PAPER

A Machine Learning-Based Approach for Selecting SpMV Kernels and Matrix Storage Formats**

Hang CUI[†*], **Shoichi HIRASAWA**[††], ***Nonmembers***, **Hiroaki KOBAYASHI**[†], ***and*** **Hiroyuki TAKIZAWA**[†,†††a)], ***Members***

SUMMARY Sparse Matrix-Vector multiplication (SpMV) is a computational kernel widely used in many applications. Because of the importance, many different implementations have been proposed to accelerate this computational kernel. The performance characteristics of those SpMV implementations are quite different, and it is basically difficult to select the implementation that has the best performance for a given sparse matrix without performance profiling. One existing approach to the SpMV best-code selection problem is by using manually-predefined features and a machine learning model for the selection. However, it is generally hard to manually define features that can perfectly express the characteristics of the original sparse matrix necessary for the code selection. Besides, some information loss would happen by using this approach. This paper hence presents an effective deep learning mechanism for SpMV code selection best suited for a given sparse matrix. Instead of using manually-predefined features of a sparse matrix, a feature image and a deep learning network are used to map each sparse matrix to the implementation, which is expected to have the best performance, in advance of the execution. The benefits of using the proposed mechanism are discussed by calculating the prediction accuracy and the performance. According to the evaluation, the proposed mechanism can select an optimal or suboptimal implementation for an unseen sparse matrix in the test data set in most cases. These results demonstrate that, by using deep learning, a whole sparse matrix can be used to do the best implementation prediction, and the prediction accuracy achieved by the proposed mechanism is higher than that of using predefined features.

key words: *SpMV, code selection, deep learning, convolutional neural network*

1. Introduction

Sparse Matrix-Vector multiplication (SpMV) is a computational kernel widely used in many applications. A sparse matrix means a large-scale matrix whose elements are almost all zero elements, and non-zero elements appear only in some specific locations of it. Figure 1 shows an example of visualizing a sparse matrix used in a circuit simulation problem. Black points in the figure express the locations of non-zero elements. In scientific computation, operation research, image processing, data mining, structural mechanics and some other fields, systems are naturally sparse, and sparse matrix algorithms are required for analyses [25]. Because of the strong demands for high performance and efficiency, many different approaches have been proposed to accelerate the SpMV computation.

Different platforms, algorithms, and storage formats can be used to build SpMV implementations. The performance characteristics of each SpMV implementation change depending also on the matrix to be processed. For example, Graphics Processing Units (GPUs) and general-purpose Central Processing Units (CPUs) have totally different architectures, and hence different features of a matrix would affect their performances on SpMV computations. Also, a storage format designed for a specific kind of sparse matrices would not work well for other kinds of sparse matrices. Thus, if the best implementation for a given matrix can be predicted in advance of the execution, the computing system is able to achieve a better performance.

One solution of this problem is to use manually-predefined features of an input sparse matrix and a machine learning model such as Support Vector Machine (SVM) [28] to build up a classifier for the prediction. For example, Muralidharan et al. [19] have used three features related to row lengths and other two features to estimate the padding size required for DIA and ELL formats, and used SVM to predict the most appropriate implementation. However, there are some weaknesses of this existing approach. First, it is challenging to manually predefine features that properly

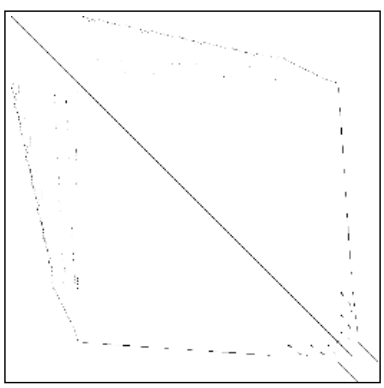


Fig. 1 An example of a visualized sparse matrix, fpga_dcop_01.mtx from [8].

Manuscript received May 29, 2017.
Manuscript revised January 11, 2018.
Manuscript publicized June 13, 2018.

[†]The authors are with Graduate School of Information Sciences, Tohoku University, Sendai-shi, 980–8578 Japan.
[††]The author is with Information Systems Architecture Science Research Division, National Institute of Informatics, Tokyo, 101–8430 Japan.
[†††]The author is with Cyberscience Center, Tohoku University, Sendai-shi, 980–8578 Japan.
[*]Presently with FANUC CORPORATION.

