Development of Open-Source Parallelized Tensor Network Softwares: mptensor and TeNeS

Satoshi MORITA and Naoki KAWASHIMA
Institute for Solid State Physics, University of Tokyo, Kashiwa, Chiba 277-8581

Abstract

In this activity report, we overview two open-source softwares, mptensor [1] and TeNeS [2], for tensor network methods. The former is a library for massively parallel computing of tensor networks. It provides useful functions for tensor operations which are commonly appeared in tensor network methods. TeNeS, named for "Tensor Network Solver", is a solver for two-dimensional quantum lattice systems based on an infinite tensor network state. TeNeS is originally developed as pTNS by Tsuyosi Okubo [3] and released under the support of Project for Advancement of Software Usability in Materials Science (PASUMS) of the Institute for Solid State Physics, the University of Tokyo [4]. TeNeS can calculate the ground state wave function for user-defined Hamiltonian and evaluate physical quantities including the magnetization and the correlation functions.

1 Introduction

In strongly correlated quantum many-body physics, numerical simulation of low-energy states is quite important for understanding of interesting phenomena including high-temperature superconductivity, quantum spin liquid, topological nature, etc. To attack these problems, many computational approaches are proposed. The exact diagonalization method (ED) provides the exact ground state for small systems [5]. However, exponentially divergence of its computational cost and memory usage tightly limits the size of systems. The quantum Monte Carlo method (QMC) also can calculate the exact results within statistical errors, but the negative sign problem occurs in the frustrated systems [6]. The variational Monte Carlo method (VMC) is another Monte Carlo method based on the variational principle [7]. Variational parameters are optimized to minimize the energy and the resulted wave function is expected to approximate the ground state well. The expectation value of physical quantities is estimated by using the Monte Carlo sampling. Results of VMC are biased by choice of the variational wave function ansatz. The density matrix renormalization group method (DMRG) is a quite accurate method for one-dimensional systems [8, 9].

Recently, tensor network methods are proposed as a powerful tool [10–16]. DMRG can be viewed as a variational method based on a one-dimensional tensor network, i.e., the matrix product state (MPS) [17]. Its higher-dimensional generalization including the projected entangled pair state (PEPS) is quite successful [18–20]. In classical systems, the partition function can be expressed as tensor networks [21, 22]. The physical properties of the systems can be obtained by the contraction of tensor networks.

Information compression based on the singular value decomposition is a key idea of tensor network methods in order to avoid exponentially divergence of computational cost. Accuracy of tensor network simulations is determined by the size of tensors. However, computational cost and memory usage still increase very rapidly as a function of tensor size. Thus parallel computation of tensor network methods is unavoidable to approach inter-

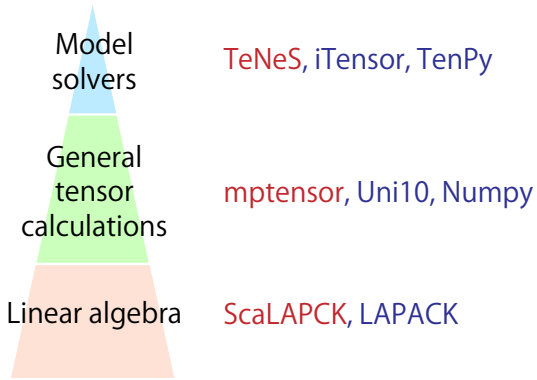

Figure 1: Hierarchical structure of tensor network calculations. Our softwares, TeNeS and mptensor, belongs to the top and middle levels. The softwares written in red support parallel computation on a distributed memory system.

esting phenomena.

Tensor network calculations are classified in hierarchical structure as shown in Fig. 1. The top level is lattice model solvers of tensor network methods. iTensor [23] and TenPy [24] are a famous software for MPS in this level. The former is written in C++ language but the latter is a Python package. The middle level provides a framework for tensor calculations including tensor contraction and tensor decomposition A C++ library, Uni10 [25] and a famous Python package NumPy [26] are in this level. Recently, a team of Google released a Python library "TensorNetwork" for physics and machine learning [27]. Since almost tensor calculations are written as matrix operations, this level is a kind of wrapper to a library for linear algebra, which is the bottom level. The well-known libraries, LAPACK [28] is located in this level.

As we mentioned before, parallel computation of tensor network methods is necessary to investigate interesting physical phenomena within sufficient accuracy. However, the softwares listed in the previous paragraph are not parallelized. To resolve this situation, we are developing two softwares, mptensor [1] and TeNeS [2] (Fig. 2). mptensor is an open-source C++ library for parallel computation of tensor networks on a distributed memory system. It provides many tensor operations commonly used in tensor network methods. mptensor uses ScaLAPACK [29] which is a paralell version of LAPACK, and supports MPI/OpenMP hybrid parallel computing. On the other hand, TeNeS is a lattice solve built on mptensor. It can solve quantum many-body problems on a two-dimensional infinite lattice based on a infinite tensor network wavefunction. TeNeS originates from pTNS developed by Tsuyoshi Okubo [3]. The official version of TeNeS (v1.0) is released on April 2020 under the support of the support of Project for Advancement of Software Usability in Materials Science (PASUMS) of the Institute for Solid State Physics, the University of Tokyo [4].

