Sparse Matrix Reordering Method Selection with Parallel Computing and Deep Learning

Rui Xia[1],Jihu Guo[1], Huajian Zhang[1], Shun Yang[1], Qinglin Wang[*1,2,3], Jie Liu[*1,2,3]
[1] College of Computer Science, National University of Defense Technology, Changsha, China
[2] Laboratory of Digitizing Software for Frontier Equipment, National University of Defense Technology
[3] National Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology
xiarui21@nudt.edu.cn, guojihu1998cs@gmail.com,{zhjsag,yangshun}@nudt.edu.cn,wangqinglin_thu@163.com, liujie@nudt.edu.cn

***Abstract*—Sparse matrix reordering is an important step in Cholesky decomposition. By reordering the rows and columns of the matrix, the time of computation and storage cost can be greatly reduced. With the proposal of various reordering algorithms, the selection of suitable reordering methods for various matrices has become an important research topic. In this paper, we propose a method to predict the optimal reordering method by visualizing sparse matrices in chunks in a parallel manner and feeding them into a deep convolutional neural network. The results show that the theoretical performance can reach 95% of the optimal performance, the prediction accuracy of the method can reach up to 85%, the parallel framework achieves an average speedup ratio of 11.35 times over the serial framework, and the performance is greatly improved compared with the traversal selection method on large sparse matrices.**

***Index Terms*—reordering method, sparse matrix, deep learning, parallel computing**

I. Introduction

Sparse matrix decomposition is an important research hotspot in the field of high-performance computing (HPC). Reordering is one of the important steps, which may consume half of the decomposition time. If an optimal reordering method that matches the characteristics of the matrix can be chosen, the number of padded values generated by the decomposition can be reduced, thus reducing the time and storage cost of subsequent computations.

Many researchers have proposed a variety of different sparse matrix reordering algorithms. Amestoy P.R. et al.[1] proposed an approximate minimum degree ordering algorithm(AMD). Davis T.A. et al.[2] proposed COLAMD, an approximate column minimum degree ordering algorithm. Based on Cuthill E. and Mckee J.'s work[3], George A. et al.[4] proposed the reverse Cuthill-Mckee (RCM) algorithm through improvement. The METIS graph partitioning software developed by Karypis Lab[5–11] is also widely used for sparse matrix reordering. The nested partitioned algorithm called NESDIS[5, 10] is also commonly used for sparse matrix reordering.

How choose the most suitable method from the many reordering methods has also become an important research topic subsequently. It is too expensive in time and space to try all reordering methods. Moreover, there are no uniform evaluation indicators for evaluating the merits of reordering methods. The above factors make the selection of reordering methods difficult.

In recent years, Deep Neural Networks (DNN) have been successful in various fields, such as meshing in computational fluid dynamics (CFD)[12, 13]. In the field of HPC, many tasks require classification and decision-making, and deep neural networks have achieved good results. To improve the efficiency of the sparse matrix reordering method selection algorithm, we introduce a deep learning method for prediction and design a multi-process parallel framework for speedup. Experiments confirm that the method achieves good results.

The main contributions of this paper are the following two points. First, this paper proposes a method for visualizing sparse matrices by parallel computing, which can process sparse matrices of large size in a relatively short time and generate the picture needed for prediction. Second, this paper proposes to use a deep neural network to select an appropriate reordering method for sparse matrices based on the visualization of sparse matrices, which achieves high accuracy and low theoretical performance loss.

Our experimental results show that using the prediction method proposed in this paper, we can theoretically achieve the best performance of 95% and achieve up to 85% accuracy. The parallel framework achieves an average speedup ratio of 11.35 times over the serial framework on the dataset and has a greater performance advantage over traditional traversal selection methods on sparse matrices of larger size.

II. Background

This section briefly introduces the reordering methods and deep neural networks involved in the research.

A. Introduction to Reordering Methods

We selected five more general sparse matrix reordering algorithms for testing. These methods are representative and the results are easily extended to other reordering algorithms with similar ideas.

* Qinglin Wang and Jie Liu is the corresponding author.

1) AMD: AMD is an approximate minimum degree ordering algorithm to find a permutation P so that the Cholesky factorization $PAP^T = LL^T$ has fewer nonzero entries than the Cholesky factorization of A. The algorithm maps the matrix into an undirected graph and completes the reordering by continuously selecting the minimum degree node as the pivot and performing elimination.

2) SYMAMD: COLAMD is a column approximate minimum degree sorting algorithm. It is extended from AMD. The algorithm computes a permutation P such that the LU decomposition of AP tends to be sparser than that of A. $P^T A^T AP$ also tends to be sparser than $A^T A$. SYMAMD is a symmetric minimum degree sorting method based on COLAMD. It constructs a matrix M such that $M^T M$ has the same mode as A, then uses COLAMD to compute the column ordering of M.

