MixedNet: Combining Sparse Matrix Image and Features for Accurate Format Selection to Accelerate SpMV

***Abstract*—Sparse Matrix-Vector Multiplication (SpMV) is a crucial kernel function in scientific and engineering computations. Given that diverse sparse matrix formats cater to different matrix structures, choosing an appropriate format can significantly enhance computational efficiency. Consequently, selecting the optimal format has become a prominent research area. However, prior studies have employed simplistic network structures for format selection, coupled with sparse matrix representations that suffer from information loss, failing to leverage neural networks' potential fully.**

This paper introduces MixedNet, a neural network designed for accurate sparse matrix format selection. It utilizes dual-channel input comprising matrix images and matrix features. We propose two innovative matrix image representations to address the challenge of information loss when transforming sparse matrices into image representations.

Our experimental results demonstrate that our MixedNet attains the highest accuracy of 87.55% and the lowest average relative loss (ARL) of 3.09% compared to various other Neural Networks. Furthermore, we analyze the time overhead induced by format selection, revealing the feasibility of format selection in practical applications.

***Index Terms*—Nerual Networks, Algorithm and Application, SpMV, Format selection**

I. Introduction

Sparse matrix-vector multiplication (SpMV) is a crucial kernel function extensively employed in scientific and engineering computing domains, including applications like PageRank and graph analysis [1], [2]. The efficiency of SpMV significantly impacts these applications, with instances where SpMV dominates processing time overhead, such as in solving linear equation systems through the Krylov subspace method and addressing eigenvalue problems.

To alleviate the time overhead associated with SpMV, various sparse matrix formats have been designed to accommodate diverse application scenarios and hardware architectures [3]–[11]. However, new formats are typically tailored for matrices with specific sparse patterns, highlighting the absence of a universal format that ensures optimal SpMV performance across different hardware architectures and matrix characteristics. Consequently, the selection of sparse matrix formats significantly influences SpMV efficiency [3]–[8], [11]–[14].

Research papers [11], [15]–[23], [23]–[25] emphasize the importance of optimal sparse matrix format selection in enhancing SpMV efficiency, as the chosen format significantly impacts cache hit rates and data locality. Consequently, several methods based on machine learning and deep learning approaches have been proposed to address the challenge of predicting the appropriate matrix format.

Matrix format selection using machine learning necessitates careful consideration of matrix features and running platform information. Prior works [13], [16]–[19], [23] have utilized decision trees, SVM, Pairwise-weighted-SVM, and XGBoost to combine hardware platform and matrix feature information. However, selecting matrix features poses challenges due to their vast number and specialization for predicting specific formats. To overcome this, deep learning methods [11], [15], [20], [21], [21]–[25] have been employed to extract matrix features using matrix image representations automatically.

Previous matrix image representations, such as using the 0-1 or density method [21]–[23], lead to memory overflow and subtle information loss in some matrices. Additionally, prior works often use simple networks with only matrix representation or features as input, resulting in decreased accuracy when candidate formats and matrices increase.

This study proposes a novel network, MixedNet, and two novel matrix image representations to realize more accurate format selection to speed up SpMV. We also provide a detailed analysis of the time overhead associated with matrix format selection. Our experimental results demonstrate superior efficiency, and the time overhead analysis underscores the value of format selection. The main contributions of this paper include:

1. Novel Dual-Channel Neural Network: We propose a dual-channel neural network, MixedNet, for format selection. MixedNet achieves an impressive 87.55% accuracy and a 3.09% average relative loss. MixedNet outperforms other neural networks in prior works on a dataset consisting of 2321 matrices collected from SuiteSparse [26].

2. Innovative Matrix Image Representations: We introduce two new matrix image representations, Multihistogram and Position, that capture more distribution information of matrix non-zero elements than previous matrix representations. Our representations reduce the information loss while transforming matrices into image representations and enhance the accuracy of models.

3. Comprehensive Time Overhead Analysis: We analyze the time overhead of format selection, emphasizing the feasibility of speeding up SpMV with format selection in practical applications.

Section II provides an overview of the sparse matrix format and related work. The intricacies of our method are detailed in Section III. Subsequently, Section IV delves into result

analysis, while Section V is dedicated to examining time overhead. Finally, we encapsulate our findings and conclude the entire paper in Section VI.

II. Background

A. Sparse matrix format

In SpMV, sparse matrices are typically represented using various sparse matrix formats. In our study, we focus on eight prominent sparse matrix formats: CSR, COO, DIA, and ELL-R [27], along with innovative, high-performing, and open-source formats such as SpV8 [11], CSR5 [5], and CVR [9]. Additionally, we incorporate MKL-CSR [28], referred to as MKL in subsequent references, provided by Intel's MKL. Figure 1 illustrates these sparse matrix formats.

COO format comprises three arrays, non-zero element array (***vals***), row indexes array of non-zero elements (***rowIdx***), and column indexes array of non-zero elements (***colIdx***).

CSR format also comprises three arrays representing the sparse matrix. ***colIdx*** array stores the columns of each non-zero element, ***vals*** array stores the value of each non-zero element, and ***rowptr*** array is used to mark the boundaries of each row. ***rowptr*** is based on the ***rowIdx*** in the COO format for further compression.

DIA format comprises a two-dimensional array ***AS*** containing in each column the non-zero elements along a diagonal of the matrix and an integer array ***offset*** that determines where each diagonal starts.

ELL-R [27] format is an enhanced version of the ELL format, in which the ELL format generates a vast number of padding elements when the number of non-zero elements in each row differs significantly. ELL-R limits the maximum number of padding elements in each row, thus improving efficiency.

CSR5 [5] translates CSR into ***tile***s to determine the critical operations, with each thread responsible for one column of the tile. Meanwhile, CSR5 [5] distributes the non-zero elements to each thread in a balanced manner and uses a segmented sum to conduct the SpMV computation.

