



MANet: An Architecture Adaptive Method for Sparse Matrix Format Selection

Zhenglun Sun , Peng Qiao , and Yong Dou

National University of Defense Technology, Changsha, China
{zhenglun_sun, pengqiao, yongdou}@nudt.edu.cn

Abstract. The proliferation of modern computer architectures brings a great challenge to sparse matrix-vector multiplication (SpMV), which is widely used in scientific computing and artificial intelligence. Providing a suitable sparse matrix format for SpMV is crucial to achieve high performance by enhance data locality and cache performance. However, for different architectures, the best sparse matrix format varies. In this paper, we propose a novel architecture adaptive sparse matrix format selection method, MANet, to select proper format to optimize performance of SpMV. This method transforms a sparse matrix into a high-dimensional image, with the matrix sparseness feature and architecture feature combined as inputs. To evaluate the effectiveness of this method, we generated a dataset that includes various scientific problems and architectures with augmentation. Results show that MANet improves sparse matrix selection accuracy by 6% compared to previous works and can achieve a speedup of up to 230% compared to methods with a fixed format. When adapting to an architecture that is not presented in the training, it can still provide 88% selection accuracy and 14% higher than the previous approaches, without further training.

Keywords: Sparse Matrix-Vector Multiplication · Sparse Matrix Format Selection · High-Performance Computing · Architecture adaptive · Deep Learning

1 Introduction

In high-performance computing, the Sparse Matrix-Vector Multiplication (SpMV) kernel plays an important role in scientific computing [15] and related numerical computing fields such as weather forecasting [8], computational fluid dynamics [10]. High computation efficiency of SpMV is hard to achieve due to the irregular memory access patterns introduced by sparse matrices.

Many researchers have sought to accelerate SpMV from both architectural and algorithmic perspectives. In terms of architecture, attempts have been made to increase CPU and memory frequencies for direct speedup, as well as to employ larger caches to reduce repetitive data I/O.

From algorithmic perspective, numerous sparse matrix formats and algorithms have been proposed to improve data locality. Here we focus particularly on the sparse matrix format level. A number of storage formats have been developed to address the heterogeneity of nonzeros distribution, with impacts on data locality, cache performance, and thus overall performance. Examples of such formats include COO, CSR, and BSR, which are designed to meet the demands of modern architectures and scientific applications. With an appropriate selection of sparse matrix format, the performance of SpMV can be significantly enhanced [6]. In this work, we aim to select an appropriate format with different architecture settings.

Existing methods for sparse matrix selection rely on machine learning algorithms [3, 12, 19], which will extract nonzeros features and predict matrix format with the support of Convolutional Neural Networks (CNN) or the decision tree. However, the prediction results of previous studies may not be well-adapted to different architectures. The rapid progress in computer architecture has made it challenging to maintain a single optimal format that can be used across different CPUs or accelerators [2, 9, 13]. Therefore, it is necessary to develop a format selection method that is able to adapt to varying architectures.

To improve the generalizability of the model across various architectures, we introduce a sparse matrix selection approach with architecture generalization entitled MANet. Our proposed method utilizes matrix pooling-like normalization as preprocessing technique and employs a multiple-input CNN to select the most suitable storage format for the matrix based on the features of the architecture and the matrix.

Our contribution consists of three parts, and we will provide the corresponding code after finishing the organization process:

- We proposed a format selection network for sparse matrices called MANet, which is adaptive to architecture. In terms of adaptability, MANet improves prediction accuracy by 20% when adapting to other platforms with different architecture settings. When adapting to platforms with architectures that have not been previously encountered, MANet yields a 14% improvement compared to related works.
- After adaptation to the approximate platform, the SpMV speedup is 230% after format selection using MANet compared to the COO storage format. Moreover, 89.3% of the matrices can reach the minimum time after using MANet for format selection.
- Our research further probes into the effect of architecture settings on the format distribution in a CPU platform. We indicate that the computation speed of SpMV is determined by the hardware architecture setting, including processor frequency, memory bandwidth, cache size, etc.

2 Background

2.1 Sparse Matrix Storage Format

The main objective of sparse matrix storage formats is to compress matrices, thereby improving locality and reducing storage overhead. Various storage formats, custom-tailored to different distributions of nonzeros, have been developed. The selection of an appropriate format entails consideration of characteristics associated with the nonzeros distribution and architecture.

Our work focuses on three commonly used formats, COO, CSR, and BSR [5].