3) SYMRCM: SYMRCM is the symmetric reverse Cuthill-McKee sorting algorithm. It can give a permutation P such that PAP^T is a sparse matrix with smaller bandwidth than A, especially for long and thin matrices. The algorithm first finds the pseudo-external vertices of the matrix graph, then performs a breadth-first search to generate a hierarchical structure, and reorders the vertices by reducing the distance to the pseudo-external vertices.

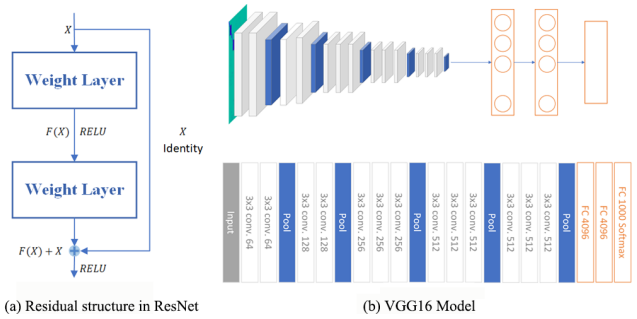
4) METIS: METIS is a powerful graph partitioning package developed by Karypis Lab for partitioning irregular graphs and grids and computing sparse matrix reordering. In SuiteSparse, METIS is used as a nested dissection algorithm for reordering sparse matrices. The algorithm coarsens the graph by folding the vertices and edges, reorders the smaller graphs, and then uncoarsens the smaller graphs by a refinement step to obtain the reordering matrix.

5) NESDIS: NESDIS is a nested segmentation algorithm provided in SuiteSparse. This method is implemented based on the vertex segmentation procedure of METIS. Unlike METIS, which reorders on just the leaves, NESDIS calls an algorithm similar to the minimum degree ordering on the whole graph. In addition, NESDIS can set whether to follow the constrained minimum degree ordering of the divided nodes. To reorder sparse matrices of various sizes, the node size divided in this experiment is small, so it is not followed to improve the running speed.

B. Introduction to Deep Neural Networks

We choose the commonly used ResNet and VGG deep neural networks as our research objects, and the experimental results of these networks are representative.

1) ResNet: The ResNet network was proposed by He Kaiming et al.[14] in 2015. ResNet is optimized for gradient disappearance and explosion problems and degradation problems that may exist in deep learning. ResNet adopts batch normalization to solve the problems of vanishing and exploding gradients, and designs the residual structure shown in Fig. 1(a) to solve the degradation problem in deep networks. The residual structure can artificially make some layers of the neural network skip the connections of neurons in the next layer to weaken the strong connections between layers. In this paper, we will use ResNet18 as the training model.



(a) Residual structure in ResNet (b) VGG16 Model

Fig. 1. Residual structure in ResNet network and VGG16 network structure

2) VGG: VGG network is a kind of convolutional neural network proposed by Oxford's Visual Geometry Group[15]. In the 2014 ImageNet competition, it won first place in the localization task and second place in the classification task. The network structure of VGG16 is shown in Fig. 1(b). The VGG network is composed of 13 convolutional layers, 5 maximum pooling layers, and 3 fully connected layers. In this paper, we use the VGG16 with batch normalization (VGG16_BN) to prevent the gradient explosion phenomenon.

III. Method

This section focuses on our approach to visualizing sparse matrices, the data enhancement strategies we used, the network enhancement strategies introduced into the network, and the parallel program framework used for prediction.

A. Visualization of Matrix Structure Information

The sparse matrix reordering method does not focus on the numerical information of the sparse matrix but focuses more on the structural information of the matrix. Its structural information depends on the non-zero elements' position, non-zero elements' density, and the number of non-zero elements. How to characterize the structural information required for classification is a research focus.

Inspired by the work of Pichel J.C. et al.[16] and Zhao Y. et al.[17] on the selection of sparse matrix compression formats, we store the key structural information of the sparse matrix in the RGB channels of the image. After a lot of experiments, we propose the following storage method, which is beneficial to represent the structural information used for the classification of the picture and achieve a better classification effect.

