High Performance Single-Site Finite DMRG on GPUs

Hao Hong[1,*], **Hao Huang**[1*], **Tao Zhang**[1], **Xiao-Yang Liu**[2†]
[1]School of Computer Engineering and Science, Shanghai University, Shanghai, China
[2]Department of Electrical Engineering, Columbia University, USA
{honghao,baxlumen,taozhang}@shu.edu.cn, xl2427@columbia.edu

Abstract

The Density Matrix Renormalisation Group (DMRG) method is widely used in quantum many-body systems. However, the time and space cost of the DMRG algorithm increases rapidly with the amount of the sites and the dimension of the system. Multi-cores GPUs are suitable for matrix and tensor computation-intensive algorithms, and the DMRG algorithm is tensor computation-intensive. In this paper, we propose a high-performance GPU implementation for the single-site finite DMRG algorithm with three optimization strategies, including efficient memory access, faster multiplication, and faster data transferring. For different numbers of sites, the optimized GPU implementation achieves an average of 4.62× and up to 13.23× speedups over the Jax-GPU implementation in the TensorNetwork library.

Keywords GPU, single-site finite DMRG, memory access, faster multiplication

1 Introduction

The Density Matrix Renormalisation Group (DMRG) method [17] is an adaptive algorithm for optimizing a matrix product state (MPS) (or tensor train) tensor network to approximate the dominant eigenvector of a large matrix. In quantum many-body systems, DMRG, as a numerical variational technique, can obtain the low-energy ground state of a quantum physics system. It was first introduced in 1992 [15][17] and now has successfully calculated some properties of low-energy states on many different one-dimensional models, such as the Ising model [3], Heisenberg model [11] and other spin models, fermion systems such as Hubbard model [14], impurity systems such as Kondo effect [4], boson system, mixed boson and fermion system [8].

In one-dimensional short-range systems, the finite DMRG algorithm could get a good convergence quickly. However, with the extra site increasing, the computation time and memory requirements increase rapidly. Meanwhile, some studies [18][13][2] have utilized DMRG to the two-dimensional(2D) systems. However, 2D systems are still hard tasks for DMRG because of the exponential computational growth of the involved matrix dimensions. Therefore, developing an efficient DMRG algorithm is a critical task.

Various improvements have been proposed, such as a real-space parallel scheme [9], the use of abelian and non-abelian symmetries [7], single-site DMRG [16], and subspace expansion strictly single-cite SMRG [5]. However, those methods are only based on CPU, whose computing power for tensors is far less than GPU. Many-cores GPUs are proposed to be more suitable for DMRG. Nemes et al. [10] firstly proposed to use GPU to speed up the Davidson diagonalization part, and Weitang Li et al. [6] proposed TD-DMRG implementation on GPU and CPU heterogeneous platform. Google [12]

[*]Equal contribution.
[†]Corresponding author: Xiao-Yang Liu.

NIPS 2020 Workshop on Quantum Tensor Networks in Machine Learning, Montréal, Canada. This is a non-archival publication - the authors may submit revisions and extensions of this paper to other publication venues.

Algorithm 1 Lanczos Eigendecomposition [1]

Require: Vector $\boldsymbol{v}_0$, 3-order tensors $\mathcal{L}$ and $\mathcal{R}$, 4-order tensor $\mathcal{M}$, the maximum number of iterations α, intermediate variable dimension r_m.
Ensure: Minimum eigenvalue ϵ and corresponding eigenvector ψ.

for $k = 1$ to α **do**
  $\boldsymbol{M}(:, 0) = \boldsymbol{v}_0 / \|\boldsymbol{v}_0\|_2$,
  for $j = 2$ to $r_m + 1$ **do**
    $\boldsymbol{M}(:, j) = \boldsymbol{M}(:, j-1) \circ \mathcal{L} \circ \mathcal{M} \circ \mathcal{R}$,
    for $i = 1$ to $j - 1$ **do**
      $\boldsymbol{A}(j-1, i) = \boldsymbol{M}(:, j) \cdot \boldsymbol{M}(:, i)$ and set $\boldsymbol{A}(i, j-1) = \boldsymbol{A}(j-1, i)$,
    end for
    for $i = 1$ to $j - 1$ **do**
      $\boldsymbol{M}(:, j) = \boldsymbol{M}(:, j) - \boldsymbol{M}(:, i) \cdot \boldsymbol{M}(:, j) \times \boldsymbol{M}(:, i)$,
      $\boldsymbol{M}(:, j) = \boldsymbol{M}(:, j) / \|\boldsymbol{M}(:, j)\|_2$,
    end for
  end for
  $[\boldsymbol{e}, \boldsymbol{E}] = \text{eigh}(\boldsymbol{A})$ and set $\boldsymbol{v}_0 = \boldsymbol{M}(:, (1, r_m)) \times \boldsymbol{E}(:, 1)$,
end for
$\psi = \boldsymbol{v}_0 / \|\boldsymbol{v}_0\|_2$ and set $\epsilon = \boldsymbol{e}(1)$.