- The Coordinate Format (COO) storage structure organizes a matrix into three arrays: *col*, *row*, and *val*. The elements in each array specify the column indices of nonzeros entries, the row indices of nonzeros entries, and their values respectively.
- The Compressed Sparse Row (CSR) format stores the column coordinates of the nonzeros into an array *ind*, as well as containing the nonzeros values themselves in another array *val*. Additionally, CSR contains a pointer array *ptr* which helps to quickly identify the intervals of each row, as if listing it out as a list. The CSR format provides a practical approach to representing sparse matrices in three relatively small arrays.
- The Block Sparse Row (BSR) format divides a matrix into multiple dense blocks and stores them using the same CSR-style indexing used by the CSR format. The index array *ptr* gives an interval for each row of each block, the value array *val* contains the nonzeros entries, and the column coordinates of the nonzeros are given by *ind*.

2.2 Influence of Nonzeros Distribution

The standard SpMV computation paradigm is $y = Ax$, where y and x are vectors, and A is a sparse matrix. When performing SpMV computations, each non-zero element A_{ij} is accessed only once. Our work explores strategies to reuse both y and x , enabling the exploitation of data locality opportunities present in SpMV sessions.

In dense storage format, SpMV multiplication is known to be inefficient due to the traversal of zero values during updating the vector. Moreover, non-optimized memory and cache access can lead to significant drops in overall performance. A suitable storage format can alleviate these issues while simultaneously providing opportunities for parallel computing [13].

By taking the matrix stored in CSR format as an example and employing CSR as a storage structure, the memory access pattern is changed. The pseudocode of SpMV when dealing with a matrix stored in CSR is presented in Algorithm 1. This approach only requires $m + 3nnz$ memory accesses when without cache for a given matrix of size $m \times n$ and nonzeros nnz , resulting in significant speed improvements compared to the traditional dense format, which

Algorithm 1. SpMV using CSR format

Input: A : input matrix with $m \times n$ size
val: nonzeros in A
ptr: row offsets in CSR
ind: column index in CSR
x: input vector
Output: y : output vector
1: **for** $i = 0, 1, 2, \dots, m - 1$ **do**
2: $y[i] \leftarrow 0$
3: **for** $j = ptr[i], \dots, ptr[i + 1] - 1$ **do**
4: $y[i] \leftarrow y[i] + val[j] * x[ind[j]]$
5: **end for**
6: **end for**
7: return y

would require $m * n$ memory accesses. And for best case, it will only require $m + n + 2 \times nnz$.

From the above, it can be found that the access optimization due to the sparse matrix storage format is directly related to the distribution of nonzeros. Different storage formats will correspond to different SpMV algorithms, which will have an impact on the access pattern of x . To select proper format, CNN, decision tree and graph neural network (GNN) are used.

3 Methodology

3.1 Overview

In this section, we present a multi-input sparse matrix format selector that is able to adapt to various architectures while overcoming several challenges. Our proposed approach combines matrix pre-processing with architecture features that are associated with the distribution of non-zero data. This leads to improved performance when adapting to platforms with varying architecture settings. We provide a detailed overview of the construction process, from data pre-processing to classification.

The left part of Fig. 1 contains 4 steps to preprocess data and construct a multi-input CNN network. Assuming there is already a sparse matrix dataset D and target architectures P_i , where $i = 1, 2, \dots, n$.

(1) This work requires a matrix training label for the first step. To achieve the best performance in sparse matrix format, the training labels are acquired by running SpMV on P_i 100 times and measuring the SpMV computing time. By measuring the SpMV computing time, this step selects a format that makes SpMV execute at the fastest speed as a label and attaches the label to with matrix ID. (2) The second step will normalize the matrix into a fixed size by pooling the matrix. With a fixed size, the sample formed by the sparse matrix can be used for CNN training and classification. (3) The third step of the process

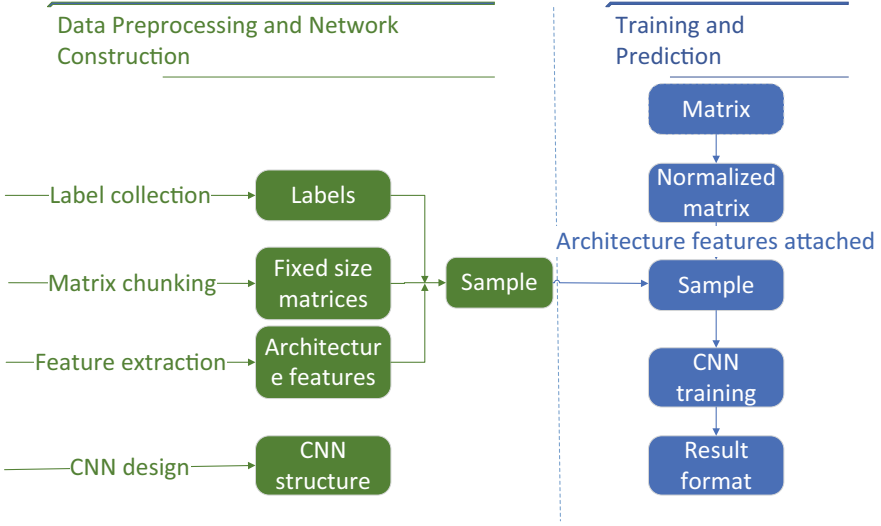


Fig. 1. Overview of the data preprocessing, network construction and training.

is focused on extracting architecture features from P_i that affect SpMV execution time. Data preprocessing allows us to obtain high-dimensional image data associated with both matrix nonzero features and architecture features. (4) The final step of the left part in Fig. 1 forms a multi-input CNN network to extract features from the sample. (5) The right part of Fig. 1 performs the CNN training and prediction procedure.