Figure 2: Logo images of mptensor and TeNeS. The latter represents a shape of a local tensor in the tensor network state.

In the next section, we briefly review tensor network methods and introduce graphical representations of tensor networks. In the third and fourth sections, we introduce our softwares and show its performance. The last section is devoted to summary.

2 Basics of tensor networks

In many-body problems, a huge tensor often appears naturally. One typical example is a coefficient in a wave function of a quantum N-body system

$$|\psi\rangle = \sum C_{i_1 i_2 \cdots i_N} |i_1\rangle |i_2\rangle \cdots |i_N\rangle, \qquad (1)$$

where $|i_x\rangle$ denotes a basis states on a site x and $C_{i_1 i_2 \cdots i_N}$ can be regarded as a N-rank tensor. The number of elements in such a tensor is $O(d^N)$ if each index takes a value from 1 to d. Here d corresponds the degrees of freedom on each site. Since computational cost grows exponentially with the

system size N, it is difficult to treat such a huge tensor directly.

A tensor network effectively represents a many-index tensor by decomposing it into small tensors only with a few indices, like as,

$$C_{i_1 i_2 \cdots i_N} = \sum_{\{k_j\}} T_{i_1 k_1 k_2} T'_{i_2 i_3 k_2 k_3} T''_{i_4 k_3 k_4 k_5} \cdots . \quad (2)$$

We introduce new indices $k_1, k_2, \cdots$, which connect between small tensors. Summation over these indices is nothing but tensor contraction. Since k_j does not represent any physical degrees of freedom, it is called a virtual index. On the other hand, the original indices in $C_{i_1 i_2 \cdots i_N}$ are named the physical indices. If the dimension of each virtual index is D and each tensor has z indices, the number of elements in each tensor is $O(D^z)$. When the number of tensors in a tensor network is polynomial in N, tensor network methods reduce exponentially large computational cost to polynomial order.

In a tensor network, many small tensors connect each other by virtual indices. Its mathematical expression becomes messy for large networks. A graphical representation of tensor networks is useful and intuitive. A tensor is represented by an object (circle, square, etc.) and its indices is shown by lines from a tensor. In other words, a tensor locates on a node of a graph and its indices correspond to edges. Several examples are shown in Fig. 3(a). Clearly, a scalar has no index, a vector has only one leg, and a matrix has two legs. By using terminology of the graph theory, an index of a tensor is also called as a bond, and the dimension of index is a bond dimension. A bond connecting between two tensors indicates tensor contraction of two tensors. For example, a graphical representation of matrix multiplication $C_{ij} = \sum_k A_{ik} B_{kj}$ is shown in Fig. 3(b).

Of course, decomposition of a huge tensor into a tensor network is approximation. Its accuracy depends on the virtual bond dimension D and a shape of network. As a variational state for the ground state of the quantum many-body problems, several kinds of structures are proposed. One of the simplest network is the matrix product state (MPS), whose wave function is written as a product of matrices,

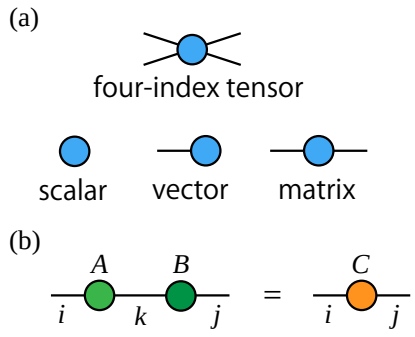

Figure 3: Graphical representation of tensor networks. (a) A tensor is represented by a node of graph and indices of a tensor corresponds to edges. (b) A graphical representation of matrix multiplication, $C_{ij} = \sum_k A_{ik} B_{kj}$.

$$C_{i_1 i_2 \cdots i_N} = \sum_{\{k_j\}} M^{(1)}_{i_1 k_1} M^{(2)}_{i_2 k_1 k_2} M^{(3)}_{i_3 k_2 k_3} \cdots M^{(N)}_{i_N k_{N-1}} . \quad (3)$$

Note that the three-index matrix $M^{(n)}_{i_n k_{n-1} k_n}$ can be regarded as a matrix when the physical index i_n fixed. Its graphical representation is shown in Fig. 4(a). This form can be easily derived by iteration of the singular value decomposition. The MPS is appeared in DMRG and achieves great success in simulations of the one-dimensional gapped systems [17]. The projected entanglement pair state (PEPS) [19] is two-dimensional generalization of MPS (Fig. 4(b)).