gives a fast GPU implementation of DMRG in the TensorNetwork library. However, those DMRG implementations are not fully optimized for the GPU architecture.

In this paper, we implement an efficient single-cite finite DMRG algorithm on GPUs. We found the main operations and mainly optimized the high overhead operations, including memory access, tensor multiplication, and data transfer. Evaluation results show that the GPU-based DMRG algorithm achieves good performance.

Our major contributions are summarized as follows.

- We design and implement the single-site finite DMRG algorithm on GPUs.
- We propose optimization strategies for memory access, reducing calculation, faster data transfer, and faster tensor multiplication, thereby improving performance.
- We evaluate the performance of the single-site finite DMRG with experiments on GPUs. We achieved an average speedup of 4.62× and up to 13.23× speedup versus the Jax-GPU implementation in the TensorNetwork library.

This paper is organized as follows. Section 2 describes the single-site finite DMRG algorithm. In Section 3, we present the GPU implementation of the single-site finite DMRG algorithm. Section 4 describes our experimental setup and presents the performance of DMRG algorithm on GPUs. Section 5 gives our conclusions.

2 Overview of the DMRG Algorithm

We briefly describe the single-site finite DMRG algorithm based on Lanczos feature decomposition. The main processes of the DMRG algorithm:

1) Randomly initialize each tensor in MPS $\mathcal{A}_{(n)}$
2) Move the orthogonal center to the n-th tensor $\mathcal{A}_{(n)}$ and calculate its corresponding effective Hamiltonian $\widehat{H}$.
3) Update $\mathcal{A}_{(n)}$ to the lowest eigenstate of $\widehat{H}$ via the Lanczos method.
4) Repeat steps 2-3 to update the n-th to the first tensor in turn; then update the first to n-th tensor in turn.
5) Every updating of a tensor, judge whether the system energy converges. When the energy converges, or the iteration reaches the maximum sweep, the algorithm stops.

