

Study on γ-ray source from the resonant reaction $^{19}F(p,\alpha\gamma)^{16}O$ at E_p=340 keV

Fu-Long Liu(刘伏龙), Wan-Sha Yang(杨婉莎), Ji-Hong Wei(魏继红), Di Wu(吴笛), Yang-Fan He(何阳帆), Yu-Chen Li(李雨尘), Tian-Li Ma(马田丽), Yang-Ping Shen(谌阳平), Qi-Wen Fan(樊启文), Chuang-Ye He(贺创业), Bing Guo(郭冰), Nai-Yan Wang(王乃彦)

Citation: Chin. Phys. B . 2020, 29(7): 070202 . **doi:** 10.1088/1674-1056/ab96a0
Journal homepage: http://cpb.iphy.ac.cn; http://iopscience.iop.org/cpb

What follows is a list of articles you may be interested in

New analytical exact solutions of time fractional KdV–KZK equation by Kudryashov methods

S Saha Ray

Chin. Phys. B . 2016, 25(4): 040204 . **doi:** 10.1088/1674-1056/25/4/040204

Simulation of the 3D viscoelastic free surface flow by a parallel corrected particle scheme

Jin-Lian Ren(任金莲) and Tao Jiang(蒋涛)

Chin. Phys. B . 2016, 25(2): 020204 . **doi:** 10.1088/1674-1056/25/2/020204

Accurate treatments of electrostatics for computer simulations of biological systems: A brief survey of developments and existing problems

Yi Sha-Sha, Pan Cong, Hu Zhong-Han

Chin. Phys. B . 2015, 24(12): 120201 . **doi:** 10.1088/1674-1056/24/12/120201

Ultra-elongated depth of focus of plasmonic lenses with concentric elliptical slits under Gaussian beam illumination

Wang Jing, Fu Yong-Qi

Chin. Phys. B . 2013, 22(9): 090206 . **doi:** 10.1088/1674-1056/22/9/090206

The complex variable meshless local Petrov–Galerkin method of solving two-dimensional potential problems

Yang Xiu-Li, Dai Bao-Dong, Zhang Wei-Wei

Chin. Phys. B . 2012, 21(10): 100208 . **doi:** 10.1088/1674-1056/21/10/100208

Improved hybrid parallel strategy for density matrix renormalization group method*

Fu-Zhou Chen(陈富州)[1], Chen Cheng(程晨)[1,2], and Hong-Gang Luo(罗洪刚)[1,2,†]

[1] School of Physical Science and Technology, Lanzhou University, Lanzhou 730000, China
[2] Beijing Computational Science Research Center, Beijing 100084, China

(Received 21 January 2020; revised manuscript received 7 April 2020; accepted manuscript online 17 April 2020)

We propose a new heterogeneous parallel strategy for the density matrix renormalization group (DMRG) method in the hybrid architecture with both central processing unit (CPU) and graphics processing unit (GPU). Focusing on the two most time-consuming sections in the finite DMRG sweeps, i.e., the diagonalization of superblock and the truncation of subblock, we optimize our previous hybrid algorithm to achieve better performance. For the former, we adopt OpenMP application programming interface on CPU and use our own subroutines with higher bandwidth on GPU. For the later, we use GPU to accelerate matrix and vector operations involving the reduced density matrix. Applying the parallel scheme to the Hubbard model with next-nearest hopping on the 4-leg ladder, we compute the ground state of the system and obtain the charge stripe pattern which is usually observed in high temperature superconductors. Based on simulations with different numbers of DMRG kept states, we show significant performance improvement and computational time reduction with the optimized parallel algorithm. Our hybrid parallel strategy with superiority in solving the ground state of quasi-two dimensional lattices is also expected to be useful for other DMRG applications with large numbers of kept states, e.g., the time dependent DMRG algorithms.

Keywords: density matrix renormalization group, strongly correlated model, hybrid parallelization

PACS: 02.70.−c, 71.10.Fd, 71.27.+a, 05.10.Cc **DOI:** 10.1088/1674-1056/ab8a42

1. Introduction

Density matrix renormalization group (DMRG) is one of the most important numerical methods for low dimensional strongly correlated lattice models.[1–3] The ground state of a one-dimensional (1D) system can be obtained variationally by DMRG with extremely high efficiency and precision. After its invention for the real space lattice model, the application of DMRG has been extended to the momentum space[4,5] and quantum chemistry.[6–8] Besides the ground state algorithm, a serial of techniques, including the time-dependent and finite-temperature DMRG, are generalized and developed.[9–15] In addition, many novel quantum phases in two-dimensional (2D) interaction systems have been investigated by DMRG, e.g., stripe phase in high temperature superconductor[16–22] and quantum spin liquid.[23–25] However, in these extended applications, especially for the two and quasi-two dimensional problems, the computational cost of DMRG increases dramatically for the convergent results. These lead to great challenges to the computing and storage capabilities of the contemporary computers.