The main challenges we face are in Steps 3 and 4. Since architectures have many features, it is necessary to conduct experiments to determine which of these features should be encoded into the sample. The high dimension of the samples makes it difficult for CNN to extract features for training. Designing an appropriate network structure to remain amenable to an architecture adaptive format selector remains a problem for our work.

3.2 Matrix Labeling

For the classification task, CNN requires labels to provide ground truth. In order to achieve this, 100 iterations of SpMV were executed on platform P_i , and the format with the shortest average time was selected as the label.

As previously discussed, data locality and other factors influencing the speed of SpMV can be highly variable, depending on the architecture configuration. In practical computing scenarios, the CPU retrieves data, including requested and surrounding data, thus creating a data locality. Variations in CPU frequency, cache size, etc. can have an effect on data locality and other factors that impact the speed of SpMV.

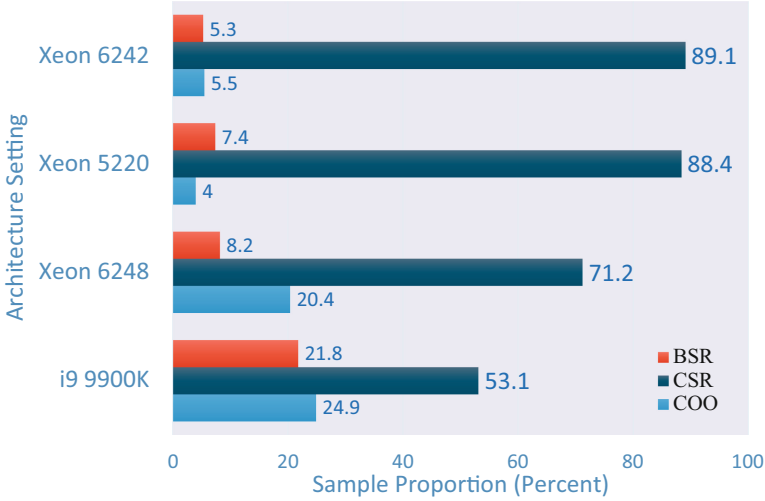


Fig. 2. The generated format label distribution on different architecture.

This experiment investigated the performance of SpMV across multiple platforms with different architectures, and also involved labeling matrices from the entire dataset. The results indicated that the proportions of samples varied depending on the architecture utilized, as demonstrated in Fig. 2. However, when selectors lack adaptability to these architecture changes, altering sample proportions could lead to poor performance results as shown in Fig 5. Notably, as depicted in Fig. 2, for certain parts of the dataset, COO format still outperforms other formats. Specifically, in the E5 platform, COO performed better than all other formats for the HB group in SuitSparse Matrix Collection [4].

3.3 Matrix Normalization and Matrix Feature Extraction

The size of matrices may vary, complicating their use in CNN training and inference. To address this issue, we propose a normalization method that extracts the matrix features and pools them into a fixed size suitable for CNN utilization. This method is based on the observation of the matrix non-zeros distribution features, as described in [2], and the BSR design proposed in [18]. In this part, we first describe the pooling method before detailing the feature extraction method.

Matrix Pooling. Algorithm 2 outlines a pooling-like Algorithm for normalizing a given matrix with a size-adjustable sliding window. Given a matrix, the sliding window slides through matrix A and maps its features to the corresponding pixel in the sample. After normalization, a three-channel sample is obtained, containing the matrix features. This sample is further processed to add the necessary architecture settings.

