + p - 1$.
- Merge A+C: the latest tensors $C(s+2, p)$ and $A(s+1, \lceil n/2 \rceil)$ are contracted together; the rank of the new tensor in this process is $m + \lceil n/2 \rceil - s - p$.
- Merge A+E: $A(s+1, \lceil n/2 \rceil)$ and $E(s+2, p+1)$ are contracted together; the rank of the new tensor in this process is still $m + \lceil n/2 \rceil - s - p$.
- Part E contracts to $E(n, \lceil n/2 \rceil)$, the order of which is $m - s$.

The maximum rank of the intermediate tensors during the entire contraction process is:

$$r = max(\lceil n/2 \rceil + 1, m - s + p - 1, m + \lceil n/2 \rceil - s - p)$$

The objective optimization function is:

$$r_{min} = f(s,p) = \begin{cases} m - s + p - 1 \leq \lceil n/2 \rceil + 1 \\ m + \lceil n/2 \rceil - s - p \leq \lceil n/2 \rceil + 1 \end{cases}$$

Since m and n are known, if we set s satisfying $2(m-s) \leq \lceil n/2 \rceil + 3$, it is sure that we can get p from the relationship:

$$m - s - 1 \leq p \leq \lceil n/2 \rceil - m + s + 2$$

At this time $r = \lceil n/2 \rceil + 1$. As explained above, if the order of the intermediate tensors does not exceed $\lceil n/2 \rceil + 1$, it is the minimize-memory one. So it is proved that in the case of s level continuous slice in the same direction, our contraction path algorithm must be the minimize-memory one.

3.1.3 Memory-limited Simulation Capacity on SW26010pro. We hope to simulate more qubits and more complex quantum circuits based on traditional high-performance computers. However, due to the limitation of the processor memory, the simulation capabilities of quantum simulators are limited. As we have found the minimize-memory contraction path, we can calculate the memory-limited simulation capacity of any processor. The maximum rank of the intermediate tensor generated during the contraction is $r = \lceil n/2 \rceil + 1$ for the $m \times n$ bits rectangular tensor network. In the contraction process, we use data multiplexing to reduce memory overhead further. At this time, only threefold spaces of r-rank tensor are needed. The main memory requirement in the minimize-memory contraction path occurs when the tensors reach the maximum rank $\lceil n/2 \rceil + 1$, which means that contractions meet from two different directions. Then there needs memory to store at least 3 maximum rank tensors. Since the circuit evolution takes every 8 depths, the bond dimension of the 2D tensor network is $2^{\lceil L/8 \rceil}$[9]. The data is stored with single precision in our simulation, which uses 8Bytes to store a complex number. The main memory usage can be expressed as the following:

$$M = 3 \times 8 \times 2^{\lceil L/8 \rceil (\lceil n/2 \rceil + 1)} / 1024^3 (GB)$$

Thus, the relationship between the maximum number of qubits $m \times n$ and quantum depth L of a specified processor satisfies:

$$\lceil L/8 \rceil (\lceil n/2 \rceil + 1) = 27 + log_2(M/3)$$

As described in Chapter 2.2, the memory of one processor of SW26010pro is 96GB, its simulation capabilities of various circuit grids can be obtained, as shown in TABLE 3.

3.2 Thread-level parallelism

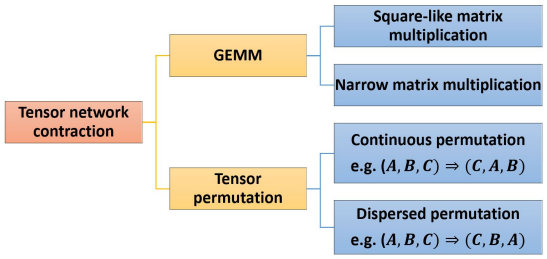


Figure 6: Thread-level parallelization of SW_Qsim

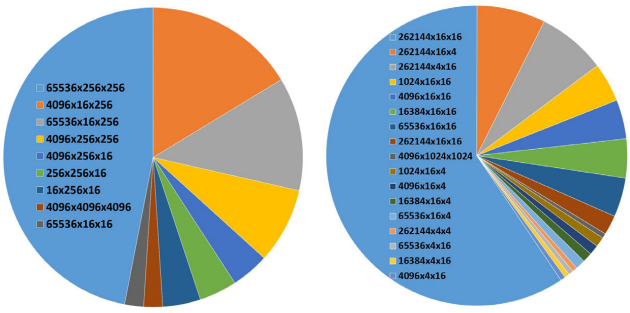
The thread-level parallelization of SW_Qsim is shown in Fig. 6, which is implemented on one CG of SW26016pro. Thread-level parallel strategies and optimization methods are essential to the overall performance of SW_Qsim, as 64 CPEs in one CG provide the main computing performance. The major time-consuming step of SW_Qsim is the tensor contraction, consisting of two main kernels GEMM and tensor permutation.

3.2.1 GEMM optimizations. In the step of high-rank tensor contraction, the most time-consuming calculation is the matrix–matrix multiplication of various dimensions. Fig. 7 shows typical statistical information about the dimension ratio of GEMM in SW_Qsim, indicating that the matrices are highly irregular. For user convenience, the New Sunway provides a high-performance extended math library SW_BLAS. SW_BLAS has extraordinary performance

Table 3: Memory-limited simulation capabilities of SW26010pro

circuit grid	slice number	maximum rank of tensors	depth	memory of three largest tensor(GB)	SW26010pro capacity
Rec-8×8	5	5	48	24	√
			56	768	×
Rec-10×10	6	6	40	24	√
			48	1536	×
Rec-12×12	8	7	32	6	√
			40	768	×
Rec-20×20	14	11	16	0.375	√
			24	192	×
Rec-32×32	24	17	8	0.00294	√
			16	384	×
Rec-50×50	36	26	8	1.5	√
			16	1.005E+8	×
Rec-60×60	44	31	8	48	√
			16	1.02E+11	×
Rec-62×62	45	32	8	96	×

for square matrices, but the performance is much worse and unsatisfactory for very narrow matrix, such as many matrix shown in Fig. 7 Therefore, we divide the matrix into two types: the square-like matrix and the narrow matrix, and study their GEMM optimization methods separately.