We selected sparse matrices larger than 512×512. The matrices are evenly cut into 512×512 regions to generate 512×512 images. The three-channel value of each pixel in the image represents the feature of the non-zero elements in its region. The R channel is shown in Eq. 1, which approximately represents the distribution range of non-zero elements in the region, and the G channel is shown in Eq. 2, which represents the density of non-zero elements in the region, the B channel is shown in Eq. 3, which represents the distance

from the average point of the abscissa and ordinate of all non-zero elements in the area to the diagonal line approximately represents the relative location of the divided area and the bandwidth of the distribution. NNZ_L represents the number of non-zero elements in the region, S_L represents the acreage of the region, X_{Max} and X_{Min} respectively represent the maximum and minimum abscissas of non-zero elements in the area, Y_{Max} and Y_{Min} represent the maximum and minimum ordinates of non-zero elements in the area. It should be noted that the values calculated below need to be normalized to $[0, 255]$ to obtain the value of each pixel on the RGB channel.

$$R = |X_{Max} - X_{Min}| + |Y_{Max} - Y_{Min}| \tag{1}$$

$$G = \frac{NNZ_L}{S_L} \tag{2}$$

$$B = |\frac{\sum X_{Nonzero}}{NNZ_L} - \frac{\sum Y_{Nonzero}}{NNZ_L}| \tag{3}$$

Due to the sparseness of the sparse matrix structure, after the matrix picture of the corresponding size is generated in the above manner, there are still a large number of black pixels on the picture. This means that the matrix area represented by this pixel is mostly filled with zero elements and does not contain enough valid information. It not only wastes image space but also is not conducive to image classification. To make full use of the image to store effective structural information, pixels that do not store any effective information can be used to store the structural information of the entire picture.

The pixels are defined as follows. The R channel is defined by Eq. 4, which approximately describes the density of the sparse matrix. The G channel is defined by Eq. 5, which approximately describes the number of nonzero elements of the sparse matrix. The B channel is defined by Eq. 6, which approximately describes the size of the sparse matrix. $Order$, S_G, and NNZ_G represent the order of the matrix, the acreage of the global area, and the number of non-zero elements in the global area. K and α are scaling factors. K is generally set to not less than $\frac{255}{\lfloor \log \max(NNZ_{max}, Order_{max}) \rfloor}$. In this paper, we empirically take $K = 10$. Since the density of all sparse matrices does not exceed 10%, we take $\alpha = \frac{1}{10\%} = 10$.

$$R = \alpha \times \frac{NNZ_G}{S_G} \times 255 \tag{4}$$

$$G = K \times \lfloor \log NNZ_G \rfloor + K \times \frac{NNZ_G - 2^{\lfloor \log NNZ_G \rfloor}}{2^{\lfloor \log NNZ_G \rfloor}} \tag{5}$$

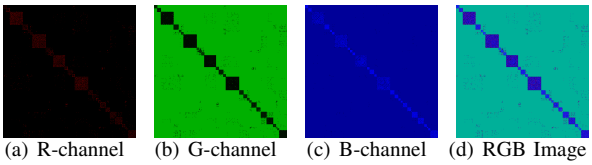
$$B = K \times \lfloor \log Order \rfloor + K \times \frac{Order - 2^{\lfloor \log Order \rfloor}}{2^{\lfloor \log Order \rfloor}} \tag{6}$$

It should be noted that since the pixel values computed above are already distributed in $[0, 255]$, these values do not need to be normalized. Fig. 2 shows the R-channel image, G-channel image, B-channel image, and the corresponding composite image generated by a sparse matrix after the above operations.

B. Data augmentation and improvement

In this paper, some methods are used to enhance the data and improve the training effect. First, since the dataset we collected is not balanced for various classes, we weighted each class according to Eq. 7 during the training process to reduce the impact of imbalanced datasets. After experiments, it is proved that giving different categories the appropriate weight is beneficial to improving the training effect. In the formula, Num_i represents the number of pictures in category i. ω_i represents the weight of category i.

$$\omega_i = \frac{\max_j Num_j}{Num_i} \tag{7}$$



(a) R-channel (b) G-channel (c) B-channel (d) RGB Image

Fig. 2. Three-channel picture and RGB picture of a sparse matrix

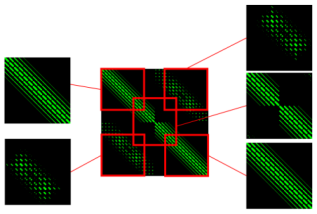


Fig. 3. Cut the matrix image along the main diagonal and anti-diagonal

Second, we consider that sparse matrices often contain rich structural information on the diagonal and antidiagonal. Experience tells us that cropping the matrix image along the diagonal and off-diagonal lines is more beneficial to preserve effective structural information. The pictures will be divided into several 224×224 pictures as shown in Fig. 3. If an image contains less than 1% of non-zero pixel points, the image will be considered as not containing enough valid information and will be eliminated. The cropped images will vote to determine which reordering method applies to the sparse matrix to which they belong. From our experimental experience, we set the voting weight of the matrix images intercepted on the diagonal to be 2 times of the other images, which can effectively improve the recognition success rate of the matrix. The specific steps will be discussed in the following parallel subsection.