CVR [9] is mainly designed to solve the load balancing problem on X86 CPUs by first distributing the non-zero elements to each thread as evenly as possible in behavioral units and then adding the remaining non-zero elements to threads with fewer non-zero elements in a balanced manner according to ***rec*** and ***tail***.

SpV8 [11] first reorders the rows according to the number of non-zero elements in the rows and then divides the data into ***Panel*** and ***Fragments*** according to the SIMD width to perform SIMD operations and scalar operations respectively.

MKL-CSR [28] is provided by Intel's highly optimized scientific computing library, MKL (Math Kernel Library). MKL provides a CSR-based SpMV kernel that obtains extraordinary performance and is always regarded as the SpMV benchmark kernel.

B. Related work

Machine learning models play a crucial role in solving classification problems. In the context of format selection, the problem can be treated as a classification task [16]–[20]. Matrices are classified based on their features, with each class corresponding to a specific sparse matrix format.

However, extracting meaningful matrix features poses a challenge since the design of features is highly empirical, and specific features are often designed to predict particular formats, which leads to redesigning features when new formats occur.

Deep learning models, extensively explored in image classification [29]–[32], offer a compelling alternative. By transforming the matrix classification problem into an image classification problem, researchers like Zhao et al. [15] utilized Histogram Representation and a 7-layer two-channel CNN for matrix image classification. Pichel et al. [21], [24] convert matrices into binary images fused with matrix features. Some other studies also adopt image representation and employ CNNs for matrix classification [22], [23].

Compared with traditional machine learning models, deep learning models excel in automatically learning matrix features, circumventing potential challenges associated with manual feature extraction and achieving superior accuracy. However, the format selection performance of the neural networks proposed in previous work is poor when the size of the sparse matrix dataset and the number of alternative formats increase.

III. Methodology

To harness the full performance of neural networks in sparse matrix format selection, we present MixedNet along with two innovative sparse matrix representations, Multihistogram and Position. In this section, we first outline the workflow of Sparse Matrix-Vector Multiplication (SpMV) with format selection and then provide a detailed introduction of each step.

A. Workflow Overview

The SpMV with format selection workflow is illustrated in Fig. 2. SpMV with our method initially necessitates an input sparse matrix in CSR format, widely recognized as the most commonly used format. Subsequently, we extract matrix features and transform the matrix into an image representation. Then, the matrix features and image representation are fed into the pre-trained MixedNet for prediction, yielding the optimal format. Finally, the matrix is transformed into the predicted optimal format to perform SpMV accordingly.

B. Sparse Matrix Representations

To address the information loss when transforming sparse matrices into specific representations, we propose two novel image representations, Multihistogram and Position Representation, aiming to exploit the sparse pattern information of matrices better. Algorithm 1 and Algorithm 2 provide details on generating the Multihistogram and Position Representations of a matrix. Figure 3 presents an illustrative example of these image generation methods.

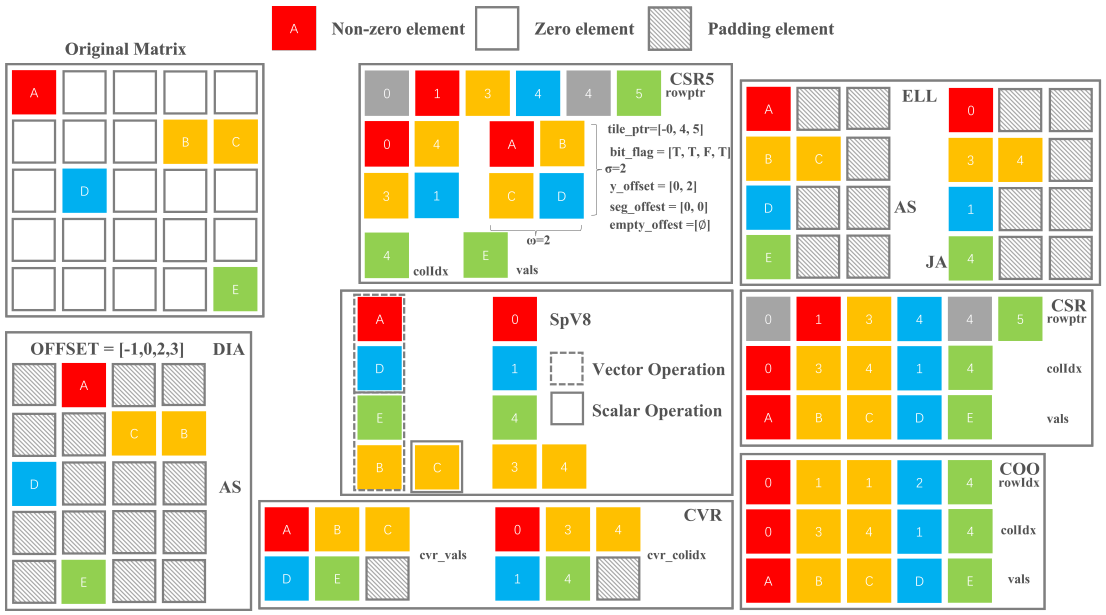

Fig. 1. Description of the matrix format

Algorithm 1 Generating Multihistogram Representation

Input: $img, block\_rows, data, row, col,$
$block\_cols, block\_height, block\_width$

$span$ = max($block\_height$, $block\_width$)
for $i = 1$ to **len**($data$) **do**
  $rowidx$ = $row[i]$
  $colidx$ = $col[i]$
  $shift$ = **abs**($rowidx$-$colidx$) / $span$
  $block\_rowidx$ = $rowidx$ / $block\_height$
  $block\_colidx$ = $colidx$ / $block\_height$
  $img[block\_rowidx][shift][0]$ += 1
  $img[shift][block\_colidx][1]$ += 1
  $img[block\_rowidx][block\_colidx][2]$ += 1
end for
return normalize(img)

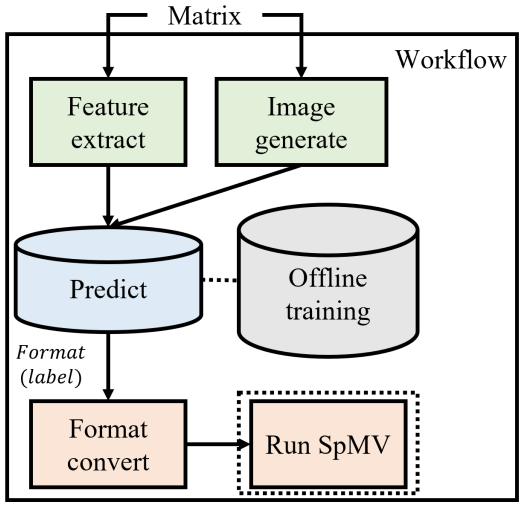

Fig. 2. SpMV with format selection for a given matrix.