(a) (b)

Figure 7: Distribution of matrix sizes of GEMM:(a) 100-bit 32-depth RQC: $10 \times 10 \times (1+32+1)$**; (b) 400-bit 16-depth RQC:** $20 \times 20 \times (1+16+1)$

(1) **Square-like matrix multiplication.** In $A_{m\times k} \times B_{k\times n} = C_{m\times n}$, if the subscripts of m, k and n are close, which means the shapes of matrix A and B are square-like, this kind of matrix multiplication is a computation-intensive problem. Fig. 8 demonstrates its optimization method. Firstly, the matrices A and B are distributed to the LDM of the 64 CPEs of one CG, and each CPE calculates block-matrix multiplication with data on its own LDM. Secondly, blocks of matrix A are broadcast on the same row by RMA. To avoid communication hotspots, the row-broadcast is initiated by the CPE on the diagonal of the 8×8 CPEs array. Correspondingly, blocks of matrix B are column-broadcast by the CPE on the opposite diagonal. And then, according to the corresponding subscript relationship of blocks of A and blocks of B, the intermediate results are computed. After 8 row-broadcast and column-broadcast, the calculation of the block matrix C is gradually completed. Finally, we accumulate these block-results to get the final result.

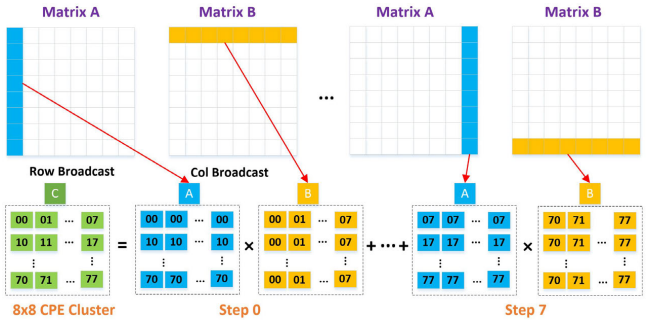


Figure 8: Optimization method of square-like matrix multiplication

(2) **Narrow matrix multiplication.** In $A_{m\times k} \times B_{k\times n} = C_{m\times n}$, if one or two of the subscripts m, k and n is much smaller than the other two, that is, the shape of matrix A or B is narrow, this kind of matrix multiplication is a bandwidth-constrained problem. Fig. 9 demonstrates its optimization method. Assuming k is much smaller than m and n, we use a one-dimensional data division algorithm, which distributes matrices A and B to each CPE in the m or n direction respectively. Each CPE broadcasts blocks of B stored on its LDM to other CPEs in turn by RMA. Then all of CPEs multiply blocks of A stored on its LDM and blocks of B received from other CPEs Finally, the block-results are accumulated to get the final result.

3.2.2 Tensor permutation optimizations. In order to improve the efficiency of high-rank tensor contractions and make full use of high-performance 2D matrix multiplication, we first reformat the tensor contractions into 2D matrix multiplications using tensor permutation. According to different dimensions of tensor permutation, we propose the following two methods.

(1) **Continuous permutation**. If the tensor permutation can be treated as 2D matrix transpose, for example: $(A, B, C) \Rightarrow (C, A, B)$can be treated as $(A', B') \Rightarrow (B', A')$, here $A' = (A, B)$ and $B' = C$, this is a continuous permutation. For the case of continuous permutation, the essence of optimization is the regular movement of datas

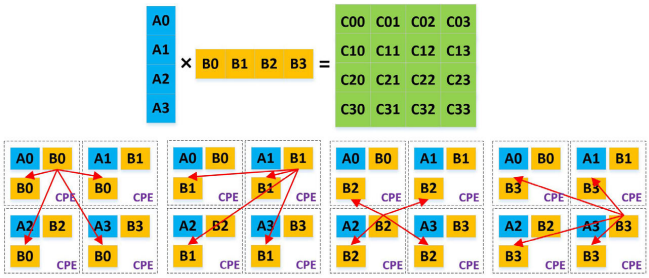


Figure 9: narrow matrix multiplication

inside the continuous two dimensions. We accelerate the regular movement of datas on SW26010pro as follows:

- load the continuous data that need to be transposed to the LDM of each CPE in blocks by DMA;
- each CPE concurrently transposes data stored on its own LDM using the SIMD "vshuff" instruction;
- each CPE writes the transposed data to the corresponding location in the main memory by DMA.

The overall process is shown in Fig. 10.

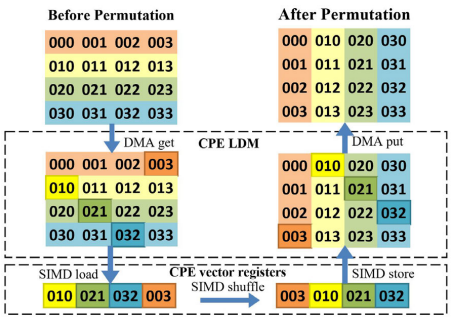


Figure 10: Continuous lowest 2D transposition.