Important feature of PEPS is the fact that it satisfies the area law for the entanglement entropy [30]. The area law states that the entanglement entropy in the ground state of a region A is proportional to $|\partial A|$, the boundary area of the region. The entanglement entropy of a tensor network state is bounded by the number of bonds which connects a region A and its complement. If we consider an $L \times L$ region, the entanglement entropy of PEPS clearly scales as $O(L)$, which satisfies the area law in two-dimensional systems. In critical (gapless) one-dimensional systems, entanglement entropy logarithmically increases with the system size. Since entanglement entropy of MPS is bounded from above by $\log D$, MPS can not represent entanglement structure of such a system.

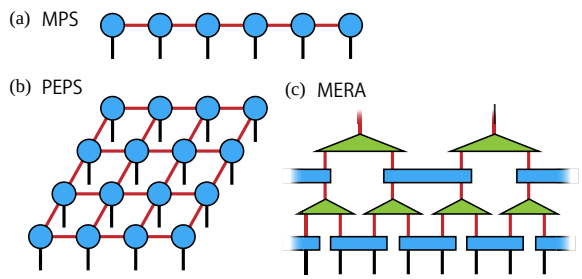

Figure 4: Some structures of tensor network states. The black (red) line indicates physical (virtual) bonds. (a) The matrix product state. (b) The projected entanglement pair state. (c) The multi-scale entanglement renormalization ansatz.

The multi-scale entanglement renormalization ansatz (MERA) shown in Fig. 4(c) resolves this problem by introducing additional dimension [31]. When we split the system into two parts, we need to cut $O(\log L)$ bonds in the MERA wavefunction, which indicates that upper bound of entanglement entropy is $O(\log L)$ instead of $O(1)$.

Important problems in a tensor network wavefunction for the ground state are how to calculate expectation values and how to optimize tensor elements based on the variational principle. These problems are touched in the fourth section.

Another important application is tensor network representation of the partition function in the statistical physics. The simplest example is the transfer matrix method for one-dimensional classical systems, in which the partition function is written as a product of transfer matrices. In the classical Ising chain with the Hamiltonian $H = -J\sum_i S_i S_{i+1}$, the transfer matrix is given as a 2×2 matrix,

$$M = \begin{pmatrix} e^{\beta J} & e^{-\beta J} \\ e^{-\beta J} & e^{\beta J} \end{pmatrix}, \qquad M_{ij} = e^{\beta J S_i S_j} \tag{4}$$

where β denotes the inverse temperature and S_i takes ± 1.

A generalization of the transfer matrix to higher dimensional systems is possible. Let us consider the square lattice tilted by 45 degrees and put a local tensor on half of plaquettes as shown in Fig.5. Obviously an index of a tensor corresponds to the direction of spins. Tensor elements for the Ising model are written as

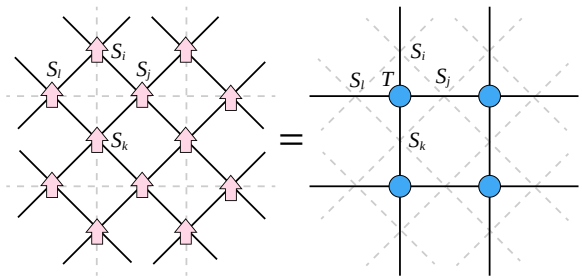

Figure 5: A tensor network representation of a classical spin model on the square lattice.

$$T_{ijkl} = e^{\beta J(S_i S_j + S_j S_k + S_k S_l + S_l S_i)}. \tag{5}$$

Another sophisticated way to obtain a tensor network representation is based on the singular value decomposition of the bond Boltzmann factor W. In the Ising model, W is 2×2 matrix with $W_{ss'} = e^{\beta J ss'}$. From the singular value decomposition, $W = U\Sigma V^\dagger$, we obtain a local tensor as

$$\tilde{T}_{ijkl} = \sum_s \sqrt{\sigma_i\sigma_j\sigma_k\sigma_l}\, U_{si} U_{sj} V^*_{sk} V^*_{sl}, \tag{6}$$

where σ_i denotes the singular value of W. In contrast to the previous example, a network of $\tilde{T}$ is the same as the original lattice of the model.

Contraction of the tensor network is a quite important task of tensor network methods. For quantum systems, it appears in the inner product of two tensor network states and the expectation value of physical quantities. In the classical systems, it directly relates to the partition function. However, exact contraction is usually impossible because of exponentially large computational cost. The tensor renormalization group method (TRG) [32] provides an efficient contraction scheme, which is based on the real-space renormalization group (Fig. 6). Let us consider a tensor network on the square lattice. A local tensor with four indices is approximated by the product of two three-index tensors, which are calculated by the truncated singular value decomposition. Then the coarse-grained tensor is obtained by contraction of four tensors. This step is regarded as a real-space renormalization with scaling factor $b = \sqrt{2}$. After

t times iteration of TRG steps, the trace of tensor, $Z = \sum_{i,j} T_{ijij}$, corresponds to the partition function with $N = 2^t$ under the periodic boundary condition. The animation of TRG method is available in Ref. [33]. Although the computational cost of the original TRG method is $O(D^6)$, we showed that it can be reduced to $O(D^5)$ by using a randomized algorithm for the singular value decomposition [34].