Algorithm 2 Generating Position Representation

Input: $row, col, height, width, size, img, background\_size$

$shift$ = $background\_size$ / 2 - $size$ / 2
for $i = 1$ to $size$ **do**
  for $j = 1$ to $size$ **do**
    r = $row[count]$
    c = $col[count]$
    $img[i + shift][j + shift][0]$ = 1
    $img[i + shift][j + shift][1]$ = (r / $height$)
    $img[i + shift][j + shift][2]$ = (c / $width$)
  end for
end for
return normalize(img)

Multihistogram Representation utilizes histogram [15] and density representations to preserve a more detailed distribution of non-zero elements. Taking the Dense Representation matrix in Figure 3 as an example, the matrix is divided into blocks of size 2×2, resulting in a 3×3×3 matrix to represent the original matrix. For a block containing non-zero elements A and B, the density is calculated as 0.5. The first block's row index ($block_rowidx$) and column index ($block_colidx$) are both 0. The shift ($shift$) is calculated as $\lfloor |0-0|/2 \rfloor = 0$. Considering element A with row and column indices both at 0, we update $img[0][0][0]$ and $img[0][0][1]$ by 1, and $img[0][0][2]$ is set to 0.5, for element B (in the same block as A), the shift is again 0, leading to updates in $img[0][0][0]$ and $img[0][0][1]$ by 1. The second block, containing only non-zero element C, has row

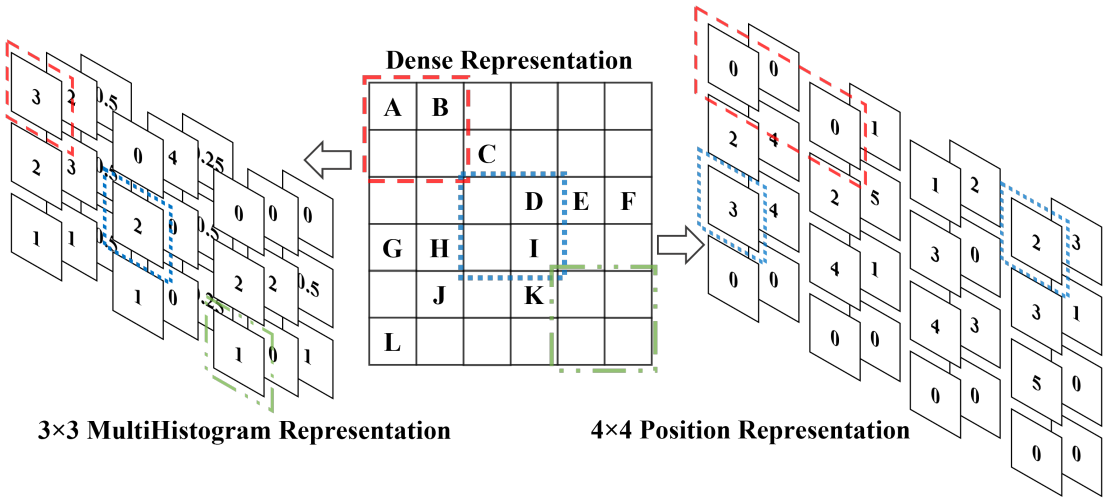

Fig. 3. Multihistogram Representation contains the density and distribution of non-zero elements in the row and column directions. Position Representation detects the sparse patterns [33] of the matrix by sampling.

and column indices 0 and 1. The shift is calculated as $\lfloor |1-2|/2 \rfloor = 0$. Consequently, $img[0][0][0]$ and $img[0][1][1]$ are updated by 1, and $img[0][1][2]$ is set to 0.25. Finally, the values in the Multihistogram Representation img are normalized to the range [0, 255].

Position Representation captures the distribution information of select elements within the sparse matrix, resembling a sampling method. This technique, akin to sparse pattern detection like OSKI [33], focuses on sampling non-zero elements of the matrix. Position Representation stores these elements' relative row and column positions in two channels, providing a straightforward representation of position information. While this approach emphasizes the distribution information of chosen non-zero elements, it inevitably leads to losing distribution information for other elements. However, this sampling-like method is adept at uncovering the sparse pattern of a matrix.

Using the Dense Representation matrix in Figure 3 as an example, where there are 12 non-zero elements, we construct a 4×4×3 matrix to represent the original matrix. The non-zero elements are considered in row-major order, with A being the first element, B the second, C the third, and so forth. For A, we set $img[0][0][0] = 0/4$, $img[0][0][1] = 0/4$, and $img[0][0][2] = 1$. For B, we set $img[0][1][0] = 0/4$, $img[0][1][1] = 1/4$, and $img[0][1][2] = 1$. For L, we set $img[2][3][0] = 5/4$, $img[2][3][1] = 0/4$, and $img[2][3][2] = 1$. When the size of the Position Representation img is fixed at H×W×2, we sample H×W non-zero elements using $shift$ to populate the matrix. If the number of non-zero elements is insufficient to fill the matrix, we pad with zeros. Finally, the values in img are normalized to the range [0, 255].