(2) **Dispersed permutation**. If the two dimensions of a tensor needed to be permuted is discontinuous, such as $(A, B, C) \Rightarrow (C, B, A)$, this is a dispersed permutation. In this case, the essence of optimization is a regular movement of a piece of dispersed data. Assuming that the data is stored continuously in low dimensions, high-dimensional data should be permuted by striding DMA, and low-dimensional data should be permuted by SIMD shuffle instruction on each CPE as much as possible. The movement of the dispersed datas segment is accelerated on SW26010pro as follows:

- according to the total size of the dimensions that a tensor needs to permute, we distribute the whole data of these dimensions on the 64 CPEs;
- each CPE loads the data distributed to its own LDM in blocks by striding DMA and completes the permutation of (A,B,C,D) to (B,A,C,D). The LDM of each CPE stores data in the order of (A',C,D), where A'=(B,A);
- each CPE performs one two-level SIMD shuffle to permute the data from (A',C,D) to (D,A',C)
- each CPE writes its permuted block data back to the corresponding position in the main memory by the stride DMA interface and permutes the data to the final order (D,B,A,C)

The overall process is demonstrated in Fig. 11 and the stride DMA is shown in Fig. 12.

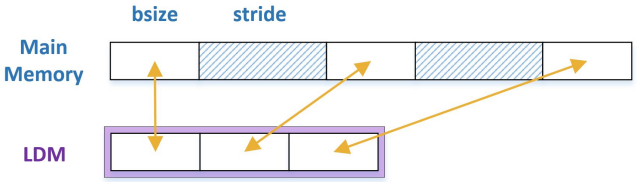


Figure 12: The stride DMA

3.3 Fault tolerance and parallel stability

As the mean time between failures of any supercomputer is limited, running jobs stably for a long time with up to 28.75 million cores is a great challenge. The parallel calculation process of SW_Qsim is shown in Fig. 13(a). If all of the nodes are continuously healthy, each process calculates in parallel. After all the calculations are finished, the results will be collected by using MPI_Reduce. However, if any node is unavailable, MPI_Reduce will fail since there is no fault tolerance mechanism in MPI. The breakpoint is a common mechanism for fault tolerance, but the frequent breakpoint operations will reduce the computational efficiency, and the intermediate data in more than 28.75 million cores need to be saved simultaneously, which brings tremendous pressure to the I/O system. and the job can only be terminated.

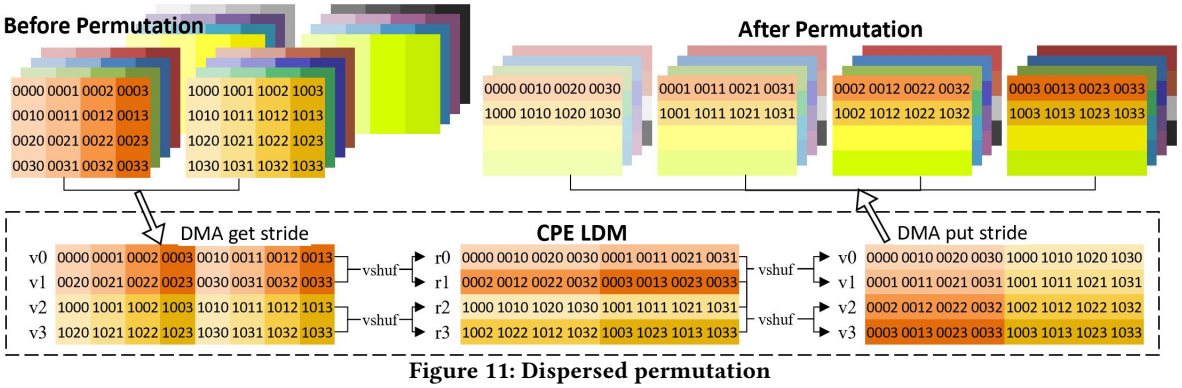


Figure 11: Dispersed permutation

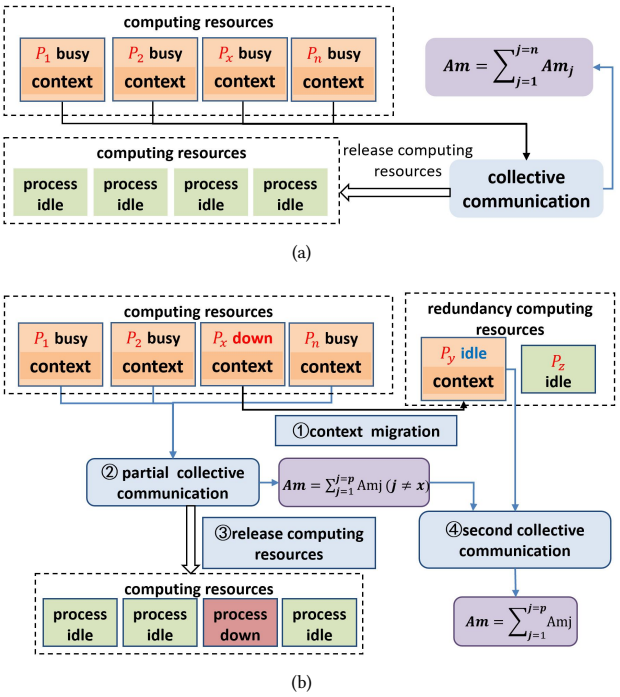


Figure 13: (a) Normal flowchart of parallel job; (b) The flowchart of exception-handling

A new parallel fault tolerance mechanism of job migration and step-by-step collective communication is proposed to improve the job robustness, combining with the parallel software environment of the New Sunway. The mechanism is based on two features of the SW_Qsim. The first is the relatively computational balance of tasks, which means we can use static task partition. As a result, the tasks computed on each node are certain. The second is the use of communication avoidance algorithms, which avoids interprocess communication in the processing of SW_Qsim except for the messages generated by MPI_Reduce. The mechanism is realized in combination with the runtime system. When the SW_Qsim is running, the runtime system will run a process to detect the node status. If a node error is detected, the process will migrate the tasks on the node to an idle healthy node through a log-file that records the tasks assigned to the error node. Meanwhile, the runtime system will set the error node status to be failed, which is used for the partial collective communication. The mechanism is shown is Fig. 13(b).