Algorithm 2. Matrix pooling with sliding window

Input: A : input matrix with $m \times n$ size
val: nonzeros in A
 m_s : input sample row size
 n_s : input sample column size
 w : sliding window with $m_w \times n_w$ and nonzeros density *density*

Output: *Sample*: CNN input sample

- 1: set $m_w = \frac{m}{m_s}$
- 2: set $n_w = \frac{n}{n_s}$
- 3: **for** $i = 0, 1, 2, \dots, \frac{m}{m_w}$ **do**
- 4: **for** $j = 0, 1, 2, \dots, \frac{n}{n_w}$ **do**
- 5: moves the top left corner index of the w to the element $A_{(i*m_w, j*n_w)}$
- 6: $density = \frac{nnz}{n_w \times m_w}$
- 7: **if** $density \geq threshold$ **then**
- 8: assigns matrix features to the position (i, j) of the sample
- 9: **else**
- 10: assigns zero to the position (i, j) of the sample
- 11: **end if**
- 12: **end for**
- 13: **end for**
- 14: return *Sample*

Taking a sparse matrix of size 100×100 as an example, and assuming that the input size of the CNN is 10×10 , the sliding window size according to Algorithm 2 is also 10×10 . With the *stride* shown in Eq. (1), the matrix is divided into 100 parts with pooling, each of which is mapped to a set of sample pixels. If the density of non-zero elements in a given part exceeds the threshold, the corresponding pixel is assigned with the matrix features; otherwise, it remains blank.

$$stride = \frac{m}{m_s} \quad (1)$$

Matrix Feature Extraction. We extract key features from the sparse matrix and store them in the relative dimensions of the sample, as demonstrated in previous work [2]. The extracted features include:

- matrix size: $m * n$ (require normalization)
- matrix density: $\frac{nonzeros}{matrix\ size}$
- nonzeros number of a row which contains most nonzeros: max_{row} (require normalization)

As these features can vary greatly, we normalize them to a range of $[0, 255]$ using Eq. (2). Limiting the feature values to the range of $[0, 255]$ enables a more accurate description of the density distribution of non-zero values covered by the sliding window across all color channels, thereby providing a more refined

description of the non-zero value distribution for the entire sample. If normalization is not used, the significant differences in feature values may have an impact on the training process.

$$\text{normalized feature} = \frac{\text{feature} - \text{min}}{\text{max} - \text{min}} \times 255 \quad (2)$$

The density threshold is an important factor to consider when describing the nonzeros distribution of a sparse matrix. If the threshold is set too low, the detail of nonzeros distribution will not be described accurately, while setting the threshold too high will result in too many empty blocks without features. It is therefore necessary to carefully consider the optimal density threshold to ensure an accurate description of the sample.

3.4 Architecture Feature Extraction

Architecture has been shown to influence SpMV performance and format selection (Fig. 2). To ensure our model can adapt to other architectures without additional training, we incorporate architecture features into extra dimensions of the sample. The resulting sample has six dimensions: the first three dimensions correspond to matrix features, while the remaining three dimensions refer to architecture features.

By concatenating the architecture feature matrix with the matrix feature matrix, we can generate a sample with 6 dimensions. Initially, employing `lscpu` and `mbw` [1], we can retrieve the CPU architecture settings and memory bandwidth. Subsequently, according to Eq. (3), the architecture feature is normalized. Then, three architecture feature matrices are constructed based on corresponding architecture features. Finally, these matrices are stacked onto the sparse matrix feature matrix, resulting in a high-dimensional sample.

Architecture feature dimensions do not pay attention to the sparse matrix-related features. Setting the corresponding dimension to a solid value allows the CNN to focus on the value, eliminating the interference of sparse matrix features, and thereby improving accuracy.

In Sect. 4.4, our experiments revealed three significant architecture features that have an impact on the SpMV speed. Features of architecture we extract are shown below, those features require normalization following Eq. (3):

- The size of L3 cache size: *L3*.
- The basement frequency of CPU: *Frequency*.
- The bandwidth of memory: *Bandwidth*.

$$\text{normalized feature} = \frac{\text{feature}}{\text{max}} \times 255 \quad (3)$$

Normalization methods for matrix features cannot be used for architecture features. As can be observed by examining the hardware architecture settings in Table 1, applying the normalization method given in Eq. (2), a transformation in

the distribution of the sample space can be observed. For instance, the Euclidean distance of the feature vector with Xeon 6248 and i9 9900K decreased from 2.2176 to 1.833 with the inclusion of the Xeon 6242 processor, indicating an overall 17% shrinkage and a corresponding shift in the position of the sample in the sample space. To this end, a scalable normalization approach is presented in Eq. (3), wherein max represents the maximum value of the matrix feature while $feature$ stands for the feature value of the matrix.

Table 1. Architecture features of each platform

Architecture Property	i9 9900K	Xeon 5220	Xeon 6248	Xeon 6242	E5 2640
Basement Frequency	3.6 GHz	2.2 GHz	2. 5 GHz	3.1 GHz	2.6 GHz
Cores	8	18	20	16	6
L3Cache	16 MB	24.75 MB	27.5 MB	22 MB	15MB
Memory	126 GB	119.2 GB	377.6 GB	119.2 GB	119.2 GB
Memory Bandwidth of Node	9137 MiB/s	4353 MiB/s	5024 MiB/s	5583 MiB/s	6069 MiB/s
Launch Date	Q4'18	Q2'19	Q2'19	Q2'19	Q1'12

3.5 Network Design

For traditional image classification tasks, images are typically input into a single-input network, with the variation in the structure of the network mainly in terms of the number of layers, the internal parameters, and the organization form. However, in the case of sparse matrix data, the feature distribution has some distinct differences from traditional image feature distributions, as the information present in each dimension may be different from other dimensions.