In addition to image representations, matrix features play a pivotal role in comprehensive matrix representation. In works such as [16]–[20], [25], matrix features are manually extracted based on expertise, describing the matrix as a vector composed of these features. In our experiments, we employ feature vectors and image representations to leverage information derived from the matrix. As demonstrated in Section IV, our hybrid approach contributes to higher model accuracy, showcasing the effectiveness of incorporating feature vectors and image representations in matrix analysis.

C. MixedNet Architecture

The network structure of MixedNet is shown in Table I.

Our MixedNet consists of two channels to utilize the matrix information better. The inference time increases with the depth of the network [29], [30], [32]. Therefore, we use the shallow depth EfficientNet-B0 [31] network as the Image Channel for processing matrix images after trading off the recognition accuracy with the time overhead. The other channel, the Feature Channel, uses a fully connected network to process the matrix's features, adding residual connections to make the network better learn the basis of matrix feature classification [30]. At the end of the network, it fuses the results of the two channels through fully connected layers for prediction. Compared with other networks with a single input of matrix representation, our MixedNet architecture ensures rapid inference time and better classification accuracy.

D. Generating Labeled Matrix Dataset

Before training neural networks, labeling all matrices with their optimal formats is essential. We chose eight sparse matrix formats and curated a dataset comprising 2687 matrices from SuiteSparse [26]. We conducted 3000 SpMV operations for each matrix on the target CPU platform (Table II shows its parameters.), utilizing each of the eight sparse matrix formats. The sparse matrix format resulting in the shortest execution time was assigned as the optimal format. This meticulous process ensured the matrices were appropriately labeled with their most efficient format. To minimize machine errors, we conducted 500 warm-up iterations before SpMV runs. The results of this process, indicating the number of times each format was labeled as the optimal choice, are presented in

TABLE I
MIXEDNET NETWORK ARCHITECTURE

MixedNet Structure (classes=C)							
Image channel (input_size=H×W)				Feature channel (input_size=N)			
Operator	Size	Channels	Layers	Operator		Size	Layers
Conv3×3	H×W	32	1	FC		N×256	1
MBConv1,k3×3	112×112	16	1				
MBConv6,k3×3	112×112	24	2	FC		256×512	1
MBConv6,k5×5	56×56	40	2				
MBConv6,k3×3	28×56	80	3	Residule FC	FcBlock	512×512	8
MBConv6,k5×5	14×14	112	3				
MBConv6,k5×5	14×14	192	4		FcBlock	512×512	8
MBConv1,k3×3	7×7	320	1				
Conv1×1& Pooling	7×7	1280	1	FC		512×C	1
FC	1280×C	-	1				
Concat&FC (2×C)×C							

TABLE II
CPU PARAMETERS INFORMATION

Specification	Intel Xeon Gold 6258R
Total Cores	28
Total Threads	56
Max Turbo Frequency	4.00 GHz
Processor Base Frequency	2.70 GHz
Cache Size	38.5 MB
Max Memory Size	1 TB
Memory Types	DDR4-2933
Maximum Memory Speed	2933 MHz

TABLE III
NUMBER OF EACH LABEL (FORMAT)

Format	Number	Number (Selected)
CSR5	247	244
CVR	492	492
SpV8	13	13
MKL	1619	1411
CSR	47	45
COO	6	5
ELL-R	21	21
DIA	242	104
Total	**2687**	**2321**

Table III. During our experiments, we identified that specific matrices could not perform SpMV with all eight formats. Using specific formats led to issues such as memory leaks, memory overflows, and other runtime problems. Consequently, we retained only the subset of matrices, totaling 2321, that could successfully run in all eight formats. This ensured a consistent and reliable dataset for further analysis and training.

E. MixedNet Training

We train our model on a Nvidia GeForce 3090Ti with 10752 CUDA cores and 24GB of memory. The dataset consists of 2,321 labeled matrices capable of performing SpMV in all eight formats. The dataset is divided into a training set, comprising 80% of the data used for training the neural network, and a test set, containing the remaining 20% for evaluation.

Training the network poses challenges due to imbalances in the dataset. As indicated in Table III, the number of matrices classified as MKL is 1411, while COO is only 5. Various methods are employed, such as constructing new matrices for data augmentation through cropping, panning, and random fusion [20], adding matrix features to images for augmentation [23], and fusing different matrix features in channels with cropping and panning to create new matrices [21], [24].

However, images derived from matrices differ from general images and cannot be augmented using traditional methods because each element in the matrix and its distribution information holds real meaning. In our experiments, we observed that constructing new matrices might cause the model to struggle to learn the features of the original matrix. For instance, the DIA format is optimal for matrices with a natural diagonal structure. Applying translation, cropping, or flipping for data augmentation can disrupt the distribution of non-zero elements and affect the model's ability to discern the original matrix's features.

Given the close relationship between matrix features and the distribution of non-zero elements, altering the distribution through operations like translation, cropping, and flipping can lead the network to learn incorrect information. Moreover, matrices in the original dataset originate from real-world applications, and the realism of newly generated matrices through augmentation methods is not guaranteed. To assess the impact of flipping on accuracy, we compared different networks, as depicted in Table IV. The results indicate a decrease in accuracy for every model when the flipping operation is applied for data augmentation.

To address these concerns, we opted for oversampling data in formats other than MKL instead of generating new matrices.

TABLE IV
ACCURACY CHANGED BY APPLYING FLIP

CNN	Representation	Accuracy (Flip)
MixedNet	Histogram	85.71% (-9.17%)
	Multihistogram	87.55% (-7.21%)
	Position	86.69% (-5.66%)
Late-Merging CNN	Histogram	54.91% (-7.73%)
	Multihistogram	52.03% (-15.07%)
	Position	58.89% (-1.05%)
AlexNet	Histogram	71.56% (-11.80%)
	Multihistogram	69.72% (-14.15%)
	Position	67.98% (-1.85%)

This approach helps alleviate the imbalance in the dataset, allowing the model to better discern differences between various data categories without introducing unrealistic matrices.