(1) During the calculation process, if one node fails, it will be detected by the runtime process and its status will be set to failed in the node status list. The tasks of the failed node have been recorded in a log-file, so that we can restart these tasks on an idle healthy node. (2) When the tasks of the original healthy nodes end, these parts of calculation results are collected by partial collective communication. The partial collective communication is based on the node status list provided by the runtime process, in which only the data of the healthy nodes are collected, and data of the failed nodes are ignored. (3) The computing resources of the ending tasks are released. (4) After the calculation, the simulation result of the migrated task is merged with others by further collective communication. At this time, the whole simulation result is obtained.

4 PERFORMANCE MEASUREMENT

4.1 Simulation validation

We use Porter–Thomas distribution and a variational quantum algorithm to validate the accuracy of the quantum simulator.

4.1.1 Amplitude distribution of the states of the random quantum circuits. We have measured 100 thousands amplitudes with a $20 \times 20 \times (1 + 8 + 1)$ scale quantum circuit, where 20×20 is 20^2-qubit square grid and 8 is the circuit depth. The output result conforms to the Porter–Thomas distribution law, as Fig. 14 shows.

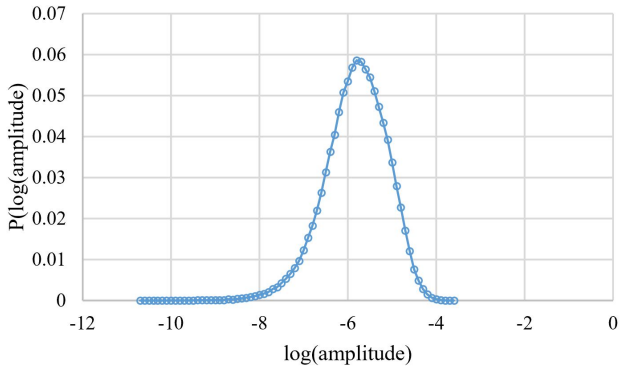


Figure 14: Amplitudes distribution law of the states of random quantum circuits.

4.1.2 Variational quantum algorithm. The variational quantum eigensolver (VQE) [24] is a promising algorithm that can be used to simulate many-body systems in chemistry, physics, and material science using near-term quantum computers. The VQE is based on the Rayleigh–Ritz variational principle, and is used to find the molecule's ground state energy. The VQE is a hybrid quantum–classical algorithm, in which a quantum computer is used to prepare and measure the wave function (state), while a classical computer is used to update the quantum circuit parameters using an optimization algorithm. Here, we use the SW_Qsim quantum simulator to prepare the ansatz state of the molecule by applying the ansatz circuit to it.

We apply the Trotter quantum ansatz circuit (denoted by $U(\theta)$, where θ indicates the variational parameters) with a jellium Hamiltonian, and apply it to the four-qubit H_2 molecule system. The circuit for the ansatz state $|\psi(\theta)\rangle = U(\theta)|0000\rangle$ is shown in Fig. 15.

The comparison of simulation results and theoretical results is shown in Fig. 16. The final ansatz state $|\psi(\vec{\theta})\rangle$ consists of the state $|0101\rangle$, $|0110\rangle$, $|1001\rangle$, $|1010\rangle$, which is exactly the same as the reference result [17].

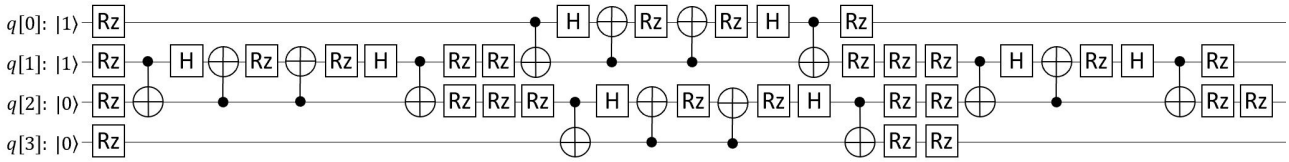


Figure 15: Circuit for the ansatz state

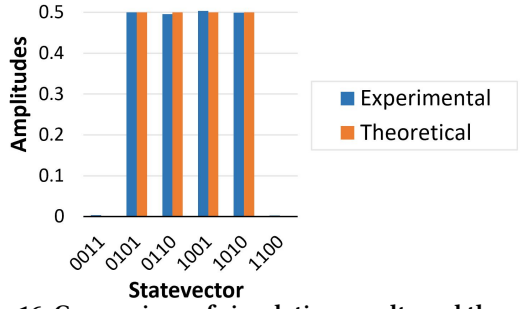


Figure 16: Comparison of simulation results and theoretical results

Figure 17: Performance of square-like matrix multiplication

4.2 PERFORMANCE RESULTS

4.2.1 Memory requirement. The contraction path of the tensor network determines the rank of the new intermediate tensors during the contraction process. Regarding the $n \times n$ bits rectangular tensor network if we take contraction simply rows by rows, there is no cut for the tensor network, and the maximum rank is n+1. If we take plain cuts vertically for s times at the middle sites of the tensor network[30], the maximum rank is $\lceil n/2 \rceil + m - s - 1$. After s cuts (taking plain cuts for s times), we found a special contraction path meeting from two directions. The contraction path reduces the maximum rank of the intermediate tensors to $\lceil n/2 \rceil + 1$, which is minimized-memory for rectangular grid tensor networks. Take a $10 \times 10 \times (1 + 32 + 1)$ quantum circuit as an example, the memory requirement for the minimized-memory contraction path stands out when comparing to other methods demonstrated in TABLE 4. The maximum rank of intermediate tensors reduces from 11 to 8, and finally to 6 through the three contraction paths. Moreover, the memory of the maximum rank tensor with "minimize-memory cut" is reduced by 256 times compared with "plain cuts" algorithm.