In order to facilitate the network’s ability to extract information from each channel, we present MANet illustrated in Fig. 3 as an abridged general view. This network decouples the elements of the samples into six inputs at the beginning. Further, the tensor with six dimensions is fed into the CNN layer after which it is concatenated and fed into a fully-connected layer. By separating the samples at the start and merging the features at the end, this architecture enables the efficient extraction of relevant features while also reducing the complexity of training.

The MANet architecture consists of four CNN layers, with each layer containing six channels. Following each CNN layer, a max pooling layer is connected. The final CNN layer is followed by three fully connected layers that output the prediction results. The output channels for the CNN layers are 96, 256, 256, and 256, respectively. Additionally, the kernel size for each CNN layer is 11×11 , 5×5 , 3×3 , and 1×1 with padding and stride, respectively. The input and output sizes for the fully connected layers are 13824×4096 , 4096×4096 , and 4096×3 .

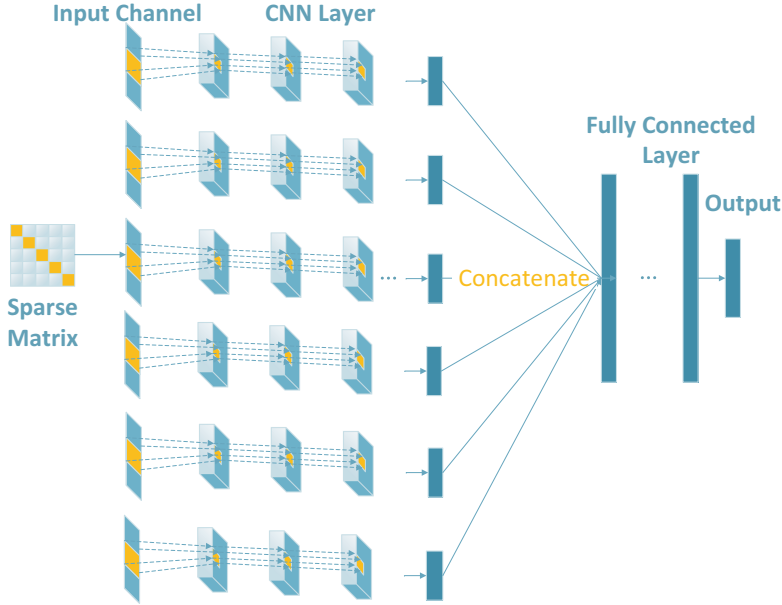


Fig. 3. Network structure of the matrix format selection method with architecture awareness.

4 Evaluation

4.1 Experiment Setting

Hardware Platform. We evaluated MANet on five platforms of varying architectures: i9 9900K (Coffee Lake), Xeon 5220 and 6248 (Cascade Lake), Xeon 6242R (Cascade Lake), and E5 2640 (Broadwell). Nvidia V100 GPU is used for training acceleration. The architecture features of the CPUs are shown in Table 1. These CPU platforms encompass a wide range of usage scenarios, from personal devices (like i9 9900k) to servers (like E5 6242), and span a considerable time period from Q1'12 to Q2'19.

Software Platform. PyTorch 1.7 [11] was utilized for constructing neural networks. To analyze the cache, the PAPI 6.0.0.1 [16] interface was employed to access the hardware counters during measuring the SpMV computation time. The memory bandwidth was then measured by the MBW [1] benchmark. To ensure stable labeling, all SpMV operations were performed with a single core using the SciPy library [17], which was also used to convert the matrix to the corresponding format.

Parameter Setting. The shape of the sample is 256×256 after preprocessing. The density used for normalization is 0.01. For training, the learning rate is

$1e-4$, we use Adam as the optimizer and use cross entropy as the loss function. The model was trained over 100 epochs with a batch size of 64. The weighting of each label’s sample was set to 5.74, 1.41, and 8.56, respectively.

Dataset. This paper utilizes the sparse matrices from the Suit Sparse Matrix Collection [4] as training, testing, and validation data. The same 2150 matrices are preprocessed and used across five CPU platforms for training, testing, and validation. Matrix with size less than 256×256 is excluded resulting in 1702 original matrix for further argumentation. Furthermore, the data set is augmented by chunking into four equal parts as four new matrices, transposing, etc., resulting in 14168 sparse matrices. Matrix with size less than 256×256 is excluded. As indicated in Sect. 3, the feature to be extracted is as follows.