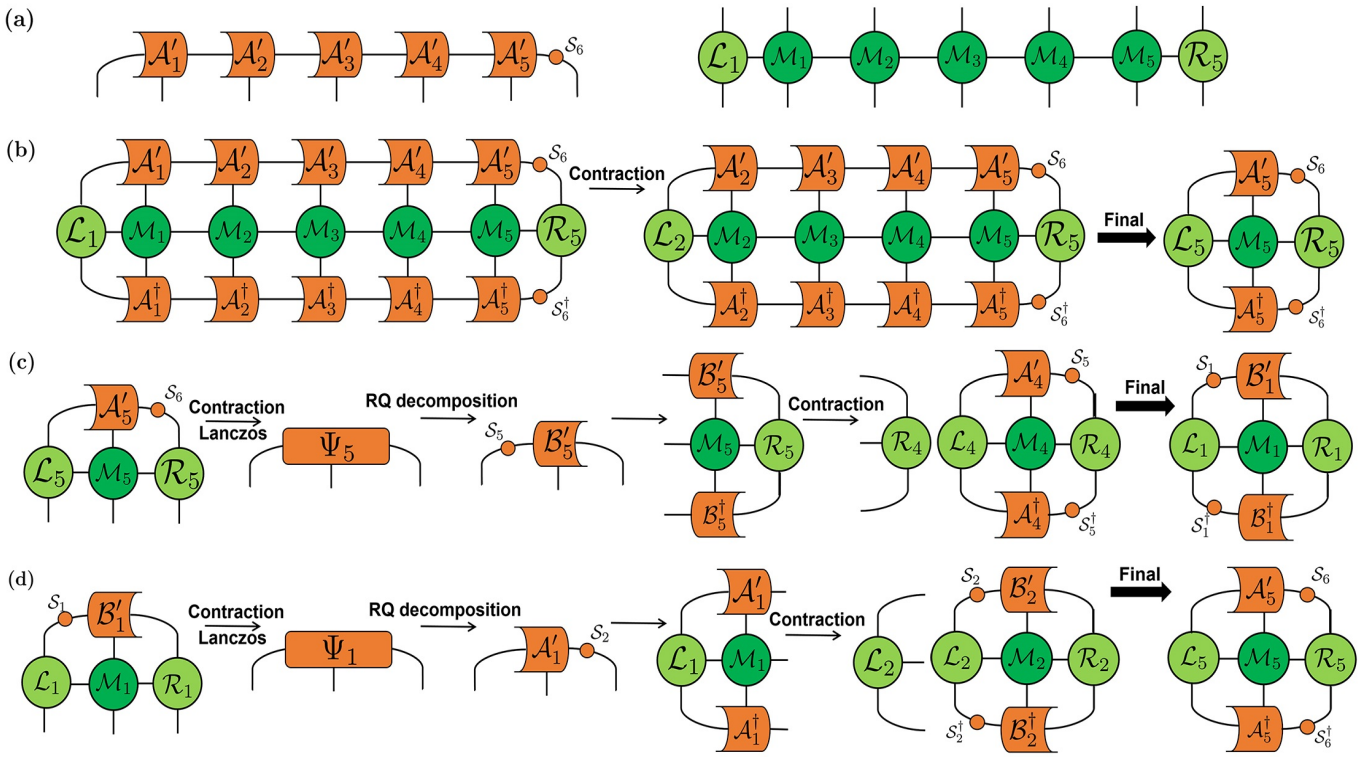

Figure 1: Illustration of the DMRG algorithm.

2.1 Notions and Operations

We use boldface lowercase letters $\boldsymbol{a} \in \mathbb{R}^n$ to denote vectors, boldface uppercase letters $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_2}$ to denote matrices and uppercase calligraphic letters $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ to denote third-order tensors. We use $\cdot$ to denote the dot product, $\circ$ to denote the tensor contraction operation, and $\| \ \|_2$ to obtain the L2 Frobenius norm.

For a vector $\boldsymbol{a} \in \mathbb{R}^n$, $\boldsymbol{a}(n)$ denotes the n-th element. For a matrix $\boldsymbol{M} \in \mathbb{R}^{n_1 \times n_2}$, $\boldsymbol{M}(i,j)$ denotes the (i,j)-th element, $\boldsymbol{M}(:,j)$ denotes the j-th column, and $\boldsymbol{M}(:,(i,j))$ denotes the matrix composed of the i-th column to the j-th column of $\boldsymbol{M}$. For a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, vec$(\mathcal{A})$ denotes the mode-1 vectorization, $\boldsymbol{A}_{(n)}$ denotes the mode-n matricization, $\mathcal{A}'$ and $\mathcal{A}^\dagger$ denotes the regular format and the conjugation format.

2.2 Lanczos Eigendecomposition

A Lanczos eigendecomposition is aimed to obtain the minimum eigenvalue e and corresponding eigenvector $\boldsymbol{v}$ of a vector $\boldsymbol{v}_0$:

$$[e, \boldsymbol{v}] = \text{Lanczos}(\boldsymbol{v}_0, \mathcal{L}, \mathcal{M}, \mathcal{R}),$$

where $\boldsymbol{v}_0 = vec(\Psi^{init})$, $\mathcal{L}$ and $\mathcal{R}$ are two 3-order tensors and $\mathcal{M}$ is a 4-order tensor. Alg. 1 describes the procedures of the Lanczos eigendecomposition [1].

2.3 Single-Site Finite DMRG

The DMRG algorithm solves the optimization problem of the physical system's ground state energy based on the smallest eigenstate. It can be understood as minimizing the following problems:

$$E_g = \min_{\langle\varphi|\varphi\rangle=1} \langle\varphi|\widehat{H}|\varphi\rangle,$$

where the Hamiltonian $\widehat{H} = t\sum_{\langle i,j\rangle} \boldsymbol{S}_p^y \boldsymbol{S}_p^y + t\sum_{\langle i,j\rangle} \boldsymbol{S}_m^x \boldsymbol{S}_m^x,\ t = 2$ under the study of the quantum XX model, and φ is a single site. The steps of the single-site finite DMRG algorithm with 5 sites are shown as Fig. 1.