IV. EXPERIMENTATION AND EVALUATION

A. Accuracy Comparison

The accuracy of our method is compared with that of previous works in [15]–[21], [24]. The results are presented in Table V. The matrix representation proposed in [21], [24] cannot accommodate all matrices since restoring a binary image of a large matrix can lead to memory overflow. Therefore, we utilized Histogram [15], Multihistogram, and Position Representation for the CNN models in [21], [24] and selected the best-performing one.

TABLE V
ACCURACY COMPARISON

Technique	Format Selection Accuracy
Decision Tree [16]	61.26%
Multi-class SVM [17]	65.80%
Pairwise weighted SVM [19]	54.11%
XGBoost [18]	70.35%
DIESEL [20]	58.57%
Late-Merging CNN [15]	54.91%
AlexNet [21]	71.56%
SpNet [24]	66.84%
EfficientNet [31] (Simplified MixedNet)	82.58%
MixedNet (Ours)	87.55%

For comparison, all the models in [16]–[20] used machine learning models. In our approach, we employed the 15 features presented in Table VI as input, encompassing all the matrix features mentioned in [16]–[20]. As observed in Table V, the accuracy of the machine learning models falls short of that achieved by the deep learning model. This underscores the superior performance of deep learning models in sparse matrix format selection.

Furthermore, the methods in [16], [17] employing decision trees and support vector machines exhibit severe overfitting with nearly 100% accuracy in the training set. Our MixedNet achieved the highest accuracy of 87.55% with our Multihistogram Representation, surpassing the mentioned networks. Zhao et al. [15] use a 6-layer convolutional neural network, while Pichel et al. [21], [24] use AlexNet and a simplified version named SpNet. The method in [21] achieves the second-best accuracy of 71.56%, using our Position Representation. In our experiment, we tested more matrices and formats compared with experiments in other studies [15], [21], [24], better indicating the model's ability to generalize. However, it increases the learning difficulty for the simple network structure in previous work. Also, we did not choose most of the traditional data augmentation methods applied in previous work to ensure the matrix structure's fidelity, so the previous work methods performed poorly. On the contrary, our method achieves better performance due to the more efficient structure and more efficient matrix representation. We also tested EfficientNet-B0 [31] and found that our dual-channel input further improves accuracy.

It is illustrated in Fig. 4 that our Multihistogram Representation and Position Representation improve the accuracy compared to Histogram Representation [15]. Compared with Histogram Representation, our method can represent more matrix information, such as row and column position information and density information, which reduces the information loss so that the model can learn the feature information of the matrix better.

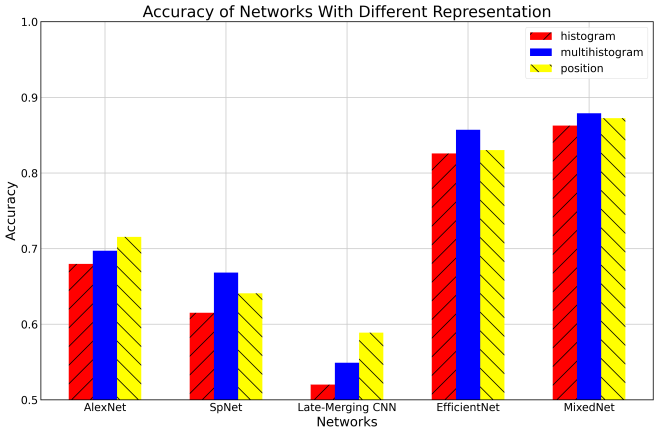

Fig. 4. Accuracy of each model when applying different matrix representations.

In summary, our work demonstrates a significant improvement in the recognition accuracy of sparse matrix format selection compared to previous methods. Utilizing the efficient neural network architecture and advanced matrix representations contributes to achieving superior results.

B. Average Relative Loss

The FLOPS achieved by a sparse matrix format when performing SpMV is a crucial metric to evaluate the performance of a format [34]. Let $F(m, f)$ be the FLOPS obtained by SpMV using format f for matrix m. $F_{max}(m, f_o)$ denotes the maximum FLOPS obtained by SpMV of the matrix using format f_o, i.e., f_o is the optimal format for matrix m. Define the FLOPS average relative loss ARL between using format f_o and f as:

$$ARL = \frac{\sum_{i=1}^{M} F_{max}(m_i,f_o) - F(m_i,f)}{\sum_{i=1}^{N} F_{max}(m_i,f_o)}$$

TABLE VI
PARTIAL FEATURES OF A MATRIX

Feature	Description	Complexity
m	The number of rows.	O(1)
n	The number of columns.	O(1)
nnz	The total number of nonzero values in the matrix.	O(1)
density	The density of the matrix, nnz/(m*n).	O(1)
mean	The mean number of nonzero values per row (npr).	O(1)
sd	The standard deviation of npr in a matrix.	O(2m)
var	The variance of npr in a matrix.	O(1)
maxnnz	The maximum of npr in a matrix.	O(m)
maxavg	The difference between maxnnz and mean.	O(1)
distavg	The mean distance between first and last nonzero values in each row.	O(m)
clusteravg	The mean of the number of distinct consecutive.	O(nnz)
ndiag	The number of matrix diagonals with one or more nonzero values.	O(nnz)
diagfill	The diagonal fill-in ratio for matrices, (m*ndiag)/nnz.	O(1)
fill	The fill-in ratio for matrices, (m*maxnnz)/nnz.	O(1)
cv	The value of sd/mean.	O(1)

Then the performance that can be obtained by executing SpMV on this dataset can be expressed as $performance$:

$$performance = 1 - ARL = 1 - \frac{\sum_{i=1}^{M} G_{max}(m_i, f_o) - G(m_i, f)}{\sum_{i=1}^{N} G_{max}(m_i, f_o)}$$

Each model has varying performance because each model has differential prediction results for the matrix format in the dataset. As shown in Fig. 4 and Fig. 5, the correlation between performance and accuracy is a significant positive correlation, and the performance obtained by a model with high accuracy will be higher on the dataset and vice versa. The performance of different models is shown in Fig. 5. Our MixedNet achieves the best 3.09% ARL with Multihistogram Representation while the second best method [21] achieves 9.61% ARL with Position Representation.