Many derivatives of TRG are proposed as a more efficient and accurate method [35–38]. Since the higher-order tensor renormalization group (HOTRG) [35] can be used in a higher-dimensional system, it attracts attention also from computational elementary particle physics. Recently we proposed a calculation method for higher-order moments of physical quantities based on HOTRG [39]. We showed that the finite-size scaling analysis provides critical exponents and distinguishes the weakly first-order and the continuous transitions. The TRG method can also be applicable to systems with open boundaries. The fixed-point boundary tensor at criticality has the information of the conformal tower described by the boundary conformal field theory [40, 41].

3 mptensor

mptensor is a C++ library for parallel computation of tensor networks and provides tensor operations used commonly in tensor network methods [1]. Its target users are researchers who want to parallelize their codes of tensor network methods. To make it easy to translate a serial code written with Python to a parallelized C++ code, interface of mptensor is as much like as NumPy [26], which is a famous

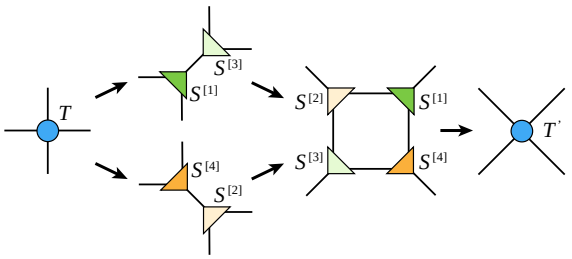

Figure 6: Diagrams of the TRG method.

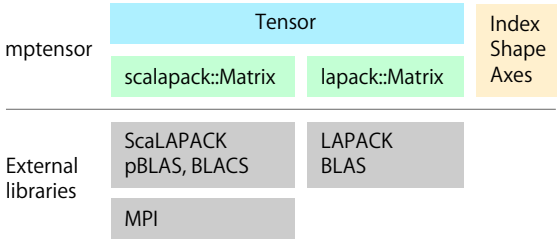

Figure 7: The class structure of mptensor.

Python module of a multi-dimensional array and is a de-facto standard in the field of machine learning.

For high-performance computing, mptensor supports hybrid parallel computing using MPI and openMP. With MPI, mptensor distributes tensor elements based on a block-cyclic matrix uses ScaLAPACK, which is a library for linear algebra on distributed-memory machines [29]. A tensor is matricized and distributed by the block-cyclic way. mptensor also supports LAPACK for a system without MPI.

3.1 Class structure

mptensor is composed of three classes as shown in Fig. 7. The `Tensor` class is a main object of mptensor, which represents a tensor and corresponds to `ndarray` in NumPy. In the `Tensor` class, a tensor is automatically matricized and it elements are stored in a `Matrix` object. mptensor has two kinds of `Matrix` classes, a wrapper class of a linear-algebra library. One uses ScaLAPACK and the other LAPACK, which are separated by C++ namespace.

The `Index` class is a short array of non-negative integers to represent an index of a tensor. The `Shape` and `Axes` classes are just an alias of the `Index` class. These classes have a simple constructor to mimic a list of Python. The index class can create the same list by `Index(1,2,3)` as well as Python can easily create a list of integer by `[1,2,3]`, . For C++11, an initializer list {`1,2,3`} is also available instead of the index class.

The `Tensor` class is implemented as a template class. Its template parameters specify a `Matrix`

class and a value type of elements (`double` or `complex`). For examples, a real-valued distributed tensor is `Tensor<scalapack::Matrix, double>` and a complex-valued non-distributed tensor is `Tensor<lapack::Matrix, complex>`. Since all functions for the tensor class accepts any patterns of template parameters, users can easily change a type of tensors. In addition, mptensor is designed to make it easy to support another library for linear algebra in the future. It is because the tensor class can accept any matrix class if it satisfies necessary interfaces.

3.2 Access to tensor elements

To access an element of a tensor, mptensor provides two ways. One is to access an element with an index of a tensor, which we call a global index. For example, we can get and set an element of a four-index tensor T by specifying (i, j, k, l). The other way is to specify a local index of a one-dimensional array where distributed elements are stored. In mptensor, a tensor is matricized like as $T_{ijkl} = M_{(ij),(kl)}$. The huge matrix M is decomposed into small matrices by using the two-dimensional block-cyclic distribution in ScaLAPACK. A small distributed matrix on each MPI process is flattened and stored as a one-dimensional array. The local index specifies a position in this one-dimensional array. In mptensor, matricization of a tensor is done in the tensor class, while translation between a index of the global matrix M and a local index is calculated in the matrix class.