At step (a), we randomly generate a MPS $\mathcal{A}_n, n \in (1, 2, 3, 4, 5)$ and then perform QR decomposition of the first tensor to obtain $\mathcal{A}_1'$ and $\mathcal{S}_2$. Replace the $\mathcal{A}_2$ with $\mathcal{S}_2 \times_1 \mathcal{A}_2$ and repeat the above steps until the $\mathcal{A}_5'$ and $\mathcal{S}_6$ to get a left regular MPS. Then according to the quantum XX model, we obtain an MPO to express the operations of the left regular MPS. In MPO,

$$\mathcal{M}_n = \begin{bmatrix} \boldsymbol{I} & 0 & 0 & 0 \\ \sqrt{2}\boldsymbol{S}_p & 0 & 0 & 0 \\ \sqrt{2}\boldsymbol{S}_m & 0 & 0 & 0 \\ 0 & \sqrt{2}\boldsymbol{S}_m & \sqrt{2}\boldsymbol{S}_p & \boldsymbol{I} \end{bmatrix}, \text{ where } \boldsymbol{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \boldsymbol{S}_p = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \boldsymbol{S}_m = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},$$

and each $\mathcal{M}_n$ represents a 4-order tensor. $\mathcal{L}_1 = [1 \quad 0 \quad 0 \quad 0]$, $\mathcal{R}_5 = [0 \quad 0 \quad 0 \quad 1]$ are considered as 3-order tensors in $\mathbb{R}^{4\times1\times1}$.

At step (b), we combine the MPO to the left regular MPS and the conjugate form of the left regular MPS to get a tensor network representation of the system's energy. According to the idea of local energy minimization, we contract the tensor network and use the Lanczos method to calculate the ground state energy. We first contract $\mathcal{A}_1',\ \mathcal{L}_1,\ \mathcal{M}_1,\ \mathcal{A}_1^\dagger$ to get $\mathcal{L}_2 : \mathcal{L}_2 = \mathcal{A}_1' \circ \mathcal{L}_1 \circ \mathcal{M}_1 \circ \mathcal{A}_1^\dagger$ and iteratively perform contraction to get the final tensor network with $\mathcal{A}_5', \boldsymbol{S}_6, \mathcal{L}_5, \mathcal{M}_5, \mathcal{R}_5, \mathcal{A}_5^\dagger, \boldsymbol{S}_6^\dagger$.

At step (c), $\mathcal{A}_5' \times_3 \boldsymbol{S}_6$ gets Ψ_5^{init} and utilize the Lancozs method with $\Psi_5^{init}, \mathcal{L}_5, \mathcal{M}_5, \mathcal{R}_5$ to obtain the ground state Ψ_5. Then use the RQ decomposition of the Ψ_5 to get the $\boldsymbol{S}_5$ and $\mathcal{B}_5'$, and contraction $\mathcal{B}_5', \mathcal{M}_5, \mathcal{R}_5, \mathcal{B}_5^\dagger$ to get $\mathcal{R}_4 : \mathcal{R}_4 = \mathcal{B}_5' \circ \mathcal{R}_5 \circ \mathcal{M}_5 \circ \mathcal{B}_5^\dagger$. Now we get the left shifted tensor network with $\mathcal{A}_4', \boldsymbol{S}_5, \mathcal{L}_4, \mathcal{M}_4, \mathcal{R}_4, \mathcal{A}_4^\dagger, \boldsymbol{S}_5^\dagger$. Repeat the above operations until the final result in (c) of Fig. 1 completing a left shift.