[**]A preliminary version of this paper was presented at ATMG of McSOC16

a) E-mail: takizawa@tohoku.ac.jp (Corresponding author)

DOI: 10.1587/transinf.2017EDP7176

represent characteristics of a matrix and available platforms for the prediction. Second, since a limited number of features are defined and used to express a sparse matrix in the classification process, some information loss would happen. It can influence the classification accuracy of the existing approach.

Recently, deep learning is being used to solve classification problems, and often outperforms the existing approaches [6], [9], [13], [20], [23]. A Convolutional Neural Network (CNN) [16] is a popular deep learning model in the field of image recognition. A CNN can automatically learn not only classification rules but also useful features of images for the classification. This learning capability is called feature learning or representation learning [15]. Therefore, if a sparse matrix is represented as an image and input to a CNN for the classification, the CNN is expected to capture the useful underlying features as well as the classification rules to predict the best SpMV implementation.

The main objective of this paper is to show that, by using deep learning, a whole sparse matrix can be used to attain SpMV code selection, and the classification accuracy can be higher than those of existing approaches with predefined features. To this end, this paper builds up a classifier using a CNN and represents sparse matrices in an image data format for the classifier. By using an appropriate image format to express the characteristics of a sparse matrix, and enlarging the data set, the proposed mechanism is trained and evaluated. The evaluation results demonstrate that useful features and classification rules of SpMV code selection can be extracted by the proposed mechanism.

The remainder of this paper is organized as follows. Section 2 briefly reviews some background knowledge such as SpMV acceleration and deep learning, and also some related studies for SpMV code selection. Section 3 describes the proposed mechanism of using deep learning on SpMV code selection in detail. Section 4 evaluates the proposed mechanism in terms of prediction accuracy and execution time of the selected SpMV implementation. Section 5 gives concluding remarks of this paper, and also mentions the future work.

2. Background and Related Work

There are various approaches to acceleration of SpMV. One approach is to use accelerators [21], [22], [24], [30], [33], and different accelerators will perform differently on the same sparse matrix. Another is to optimize sparse storage formats [10], [12], [17], [25], [26]. Sparse storage formats allocate a continuous memory region for nonzero elements of the sparse matrix, and also for a limited number of zeros in general. The ELL, CSR, HYB and COO formats [4] are sparse storage formats used in this paper. Similar to different processors, different storage formats will also work differently on the same sparse matrix. In [14], [32], Zhang et al. and Langr et al. analyzed the characteristics of different accelerators and different storage formats, respectively. However, both of them did not analyze the situation where the platforms and storage formats are evaluated together.

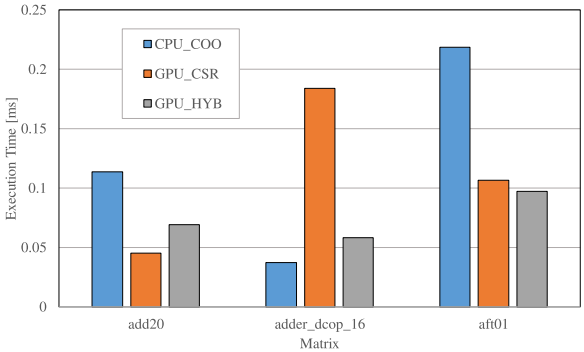


Fig. 2 Performance evaluation results of three SpMV implementations.

A program that has a so-called autotuning ability can automatically optimize itself for a given input to gain a better performance. In [27], Thomas et al. developed a framework to select an adaptive algorithm according to the input size and data type. In [3], some heuristics are used in exploring the feasibility of autotuning SpMV. Also, machine learning approaches [5], [18], [19] are available for SpMV code selection. Besides, Vuduc et al. have proposed a framework called OSKI [29] to tune data structures and code transformations that lead to the fastest implementation of a kernel for a given machine and a given matrix.

In this paper, it is assumed that some different SpMV implementations are available. Figure 2 shows the performance evaluation results of three SpMV implementations on different matrices on a computing system of an Intel Core i7-3770 CPU, an NVIDIA GeForce GT430 GPU and an 8GB main memory. In this figure, CPU_COO, GPU_CSR, and GPU_HYB indicate the CPU implementation for the COO format, the GPU implementation for the CSR format, and the GPU implementation for the HYB format, respectively. These results show that the best combination of an SpMV kernel and a matrix format changes for each matrix. An autotuning system is expected to empirically evaluate the performance of each implementation on the computing system, and map the SpMV computation to its most appropriate implementation.