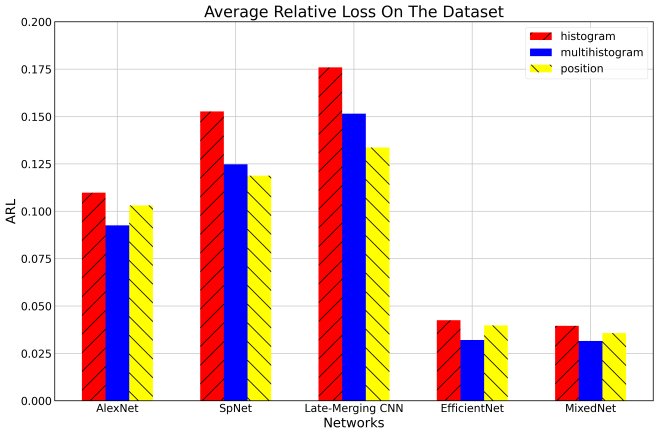

Fig. 5. ARL achieved by each model on the dataset.

Compared to other models, our MixedNet with Multihistogram Representation achieves the lowest average relative loss. Since model performance and accuracy are positively correlated, our model has the highest accuracy and, therefore, has a higher probability of predicting the correct format. Our experimental results also confirm the importance of the accuracy of the model to improve the performance to some extent.

C. Average Speedup Rate

Let $T(m, f)$ be the time overhead of SpMV using format f for matrix m. Suppose the data set contains M matrices. Define the time overhead of performing SpMV in the entire dataset as:

$$TotalOverhead = \sum_{i=1}^{M} T(m_i, f)$$

Executing SpMV using the format predicted by the model can be viewed as employing a novel format. f_p denotes the predicted format of the model. Define the time overhead of using the model predicted format to perform SpMV in the entire dataset as:

$$TotalOverhead = \sum_{i=1}^{M} T(m_i, f_p)$$

Define the $AverageSpeedup$ of using model or specific format f to using CSR format to performing SpMV as:

$$AverageSpeedup = \frac{\sum_{i=1}^{M} T(m_i, f_{CSR})}{\sum_{i=1}^{M} T(m_i, f_p)} \ or \ \frac{\sum_{i=1}^{M} T(m_i, f_{CSR})}{\sum_{i=1}^{M} T(m_i, f)}$$

As shown in Table VII, we measured the $AverageSpeedup$ of performing SpMV with five deep learning models and performing SpMV with eight formats compared to CSR format, respectively. According to the results, our method achieves the highest (1.508x) $AverageSpeedup$ compared to the CSR format.

TABLE VII
AVERAGE SPEEDUP OVER CSR FORMAT

Technique	Type	Speedup
Our method	CNN	1.5087x
AlexNet [21]	CNN	1.4081x
SpNet [24]	CNN	1.3586x
Late-Merging CNN [15]	CNN	1.3288x
CSR5	Single Format	1.1484x
CVR	Single Format	1.1161x
SpV8	Single Format	0.7698x
MKL-CSR	Single Format	1.2838x
COO	Single Format	0.6810x
CSR	Single Format	1.0000x
DIA	Single Format	0.0145x
ELL-R	Single Format	0.6782x

Compared to implementing SpMV with a single format, applying a model to select the format provides additional

alternatives. Since the optimal format varies from matrix to matrix, applying format selection further enhances the performance of SpMV in cases when the default format is not optimal. In summary, using a deep learning model improves the performance of SpMV relative to using a single format.

V. Time Overhead Analysis

Format selection results in additional time overhead, including model inference, feature extraction time, and format conversion time. Before starting the analysis, we introduce $CSRAvgTime$. Let $T(m_i, f)$ be the time overhead of SpMV using format f for matrix m_i. The CSR-based SpMV average time can be denoted as $CSRAvgTime$:

$$CSRAvgTime = \frac{1}{M}\sum_{i=1}^{M} T(m_i, CSR)$$

Here, we normalize all the types of time overhead to $CSRAvgTime$ for convenient analysis in Section V-E.

A. Model Inference Time Overhead

The inference process is performed with the same GPU we used for training our model. Let $T(m, model)$ be the time to infer the format of matrix m through $model$. M represents the number of matrices in the dataset. The Inference time overhead $AvgInfTime$ is expressed as follows:

$$AvgInfTime = \frac{1}{M}\sum_{i=1}^{M} T(m_i, model)$$

The average time required to predict the matrix format for each of the five models is shown in Fig. 6. From Fig. 6, we can see that the methods of [15], [21], and [24] can predict results in a shorter time, and the method in [15] requires a minimum time of 10.41 $CSRAvgTime$ with Pytorch.

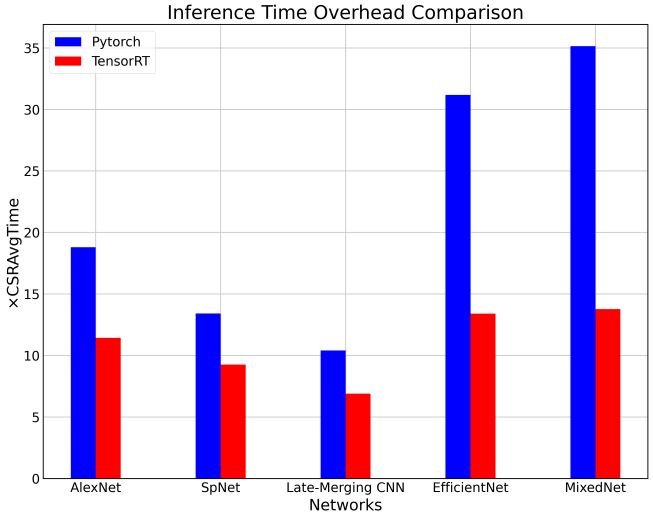

Fig. 6. Inference time overhead of each model using PyTorch and TensorRT

These methods employ relatively simplified neural networks, while our network is rather elaborate, so it needs a longer time to complete the prediction process. However, with the acceleration using TensorRT, the difference in the total time consumed by each model is minor. Our MixedNet's format prediction time is 35.14 $CSRAvgTime$ with Pytorch and 13.78 $CSRAvgTime$ with TensorRT (maintaining the original accuracy). As can be seen, the acceleration effect of using TensorRT is pronounced, and it also indicates that using an approach like TensorRT can effectively cut down the time overhead for inference.