At step (d), $\boldsymbol{S}_1 \times_1 \mathcal{B}_1'$ gets Ψ_1^{init} and utilize the Lanczos method with $\Psi_1^{init}, \mathcal{L}_1, \mathcal{M}_1, \mathcal{R}_1$ to obtain the ground state Ψ_1. Then use the QR decomposition of the Ψ_1 to get the $\boldsymbol{S}_2$ and $\mathcal{A}_1'$, and contraction $\mathcal{A}_1', \mathcal{M}_1, \mathcal{L}_1, \mathcal{A}_1^\dagger$ to get $\mathcal{L}_2 : \mathcal{L}_2 = \mathcal{A}_1' \circ \mathcal{L}_1 \circ \mathcal{M}_1 \circ \mathcal{A}_1^\dagger$. Now we get the right-shifted tensor network with $\mathcal{B}_2', \boldsymbol{S}_2, \mathcal{L}_2, \mathcal{M}_2, \mathcal{R}_2, \mathcal{B}_2^\dagger, \boldsymbol{S}_2^\dagger$. Repeat the above operations until the final result in (d) of Fig. 1 completes a right shift.

Completing a left and right shift is called a sweep and if the difference between the last ground state energy and the current ground state energy is lower than the accuracy, the convergence is considered. When the system converges or reaches the maximum sweep, the algorithm stops and gives the final system's ground state energy.

3 Efficient Single-Site Finite DMRG on GPU

We describe our optimization strategies in detail, including faster memory access, faster multiplication, and data transfer.

3.1 Faster Memory Access

We observe that the slice-by-slice column-major layout tensor storage format in GPU can save computation and memory space of tensor matricizations. Fig. 2 illustrates the layout of a $3 \times 3 \times 2$ tensor $\mathcal{X}$ in a 1D array $\boldsymbol{x}$ and its three matricizations. Mode-1 matricization column-major storage and Mode-3 matricization row-major storage can be obtained directly by $\boldsymbol{x}$. Mode-2 matricization row-major storage can be obtained by the first 12 and second 12 elements of $\boldsymbol{x}$.

Suppose there are N sites in the DMRG algorithm, each tensor of MPS needs to be matricized with mode-3 at step (a). At step(c) and step(d), matricization with mode-1 and mode-3 is required respectively. Therefore, a total of $3 \times N$ times tensor matricization processes are required in the entire algorithm. Using the storage method in Fig.2, the $3 \times N$ times matricization process can be omitted, thereby improving operating efficiency and reducing operating time.

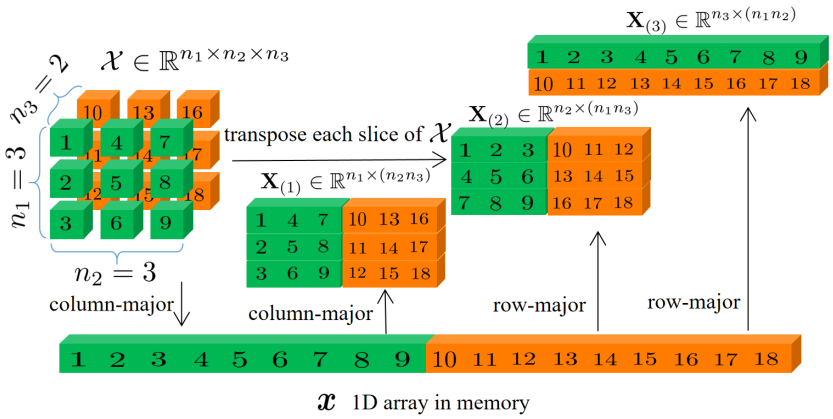

Figure 2: Tensor's storage as a 1D array in GPU memory to avoid explicit tensor matricization.

3.2 Faster Multiplication

3.2.1 Mode-1 And Mode-3 Tensor Contraction

In the DMRG algorithm, there are mode-1 tensor contraction $\Psi_n^{init} = \boldsymbol{S}_n \times_1 \mathcal{B}'_n$ at the steps (a) and (d) and mode-3 tensor contraction $\Psi_n^{init} = \mathcal{A}'_n \times_3 \boldsymbol{S}_{n+1}$ at the step (c). The conventional approach includes the following steps:

- Matrixing tensors $\mathcal{B}$ and $\mathcal{A}$ into matrices $\boldsymbol{B}_{(1)}$ and $\boldsymbol{A}_{(3)}$
- Calculating matrix multiplication: $\psi^{init} = \boldsymbol{S}\boldsymbol{B}_{(1)}$, or $\psi^{init} = \boldsymbol{S}\boldsymbol{A}_{(3)}$
- Folding the matrix into tensors: $\psi^{init} \rightarrow \Psi^{init}$