Table 4: The maximum rank of tensors and the memory requirement of RQC $10 \times 10 \times (1 + 32 + 1)$

Contraction Paths	Cut	Max rank of intermediate tensors	Memory of the maximum rank tensor (GB)
conventional (rows by rows)[9]	0	11	131072
plain cuts[30]	6	8	32
minimize-memory cuts(Our work)	6	6	0.125

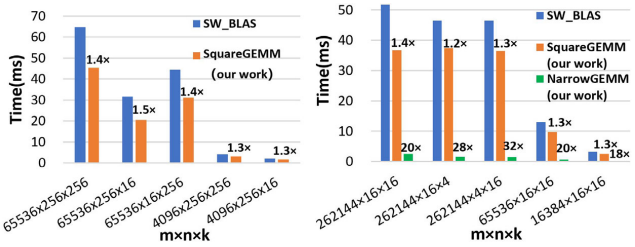


Figure 18: (a) Performance of square-like matrix multiplication

4.2.2 GEMM acceleration on SW26010pro. We firstly examine the performance of GEMM, which is the main kernel of SW_Qsim. The comparison of GEMM performance between SW_BLAS and our square-like matrix multiplication is illustrated in Fig. 17. We can observe that the average performance of the square-like matrix can deliver 2000 GFLOPS, and the maximum performance can reach 2200 GFLOPS, about 94% of peak performance. In general, the performance of our algorithm is 10%-40% higher than that of SW_BLAS.The comparison of GEMM performance in RQCs is illustrated in Fig. 18(a) and Fig. 18(b). Fig. 18(a) shows the run-time of the key matrix multiplication of $10 \times 10 \times (1 + 32 + 1)$ RQC. These are square-like matrices, and the performance of our algorithm is 30%-50% higher than the performance of SW_BLAS. Fig. 18(b) shows the run time of the key matrix multiplication of RQC $20 \times 20 \times (1 + 16 + 1)$, which contains mostly narrow matrices. The performance of square-like matrix multiplication is 30% higher than that of SW_BLAS, which does not meet our expectations. However, the performance of our narrow matrix multiplication is 20 times better than that of SW_BLAS on average, which illustrates the effectiveness of our optimization techniques.

4.2.3 Tensor permutation acceleration on SW26010pro. Secondly we evaluate the performance of tensor permutation, the other time-consuming kernel of SW_Qsim. Tensor permutation is a classical anti-storage problem. The bandwidths of continuous permutation and dispersed permutation are shown in Fig. 19. We can observe that the performance of continuous tensor permutation is over 40GB/s (about 80% of the theoretical value) except some small tensor while it is about 30GB/s for the case of dispersed tensor permutation, because the difficulty and the optimization methods of the two kinds of problems are different.

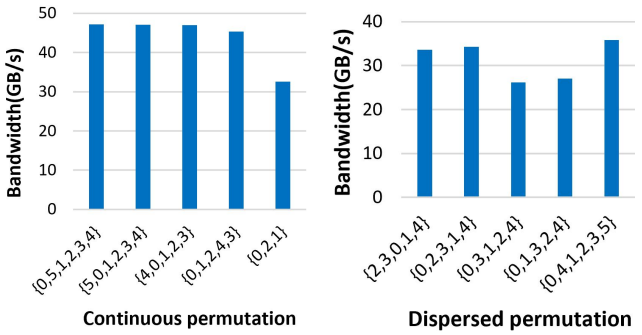


Figure 19: Performance of tensor permutation: (a) continuous permutation; (b) dispersed permutation. The abscissa indicates the order of the tensor after permutation while its original order is $\{0, 1, 2, 3, 4, 5\}$, $\{0, 1, 2, 3, 4\}$ and so on.

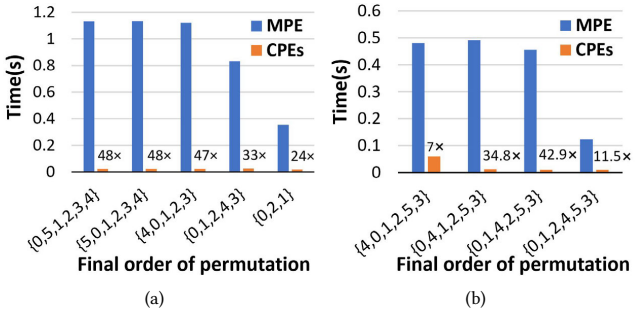


Figure 20: tensor permutation performance between MPE and CPEs: (a) continuous permutation; (b) dispersed permutation. The abscissa indicates the order of the tensor after permutation while its original order is $\{0, 1, 2, 3, 4, 5\}$, $\{0, 1, 2, 3, 4\}$ and so on.

The comparisons of tensor permutation performance between MPE and CPEs are illustrated in Fig.26 and Fig.27. For the continuous permutation, due to the 64 CPEs and the use of SIMD vshuff, the acceleration of one CG to one MPE is about 20 to 48 times. For the dispersed permutation, the acceleration of one CG to one MPE is from 7 to 43 times according to various cases of tensor permutation.