For the problem of SpMV code selection, some approaches using manually-predefined features have been proposed so far. Muralidharan et al. used five predefined features for code selection [19]. They used SVM as the machine learning algorithm. Their approach can achieve a prediction accuracy of approximately 80% when predicting the best implementation of each of 100 different sparse matrices.

The existing approach that uses manually-predefined features and a machine model has some weaknesses when building a classifier. First, it is challenging to manually predefine some features that can perfectly express the features of any sparse matrix. Second, it is difficult to prevent the parameterization from losing information available to predict the best SpMV implementation for an input sparse matrix.

On the other hand, a deep learning approach can over-

come the weaknesses mentioned above. Since deep learning can extract useful features as well as classification rules automatically from training data, the whole of a sparse matrix can be used for the classification. Deep learning is a kind of machine learning that attempts to model high-level abstraction of data by using multiple processing layers. As with other machine learning algorithms, deep learning can also be treated as a black box and used as a classifier. Recently, deep learning is being widely used in image recognition [13], [20], [23] and some other classification problems such as sentiment classification [9] and natural language processing [6], and often achieves a higher classification accuracy than traditional approaches.

A CNN is a kind of deep learning model that is used in the field of image recognition. In [13], a CNN is used to classify images from ImageNet [1] into 1,000 categories and achieves a higher accuracy than ever. A different input data size, convolution kernel size, pooling kernel size and fully-connected layer size can make different CNN topologies. Different CNN topologies perform differently on the same data set of a classification problem. Therefore, an appropriate topology is usually essential to achieve accurate classification. Finding an appropriate topology is an important stage to gain a good performance by using a CNN. If a sparse matrix can be expressed as an image, a CNN can also be a good classifier for SpMV code selection.

In our preliminary work [7], SpMV code selection was performed as an image classification problem by converting a sparse matrix to an image. However, the classification accuracy was not high because several design options of the proposed approach such as the image format and the CNN model parameters were not well examined yet. Therefore, in this paper, we present a new image format that contains additional information to achieve a higher classification accuracy. Moreover, we propose a systematic way of artificially generating images for training and testing of a CNN model.

3. A Code Selection Mechanism Using Deep Learning

In this section, the proposed mechanism is described in detail. Section 3.1 presents an overview of the proposed SpMV code selection mechanism using a CNN, and shows that there are two technical challenges to use. One is how to translate the problem into an image recognition problem. The other is how to obtain a training data set that is large enough to train a CNN to achieve a high prediction accuracy. Therefore, we first describe how to convert the problem to an image recognition problem in Sect. 3.2, and then discuss how to artificially generate training data effective to improve the prediction accuracy of a CNN in Sect. 3.3.

3.1 Proposed Mechanism

Figure 3 illustrates an overview of the SpMV code selection process using the proposed mechanism. As shown in the figure, the proposed mechanism selects one of the available

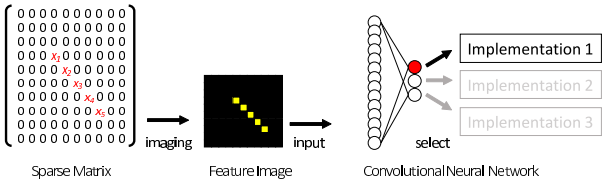


Fig. 3 An overview of an SpMV computing system using the proposed mechanism.

Table 1 Sparse matrix features and best implementation.

Matrix Name	M	N	nnz	Best Implementation
fpga_dcop_13	1220	1220	5892	GPU_CSR
fpga_dcop_14	1220	1220	5892	CPU_COO
dw2048	2048	2048	10114	GPU_CSR
dw4096	8192	8192	41746	GPU_HYB

SpMV implementations in an implementation set, which is expected to complete the SpMV computation in the shortest execution time. As shown in the center part of Fig. 3, the proposed mechanism uses a feature image generated from a sparse matrix, and a CNN for the classification. The input of the CNN is a feature image, and the output of the CNN is a vector whose elements correspond to the available SpMV implementations. That is, the dimension of the output vector is the number of the SpMV implementations, to which sparse matrices are mapped. Each element of the output vector is expected to represent the possibility that its corresponding SpMV implementation is the best implementation for a given sparse matrix. The implementation corresponding to the output vector element with the maximum value is selected as the best implementation by the proposed mechanism.