We employ an 80%–20% split for training and testing, respectively, with validation data directly sampled from the Suit Sparse Matrix Collection and excluded from the training and test sets.

Our work mainly focuses on three commonly used formats, COO, CSR, and BSR, as the storage format for sparse matrices. The three storage formats are implemented using the methods in Python Scipy Sparse V1.5.4 [17]. BSR is stored in blocks, and the block size of BSR in this experiment is selected by the heuristic Algorithm in Scipy Sparse library.

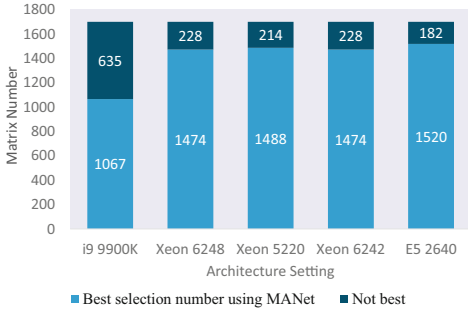
We have attempted to use other formats such as DIA, DOK, and LIL as labels, but we discovered that these formats only cover a mere 4% of the dataset. As a result, for SpMV in CPU, the COO, CSR, and BSR formats are capable of meeting most needs. Therefore, we have chosen to use COO, CSR, and BSR as labels for our dataset.

Due to the wide range of sparse matrices, the dataset used in this paper is more universal and challenging than earlier approaches. As matrices are utilized in various scientific computing applications, their feature distributions vary greatly and exhibit a strong long-tail effect when normalized.

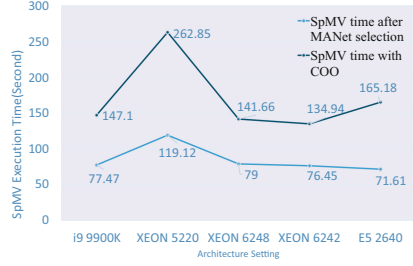
4.2 Speedup

In this section, we use sparse matrix samples from several domains to produce their format prediction results and compare the time performance of SpMV using the corresponding formats of our method and two comparison methods.

In light of the potential for architecture adaptation and release time considerations, we compare the two methods proposed by Pichel et al. [12] and Zhao et al. [19]. The MANet is trained using the data set output from the i9 9900K and Xeon 6248 platforms, while both the Pichel et al. [12] and Zhao et al. [19] methods are trained using the data set output from the i9 9900K. Comparing datasets from two different architectures using traditional methods is not feasible since the labels for the samples may not match. However, MANet is designed to extract architecture-specific features, which eliminates this issue. We compare the results of these two methods on five different platforms and measure the number of best formats each method selects. The final results are shown



(a) Best selection number of MANet.



(b) SpMV time using MANet compared with COO format.

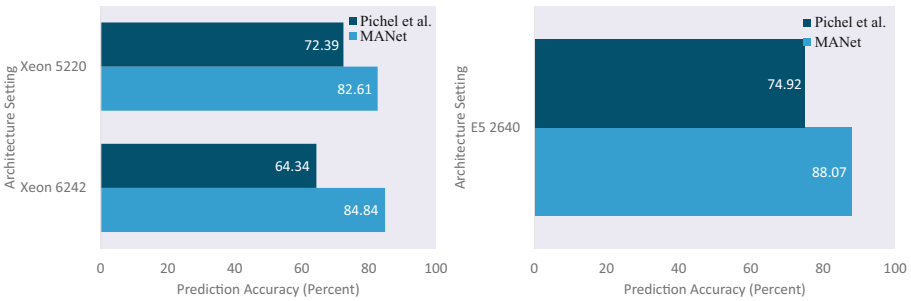
Fig. 4. SpMV speedup evaluation with MANet selector.

in Fig. 4. Figure 4a demonstrates that MANet is capable of selecting the best format in common scenarios with an accuracy of 89.3%.

When compared to the commonly used COO format, our proposed MANet shows a significant increase in SpMV computation speed of up to 230%, as demonstrated in Fig. 4b. As demonstrated in Sect. 3.2, the COO format remains an appropriate choice for enhancing the speed of SpMV.

4.3 Adaptation and Accuracy

In this section, we conduct an experiment to investigate the adaptation performance of our proposed method, MANet, in both approximate and non-approximate architecture adaptation scenarios. The comparison methods are the same as in the previous section.



(a) Adapt to approximate architecture. (b) Adapt to non-approximate architecture.

Fig. 5. Prediction accuracy when adapting to other architecture.

By utilizing a novel CNN structure and preprocessing technique, we demonstrate that MANet is capable of selecting the sparse matrix format in an environment that has not been previously encountered.