For mode-1 tensor contraction, $\Psi_n^{init} = \boldsymbol{S}_n \times_1 \mathcal{B}'_n$. According to Fig.2, We can directly calculate $\boldsymbol{S}_n \boldsymbol{B}'_{n(1)}$ in GPU utilizing cublasSgemm() in CUDA programming. For mode-3 tensor contraction, $\Psi_n^{init} = \mathcal{A}'_n \times_3 \boldsymbol{S}_{n+1}$. We calculate $\psi^{init\top} = \boldsymbol{A}_{(3)}^{\top} \boldsymbol{S}^{\top}$ because of the column-major storages of $\boldsymbol{A}_{(3)}^{\top}$ and $\mathcal{A}$ are identical. In CUDA programming, we use cublasSgemm() to calculate mode-3 tensor contraction and avoid mode-3 matricization.

3.2.2 Tensor Core

In order to obtain a faster acceleration effect, we use tensor cores to implement the tensor multiplication calculation. Tensor cores are NVIDIA's advanced technology, which can realize mixed-precision calculations, and dynamically adjust computing power according to the decrease in accuracy, so as to increase throughput while maintaining accuracy. The Tesla V100 GPU which is used in our experiments, contains 640 tensor cores: 8 per streaming multiprocessor (SM). Tensor cores are custom-crafted to dramatically increase floating-point compute throughput at only modest area and power costs. During program execution, multiple tensor cores are used concurrently by a full warp of execution. The threads within a warp provide a larger $16 \times 16 \times 16$ matrix operation to be processed by the tensor cores.

In NVIDIA's official library cuBLAS, the tensor core can be used to accelerate matrix multiplication operations. It requires that the matrix size involved in the operation is a multiple of 8. We found that the scale of the matrix in the calculation of DMRG is suitable for the calculation mode of the tensor core. When the size of the matrix does not meet the requirements of tensor cores, the GPU will switch to using CUDA cores to complete the calculation.

3.3 Data Transfer

In the DMRG algorithm, the input of the algorithm is an MPS, and the MPS's storage increases rapidly with the increase of the number of cites in the system, which will cause a significant time

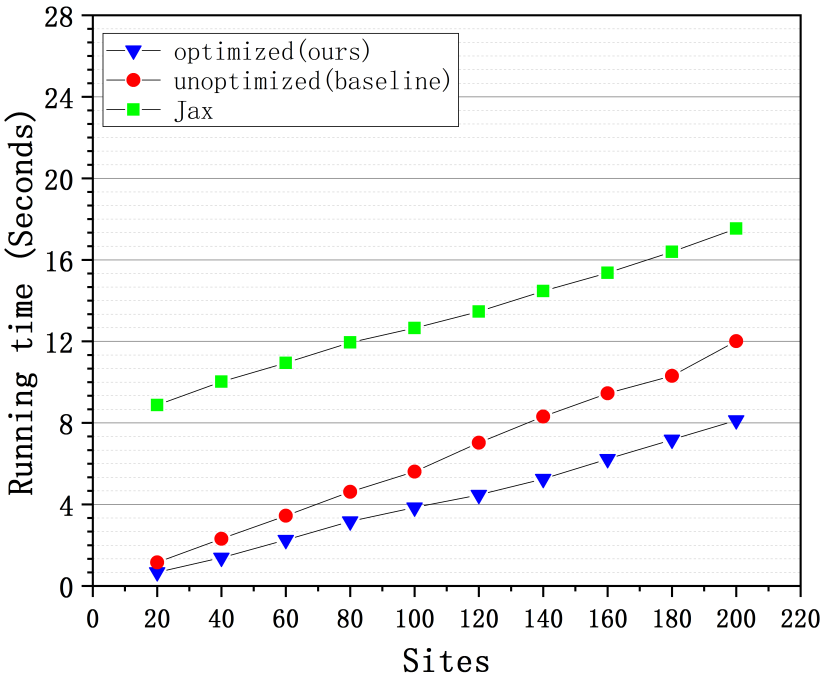

Figure 3: Running time of the DMRG algorithm.

overhead for data transferring between GPU and CPU. Therefore, it makes sense to accelerate data transferring through an effective strategy.