C. Network Enhancement Strategy

To improve the classification effect, we introduced an implementation of ECA-Net, a form of channel attention proposed by Wang et al. in 2020[18], on the original ResNet network.

$$\omega = \sigma(C1DK_k(y)) \tag{8}$$

ECA-Net is an extremely lightweight attention mechanism module. The input feature map is subjected to a global average pooling operation, a 1-dimensional convolution operation with a convolution kernel size of k determined by a nonlinear mapping of the channel dimensions, and a $Sigmoid$ activation function to obtain the weights ω of each channel (as in Eq. 8), and the weights are multiplied with the corresponding elements of the original input feature map to obtain the final output feature map. Since this module involves fewer parameters and has a relatively obvious performance gain, it can significantly reduce the complexity of the model while ensuring correct results when interacting across channels.

D. Parallel Program Framework

In the research, it is found that the route of dividing the generated matrix image and predicting the suitable reordering method through voting is suitable for parallelism. This paper improves the efficiency of the algorithm and shortens the running time of the algorithm through multi-process parallelism.

Since five sub-images need to be generated for prediction, the master process divides the five parts of the sparse matrix and transmits them to the five child processes. As shown in Fig. 4, each child process is used to generate each sub-image and make predictions using the trained model, the master process aggregates the prediction results of each child process and votes to give the final prediction result. This is the 6-process parallel framework used for the experiments. To further improve the efficiency of the child processes, as shown in Fig. 4, each child process computes pixel information in parallel with itself and 2 grandchild processes, and then the child processes collect and perform pixel normalization. This is the 16-process parallel framework used for the experiment. The pseudocode is shown in Algorithm 1.

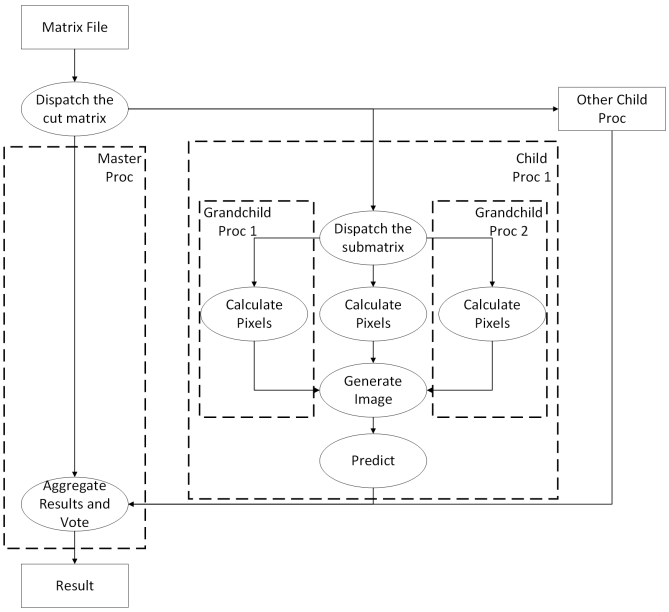


Fig. 4. The multi-process structure used by the prediction of sparse matrix reordering method

Algorithm 1: Sparse Matrix Reordering Method Prediction Parallel Algorithm

Input: Matrix Sp
Output: Reordering Algorithm Method $Method$

```
if Proc is the Master Proc then
    SubMatrix=SplitMatrix(Sp);
    Send(SubMatrix,dest=ChildProc);
    Method=Vote(Recv(source=ChildProc));
    return Method;
end
if Proc is a Child Proc then
    Matrix=Recv(source=MasterProc);
    SubMatrix=SplitMatrix(Matrix);
    Send(SubMatrix[1:],dest=GrandChildProc);
    Pic=CalculatePixel(SubMatrix[0]);
    Pic+=Recv(source=GrandChildProc);
    Send(Predict(Pic),dest=MasterProc);
end
if Proc is a GrandChild Proc then
    SubMatrix=Recv(source=ChildProc);
    Pic=CalculatePixel(SubMatrix);
    Send(Pic,dest=ChildProc);
end
```

IV. Experiment Setup

This section focuses on the dataset composition, the evaluation benchmarks, the experimental environment, and the training hyperparameters.