Since tensor elements are distributed on MPI processors, access by the global index needs to check which process has an element T_{ijkl}. On the other hands, such a check is not necessary for the local index. Thus access by the local index has simpler interface like as usual array access, `T[i]`. We recommend a for-loop with the local index to access all the elements of a tensor. Member functions of the tensor class, `global_index` and `local_position` perform translation between the global and local indices. We note that `local_position` also calculates which process has the element with the given global index.

3.3 Tensor operations

In mptensor, many tensor operations are already implemented. Some typical operations are listed as follows.

- Tensor contraction (`tensordot`)
- Tensor decomposition
  - Singular value decomposition (`svd`)
  - QR decomposition (`qr`)
  - Diagonalization (`eigh`)
- Linear-equation solver (`solve`)
- Arithmetic operators ($T + T'$, $a \times T$, etc.)
- Element-wise vector multiplication
- Find minimum and maximum elements

These are sufficient to implement various tensor network methods including PEPS and TRG.

Tensor contraction is one of the main operation in tensor network methods. The `tensordot` function takes the almost same arguments of that in Numpy. For example, let us consider the following tensor contraction,

$$C_{abc} = \sum_{i,j} A_{aibj} B_{jci}. \tag{7}$$

In mptensor, it is written as

```
C = tensordot(A, B,
              Axes(1, 3),
              Axes(2, 0));
```

The third argument indicates that the second and fourth indices of the first tensor A will be contracted. (Note that array index starts from zero in C++.) The fourth argument is for the second tensor B. The third index i in B (specified by the first element in the fourth argument) will be contracted with the second index of A and the first index j in B will connects with the fourth index of A. The resulted tensor C has three indices abc which consists of the non-contracted indices of the first tensor, followed by the non-contracted indices of the second.

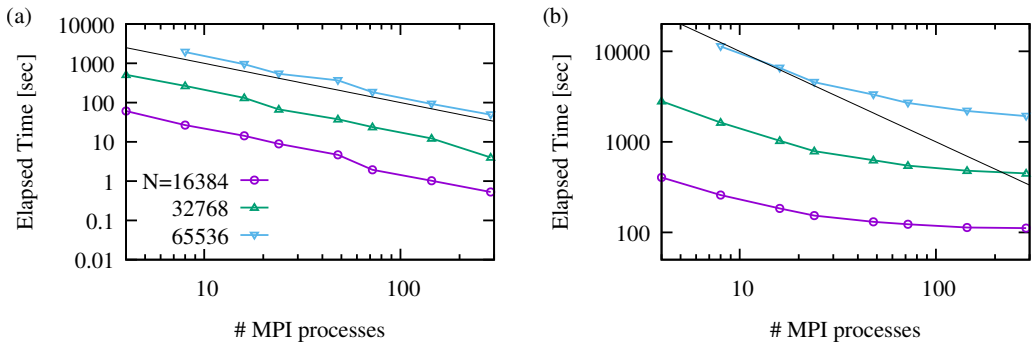

Figure 8: Elapsed time of ScaLAPACK routines, (a) a matrix-matrix multiplication (`pdgemm`) and (b) the singular value decomposition (`pdgesvd`). N is the linear size of a square matrix. The number of OpenMP threads per MPI process is 12. The solid lines indicate the perfect strong scaling.

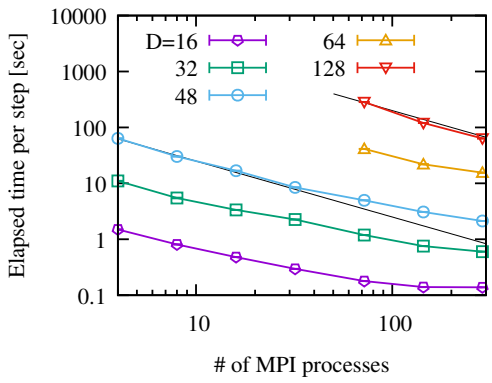

Figure 9: Elapsed time per step of the HOTRG method on the two-dimensional Ising model. The number of OpenMP threads per MPI process is 12. The solid lines indicate the perfect strong scaling.