Nowadays, variety of optimized methods have been proposed to reduce the computational cost of DMRG, for example, targeting the states in smaller Hilbert subspace in the presence of multiple symmetries,[5,26,27] the dynamical block state selection approach,[28,29] one-site sweep strategy,[30,31] and good initial wave function for the eigensolver.[32] On the other hand, to take full advantages of the advanced high-performance computing technique and further shorten the time cost of DMRG, there have been different parallelization strategies on high-performance computers, e.g., real space parallel DMRG,[33] shared/distributed-memory multi-core parallelization,[34,35] and parallelization on heterogeneous architectures.[36,37] Especially, the heterogeneous architectures employing both central processing unit (CPU) and graphics processing unit (GPU) have potential to significantly improve the efficiency, since the floating-point performance and memory bandwidth of GPU are usually many times higher than the host part.

Along this direction, a pioneer work has implemented DMRG on the hybrid CPU–GPU architecture, and achieved considerable speedup.[36] However, this implementation is based on the two-subblock configuration, the memory cost of which is d^2 (d is the local Hilbert space dimension of a single site) times larger than that of the four-subblock expression. Recently, we propose a new hybrid CPU–GPU parallel strategy with the Hamiltonian expressed by the operators of four subblocks, which greatly reduces the GPU memory cost.[37] In our implementation, one is able to use much larger number of

*Project supported by the National Natural Science Foundation of China (Grant Nos. 11674139, 11834005, and 11904145) and the Program for Changjiang Scholars and Innovative Research Team in University, China (Grant No. IRT-16R35).

†Corresponding author. E-mail: luohg@lzu.edu.cn

© 2020 Chinese Physical Society and IOP Publishing Ltd

DMRG kept states (upto 10^4) to investigate more complex systems, such as spinfull fermions on the quasi-two-dimensional lattice.

In our previous work, the hybrid strategy focuses on the diagonalization of the Hamiltonian, which is the most time-consuming part for the DMRG procedure with large number of kept states. According to a dynamical load balancing strategy, the matrix and vector operations in diagonalizing the Hamiltonian are distributed to CPU and GPU, and are performed using parallel MKL[38] and CUBLAS[39] subroutines, respectively. While these libraries generally achieve high performance in most cases, they are not always the best choice. For example, we get low performance when using these parallel subroutines for small matrix and vector operations. Therefore, in this work we improve our parallelization strategy by optimizing the diagonalization part. On the host, when acting the Hamiltonian on the wavefunction, we adopt OpenMP application programming interface[40] and sequential MKL subroutines, instead of parallel MKL subroutines. On the GPU, for vector operations in acting the Hamiltonian on the wavefunction, we use our own subroutines which gets higher bandwidth for linear combination of vectors. In addition, the previous DMRG truncation procedure with lots of matrix operations is only performed by the CPU. In the current work, we perform most of these operations on the GPU to further reduce the total computational time.

This paper is organized as follows. In Section 2, we review the finite-system DMRG and our previous parallel strategy, and get the time cost for each part in calculating the ground state of the 4-leg Hubbard model with next-nearest hopping. In Section 3, we introduce specifically the optimized hybrid parallel strategy for two computing sections, the diagonalization of the Hamiltonian and the truncation procedure. In Section 4, we show the numerical results and the performance benchmark comparing with the previous work. The conclusion is given in Section 5.

2. The DMRG method and benchmark model

A DMRG routine for finite lattice usually contains 1) infinite-system DMRG with a small number of states to produce a good initial wavefunction, and 2) a few finite-system DMRG sweeps to get the convergent result. For the quasi-two-dimensional lattice models, finite DMRG sweeps take most of the computational time. In each sweep, the wavefunction is updated through the lattice system from left to right, and then back to the left side. Without loss of generality, we briefly introduce one DMRG step with sweeping through the lattice system from left to right as an example. In our implementation, the lattice system is partitioned into four subblocks (see Fig. 1): block S, block E, site m_1, and site m_2.