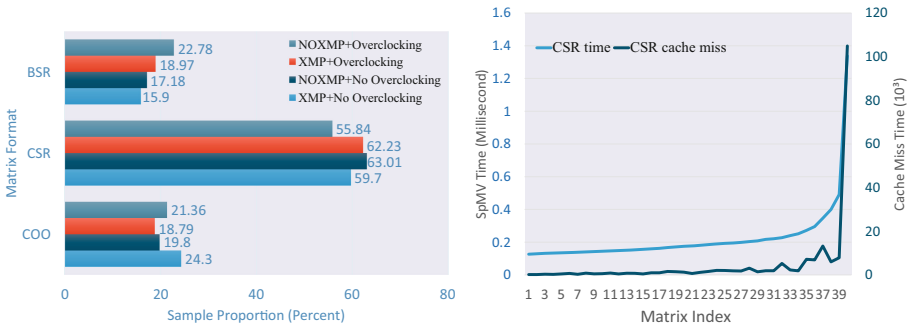
Approximate Adaptation. In approximate architecture adaptation, the test-set generated in Xeon 5220 and Xeon 6242 is used for testing. The architecture of the Xeon 5220 and Xeon 6242 is more similar to that of the Xeon 6248. Thus we choose Xeon 5220 and Xeon 6248 for the approximate architecture adaptation experiment.

The results obtained from the Intel Xeon 6242 and Xeon 5220 platforms indicate that the MANet architecture significantly outperforms the comparison works, with an improvement in prediction accuracy of up to 20% as shown in Fig 5a.

Non-approximate Adaptation. In non-approximate architecture adaptation, the data generated on an Intel Xeon E5 2640 Broadwell architecture platform was used to evaluate the performance of the MANet system. As shown in Fig. 5b, the system achieved a prediction accuracy of over 14%.

4.4 The Influence of Architecture

Architecture setting influence on the sample proportion is investigated in this section. CPU frequency, memory bandwidth, and cache size as architecture parameters are discussed in detail. Results indicate that those parameters have a significant impact on the sample distribution.



(a) The influence of CPU and memory overclocking .

(b) Relationship between cache miss and SpMV performance.

Fig. 6. The architecture setting influence towards SpMV and matrix labeling.

The effect of memory frequency and CPU frequency on the dataset from Intel i9 9900K was analyzed. Memory overclocking (XMP) and CPU overclocking were used to study the sample proportion change. The results of the sample proportion after using memory overclocking (XMP) and CPU overclocking, respectively, are presented in Fig. 6a. With lower memory frequency leading to a lower share of COO format and a greater bias towards CSR and BSR formats. Furthermore,

higher CPU main frequencies have been found to have a greater bias towards BSR formats.

Experimental results demonstrate a direct correlation between the total number of L2 cache misses and the elapsed time for SpMV, as illustrated in Fig. 6b. This relationship is more evident, due to the relatively small size of the L2 cache, which helps to reduce the impact of noise.

It has been demonstrated that there is a direct correlation between the number of cache misses and the SpMV computation time, as illustrated in the above Figure. Utilizing the larger size of the L3 cache, this paper takes advantage of its more stable performance label during labeling, making it an important architecture parameter.

5 Related Work

Recent research has focused on utilizing machine learning methods, such as decision tree and deep neural networks (DNNs), to select the appropriate sparse matrix format for optimal performance of the sparse matrix-vector (SpMV) product. More recently, convolutional neural networks (CNNs) have been employed to address this problem.

Zhao et al. [19] introduced DNN into the task of format selection for sparse matrices. However, the feature extraction method only considers the spatial distribution of non-zeros without architecture features. Pichel et al. [12] propose an approach that directly turns the matrix into a fixed-size image and encodes relative features into different dimensions. The single input CNN used are unable to handle the information of each dimension.

Qiu et al. [14] utilize the features of sparse matrices in GNNs, and use XGBoost [3] to build a flexible model for format selection during runtime. Nonetheless, the decision tree-based approach requires the pre-definition of which feature to use before training.

Due to the rapid development of architecture, the adaptation faces some challenges including feature extraction and network design. To the best of our knowledge, this work is the first that aims to solve the CNN format selector adaptation problem.

6 Conclusion

Previous models for selecting sparse matrices did not consider the influence of architecture on their performance and typically required retraining or fine-tuning when adapting to different platforms. To address this limitation, we propose MANet, a sparse matrix selection model with architectural adaptability. Our approach incorporates architecture-specific data preprocessing and a multi-input neural network, enabling MANet to achieve superior accuracy without the need for retraining or fine-tuning during migration to diverse platforms. Consequently, it enhances the interoperability of sparse matrix selection models across architectures. In our future work, we will focus on adaptation across heterogeneous

computation devices and addressing challenges in unbalanced sparse matrix classification that have not been adequately resolved.

Acknowledgements. We express our thanks to the anonymous reviewers for their insightful comments that improved the quality of the manuscript and Suit Sparse Matrix Collection for the dataset.