Another important operation in tensor network methods is the singular value decomposition (SVD). For SVD of a tensor, we need to specify the way of matricization. For example, let us consider the following decomposition,

$$T_{ijkl} = \sum_a U_{ika} S_a (V^\dagger)_{alj}. \tag{8}$$

To perform this operation in NumPy, a tensor T is matricized as $T_{(ik),(lj)}$ before SVD. The resulted singular value vectors are returned as a matrix, $U_{(ik),a}$ and $(V^\dagger)_{a,(lj)}$. Thus we need to reshape from a matrix to a tensor. mptensor provides a useful function for SVD of a tensor and the above decomposition is written in single line as

```
svd(T, Axes(0, 2), Axes(3, 1),
    U, S, VT);
```

The second and third arguments specify the way of matricization. The first and third indices of T become the row index and the others do the column index. The order of indices specified in the second and third arguments also determines that of U and VT. The result of SVD is stored in the last three arguments. U and VT are an isometric tensor corresponding to the singular value vectors, while S is a one-dimensional array and has the singular values.

The performance of mptensor strongly depends on that of ScaLAPACK. The tensor contraction is performed by using the matrix-matrix multiplication routines (`pdgemm` and `pzgemm`). Thus it shows very high execution efficiency and parallel efficiency. On the other hand, parallel performance of SVD becomes worse in many processes as shown in Fig. 8.

We show the parallel efficiency of the HOTRG method implemented by using mptensor (Fig. 9). Clearly the strong scaling is satisfied for larger bond dimension D. It is because the heaviest part of HOTRG is contraction of a tensor network, which is scaled as $O(D^7)$. We note that deviation from the strong scaling in small bond dimensions is due to the singular value decomposition which has $O(D^6)$ scaling.

4 TeNeS

TeNeS is a tensor-network solver for quantum many-body problems on an infinite two-dimensional lattice [2]. TeNeS can calculate the ground-state wavefunction for user-defined Hamiltonian and evaluate user-defined physical quantities, including the magnetization and the correlation functions. TeNeS uses TOML [42] as the format of input files. This is a very simple configuration file format, easy to parse into data structures and also human-readable. The input file of the main program, `tenes`, defines simulation parameters, unit cell information for TNS, observables, and time-evolution operators. Since it is rather complicated for beginners, we provide two helper programs, `tenes_simple` and `tenes_std`, to generate an input file as similar to HΦ [5] and mVMC [7]. By using `tenes_simple`, users easily create an input file of the predefined models and lattices. TeNeS (v1.0) supports generation of the input file for the spin-S Hamiltonian

$$H = \sum_{\langle ij \rangle} \left[\sum_{\alpha=x,y,z} J_{ij}^{\alpha} S_i^{\alpha} S_j^{\alpha} + B \left(\vec{S}_i \cdot \vec{S}_j \right)^2 \right] - \sum_i \left[H S_i^z + \Gamma S_i^x - D (S_i^z)^2 \right] \tag{9}$$

on the square, triangular or honeycomb lattice. A more complicated model is also possible by editing an input file directly.

The variational wave function of TeNeS is represented by infinite tensor network called an infinite projected entangled paired state (iPEPS) or an infinite tensor network state (iTNS). Each tensor has five indices. One corresponds to the physical degrees of freedom. The others, called virtual bonds, connect with tensors on the nearest neighbor sites. The bond dimensions of the physical and virtual bonds are denoted by d and D, respectively. In iTNS, we need to assume a lattice translational symmetry with a certain period. Figure 10(a) shows a diagram of iTNS with 2×2 sublattice structure. We note that TeNeS always uses a tensor network on the square lattice. Other two-dimensional lattices including the honeycomb and triangular lattices is properly mapped to the square one.

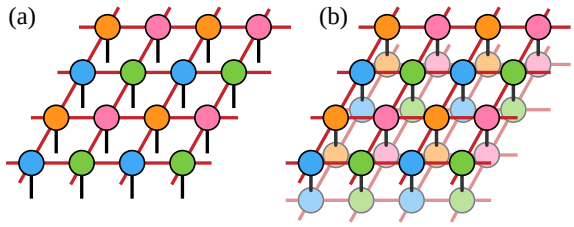

Figure 10: (a) Graphical representation of the iTNS wavefunction with 2×2 sublattice structure. (b) Diagram of the inner product $\langle \Psi | \Psi \rangle$, which is called the double layered tensor network.

To obtain the expectation value of a physical quantity in iTNS, we need to calculate tensor networks corresponding to $\langle \Psi | O | \Psi \rangle$ and $\langle \Psi | \Psi \rangle$. The diagram of $\langle \Psi | \Psi \rangle$ is called the double layered tensor network (Fig. 10(b)). To calculate contraction of this infinite network, TeNeS uses the corner transfer matrix method (CTMRG) [19,20]. We focus on a part of the network and the other part is approximated by the corner transfer matrices and the edge tensors [43, 44]. These environment tensors are optimized by absorbing the local tensor until convergence. The accuracy of the environment tensors is determined by its bond dimension χ. Since the bond dimension after contraction of the physical bond in the double layered tensor network is D^2, we usually take that χ is proportional to D^2. The computational cost of CTMRG is $O(D^{10})$ or $O(D^{12})$ depending on the way of the partial (truncated) SVD in CTMRG.