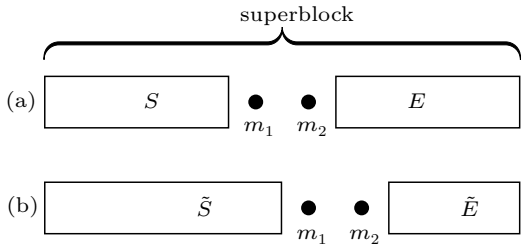


Fig. 1. Schematic one-step procedure of finite-system DMRG sweeps in four-subblock formulation.

A wave function $|\psi\rangle$ can be expressed in four-subblock basis as

$$|\psi\rangle = \sum_{\alpha,\sigma_{m_1},\sigma_{m_2},\beta} \psi_{\alpha,\sigma_{m_1},\sigma_{m_2},\beta} |\alpha\rangle|\sigma_{m_1}\rangle|\sigma_{m_2}\rangle|\beta\rangle, \quad (1)$$

where $|\alpha\rangle$, $|\beta\rangle$, $|\sigma_{m_1}\rangle$, and $|\sigma_{m_2}\rangle$ are the states on subblocks S, E, m_1, and m_2, respectively. A generic Hamiltonian H is expressed as

$$H = \sum_{i,j,k,l} W_{ijkl} O_i^S O_k^{m_1} O_l^{m_2} O_j^E, \quad (2)$$

where $O_i^S, O_j^E, O_k^{m_1}$, and $O_l^{m_2}$ are operators on subblocks S, E, m_1, and m_2, respectively. In each step of DMRG sweep, we first get the minimal eigenvalue E_0 and its eigenvector $|\psi_0\rangle$ by diagonalizing the Hamiltonian. Then we build the new subblock $\tilde{S}$ for the next step by adding a site, and truncate the basis according to the eigenvalues of the reduced density matrix of $\tilde{S}$. The next step repeats the similar procedure with four subblocks $\tilde{S}$, $\tilde{E}$, $\tilde{m_1}$, and $\tilde{m_2}$, as shown in Fig. 1.

To demonstrate the performance of our improved implementation, we apply it to get the ground state of the 4-leg Fermi Hubbard ladder with the next-nearest hopping, which is defined as

$$\hat{H} = -t \sum_{\langle i,j\rangle,\sigma} \hat{c}_{i\sigma}^\dagger \hat{c}_{j\sigma} - t' \sum_{\langle\langle i,j\rangle\rangle,\sigma} \hat{c}_{i\sigma}^\dagger \hat{c}_{j\sigma} + U \sum_i \hat{n}_{i\uparrow}\hat{n}_{i\downarrow}, \quad (3)$$

where $\hat{c}_{i\sigma}^\dagger$ ($\hat{c}_{i\sigma}$) is the electron creation (annihilation) operator at site i with spin $\sigma \in \{\uparrow,\downarrow\}$, and $\hat{n}_\sigma$ is the electron number operator with spin σ. The model has two conserved quantities: the total number of electrons $N_e = \sum_{i\sigma} \hat{n}_{i\sigma}$ and the z-component of the total spin $S_{tot}^z = \frac{1}{2}\sum_i (\hat{n}_{i\uparrow} - \hat{n}_{i\downarrow})$. In our implementation, we employ these two good quantum numbers to reduces the computational and memory cost. In this paper, we set the nearest hopping $t = 1$ as the energy scale, and fix the next-nearest hopping $t' = 0.25$ and the Coulomb repulsion $U = 6$. For the performance benchmark, we focus on the 4-leg ladder with leg-length $L_x = 16$ and particle filling $7/8$. The cylinder geometry is adopted, with open and periodic boundaries along the leg and rung, respectively. For all calculations, we take 5 sweeps to obtain the convergent energy.

Employing the previous hybrid parallel strategy[37] on the next-nearest hopping Hubbard ladder, we get the computational time proportion of several computing sections, as shown in Fig. 2. In the previous parallel scheme, applying the Hamiltonian to the wavefunction takes more than 60% of the total time cost. On the CPU, totally relying on MKL subroutines may not take good advantage of the computational resources for the small scale matrix and vector operations. On the GPU, we get low memory bandwidth for linear combination of vectors with CUBLAS subroutines based on Computer-Unified Device Architecture (CUDA). The time cost ratio of the truncation procedure is about 20%, which contains diagonalizing the reduced density matrix and projecting the operator on the new basis. Applying the Hamiltonian to the wavefunction and the truncation procedure takes most of the computational time (about 80% to 85%), and these two parts still have space to be improved. In the next section, we introduce explicitly how we further optimize the hybrid parallelization of the DMRG program with a large number of kept states.