A. Dataset

The original data used for the dataset comes from SuiteSparse. Some of these matrices were expanded. All sparse matrices have no more than 10% of non-zero elements. The dataset eliminates sparse matrices that have multiple optimal methods and that the optimal method has no obvious advantage in the number of padding elements compared to the suboptimal method, that is, matrices with an improvement of less than 1% in Eq. 9 will be eliminated. $Lnnz_1$ and $Lnnz_2$ represent the number of non-zero elements after suboptimal and optimal reordering respectively and the Cholesky decomposition.

$$Improve = \frac{Lnnz_1 - Lnnz_2}{Lnnz_1} \tag{9}$$

Since the reordering method we studied is aimed at Cholesky decomposition, the matrix we selected is a square matrix that is structurally symmetric. A total of 1076 sparse matrices are selected in the dataset. There are 569 optimal matrices of AMD, 84 optimal matrices of METIS, 342 optimal matrices of NESDIS, 63 optimal matrices of SYMAMD, and 18 optimal matrices of SYMRCM in the dataset. The training set, test set, and validation set are divided in the ratio of 6:2:2. For 5-fold cross-validation, the ratio of the training set to the test set is 8:2.

B. Benchmark

In this paper, $Ratio$ is used as a benchmark to measure the quality of the reordering algorithm. Its definition is shown in Eq. 10. $Annz$ is the number of non-zero elements of the original matrix. $Lnnz$ is the number of non-zero elements of the decomposed matrix.

$$Ratio = \frac{Lnnz}{Annz} \tag{10}$$

Taking this indicator as a benchmark is mainly based on the following considerations. The complexity of the reordering algorithm is proportional to the number of non-zero elements of the original matrix and decomposed matrix. By reordering the matrix, it is possible to reduce the number of padding elements due to decomposition, thereby reducing the time and storage cost of subsequent computations. Moreover, this indicator is not affected by the performance and configuration of different computing platforms and is suitable as an indicator for measuring the complexity of different algorithms.

Besides, we give the performance loss formula for a sparse matrix (Eq. 11). The formula with its variant will be used for the loss analysis in the evaluation.

$$Performance_{Loss} = \frac{Lnnz_{Predict} - Lnnz_{Best}}{Lnnz_{Best}} \tag{11}$$

C. Platform and Hyperparameters

The provided GPU is 1 NVIDIA GeForce RTX 3090, and the video memory size is 24 GB. The provided CPU is an 18-core Intel Xeon E5-2696 v3 and the instance memory is 78 GB. The learning rate is $5e^{-4}$. The momentum is 0.9. We used a learning rate decay strategy in the training process and the gamma is 1. The weight decay is $5e^{-4}$. The batch size is 32. The training epoch is 100, and the step size is 1. The loss function is the cross-entropy loss function.

V. Evaluation

This section shows the prediction accuracy and theoretical performance loss under different architectures of neural networks and analyzes the efficiency improvement under the designed parallel framework.

A. Evaluation of Different Networks Structure

Because the selected sample size is not large and the information density is sparse, using a deeper neural network for training may aggravate the overfitting of the model, while the shallow depth ResNet18 and VGG16 with batch normalization (VGG16_BN) are provided in PyTorch for our use. We will use the provided pre-trained models to migrate learning based on the generated matrix images, and the generated matrix images are classified. The loss function and accuracy of each network are shown in Table I and Fig. 5.

According to the comparison of training results shown in Table I and Fig. 5, in this image classification task, the precision of ResNet18 is about 0.11% lower than that of VGG16_BN, and the recall of ResNet18 is 0.45% higher than that of VGG16_BN. On the F1 score, ResNet18 improved by 0.18% over VGG16_BN. At the same time, it also enlightens us that when the network structure is replaced with a neural network such as VGG16_BN, the classification effect similar to that of ResNet18 can also be obtained. However, combining the model effect and model size, we believe that ResNet18, which has slightly higher accuracy and is more lightweight, may be more suitable.

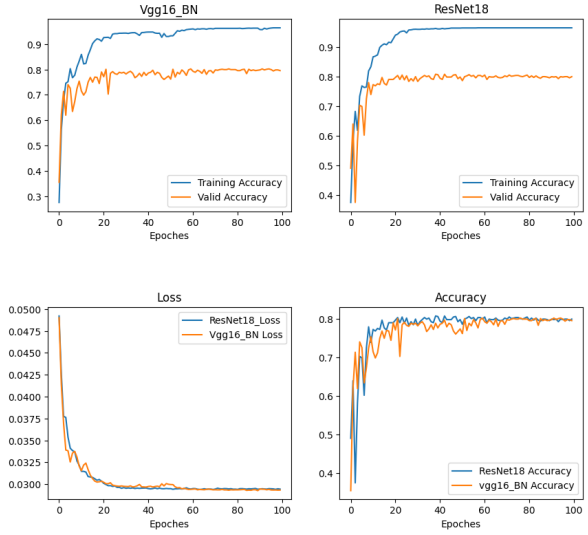


Fig. 5. The comparison of ResNet18 and VGG16_BN

Fig. 6 shows the comparison of the ResNet18 using the ECA attention mechanism with the model without ECA. The convergence of the model is slightly slower than ResNet18. As shown in Table I and Fig. 6, based on the ResNet18 model which has already been improved over the VGG16_BN model, the ResNet18 network using the ECA mechanism improved 3.18%, 3.42%, and 3.30%, respectively, over ResNet18 in terms of recall, accuracy, and F1 score. It proves that our use of the ECA attention mechanism in the model is effective.