4.2.4 Strong scaling results. In the strong scaling tests, we record the total simulation time of two RQCs of $10 \times 10 \times (1 + 32 + 1)$ and $20 \times 20 \times (1 + 16 + 1)$, representing 100-bit 32-depth and 400-bit 16-depth respectively. While the problem size is fixed, we increase the number of processor cores from 1,064,960 to 28,753,920 to investigate the parallel efficiency. The test results are shown in Fig. 21, in which the proportions of main kernels are also presented. From the figures, we can observe that the major time-consuming kernels of SW_Qsim are GEMM and tensor permutation. For the RQC of $10 \times 10 \times (1 + 32 + 1)$, GEMM and tensor permutation take up approximately 86% and 10% of the total time respectively. And for the RQC of $20 \times 20 \times (1 + 16 + 1)$, contribution of GEMM and tensor permutation become approximately 90% and 8% respectively. This is because on the whole, the many-core acceleration effect of GEMM on large matrices is better than that of smaller matrices. During the contraction of the tensor network, the matrix scale of the matrix multiplication in the RQC of $10 \times 10 \times (1 + 32 + 1)$ is larger than that in the RQC of $20 \times 20 \times (1 + 16 + 1)$, so the former RQC performs better with relatively short time. In the strong scaling test, GEMM and tensor permutation run many times, but the evalutation and MPI_Reduce of the other two kernels run only once, so their time proportion is very small. Meanwhile, as the number of cores increases by multiplication, the execution times of GEMM and tensor permutation continue to decrease in multiples, so their time proportion decreases gradually. In terms of the strong scalability, the over 95% parallel efficiency is achieved with up to 28.75 million cores.

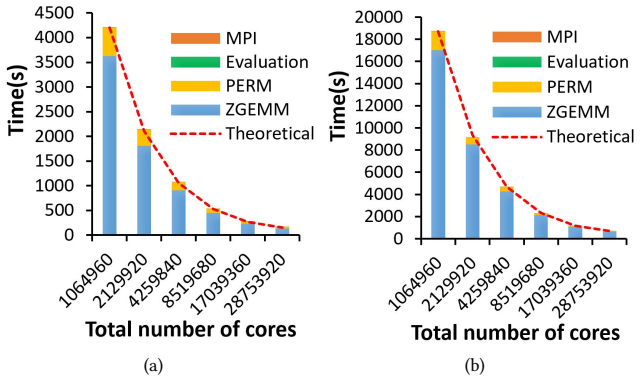


Figure 21: Strong scaling results from 1064960 cores to 28753920 cores:(a) RQCs of $10 \times 10 \times (1 + 32 + 1)$ (b) RQCs of $20 \times 20 \times (1 + 16 + 1)$

4.2.5 Weak scaling results. Different from the strong scaling run, while the number of processor cores increases, each processor of weak scaling tests does the same number of tasks. Considering the same two RQCs: $10 \times 10 \times (1 + 32 + 1)$ and $20 \times 20 \times (1 + 16 + 1)$, the test results are presented in Fig. 22. As the number of cores increases, the execution time of all kernels keep constant, and so their time proportions are basically unchanged. For both of the two cases, scaling from 1,064,960 cores to 28,753,920 cores, the parallel efficiency maintains over 95%.

Table 5: Computational efficiency between SW_Qsim and Guochu's work

Authous	Circuit	Nodes	Peak performance /node	Runime (min)
Guochu[9]	$7\times7\times(1+40+1)$	4096	3.4Tflops	31
Our work	$7\times7\times(1+40+1)$	995	14Tflops	23.8
Guochu[9]	$8\times8\times(1+37+1)$	4096	3.4Tflops	68
Our work	$8\times8\times(1+37+1)$	995	14Tflops	48.6
Guochu[9]	$9\times9\times(1+31+1)$	2048	3.4Tflops	22
Our work	$9\times9\times(1+31+1)$	498	14Tflops	20.2

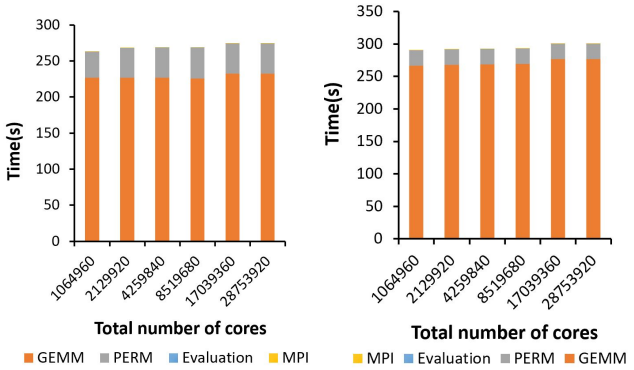


Figure 22: Weak scaling results from 1064960 cores to 28753920 cores:(a) RQCs of $10\times10\times(1+32+1)$ (b) RQCs of $20\times20\times(1+16+1)$

4.2.6 Simulation time. In recently published articles, Guochu et al. solved the problem similar to SW_Qsim, and completed the simulation of single-amplitude RQCs. The computational efficiency of a quantum simulator is defined as the computing time it takes to complete the same work using equivalent computing power. The comparisons of computational efficiency between SW_Qsim and Guochu's simulator are listed in TABLE 5. The peak performance of a single node of Tianhe-2 is 3.4 TFLOPS, while the peak performance of SW26016pro is 14 TFLOPS, which means 995 SW26010pro nodes can provide equivalent computing power as 4096 nodes on Tianhe-2. It can be seen from Table.4 that SW_Qsim uses less time and is more efficient than Guochu's with the same peak performance.