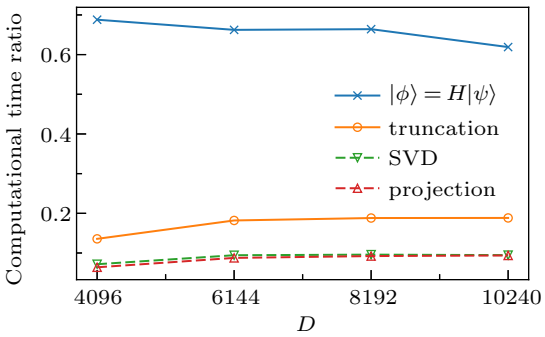


Fig. 2. The computational time ratio to the total time cost for the most time-consuming parts. Applying the Hamiltonian to the wavefunction ($|\phi\rangle = H|\psi\rangle$, denoted by $\times$) is the core operation for diagonalizing the Hamiltonian. The truncation procedure ($\circ$) contains diagonalizing the reduced density matrix ($\triangledown$) and projecting the operator on the new basis ($\triangle$). The former can be considered as the singular value decomposition (SVD) of the superblock wavefunction.

3. Hybrid parallel strategy

3.1. The diagonalization of the Hamiltonian

Among variety of subspace iterative projection methods for large-scale sparse matrix, we choose Davidson method[41] to diagonalize the Hamiltonian. In each iteration one needs to perform the action of the Hamiltonian on the wavefunction, which contributes most of the computational complexity. In practice, it is unnecessary to construct the matrix representation of the Hamiltonian. Instead, we apply the Hamiltonian directly to the wave function based on operators in subblocks, as shown in Algorithm 1. This operation is constituted by three serial steps with $\{\tilde{\psi}_j\}$, $\{\hat{\psi}_i\}$, and ϕ being calculated in each step. The vectors ψ, ϕ, $\tilde{\psi}_j$, and $\hat{\psi}_i$ can be partitioned to several subvectors labeled by the subspaces of S and E, and the states of m_1 and m_2.

Algorithm 1 Acting the Hamiltonian (see Eq. (2)) on the wavefunction (see Eq. (1)): $\phi = H\psi$.

Input: ψ is the vector representation of input wave function.
Output: ϕ is the vector representation of the output wave function.

1: **for** s **do**
2: **for** σ_{m_1}, σ_{m_2}, e and j **do**
3: $\tilde{\psi}_j^{[s,\sigma_{m_1},\sigma_{m_2},e']} = \hat{O}_j^E \psi^{[s,\sigma_{m_1},\sigma_{m_2},e]}$ ▷ step1
4: **end for**
5: **for** $\sigma'_{m_1}, \sigma'_{m_2}$, e' and i **do**
6: $\hat{\psi}_i^{[s,\sigma'_{m_1},\sigma'_{m_2},e']} = \sum_{j,k,l,\sigma_{m_1},\sigma_{m_2}} W_{ijkl} \times \hat{O}^{m_1}_{k,\{\sigma'_{m_1}\sigma_{m_1}\}} \hat{O}^{m_2}_{l,\{\sigma'_{m_2}\sigma_{m_2}\}} \tilde{\psi}_j^{[s,\sigma_{m_1},\sigma_{m_2},e']}$ ▷ step2
7: **end for**
8: **for** $\sigma'_{m_1}, \sigma'_{m_2}$ and e' **do**
9: $\phi^{[s',\sigma'_{m_1},\sigma'_{m_2},e']} = \sum_i \hat{O}_i^S \hat{\psi}_i^{[s,\sigma'_{m_1},\sigma'_{m_2},e']}$ ▷ step3
10: **end for**
11: **end for**

To achieve high total performance, we partition the action $|\phi\rangle = H|\psi\rangle$ into two parts according to the subspaces of subblock S (see line 1 in Algorithm 1), and perform them on CPU and GPU independently. We tend to distribute larger matrix and vector operations on the GPU in order to take advantages of its high performance and memory bandwidth. In our previous work, on the CPU, we use parallel MKL subroutines for each subvectors labeled by different quantum numbers $[s, \sigma_{m_1}, \sigma_{m_2}, e]$, and compute them one by one sequentially. These different subvectors may have different sizes in a very wide range, which results in low usage of the CPU power for small scale operations. Therefore, in the present work, we employ OpenMP application programming interface to further improve the performance on the CPU part. Steps 1–3 in Algorithm 1 have to be executed sequentially, but within each step, parallelization is available since these output subvectors with different quantum numbers can be calculated independently. Specifically, we distribute the calculation of each output subvector to one OpenMP thread, and balance the loads of the threads by using the dynamical scheduling strategy.