B. Feature Extraction Time Overhead

As shown in Table VI, we can extract the matrix features from the sparse matrix format. The time complexity of computing the features varies from $O(1)$, $O(m)$, $O(2m)$ to $O(nnz)$, which indicates that the time for extracting features increases the larger the matrix is. Mohammed et al. [20] state that different features are available to distinguish different types of matrices and verify by analyzing the classification process that $ndiag$ is the most appropriate feature to evaluate whether the matrix is suitable for the DIA format. However, as the types of formats increase, the number of features required increases, leading to an additional time overhead for extracting features. Table VIII shows the time consumed for extracting various features.

TABLE VIII
Feature Extraction Time Overhead

Selected Features	Time Overhead
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	62.77x
1,2,3,4,5,6,7,8,9,10,14,15	1.77x

As shown in Table VIII, computing $clusteravg$, $ndiag$ demands considerable time since these features demand a time complexity of $O(nnz)$. Therefore, we should weigh whether certain features are worth extracting based on the time required. We also discover that extracting more features is not always advisable since the accuracy archived using features in row 1 is less than row 2 of Table VIII using MixedNet and Multihistogram representation. It indicates that feature extraction is still a delicate task. Extracting some features aggravates the time overhead and provides no improvement in format prediction accuracy.

C. Format Conversion Time Overhead

The default format for the 2687 matrices obtained from SuiteSparse [26] is CSR. Out of these, 2475 matrices can be converted from CSR to 7 other formats, while the remaining matrices cannot satisfy all format conversion operations simultaneously due to data and memory issues. We calculated the average time to convert 2475 matrices to COO, MKL, CSR5, SpV8, ELL-R, and CVR. Additionally, for the DIA format, which is often not recommended when the parameter $ndiag$ is too large, we calculated the average time to convert 367 matrices with $ndiag$ less than 500 to the DIA format. The average time to convert a matrix from CSR format to each of the other seven formats is shown in Table IX. The COO and CSR formats are so similar in structure that the conversion can be completed instantly.

TABLE IX
FORMAT CONVERSION TIME OVERHEAD

Transform To Format	Time Overhead
COO	0x
DIA	1.932x
MKL	7.331x
CSR5	22.247x
SpV8	32.662x
ELL-R	57.09x
CVR	226.273x

The MKL, CSR5, and SpV8 achieve efficient format conversion. The ELL-R format requires a specific amount of 0-element padding, so the format conversion is slightly slower. The CVR format demands rather complex preprocessing operations, and we include the preprocessing operations in the format conversion time.

D. Image Generation Time Overhead

The average time to convert each of the 2687 matrices in the dataset into four types of matrix images are shown in Table X.

TABLE X
IMAGE GENERATION TIME OVERHEAD

Matrix Image Representation	Time Overhead
Density	46.734x
Histogram	60.294x
Multihistogram	74.127x
Position	67.325x

E. Total Time Overhead

We analyze the time data of SpMV based on 2687 matrices on SuiteSparse [26] and count the average time overhead required for each processing part.

For a matrix stored in the CSR format, the time overhead to perform N times SpMV can be expressed as $T_{SpMV}(CSR, N)$. Let the additional time overhead portion of $T_{SpMV}(model, N)$ over $T_{SpMV}(CSR, N)$ as the preprocessing time. When performing SpMV using the predicted format of the neural network, the time overhead $T_{SpMV}(model, N)$ can be expressed as follows:

$$T_{SpMV}(model, N) = T_{feature}(m) + T_{img}(m, img_{type}) + T_{inf}(model, m) + T_{con}(CSR, f_p) + T_{SpMV}(f_p, N)$$

Let $T_{feature}(m)$ (feature extraction time overhead) and $T_{img}(m, img_{type})$ (image generation time overhead) take the values when the model achieves the highest accuracy, 1.77 and 74.127, respectively. Let $T_{con}(CSR, f_p)$ (format conversion time overhead) take the maximum average conversion time of 226.273 for all seven formats. $T_{inf}(model, m)$ (model inference time overhead) takes the model's average format prediction time. $T_{SpMV}(f_p, N)$ is according to the result of the SpMV benchmark run. The variation of the time taken to plot SpMV with the number of executions is shown in Fig. 7. Compared with the direct SpMV using the default format, both the feature representation and image representation methods require a series of preprocessing operations. When N is small, these preprocessing overheads may take up most of the time of the entire SpMV process. In contrast, using the predicted format as N increases will be more efficient. In our experiment, we observe that when N is greater than 1200, the format selection method is better.

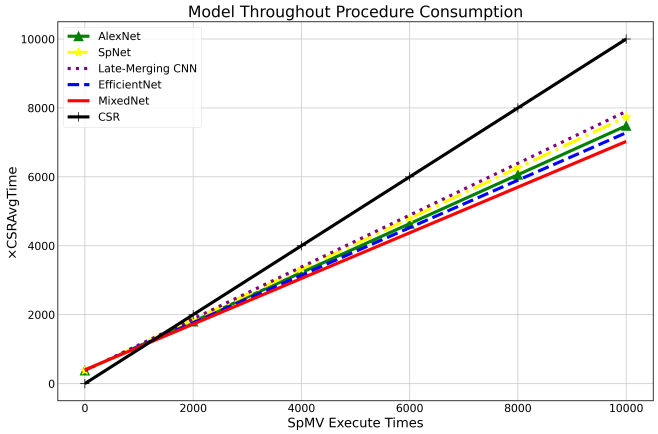

Fig. 7. Time overhead of SpMV with format selection.