In order to approximate the ground state by iTNS, we need to optimize elements of tensors. TeNeS supports the imaginary-time evolution method based on

$$|\Psi_{\mathrm{iTNS}}\rangle \simeq e^{-\tau H} |\Psi_0\rangle, \tag{10}$$

where $|\Psi_0\rangle$ is the initial iTNS. If τ is sufficiently large, the right hand side converges to the ground state and then the left hand side is expected to be a good approximation of the ground state with iTNS ansatz. TeNeS assumes that the Hamiltonian can be written as a sum of short range two-body interactions and uses the Suzuki-Trotter decompo-

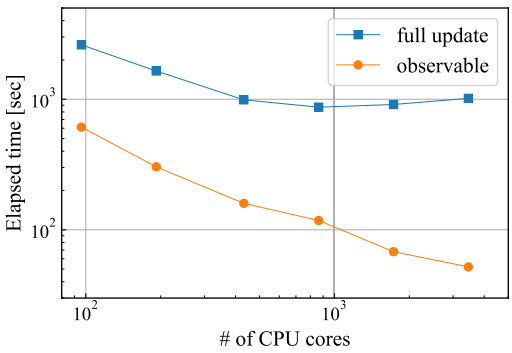

Figure 11: Elapsed time of TeNeS simulations with the virtual bond dimension $D = 10$. The blue square denotes elapsed time per step of the full update. The orange circle is tensor contraction for calculation of the expectation value.

sition. To avoid divergence of the dimension of virtual bonds, TeNeS supports two kinds of truncation approaches, so-called the full update [46] and the simple update [47]. The former solves the optimization problems of the whole network by using the CTM environment and has higher accuracy than the latter. Its computational cost is the same as CTMRG, that is, $O(D^{10})$ or $O(D^{12})$. On the other hand, the simple update considers only a part of network. Its computational cost is $O(D^5)$, which is much cheaper than the full update. It is known that the simple update has strong dependence on the initial value and usually overestimates the magnetization. Thus we need to take care of results obtained by the simple update.

The accuracy of iTNS calculation depends on the bond dimension D. However, the computational cost is rapidly increases. Thus parallel computation is necessary to obtain accurate results. TeNeS supports parallel computation since tensor operations are done by using mptensor. We show elapse time of TeNeS in Fig. 11, where we simulate the transverse field Ising model on the square lattice. We use a real-valued iTNS with the virtual bond dimension $D = 10$ and set $\chi = D^2$ for CTM. Since calculation of the expectation value is tensor contraction, its parallel efficiency is quite good. On the other hand, the full update includes the singular value decomposition. Its performance is similar to that of SVD in ScaLAPACK (Fig. 8). We note that elapse time of the simple update per step is 0.01% of the full update.

5 Summary

In this paper, we have reported recent activities of developing tensor network softwares, mptensor and TeNeS. The former is an MPI/OpenMP hybrid parallelized tensor operation library. It provides many tensor operations commonly used in the tensor network methods and is already used in the published papers [48, 49]. One of future issues for mptensor is implementation of quantum number conserving tensors. If the system has a certain symmetry, its tensor network representation has a corresponding non-trivial structure [52]. This technique not only improves accuracy but also reduces computational cost and memory usage because such a tensor is block-diagonal. Another future issue is automatic differentiation, which becomes a common tool in a field of machine learning. Recently, application of automatic differentiation to tensor network methods was proposed and succeeded to obtain accurate results [53]. Since it requires huge memory, its parallelization should become important for large-scale calculation.

TeNeS is an open-source program package for calculation of two-dimensional quantum states based on iTNS. An advantage of TeNeS over other numerical approaches is that it can treat infinite systems. In this article, we only show the parallel performance of TeNeS because of space limitations. Accuracy of TeNeS simulations will be appeared elsewhere [50]. We hope that TeNeS becomes a useful standard tool for a wide range of researchers who are interested in the strongly-correlated many-body problems.

Acknowledgements

TeNeS is developed by Tsuyoshi Okubo, Yuichi Motoyama, Kazuyoshi Yoshimi, Takeo Kato and the authors. The original version of TeNeS is developed as pTNS by Tsuyoshi Okubo [3]. Development and

release of TeNeS are supported by "Project for advancement of software usability in materials science" of the Institute for Solid State Physics, the University of Tokyo. We would like to thank Kenji Harada, Hiroshi Watanabe and Synge Todo for collaborations in developing mptensor. The numerical calculations of this work were performed with the supercomputer System B (sekirei) in the Institute for Solid State Physics, the University of Tokyo. This work was supported by MEXT as "Exploratory Challenge on Post-K computer" (Frontiers of Basic Science: Challenging the Limits) and Priority Issue on Post-K Computer (Creation of New Functional Devices and High-Performance Materials to Support Next-Generation Industries), and by JSPS KAKENHI Grant Number 19H01809.