5 CONCLUSIONS

In this work, we propose SW_Qsim, a minimize-memory quantum simulator with high-performance on a New Sunway Supercomputer. By adopting a novel parallel contraction path algorithm that can reduce the maximum rank of the intermediate tensors generated during the tensor contraction process, the memory overhead is significantly reduced, increasing the number of qubits and the depth that can be simulated. Furthermore, in order to efficiently use the New Sunway distributed system, we have applied a set of parallel methods to increase parallel efficiency and stability. Meanwhile, some architecture-specific optimizations are applied to increase the locality and instruction-level parallelism. SW_Qsim can perform simulations of random circuits of up to 100 and 400 qubits with high performance, and can achieve linear strong and weak scaling with up to 28.75 million cores. We validated the simulator via the Porter–Thomas distribution simulation and variational quantum algorithms. In future work, we will add support to emulate the effects of noise, and expand the applications of the present simulator to other fields such as quantum machine learning.

6 ACKNOWLEDGEMENT

We would like to thank Chu Guo, Heliang Huang, Lixin He, Guoping Guo, Xiaobo Zhu for discussions and help.

The work is supported by National Key Research & Development Program of China (2020YFB0204800). The corresponding author is Xin Liu(yyylx@263.net).

REFERENCES

[1] Frank Arute, Kunal Arya, and et al Babbush. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019.
[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, and etc. Hartree-Fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, August 2020. arXiv: 2004.04174.
[3] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. Characterizing Quantum Supremacy in Near-Term Devices. *Nature Physics*, 14(6):595–600, June 2018. arXiv: 1608.00263.
[4] Jianxin Chen, Fang Zhang, Cupjin Huang, Michael Newman, and Yaoyun Shi. Classical Simulation of Intermediate-Size Quantum Circuits. *arXiv:1805.01450 [quant-ph]*, May 2018. arXiv: 1805.01450.
[5] A.B. Finnila, M.A. Gomez, C. Sebenik, C. Stenson, and J.D. Doll. Quantum annealing: A new method for minimizing multidimensional functions. *Chemical Physics Letters*, 219(5-6):343–348, March 1994.
[6] E. Schuyler Fried, Nicolas P. D. Sawaya, Yudong Cao, Ian D. Kivlichan, Jhonathan Romero, and Alán Aspuru-Guzik. qTorch: The Quantum Tensor Contraction Handler. *PLOS ONE*, 13(12):e0208510, December 2018. arXiv: 1709.03636.
[7] Vlad Gheorghiu. Quantum++: A modern c++ quantum computing library, 2018.
[8] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, March 2021. arXiv: 2002.01935.
[9] Chu Guo, Yong Liu, Min Xiong, Shichuan Xue, Xiang Fu, Anqi Huang, Xiaogang Qiang, Ping Xu, Junhua Liu, Shenggen Zheng, He-Liang Huang, Mingtang Deng, Dario Poletti, Wan-Su Bao, and Junjie Wu. General-purpose quantum circuit simulator with Projected Entangled-Pair States and the quantum supremacy frontier. *Physical Review Letters*, 123(19):190501, November 2019. arXiv: 1905.08394.
[10] Chu Guo, Youwei Zhao, and He-Liang Huang. Verifying Random Quantum Circuits with Arbitrary Geometry Using Tensor Network States Algorithm. *Physical Review Letters*, 126(7):070502, February 2021. arXiv: 2011.02621.
[11] Lixin He, Hong An, Chao Yang, Fei Wang, Junshi Chen, Chao Wang, Weihao Liang, Shaojun Dong, Qiao Sun, Wenting Han, Wenyuan Liu, Yongjian Han, and Wenjun Yao. PEPS++: Towards Extreme-scale Simulations of Strongly Correlated Quantum Many-particle Models on Sunway TaihuLight. *arXiv:1806.03761 [cond-mat, physics:quant-ph]*, June 2018. arXiv: 1806.03761.
[12] Jianyu Huang, D. Matthews, and R. Geijn. Strassen's algorithm for tensor contraction. *SIAM J. Sci. Comput.*, 40, 2018.
[13] Thomas Häner and Damian S. Steiger. 0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, November 2017. arXiv: 1704.01127.
[14] Lijuan Jiang, Chao Yang, Yulong Ao, Wanwang Yin, Wenjing Ma, Qiao Sun, Fangfang Liu, Rongfen Lin, and Peng Zhang. Towards highly efficient dgemm on the emerging sw26010 many-core processor. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 422–431, 2017.
[15] Lijuan Jiang, Chao Yang, and Wenjing Ma. Enabling highly efficient batched matrix multiplications on sw26010 many-core processor. *ACM Trans. Archit. Code Optim.*, 17(1), March 2020.
[16] Tyson Jones, Anna Brown, Ian Bush, and Simon C. Benjamin. Quest and high performance simulation of quantum computers. *Scientific Reports*, 9(1), Jul 2019.
[17] Ang Li and Sriram Krishnamoorthy. QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation. *arXiv:2005.13018 [quant-ph]*, May 2020. arXiv: 2005.13018.
[18] Riling Li, Bujiao Wu, Mingsheng Ying, Xiaoming Sun, and Guangwen Yang. Quantum Supremacy Circuit Simulation on Sunway TaihuLight. *arXiv:1804.04797*

[quant-ph], August 2018. arXiv: 1804.04797.

[19] Igor L. Markov, Aneeqa Fatima, Sergei V. Isakov, and Sergio Boixo. Quantum Supremacy Is Both Closer and Farther than It Appears. *arXiv:1807.10749 [quant-ph]*, September 2018. arXiv: 1807.10749.

[20] Igor L. Markov and Yaoyun Shi. Simulating Quantum Computation by Contracting Tensor Networks. *SIAM Journal on Computing*, 38(3):963–981, January 2008.