SpMV must be performed thousands of times or even more in various domains. Therefore, in applications that need to run iterative SpMVs or significant times of SpMV, applying format selection is better than using the default format only.

VI. CONCLUSION

This paper proposes MixedNet, a novel neural network for efficient sparse matrix format selection. Combining matrix image representations and matrix features as inputs, MixedNet predicts the optimal format with better accuracy. To address the information loss problem, we propose two innovative matrix image representations, Multihistogram, and Position, demonstrating comparable generation times, significantly lower memory overhead, and enhanced matrix information compared to previous matrix representation methods.

Our approach achieves an outstanding accuracy of 87.55%, with an average relative loss of 3.09%, surpassing other works. Furthermore, we conduct a detailed analysis of the time overhead in different steps, providing valuable insights for future studies and practical applications. The results underscore the efficiency of our model compared to other approaches, emphasizing the practical feasibility of format selection in real-world applications.

REFERENCES

[1] M. Ravishankar, R. Dathathri, V. Elango, L.-N. Pouchet, J. Ramanujam, A. Rountev, and P. Sadayappan, "Distributed memory code generation for mixed irregular/regular computations," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2015, pp. 65–75.

[2] A. Venkat, M. Hall, and M. Strout, "Loop and data transformations for sparse matrix code," *ACM SIGPLAN Notices*, vol. 50, no. 6, pp. 521–532, 2015.

[3] B.-Y. Su and K. Keutzer, "clspmv: A cross-platform opencl spmv framework on gpus," in *Proceedings of the 26th ACM international conference on Supercomputing*, 2012, pp. 353–364.

[4] K. Kourtis, V. Karakasis, G. Goumas, and N. Koziris, "Csx: an extended compression format for spmv on shared memory systems," *ACM SIGPLAN Notices*, vol. 46, no. 8, pp. 247–256, 2011.

[5] W. Liu and B. Vinter, "Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 339–350.

[6] Z. Xie, G. Tan, W. Liu, and N. Sun, "Ia-spgemm: An input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 94–105.

[7] R. W. Vuduc and H.-J. Moon, "Fast sparse matrix-vector multiplication by exploiting variable block structure," in *High Performance Computing and Communications: First International Conference, HPCC 2005, Sorrento, Italy, September 21-23, 2005. Proceedings 1*. Springer, 2005, pp. 807–816.

[8] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on gpus," *ACM sigplan notices*, vol. 45, no. 5, pp. 115–126, 2010.

[9] B. Xie, J. Zhan, X. Liu, W. Gao, Z. Jia, X. He, and L. Zhang, "Cvr: Efficient vectorization of spmv on x86 processors," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 149–162.

[10] W. Liu and B. Vinter, "Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors," *Parallel Computing*, vol. 49, pp. 179–193, 2015.

[11] C. Li, T. Xia, W. Zhao, N. Zheng, and P. Ren, "Spv8: Pursuing optimal vectorization and regular computation pattern in spmv," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 661–666.

[12] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the conference on high performance computing networking, storage and analysis*, 2009, pp. 1–11.

[13] J. Li, G. Tan, M. Chen, and N. Sun, "Smat: An input adaptive auto-tuner for sparse matrix-vector multiplication," in *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, 2013, pp. 117–126.

[14] D. Merrill and M. Garland, "Merge-based sparse matrix-vector multiplication (spmv) using the csr storage format," *Acm Sigplan Notices*, vol. 51, no. 8, pp. 1–2, 2016.

[15] 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.

[16] N. Sedaghati, T. Mu, L.-N. Pouchet, S. Parthasarathy, and P. Sadayappan, "Automatic selection of sparse matrix representation on gpus," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 99–108.

[17] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Sparse matrix format selection with multiclass svm for spmv on gpu," in *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, 2016, pp. 496–505.

[18] I. Nisa, C. Siegel, A. S. Rajam, A. Vishnu, and P. Sadayappan, "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, 2018, pp. 1056–1065.

[19] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Bestsf: a sparse meta-format for optimizing spmv on gpu," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 3, pp. 1–27, 2018.

[20] T. Mohammed, A. Albeshri, I. Katib, and R. Mehmood, "Diesel: A novel deep learning-based tool for spmv computations and solving sparse linear equation systems," *The Journal of Supercomputing*, vol. 77, no. 6, pp. 6313–6355, 2021.

[21] 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.

[22] P. Guo and C. Zhang, "Sparse matrix selection for csr-based spmv using deep learning," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. IEEE, 2019, pp. 2097–2101.

[23] H. Cui, S. Hirasawa, H. Kobayashi, and H. Takizawa, "A machine learning-based approach for selecting spmv kernels and matrix storage formats," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 9, pp. 2307–2314, 2018.

[24] J. C. Pichel and B. Pateiro-Lopez, "Sparse matrix classification on imbalanced datasets using convolutional neural networks," *IEEE Access*, vol. 7, pp. 82 377–82 389, 2019.

[25] W. Zhou, Y. Zhao, X. Shen, and W. Chen, "Enabling runtime spmv format selection through an overhead conscious method," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 80–93, 2019.

[26] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[27] F. Vázquez, J.-J. Fernández, and E. M. Garzón, "A new approach for sparse matrix vector product on nvidia gpus," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 8, pp. 815–826, 2011.

[28] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, Y. Wang, E. Wang, Q. Zhang, B. Shen *et al.*, "Intel math kernel library," *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*, pp. 167–188, 2014.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[30] 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.

[31] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

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

[33] R. Vuduc, J. W. Demmel, and K. A. Yelick, "Oski: A library of automatically tuned sparse matrix kernels," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 521.

[34] D. Langr and P. Tvrdik, "Evaluation criteria for sparse matrix storage formats," *IEEE Transactions on parallel and distributed systems*, vol. 27, no. 2, pp. 428–440, 2015.