We use streaming to transfer data in parallel. First, we transfer $\mathcal{L}$, $\mathcal{M}$, $\mathcal{R}$ of the MPO, and we reuse the $\mathcal{M}$ because of $\mathcal{M}_n = \mathcal{M}$, $n \in \mathbb{R}$. For data input, we create several streaming transmission modules, and each module performs the special part of the input MPS. When all streams finish, we join the MPS short chains to form the original input MPS.

4 Performance Evaluations

All experiments were performed on an NVIDIA Tesla V100 GPU. The GPU has 32GB device memory, 5120 CUDA cores, and 640 tensor cores.

We use running time as performance metric. The speedup of our GPU implementation over a reference GPU implementation is calculated as: (running time of a reference GPU implementation) / (running time of our GPU implementation). The compared GPU implementations are listed out as follows:

- **Jax:** We use the GPU implementation of the DMRG algorithm in TensorNetwork, which is a library maintained by Google on Github for easy and efficient manipulation of tensor networks. It runs on Jax.
- **GPU Baseline:** We provide baseline implementation on GPU using the cuBLAS and cuSOLVER libraries. This implementation does not utilize the optimization techniques in Sections 3.
- **Our GPU implementation (Ours):** Our GPU optimized algorithm runs on CUDA 10.1 and uses official NVIDIA function libraries including cuBLAS, cuSOLVER, cuTENSOR. Moreover, we use the optimization techniques in Section3 to implement the GPU optimization algorithm.

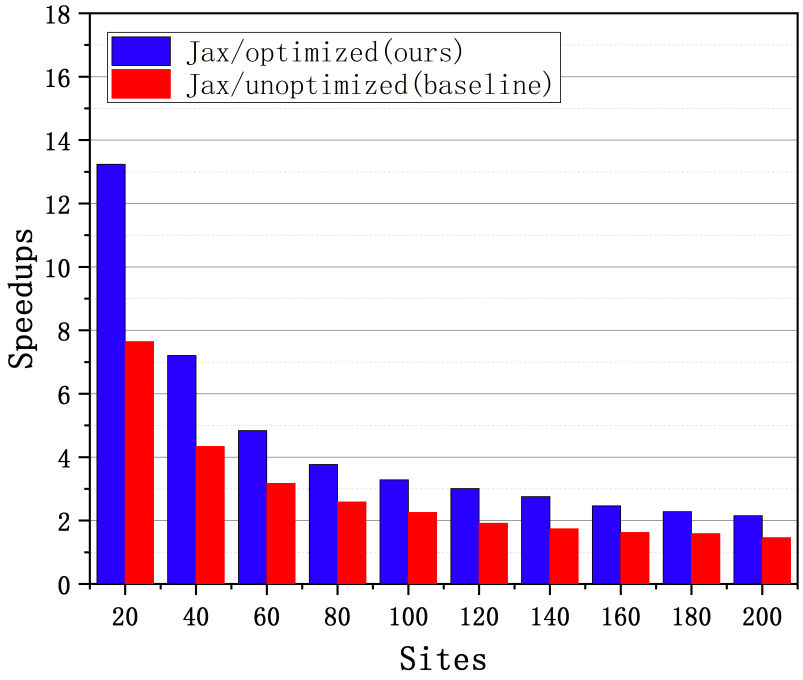

Figure 4: Speedups of the DMRG algorithm.

We set the sites from 20 to 200 and the interval to 20. In order to control a single variable, we control the precision to 1×10^{-6}, and the number of iterations to 5 times. The data here is all single-precision, and three versions of the algorithm are executed 10 times. The average is taken as the final running time.

Fig. 3 shows the running time. We can observe that the running time of our optimized GPU algorithm is shorter in the case of a different number of sites. The baseline GPU version has a smaller running time gap with the optimized GPU algorithm when the number of sites is small. As the number of sites increases, this gap gradually increases. The algorithm implemented by Jax has the longest running time.

Fig. 4 shows the speedups. When the number of sites increases, the speedups will drop. The baseline GPU algorithm achieves an average of $2.84\times$ and up to $7.64\times$ speedups versus Jax. In contrast, the optimized GPU algorithm achieves an average of $4.62\times$ and up to $13.23\times$ speedups versus Jax.