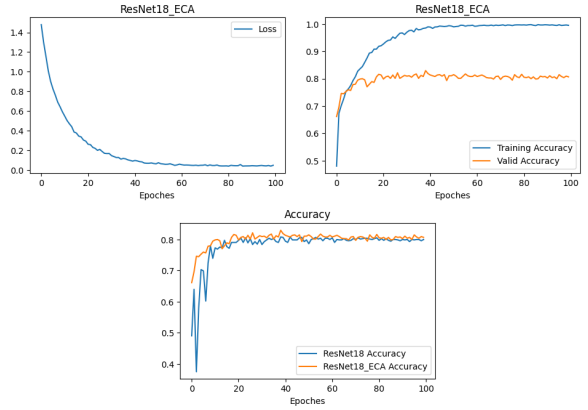


Fig. 6. The training process of ResNet18_ECA and the comparison of accuracy between ResNet18_ECA and ResNet18

TABLE I
RECALL, PRECISION, AND F1 FOR EACH METHOD UNDER DIFFERENT MODELS

Method	VGG16_BN			ResNet18			ResNet18_ECA		
	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
AMD	0.8889	0.8739	0.8814	0.9060	0.8760	0.8908	0.9402	0.8800	0.9091
METIS	0.7857	0.5789	0.6667	0.8571	0.5714	0.6857	0.8571	0.5714	0.6857
NESDIS	0.7917	0.7703	0.7808	0.7361	0.8030	0.7681	0.7917	0.9048	0.8444
SYMAMD	0.4667	1.0000	0.6367	0.6000	0.8182	0.6923	0.5333	0.8000	0.6400
SYMRCM	0.5000	1.0000	0.6667	0.5000	1.0000	0.6667	0.5000	1.0000	0.6667
TOTAL	0.8182	0.8310	0.8245	0.8227	0.8299	0.8263	0.8545	0.8641	0.8593

To further verify the reliability of the findings, we perform a 5-fold cross-validation of the three different models in this dataset. The accuracy of 5-fold cross-validation on the ResNet18_ECA, ResNet18, and VGG16_BN models was 85.21%, 83.63%, and 82.32%, respectively. As shown in Fig. 7, the validation results strongly support the conclusion that using the ResNet18_ECA model outperforms other models on this classification task.

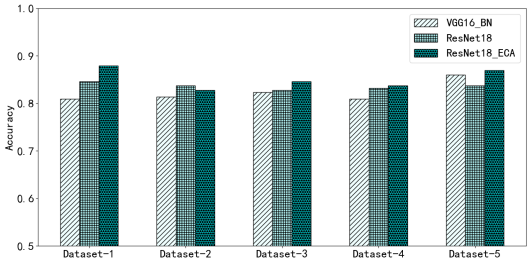


Fig. 7. 5-fold cross-validation results for the three models

We analyze the model performance specifically for each category. As shown in Table I, it can be found that ResNet18_ECA outperforms the other two models in terms of F1 scores in the AMD and NESDIS categories, and is equal to or slightly lower in the other categories. As shown in Fig. 8, the results of the confusion matrix show that ResNet18_ECA has improved over ResNet18 and VGG16_BN for the two categories of AMD and NESDIS, which have more data than other categories. In the other categories, the classification efficiency has remained basically flat or slightly lower. However, ResNet18_ECA performs better than VGG16_BN and ResNet18 networks in general, and the network size is lighter, which is easy to train.

B. Evaluation of Performance Loss

We also focus on sparse matrices where prediction fails. We use the method of converting the structural information of the sparse matrix into RGB three-channel images and using deep learning for training prediction. Although the prediction of these sparse matrix reordering methods fails, the predicted method has a certain probability to map to the sub-optimal method, and the predicted method often has a small performance gap compared with the optimal method.

Through the data in Table II, the average loss of the ResNet18_ECA model is only about 3.41%, compared to 4.95% and 4.38% for the ResNet18 and VGG16_BN models. In the case of prediction failure, the model has a 50% probability of mapping to a suboptimal method. And the performance loss of most failure matrices can be controlled below 50% of the performance of the optimal algorithm, and about half of the matrices can be controlled below 10%. The average performance loss is only about 4% of the optimal performance, which is enough to demonstrate the validity of the model prediction results.