To the best of our knowledge, there is no existing approach that uses a whole matrix for the selection. On the other hand, in the proposed mechanism, a whole matrix is converted to a feature image, and then a CNN is used to extract underlying matrix features from the image. Namely, a whole matrix is used for the SpMV code selection. Therefore, in the proposed mechanism, it is not needed to predefine the features of a matrix as a number of parameters. This is an important benefit of using the proposed mechanism, because the necessity of predefining features is one severe drawback of the existing approaches.

3.2 Conversion to an Image Recognition Problem

All the predefined features used in [19] can be summarized as the size (*AvgNZPerRow*, *RL_SD*, *MaxDeviation*) of an input sparse matrix and the layout of the non-zero elements (*DIA_Fill-in*, *ELL_Fill-in*) in the sparse matrix. Therefore, the size of a spare matrix and the layout of the non-zero elements in the sparse matrix would basically be important to determine the best implementation for the matrix.

Four sparse matrix examples are shown in Table 1 to discuss the relationship between the sparse matrix features and the best implementation. In the first two examples in Table 1, *fpga_dcop_13* and *fpga_dcop_14* have the same di-

mension and quantity of non-zero elements. However, their layouts of the non-zero elements are different. From these two examples, it is confirmed that a different implementation would perform best on each of the two matrices. These two examples demonstrate that the layout of the non-zero elements can influence the best implementation. In the last two examples in Table 1, although *dw2048* and *dw4096* have the same layout of the non-zero elements, their dimensions are different. Thus, a different implementation should be selected for each of the two matrices. These two examples demonstrate that the dimension of a sparse matrix can influence the best implementation. Therefore, if a CNN is used to predict the best implementation for a sparse matrix, the input image data should be able to contain these two features, the dimension of the original sparse matrix and the layout of the non-zero elements.

An image representing the features of a sparse matrix, called a *feature image* in this paper, is generated as the input of a CNN to select the best implementation for a given sparse matrix. Table 1 clearly indicates that the width and height of a matrix, M and N, are important features for SpMV code selection. On the other hand, for a CNN, the image needs to be of a fixed size. In addition, since the distribution of non-zero elements in a sparse matrix also affects the best SpMV implementation of the matrix, a feature image should be able to express the distribution as well as the matrix size. Moreover, in our preliminary evaluation, it is empirically proved that the number of non-zero elements (nnz) in a sparse matrix is also an important feature of the matrix for SpMV code selection. Accordingly, the feature image of a sparse matrix has to be of a fixed size, and represents the distribution of non-zero elements while keeping three important features of the original matrix, M, N, and nnz.

Figure 4 illustrates the process proposed to generate a 3-channel feature image of 64×64 pixels from a sparse matrix. In order to insert the matrix size information as a kind of feature into the feature image of a fixed size, an intermediate image is first generated. The intermediate image has the same dimension as that of the original sparse matrix. In the intermediate image, every image pixel corresponding to a non-zero element is used to express the three parameterized features, M, N, and nnz. On the other hand, every pixel corresponding to a zero element is set to zero. As a result, the intermediate image represents the distribution of non-zero elements, while keeping the three important features as the pixel color values. The intermediate image is then resized to the target size of 64×64 pixels to generate a feature image.

3.3 Artificial Sparse Matrix Generation

As mentioned in Sect. 3.1, a CNN needs a big training data set to extract useful features and learn classification rules. To increase data to organize big training and test data sets, this paper presents a method to generate artificial sparse matrices from other sparse matrices. Sparse matrices are generally used in real applications, and the distributions of their non-zero elements are structured and meaningful. Therefore, in this paper, artificial matrices are generated by resizing a sparse matrix so as to roughly keep the distribution of the non-zero elements in the sparse matrix.

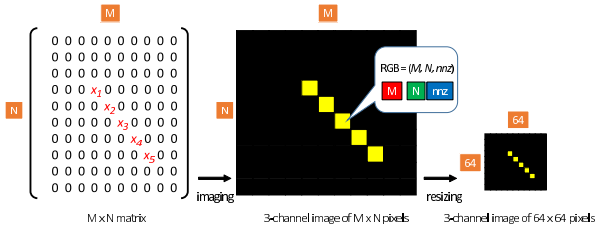


Fig. 4 An overview of the 3-channel feature image generation process.