[21] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.

[22] Feng Pan, Pengfei Zhou, Sujie Li, and Pan Zhang. Contracting Arbitrary Tensor Networks: General Approximate Algorithm and Applications in Graphical Models and Quantum Circuit Simulations. *Physical Review Letters*, 125(6):060503, August 2020. arXiv: 1912.03014.

[23] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, Thomas Magerlein, Edgar Solomonik, Erik W. Draeger, Eric T. Holland, and Robert Wisnieff. Pareto-Efficient Quantum Circuit Simulation Using Tensor Contraction Deferral. *arXiv:1710.05867 [quant-ph]*, August 2020. arXiv: 1710.05867.

[24] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1):4213, September 2014.

[25] Mikhail Smelyanskiy, Nicolas P. D. Sawaya, and Alán Aspuru-Guzik. qhipster: The quantum high performance software testing environment, 2016.

[26] P. Springer and P. Bientinesi. Design of a high-performance gemm-like tensor–tensor multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 44:1 – 29, 2018.

[27] Damian S. Steiger, Thomas Häner, and Matthias Troyer. Projectq: an open source software framework for quantum computing. *Quantum*, 2:49, Jan 2018.

[28] F. Verstraete and J. I. Cirac. Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions. *arXiv:cond-mat/0407066*, July 2004. arXiv: cond-mat/0407066.

[29] Guifré Vidal. Efficient Classical Simulation of Slightly Entangled Quantum Computations. *Physical Review Letters*, 91(14):147902, October 2003.

[30] Benjamin Villalonga, Sergio Boixo, Bron Nelson, Christopher Henze, Eleanor Rieffel, Rupak Biswas, and Salvatore Mandrà. A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware. *npj Quantum Information*, 5(1):86, December 2019.

[31] Benjamin Villalonga, Dmitry Lyakh, Sergio Boixo, Hartmut Neven, Travis S. Humble, Rupak Biswas, Eleanor G. Rieffel, Alan Ho, and Salvatore Mandrà. Establishing the Quantum Supremacy Frontier with a 281 Pflop/s Simulation. *Quantum Science and Technology*, 5(3):034003, April 2020. arXiv: 1905.00444.

[32] Steven R. White. Density matrix formulation for quantum renormalization groups. *Physical Review Letters*, 69(19):2863–2866, November 1992.

[33] Robert Wille, Rod Meter, and Y. Naveh. Ibm's qiskit tool chain: Working with and developing for real quantum computers. pages 1234–1240, 03 2019.

[34] Zheng Wu, Mingfan Li, Mengxian Chi, Le Xu, and Hong An. Runtime adaptive matrix multiplication for the sw26010 many-core processor. *IEEE Access*, 8:156915–156928, 2020.

[35] Fang Zhang, Cupjin Huang, Michael Newman, Junjie Cai, Huanjun Yu, Zhengxiong Tian, Bo Yuan, Haihong Xu, Junyin Wu, Xun Gao, Jianxin Chen, Mario Szegedy, and Yaoyun Shi. Alibaba Cloud Quantum Development Platform: Large-Scale Classical Simulation of Quantum Circuits. *arXiv:1907.11217 [quant-ph]*, September 2019. arXiv: 1907.11217.

[36] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *arXiv:2012.01625 [cond-mat, physics:physics, physics:quant-ph]*, December 2020. arXiv: 2012.01625.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

1. Abstract

We ran the sw_qsim quantum simulator on a new Sunway Supercomputer. In order to generate a RQC of arbitrary sizes, we need to run gen-circuit.cpp; The tensor network of a final state is calculated in evaluation.cpp; Minimize-memory contraction path of tensor network is generated in gen-path.cpp. In order to verify the calculating results, we applied pt.cpp to draw the Porter-Tomas distribution.

2. Installation

You can go to the FHIaims GitLab server to download the code once you have an account : https://github.com/Daisyforest/sw_qsim

Then you can install the code in the following way: tar zxvf sw_qsim.tgz, cd sw_qsim/src, make.

After compiling the sw_qsim, there will be a binary file in the sw_qsim/bin directory, which is called qsim.x

3. Run the experiments

In order to perform the RQC calculations, the RQC circuit need to be created by using open gen-circuit.cpp, change the size and depth of the circuit, then we will have the RQC circuits in the sw_qsim/bin directory. After finishing the compilation, we can go to the directory which contains the circuits and minimized-path, and perform the calculation. We can submit the calculation to the queue using : bsub -b -m 1 -q q

$_sw$

$_share - n - share$

$_size4096 - host$

$_stack1024 - ooutput\ /qsim.xpath.txtcircuit.txt$

Author-Created or Modified Artifacts:

```
Persistent ID:
↪  https://github.com/Daisyforest/sw\_qsim
Artifact name: software
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: New Sunway: the new Sunway supercomputer adopts a new-generation domestic high-performance heterogeneous many-core processors and interconnection network chips in China. The new Sunway many-cores processor (SW26010pro) is designed for massive thread and data parallelism and delivers high performance on parallel workloads. Each processor contains 6 core-groups (CGs), with 65 cores in each CG, and in total 390 cores. Each CG has one management processing element (MPE), one cluster of computing processing elements (CPEs), and one memory controller (MC). The MPE within each CG is used for computations, management, and communication. The CPEs are organized as an 8*8 mesh (64 cores) and are designed to maximize the aggregated computing power and minimize the micro-architecture complexity.

Operating systems and versions: Sunway customized Linux with kernel version 3.10.0

Compilers and versions: swg++ (-std=c++14), swgcc, swmpi

Applications and versions: sw_qsim v1.0

Libraries and versions: lapack-v3.8.0