C. Evaluation of Parallel Frameworks

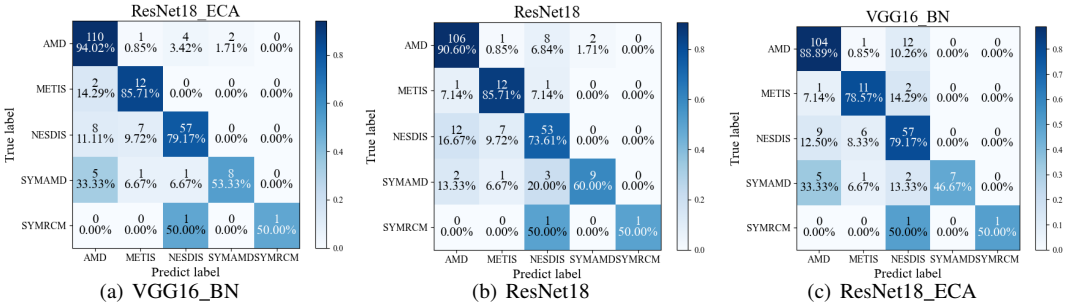
We compare the performance of all sparse matrices of the dataset in serial and parallel. The results show that the average speed-up ratio is 11.35 times, the maximum speed-up ratio is 14.92 times, and the minimum speed-up ratio is 2.95 times. Most sparse matrix speedups are distributed between 12 times and 14 times. Its specific distribution is shown in Fig. 9.

The 6-process parallel framework uses different processes to generate sub-pictures. As shown in Fig. 10, it greatly improves performance compared with serial programs. Compared with the 6-process parallel framework, the child process divides the pixel points of each subgraph into itself and two grandchild processes for parallel computation in the 16-process parallel framework. It can be seen from Fig. 10 that the 16-process parallel framework has achieved a good acceleration effect, and as the size of the sparse matrix increases, the acceleration effect is more obvious.

The method proposed in this paper has more advantages in the selection of large-scale sparse matrix reordering methods. It takes even less time to predict the appropriate method than the time consumed by running one of the reordering methods when the sparse matrix size is large. This paper selects sparse matrices with more than 500,000 non-zero elements in the dataset and selects the SYMRCM method as a comparison. The experimental results are shown in Fig. 11. The method in this paper can predict the reordering method adapted to the sparse matrix before the SYMRCM reordering method is

TABLE II
THE STATISTICS TABLE OF PERFORMANCE LOSS

Network	Second optimal hit rate	Performance Loss			Average Loss
		<10%	10%-50%	>50%	
VGG16_BN	51.28%	41.18%	33.33%	25.49%	4.38%
ResNet18	51.28%	48.72%	35.90%	15.38%	4.95%
ResNet_ECA	55.88%	47.06%	44.12%	8.82%	3.41%



(a) VGG16_BN (b) ResNet18 (c) ResNet18_ECA

Fig. 8. The confusion matrix of ResNet18, ResNet18_ECA, and VGG16_BN

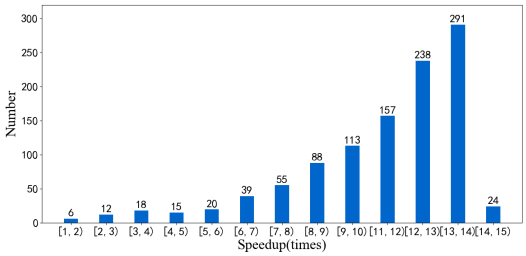


Fig. 9. Speedup distribution for sparse matrices in the dataset

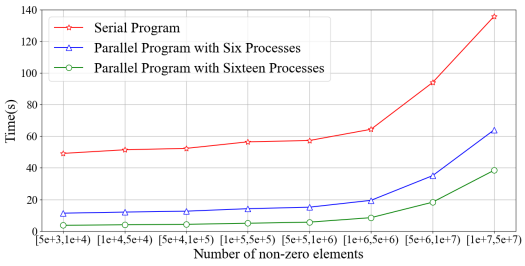


Fig. 10. Time of serial and parallel with sparse matrices of different sizes

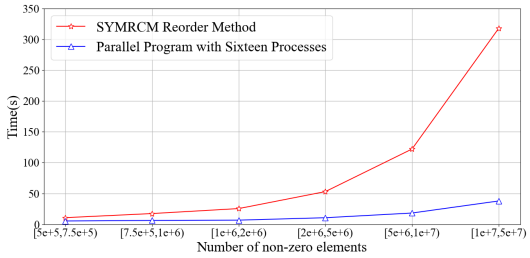


Fig. 11. Time of SYMRCM and parallel with sparse matrices of different sizes

completed, and the larger the size of the sparse matrix, the more obvious the acceleration effect. It can be inferred that the performance of the method proposed in this paper is greatly improved compared with the traversal selection method.