Lots of large vector operations are executed on the GPU for DMRG calculations with a large number of kept states. In our previous strategy, the operations about one output subvector are performed using CUBLAS subroutines in one CUDA stream, and different subvectors are calculated in multiple CUDA streams concurrently. This parallel scheme is good for steps 1 and 3 in Algorithm 1. However, in step 2, operations for each output subvector based on the CUBLAS subroutine need to access the input and output subvectors several times, which requires reading and writing the GPU memory

frequently. In the present work, we implement a new CUDA kernel for this step, in which all the subvectors for the fixed subspace $\{s, e'\}$ are calculated. Then multiple kernels about different subspaces $\{s, e'\}$ are executed in multiple streams. In this function, each element of the output subvector is computed by one thread of the GPU (see Algorithm 1). The intermediate result *val* of the linear combination of the input subvectors is stored in the registers of each thread, which has much higher bandwidth comparing to the global GPU memory. Each output subvector is written to the global memory until we get the final result of the linear combination. Therefore, the global memory accesses are greatly reduced in our new subroutine compared to the previous work.

Algorithm 2 The kernel for step 2 in Algorithm 1.

Input: $N_{\text{in_vec}}$: The number of input subvectors
Input: *in_vec*: The input subvectors $\tilde{\psi}_j^{[s,\sigma_{m_1},\sigma_{m_2},e']}$
Input: threadIdx.x: The index of the current CUDA thread
Input: blockIdx.x: The index of the current CUDA block
Input: blockDim.x: The number of threads in the current CUDA block
Input: n: The dimension of output subvectors
Input: $N_{\text{out_vec}}$: The number of output subvectors
Output: *out_vec*: The output subvectors $\hat{\psi}_i^{[s,\sigma'_{m_1},\sigma'_{m_2},e']}$

```
sI = blockIdx.x × blockDim.x
if threadIdx.x+sI < n then
    for 0 ≤ j < N_out_vec do
        val = 0
        out_vec[j][threadIdx.x+sI] = 0
        for 0 ≤ i < N_in_vec do
            val=val+in_vec[j][i][threadIdx+sI]×coeff[j][i]
        end for
        out_vec[j][threadIdx.x+sI] = val
    end for
end if
```

The performance improvement of our new parallel strategy in acting the Hamiltonian on the wavefunction is displayed in Fig. 3. Here the performance is quantified as the ratio of the total floating-point operations to the running time. Employing OpenMP on the CPU part instead of parallel MKL subroutines, we get a moderate performance improvement around 20% compared to the previous work. Then we replace CUBLAS subroutines with our own CUDA kernel, and further enhance the improved performance ratio upto more than 40% for the smallest $D = 4096$ in our benchmark test. When the number of DMRG kept states is large, the total relative performance improvement of Algorithm 1 is about 25%. Considering the time cost proportion of this part, the new parallel strategy would save appreciable fraction of the total computational time.

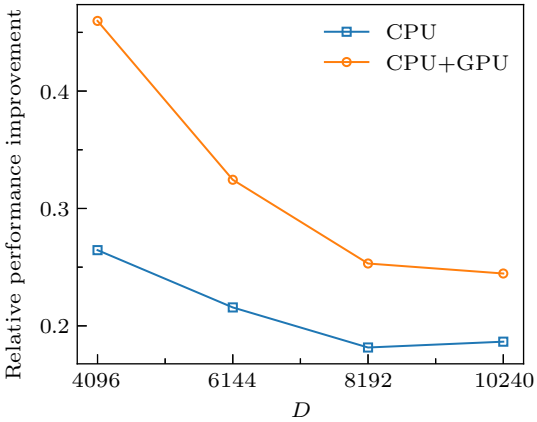


Fig. 3. The relative performance improvement of Algorithm 1 with different numbers of DMRG kept states, comparing to the previous work. Optimization on the CPU (□) increases about 20% performance for our largest D. The total improvement (○) is above 40% for $D = 4096$, and decreases as D increases.