References

[1] https://github.com/smorita/mptensor
[2] https://www.pasums.issp.u-tokyo.ac.jp/tenes
[3] https://github.com/TsuyoshiOkubo/pTNS
[4] https://www.pasums.issp.u-tokyo.ac.jp/
[5] M. Kawamura, et al., Comput. Phys. Commun. **217**, 180 (2017).
[6] https://www.pasums.issp.u-tokyo.ac.jp/dsqss
[7] T. Misawa, et al., Comput. Phys. Commun. **235**, 447 (2019).
[8] S. R. White, Phys. Rev. Lett. **69**, 2863 (1992).
[9] U. Schollwöck, Rev. Mod. Phys. **77**, 259 (2005).
[10] R. Orús, Ann. Phys. **349**, 117 (2014).
[11] R. Orús, Nature Rev, Phys. **1**, 538 (2019).
[12] J. C. Bridgeman and C. T. Chubb, J. Phys. A: Math. Theor. **50** 223001 (2017).
[13] J. Biamonte and V. Bergholm, arXiv:1708.00006 (2017)
[14] S.-J. Ran, et al., *"Tensor Network Contractions: Methods and Applications to Quantum Many-Body Systems"*, Lecture Notes in Physics, vol. 964, Springer (2020).
[15] G. Evenbly, https://www.tensors.net/
[16] http://tensornetwork.org/
[17] U. Schollwöck, Ann. Phys. **326**, 96 (2011).
[18] F. Verstraete, J. I. Cirac, arXiv:cond-mat/0407066 (2004).
[19] R. Orús and G. Vidal, Phys. Rev. B **80**, 094403 (2009).
[20] P. Corboz, T. M. Rice, and M. Troyer, Phys. Rev. Lett. **113**, 046402 (2014)
[21] R. J. Baxter, *Exactly Solved Models in Statistical Mechanics*, Academic Press, London (1982).
[22] H. H. Zhao, et al., Phys. Rev. B **81**, 174411 (2010).
[23] https://itensor.org
[24] https://github.com/tenpy/tenpy
[25] https://uni10.gitlab.io
[26] https://numpy.org
[27] https://github.com/google/tensornetwork
[28] http://www.netlib.org/lapack
[29] http://www.netlib.org/scalapack
[30] J. Eisert, M. Cramer, and M. B. Plenio, Rev. Mod. Phys. **82**, 277 (2010).
[31] G. Vidal, Phys. Rev. Lett. **99**, 220405 (2007).
[32] M. Levin and C. P. Nave, Phys. Rev. Lett. **99**, 120601 (2007).
[33] https://smorita.github.io/TN_animation
[34] S. Morita, R. Igarashi, H.-H. Zhao, and N. Kawashima, Phys. Rev. E **97**, 033310 (2018).
[35] Z. Y. Xie, et al., Phys. Rev. B **86**, 045139 (2012).
[36] G. Evenbly and G. Vidal, Phys. Rev. Lett. **115**, 180405 (2015).
[37] S. Yang, Z.-C. Gu, and X.-G. Wen, Phys. Rev. Lett. **118**, 110504 (2017).
[38] G. Evenbly, Phys. Rev. B **98**, 085155 (2018)
[39] S. Morita and N. Kawashima, Comput. Phys. Commun. **236**, 65 (2019).
[40] S. Iino, S. Morita, and N. Kawashima, Phys. Rev. B **100**, 035449 (2019).
[41] S. Iino, S. Morita, and N. Kawashima, Phys. Rev. B **101**, 155418 (2020).
[42] https://github.com/toml-lang/toml
[43] T. Noshino and K. Okunishi, J. Phys. Soc. Jpn. **65**, 891 (1996).
[44] T. Noshino and K. Okunishi, J. Phys. Soc. Jpn. **66**, 3040 (1997).
[45] N. Halko, P. G. Martinsson, and J. A. Tropp, SIAM Rev. **53**, 217 (2011).
[46] J. Jordan, et al., Phys. Rev. Lett. **101**, 250602 (2008).
[47] H. G. Jiang, et al., Phys. Rev. Lett. **101**, 090603 (2008).
[48] H.-Y. Lee and N. Kawashima, Phys. Rev. B **97**, 205123 (2018).
[49] H.-Y. Lee, R. Kaneko, T. Okubo, and N. Kawashima, Phys. Rev. Lett. **123**, 087203 (2019).
[50] Y. Motoyama, T. Okubo, K. Yoshimi, S. Morita, T. Kato, and N. Kawashima, in preparation.
[51] P. Corboz, Phys. Rev. B **94**, 035133 (2016)
[52] S. Singh, R. N. C. Pfeifer, and G. Vidal, Phys. Rev. A **82**, 050301, Phys. Rev. B **83**, 115125
[53] H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang, Phys. Rev. X **9**, 031041 (2019)