VI. RELATED WORK

Reordering based on deep learning is also a potential research hotspot in recent years, and many related works have explored this issue before.

Watanabe C. et al.[19] proposed a new matrix reordering method that can be used to visualize the global structure of the reordered observation matrix. Kwon Oh-Hyun et al.[20] realize the visualization of reordering algorithms under different indicators. Based on this work, they propose a deep generative model for reordering adjacency matrices[21]. These works focus on the visualization of matrix reordering. In this paper, we also visualize and classify the matrix.

Some works make the selection based on a user framework. Behrisch M. et al.[22] proposed a user-oriented matrix reordering method selection model GUIRO, which helps users better understand the complex reordering process and help users pick out suitable reordering algorithms. In comparison, the filled metric evaluation metrics we proposed are related to the algorithm complexity itself, which is more objective for the evaluation of the algorithm.

Mehrabi A. et al.[23] built a simple two-layer fully connected model to realize the simple classification of the column reordering algorithm on the GPU by some sparse matrices. However, this approach is highly dependent on the architecture of the GPU and the runtime parameters. In contrast, our method is platform-independent and does not require runtime parameters as a supplement.

VII. Conclusion

We propose a method to select the optimal reordering method of sparse matrices by parallel visualization and deep learning. The results show that the performance can theoretically reach 95% of the optimal performance and the prediction accuracy can reach 85%. The parallel framework achieves an average speedup ratio of 11.35 times over the serial framework, and on large sparse matrices, the performance is greatly improved compared to the traversal selection method. Future research will focus on optimizing the network structure to improve the prediction accuracy and optimizing the parallel structure to improve the efficiency of parallelism.

Acknowledgment

The work is supported by National Key Research and Development Program of China under Grant No. (2021YFBO300101).

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[2] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "A column approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 3, pp. 353–376, 2004.

[3] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, 1969, pp. 157–172.

[4] A. George and J. W. Liu, "An implementation of a pseudoperipheral node finder," *ACM Transactions on Mathematical Software (TOMS)*, vol. 5, no. 3, pp. 284–295, 1979.

[5] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[6] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed computing*, vol. 48, no. 1, pp. 96–129, 1998.

[7] G. Karypis, "Multi-constraint mesh partitioning for contact/impact computations," in *SC'03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. IEEE, 2003, pp. 56–56.

[8] D. LaSalle and G. Karypis, "Multi-threaded graph partitioning," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 225–236.

[9] D. LaSalle and G. Karypis, "Efficient nested dissection for multicore architectures," in *European Conference on Parallel Processing*. Springer, 2015, pp. 467–478.

[10] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis, "Improving graph partitioning for modern graphs and architectures," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, 2015, pp. 1–4.

[11] D. LaSalle and G. Karypis, "A parallel hill-climbing refinement algorithm for graph partitioning," in *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, 2016, pp. 236–241.

[12] X. Chen, J. Liu, C. Gong, S. Li, Y. Pang, and B. Chen, "Mve-net: An automatic 3-d structured mesh validity evaluation framework using deep neural networks," *Computer-Aided Design*, vol. 141, p. 103104, 2021.

[13] X. Chen, T. Li, Q. Wan, X. He, C. Gong, Y. Pang, and J. Liu, "Mgnet: a novel differential mesh generation method based on unsupervised neural networks," *Engineering with Computers*, pp. 1–13, 2022.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] J. C. Pichel and B. Pateiro-López, "A new approach for sparse matrix classification based on deep learning techniques," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 46–54.

[17] Y. Zhao, J. Li, C. Liao, and X. Shen, "Bridging the gap between deep learning and sparse matrix format selection," in *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, 2018, pp. 94–108.

[18] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "Eca-net: Efficient channel attention for deep convolutional neural networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 531–11 539.

[19] C. Watanabe and T. Suzuki, "Deep two-way matrix reordering for relational data analysis," *Neural Networks*, vol. 146, pp. 303–315, 2022.

[20] O.-H. Kwon, C.-H. Kao, C.-h. Chen, and K.-L. Ma, "A deep generative model for matrix reordering," *arXiv preprint arXiv:2110.04971*, 2021.

[21] O.-H. Kwon, C.-H. Kao, C.-H. Chen, and K.-L. Ma, "A deep generative model for reordering adjacency matrices," *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[22] M. Behrisch, T. Schreck, and H. Pfister, "Guiro: user-guided matrix reordering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 184–194, 2019.

[23] A. Mehrabi, D. Lee, N. Chatterjee, D. J. Sorin, B. C. Lee, and M. O'Connor, "Learning sparse matrix row permutations for efficient spmm on gpu architectures," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 48–58.