3.2. The optimization of the truncation procedure

After the diagonalization of the Hamiltonian, the number of states of the new subblock $\tilde{S}$ is truncated from dD to D according to its reduced density matrix. Due to the presence of several symmetries of the model in Eq. (3), the reduced density matrix can be represented as a block diagonal matrix. Then independent operations of different subspaces can be calculated simultaneously. The submatrix labeled by quantum number $\tilde{s}$ is expressed as

$$\rho^{[\tilde{s}]} = \sum_{\tilde{s}=s+\sigma_{m_1},\sigma_{m_2}e} \psi^{[s\sigma_{m_1},\sigma_{m_2}e]} \psi^{\dagger[s\sigma_{m_1},\sigma_{m_2}e]}. \tag{4}$$

In this work, we compute the reduced density matrix on the GPU for its high performance of the large matrix operations. Each submatrix $\rho^{[\tilde{s}]}$ is obtained in one stream of the GPU, and multiple submatrices are calculated in multiple streams concurrently. Next, one submatrix $\rho^{[\tilde{s}]}$ is diagonalized on one CPU core and one GPU stream by using the divide-and-conquer subroutine in the library MAGMA.[42] These matrix diagonalization operations are executed in multiple OpenMP threads.

In the projection procedure, we obtain the new basis set by keeping states corresponding to the largest D eigenvalues of the whole reduced density matrix. Benefit from good quantum numbers, the projection for each subspace is independent. In subspace $\tilde{s}$, the new basis set is $|\tilde{\alpha}\rangle = \sum_{\alpha\in s,\tilde{s}=s+\sigma_{m_1}} U^{[\tilde{s}]}_{\alpha\sigma_{m_1},\tilde{\alpha}} |\alpha\rangle|\sigma_{m_1}\rangle$, where the columns of $U^{[\tilde{s}]}$ are the eigenvectors of $\rho^{[\tilde{s}]}$ with the largest eigenvalues. Then each operator $O^{\tilde{S}}$ in subblock $\tilde{S}$ is projected onto the new subspace by

$$\hat{\tilde{O}}^{\tilde{S}[\tilde{s},\tilde{s}']} = \tilde{U}^{\dagger[\tilde{s}]} \hat{O}^{\tilde{S}[\tilde{s},\tilde{s}']} \tilde{U}^{[\tilde{s}']}. \tag{5}$$

In applications with lots of DMRG kept states, matrices in the projection procedure can be very large. Therefore, to achieve better performance, we employ GPU for projections in the current hybrid parallel strategy instead of CPU in our previous

work. Considering the limitation of the GPU memory capacity, different operators are projected onto the statespace of $\tilde{S}$ one after the other. For each operator, one output submatrix is calculated in one CUDA stream, and multiple submatrices are calculated concurrently on the GPU.

As shown in Fig. 2, the relative time cost of the truncation part is about 20% for solving the ground state of a fermions ladder. In some other generalized DMRG methods, like time-evolving block decimation (TEBD),[11] this part takes most of the computational time. Optimization in this subsection is of great importance for our later work realizing TEBD method in the hybrid CPU–GPU architecture.

4. Result and benchmark

To test the performance of our optimized parallel strategy, we apply it to calculate the ground state of the model (3), with different DMRG kept states ($D \in \{4096, 6144, 8192, 10240\}$). In our hybrid parallel environment, there are 1 CPU (Intel Xeon E5-2650v3, 10 cores, 2.3 GHz) and 1 GPU (12 GB GPU memory) of the NVIDIA Tesla K80 graphic card. The maximum number of GPU kernels executed concurrently in our calculations is 16. To make sure our numerical code is correct, we compare our results with the DMRG method in Itensor[43] by calculating the energy per site of the ground state. As shown in Table 1, these two implementations give the accordance results.

Table 1. The ground state energy per site E_0/L. L is the total site number of the lattice. CPU–GPU denotes our results.

D	CPU–GPU	Itensor
4096	−0.9280790	−0.9280679
6144	−0.9284130	−0.9284069
8192	−0.9285577	−0.9285539
10240	−0.9286339	−0.9286315

We also compute the charge density $\langle \hat{n}_i \rangle = \sum_\sigma \langle \hat{n}_{i\sigma} \rangle$ of the ground state, and display the hole density $1 - \langle \hat{n}_i \rangle$ in Fig. 4. As addressed in Ref. [22], the ground state of the model with fixed parameters (see Section 2) displays a stripe phase with charge density waves, which is usually connected to high temperature superconductors. As shown in Fig. 4, the locations of the maximum hole densities, which are also believed to be the locations of the magnetic domain walls, divide the real space into unit-cells with wavelength $\lambda = 4$. These results are in agreement with studies in Ref. [22].