References

1. Mbw: Memory bandwidth benchmark (2010). <http://manpages.ubuntu.com/manpages/lucid/man1/mbw.1.html>
2. Chen, D., Fang, J., Chen, S., Xu, C., Wang, Z.: Optimizing sparse matrix-vector multiplications on an armv8-based many-core architecture. *Int. J. Parallel Prog.* **47**(3), 418–432 (2019). <https://doi.org/10.1007/s10766-018-00625-8>
3. Chen, T., et al.: Xgboost: extreme gradient boosting. R package version 0.4-2 **1**(4), 1–4 (2015)
4. Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. *ACM Trans. Math. Softw.* **38**(1), 1–25 (2011)
5. Grossman, M., Thiele, C., Araya-Polo, M., Frank, F., Alpak, F.O., Sarkar, V.: A survey of sparse matrix-vector multiplication performance on large matrices. arXiv preprint [arXiv:1608.00636](https://arxiv.org/abs/1608.00636) (2016)
6. Langr, D., Tvrdik, P.: Evaluation criteria for sparse matrix storage formats. *IEEE Trans. Parall. Distrib. Syst.* **27**(2), 428–440 (2016). <https://doi.org/10.1109/tpds.2015.2401575>, <https://ieeexplore.ieee.org/document/7036061/>
7. Li, M.L., Chen, S., Chen, J.: Adaptive learning: a new decentralized reinforcement learning approach for cooperative multiagent systems. *IEEE Access* **8**, 99404–99421 (2020). <https://doi.org/10.1109/ACCESS.2020.2997899>
8. Muhammed, T., Mehmood, R., Albeshri, A., Katib, I.: Suraa: A novel method and tool for loadbalanced and coalesced spmv computations on gpus. *Appl. Sci.* **9**(5), 947 (2019)
9. Nisa, I., Siegel, C., Rajam, A.S., Vishnu, A., Sadayappan, P.: Effective machine learning based format selection and performance modeling for spmv on gpus. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE. <https://doi.org/10.1109/ipdpsw.2018.00164>, <https://ieeexplore.ieee.org/document/8425531/>
10. Oyarzun, G., Peyrolon, D., Alvarez, C., Martorell, X.: An fpga cached sparse matrix vector product (spmv) for unstructured computational fluid dynamics simulations. arXiv preprint [arXiv:2107.12371](https://arxiv.org/abs/2107.12371) (2021)
11. Paszke, A., et al.: Pytorch: An imperative style, high-performance deep learning library. In: *Advances in NeurIPS 32*, pp. 8024–8035 (2019). <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
12. Pichel, J.C., Pateiro-Lopez, B.: A new approach for sparse matrix classification based on deep learning techniques. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE. <https://doi.org/10.1109/cluster.2018.00017>, <https://ieeexplore.ieee.org/document/8514858/>
13. Pichel, J.C., Pateiro-Lopez, B.: Sparse matrix classification on imbalanced datasets using convolutional neural networks. *IEEE Access* **7**, 82377–82389 (2019). <https://doi.org/10.1109/access.2019.2924060>, <https://ieeexplore.ieee.org/document/8742660/>

14. Qiu, S., You, L., Wang, Z.: Optimizing sparse matrix multiplications for graph neural networks. In: Li, X., Chandrasekaran, S. (eds.) Languages and Compilers for Parallel Computing: 34th International Workshop, LCPC 2021, Newark, DE, USA, October 13–14, 2021, Revised Selected Papers, pp. 101–117. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-99372-6_7
15. Sun, X., Zhang, Y., Wang, T., Zhang, X., Yuan, L., Rao, L.: Optimizing spmv for diagonal sparse matrices on gpu. In: 2011 International Conference on Parallel Processing, pp. 492–501 (2011). <https://doi.org/10.1109/ICPP.2011.53>
16. Terpstra, D., Jagode, H., You, H., Dongarra, J.: Collecting performance data with PAPI-C. In: Müller, M.S., Resch, M.M., Schulz, A., Nagel, W.E. (eds.) Tools for High Performance Computing 2009: Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing, September 2009, ZIH, Dresden, pp. 157–173. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11261-4_11
17. Virtanen, P., et al.: SciPy 1.0 Contributors: SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* **17**, 261–272 (2020). <https://doi.org/10.1038/s41592-019-0686-2>
18. Vuduc, R., Demmel, J.W., Yelick, K.A.: Oski: A library of automatically tuned sparse matrix kernels. In: *Journal of Physics: Conference Series*. vol. 16, p. 071 (2005)
19. Zhao, Y., Li, J., Liao, C., Shen, X.: 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. ACM. <https://doi.org/10.1145/3178487.3178495>, <https://dl.acm.org/doi/pdf/10.1145/3178487.3178495>