5 Conclusion

In this paper, we analyzed the principle of the single-site finite DMRG algorithm and the Lanczos eigendecomposition method. According to the characteristics of the algorithms and the architecture of GPU, we designed three optimization strategies for reducing calculation and memory consumption, accelerating tensor operations. Through experiments, we obtained an average of $4.62\times$ and up to $13.23\times$ speedups over the Jax-GPU implementation in the TensorNetwork library on a Tesla V100 GPU.

Future work will focus on the implementation of infinite DMRG algorithm and distributed the single-site DMRG algorithm to take full advantages of the multi-GPU platforms and multi-node GPU clusters.

References

[1] D. Calvetti, L. Reichel, and Sorensen D.C. An implicitly restarted lanczos method for large symmetric eigenvalue problems. *Electronic transactions on numerical analysis ETNA*, 2:1–21, 1994.

[2] Stefan Depenbrock, Ian P. Mcculloch, and Ulrich Schollwöck. Nature of the spin-liquid ground state of the s=1/2 heisenberg model on the kagome lattice. *Physical Review Letters*, 109(6):067201, 2012.

[3] René Derian, Andrej Gendiar, and Tomotoshi Nishino. Modulation of local magnetization in two-dimensional axial-next-nearest-neighbor ising model. *Journal of the Physical Society of Japan*, 75(11), 2006.

[4] C. J. Gazza, M. E. Torio, and J. A. Riera. Quantum dot with ferromagnetic leads: a dmrg study. *Physical Review B*, 73(19):2549–2555, 2006.

[5] C. Hubig, I. P. Mcculloch, U. Schollwoeck, and F. A. Wolf. A strictly single-site dmrg algorithm with subspace expansion. *Physical review*, 91(15):155115.1–155115.10, 2015.

[6] Weitang Li, Jiajun Ren, and Zhigang Shuai. Numerical assessment for accuracy and gpu acceleration of td-dmrg time evolution schemes. *The Journal of Chemical Physics*, 152(2):024127–, 2020.

[7] I. P McCulloch and M Gulácsi. The non-abelian density matrix renormalization group algorithm. *Europhysics Letters (EPL)*, 57(6):852–858, mar 2002.

[8] A. Mering and M. Fleischhauer. The one-dimensional bose-fermi-hubbard model in the heavy-fermion limit. *Physical Review A*, 77(2):681–700, 2007.

[9] S Moukouri, L. G Caron, C Bourbonnais, and L Hubert. Real-space density-matrix renormalization-group study of the kondo necklace. *Physical Review B Condensed Matter*, 51(22):15920, 1995.

[10] Csaba Nemes, Gergely Barcza, Zoltán Nagy, ?Rs Legeza, and Péter Szolgay. The density matrix renormalization group algorithm on kilo-processor architectures: Implementation and trade-offs. *Computer Physics Communications*, 185(6):1570–1581, 2014.

[11] M. Kaulkei. Peschel. A dmrg study of the -symmetric heisenberg chain. *The European Physical Journal B*, 1998.

[12] Chase Roberts, Ashley Milsted, Martin Ganahl, Adam Zalcman, Bruce Fontaine, Yijian Zou, Jack Hidary, Guifre Vidal, and Stefan Leichenauer. Tensornetwork: A library for physics and machine learning. 2019.

[13] E. M. Stoudenmire and Steven R. White. Studying two dimensional systems with the density matrix renormalization group. *Annual Review of Condensed Matter Physics*, 3(3):111–128, 2011.

[14] Laura Urba, Emil Lundh, and Anders Rosengren. One-dimensional extended bose-hubbard model with a confining potential: a dmrg analysis. *Journal of Physics B Atomic Molecular and Optical Physics*, 39(24), 2006.

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

[16] White and R. Steven. Density matrix renormalization group algorithms with a single center site. *Physical review. B, Condensed Matter And Materials Physics*, 72(18):p.180403.1–180403.4, 2005.

[17] Steven R White. Density-matrix algorithms for quantum renormalization groups. *Physical Review B Condensed Matter*, 48(14):10345, 1993.

[18] Simeng Yan, David A. Huse, and Steven R. White. Spin-liquid ground state of the s = 1/2 kagome heisenberg antiferromagnet. *ence*, 332(6034):1173–6, 2011.