For different DMRG kept states D, we set the total computational time of our previous strategy as the unit, and demonstrate the performance improvement of each optimized section by showing the relative computational time reduction, as displayed in Fig. 5. In acting the Hamiltonian on the wavefunction, we can save more than 20% of the total time cost for $D = 4096$ by optimizing operations on both CPU and GPU. The performance of parallel MKL subroutines we used before on the CPU has better performance for larger scale operations, so that the improvement in this part looks more prominent for smaller kept states. In the truncation procedure, the matrix operations always achieve high performance on GPU than CPU for large scale matrices. Therefore, we further take advantage of the GPU in our new parallel scheme to save more computational time. In this part, the performance improvement increases as D increases. The total computational time of our improved parallel strategy is about 23%–29% less than that in the previous work.

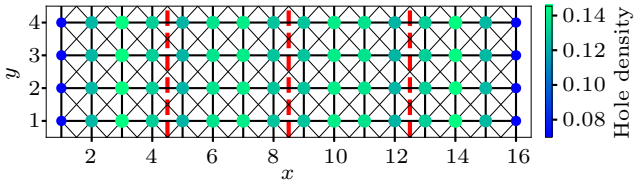


Fig. 4. The hole density on each site. Hole density is represented by both the area and the color depth of the disk. Black solid lines denote hopping, and red dashed lines represent the positions of maximum hole density. Here $D = 10240$.

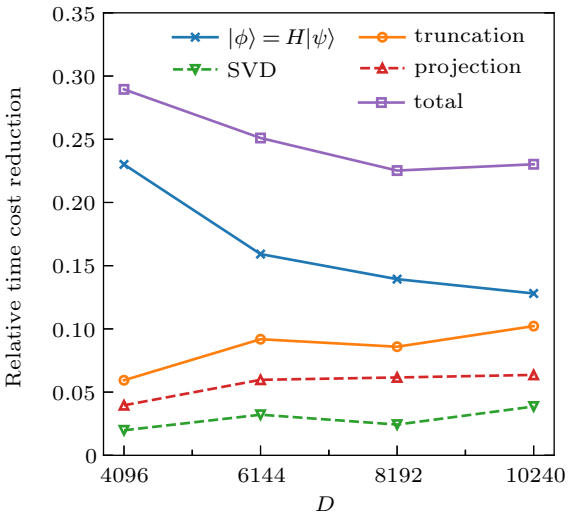


Fig. 5. The relative time cost reduction of different computing sections compared to the total computational time in our previous work. Truncation procedure (○) contains SVD (▽) and projection (△).

In our hybrid parallel strategy, we have considerably improved the performance of matrix and vector operations in DMRG calculation of the quasi-2D Hubbard model. Meanwhile, in the procedure of acting the Hamiltonian on the wave function (see Algorithm 1), which is the most time-consuming part, one may achieve better performance in the following aspects. First, for some models, there might be many small subspaces involved, which can lead to low performance in steps 1 and 3. In these cases, we can apply high performance subroutines specially for small matrix operations. Second, high host memory bandwidth may be achieved by extending the strategy optimizing the operations of step 2 in GPU to the host part

as well. Third, if the floating-point operations fall in a small number of subspaces, it is always difficult to balance the load between CPU and GPU. At this point, we can divide the states in large subspaces into many parts, and distribute the according operations into many smaller tasks.

5. Conclusion

In this paper, based on our previous work, we propose an optimized parallel scheme of DMRG method on the hybrid CPU–GPU architecture. Applying it to solve the ground state of a 4-leg Hubbard model with next-nearest hopping, we obtain the charge stripes usually being observed in high temperature superconductors, and demonstrate the performance improvement. Compared to the previous strategy, both the diagonalization and the truncation procedure are optimized, and these optimizations significantly reduce the total computational time in the DMRG calculation. Our results indicate the advantage of the new hybrid parallel strategy and the importance of high performance computing in strongly correlated algorithms. We expect our hybrid parallel strategy benefits DMRG applications with large number of kept states, for example, ground state DMRG for quasi-2D lattice models and time-dependent DMRG methods.

References

[1] White S R 1992 *Phys. Rev. Lett.* **69** 2863
[2] White S R 1993 *Phys. Rev. B* **48** 10345
[3] Schollwöck U 2005 *Rev. Mod. Phys.* **77** 259
[4] Xiang T 1996 *Phys. Rev. B* **53** R10445
[5] Ehlers G, White S R and Noack R M 2017 *Phys. Rev. B* **95** 125125
[6] White S R and Martin R L 1999 *J. Chem. Phys.* **110** 4127
[7] Luo H G, Qin M P and Xiang T 2010 *Phys. Rev. B* **81** 235129
[8] Yang J, Hu W, Usvyat D, Matthews D, Schütz M and Chan G K L 2014 *Science* **345** 640
[9] Cazalilla M A and Marston J B 2002 *Phys. Rev. Lett.* **88** 256403
[10] Luo H G, Xiang T and Wang X Q 2003 *Phys. Rev. Lett.* **91** 049701
[11] White S R and Feiguin A E 2004 *Phys. Rev. Lett.* **93** 076401
[12] Verstraete F, García-Ripoll J J and Cirac J I 2004 *Phys. Rev. Lett.* **93** 207204
[13] Feiguin A E and White S R 2005 *Phys. Rev. B* **72** 220401
[14] Stoudenmire E M and White S R 2010 *New J. Phys.* **12** 055026
[15] White S R 2009 *Phys. Rev. Lett.* **102** 190601
[16] Dagotto E 1994 *Rev. Mod. Phys.* **66** 763
[17] Keimer B, Kivelson S A, Norman M R, Uchida S and Zaanen J 2015 *Nature* **518** 179
[18] Fradkin E, Kivelson S A and Tranquada J M 2015 *Rev. Mod. Phys.* **87** 457
[19] Zheng B X, Chung C M, Corboz P, Ehlers G, Qin M P, Noack R M, Shi H, White S R, Zhang S and Chan G K L 2017 *Science* **358** 1155
[20] Huang E W, Mendl C B, Liu S, Johnston S, Jiang H C, Moritz B and Devereaux T P 2017 *Science* **358** 1161
[21] Cheng C, Mondaini R and Rigol M 2018 *Phys. Rev. B* **98** 121112
[22] Huang E W, Mendl C B, Jiang H C, Moritz B and Devereaux T P 2018 *npj Quantum Materials* **3** 22
[23] Yan S, Huse D A and White S R 2011 *Science* **332** 1173
[24] Savary L and Balents L 2016 *Reports on Progress in Physics* **80** 016502
[25] Wang L and Sandvik A W 2018 *Phys. Rev. Lett.* **121** 107202
[26] Alvarez G 2012 *Comput. Phys. Commun.* **183** 2226
[27] Tzeng Y C 2012 *Phys. Rev. B* **86** 024403
[28] Legeza O, Röder J and Hess B A 2003 *Phys. Rev. B* **67** 125114
[29] Legeza O and Sólyom J 2003 *Phys. Rev. B* **68** 195116
[30] Hubig C, McCulloch I P, Schollwöock U and Wolf F A 2015 *Phys. Rev. B* **91** 155115
[31] White S R 2005 *Phys. Rev. B* **72** 180403
[32] White S R 1996 *Phys. Rev. Lett.* **77** 3633
[33] Stoudenmire E M and White S R 2013 *Phys. Rev. B* **87** 155137
[34] Hager G, Jeckelmann E, Fehske H and Wellein G 2004 *J. Comput. Phys.* **194** 795
[35] Romero E and Roman J E 2014 *ACM Trans. Math. Software* **40** 13:1
[36] Nemes C, Barcza G, Nagy Z, Legeza O and Szolgay P 2014 *Comput. Phys. Commun.* **185** 1570
[37] Chen F Z, Cheng C and Luo H G 2019 *Acta Phys. Sin.* **68** 120202 (in Chinese)
[38] Intel 2019 *Intel Math Kernel Library Developer Reference*
[39] NVIDIA 2018 *CUBLAS Library v. 9.2*
[40] *OpenMP Application Programming Interface*
[41] Davidson E R 1975 *J. Comput. Phys.* **17** 87
[42] *Matrix Algebra on GPU and Multicore Architectures (MAGMA)*, http://icl.cs.utk.edu/magma/
[43] *ITensor Library* (version 3.1.0)