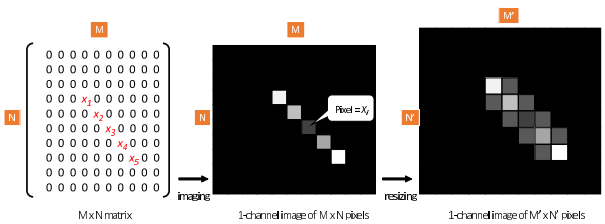


Fig. 5 An overview of the artificial sparse matrix generation process.

Table 2 Best implementations of original and artificial matrices.

Matrix Name	M	N	nnz	Best Implementation
add32	4960	4960	23884	GPU_CSR
add32_1182	1182	1182	1992	CPU_CSR
add32_2797	2797	2797	7949	CPU_COO
add32_7555	7555	7555	55749	GPU_HYB

The process to generate artificial matrices is shown in Fig. 5. The idea behind this process is to generate a new matrix by randomly resizing an existing sparse matrix so that the distribution of non-zero elements in the generated matrix still remains structured at a certain level. The process in Fig. 5 is similar to the feature image generation processes in Fig. 4. However, in the intermediate image, every pixel of a non-zero element does not have M, N, nor nnz. First, a single-channel intermediate image of the original sparse matrix is generated, and every pixel has the value of one matrix element. Then, the intermediate image is resized to a random size. In this way, the distribution of non-zero elements in a new artificial matrix roughly retains that of the original matrix. By using this process, the size of a sparse matrix is changed, while the non-zero element distribution is kept. The best implementation of a new artificial matrix can be different from that of the original one, resulting in increasing the number and variety of data available for training.

In this work, original matrices are randomly classified into training and test data sets, and artificial matrices are generated from only matrices in the training data set. Table 2 shows an example of one original sparse matrix and three artificial sparse matrices generated from it. The name of a new artificial sparse matrix is decided according to the width of the new size. It is shown that, by changing the size of

a sparse matrix, a different implementation works best for the resized one. As a result, the data set is enlarged, and a CNN can use more data to extract useful features and learn classification rules for SpMV code selection.

A pair of a matrix and its best implementation is called an *execution record*. The best implementation of each sparse matrix is obtained by executing every available implementation for the matrix and measuring the execution time. Using execution records in the training data set, a CNN is trained so that it can predict the best implementation of a sparse matrix that is unseen during the training process. Execution records in the test data set are used to evaluate the classification accuracy of the trained CNN.

4. Evaluation and Discussions

This section discusses the performance of the proposed mechanism in terms of two evaluation metrics. One is the classification accuracy, and the other is the expected performance gain by using the proposed approach.

In the evaluation, original matrices are selected from a collection of sparse matrices from a wide variety of scientific and engineering applications, UFSMC [8], and then randomly split into training and test data sets. Then, artificial matrices are generated from matrices in the training data set, as described in Sect. 3.3, i.e., the test data set does not include any artificial matrices to discuss the classification accuracy for real sparse matrices. After that, a training process is executed by monitoring the training and test accuracies and data losses.

4.1 Evaluation Setup

This paper uses a deep learning framework, Caffe [31], for training and testing. Each element of the output vector corresponds to one SpMV implementation, and the CNN is trained so that the element of the best implementation becomes the largest. A basic CNN model called LeNet [16] is used as the baseline topology, its parameters are adjusted in a try-and-error manner, and then the best one among the examined topologies is selected for the evaluation. Figure 6 shows the CNN topology used to predict the best SpMV implementation of each feature image.

In this paper, three different SpMV implementations are implemented for both GPU and CPU platforms. The execution time of each SpMV implementation for a sparse matrix is recorded to obtain the best implementation for the matrix. The performance evaluation of this mechanism is executed on a system with an Intel Core i7-3770 CPU, an NVIDIA GeForce GT430 GPU and an 8GB main memory.

First, sparse matrices whose row and column are both between 100 and 20,000 are selected from UFSMC. The number of matrices selected from UFSMC is 877. Then, 207 matrices are used for testing, and the other 670 matrices are used for training. As a result of artificial matrix generation from the 670 training data, the number of training data is increased to 4,811. Then, the execution record of every matrix is generated for training and testing.

The `cvResize` function of OpenCV [2] is used with the `CV_INTER_LINEAR` mode for the feature image generation process and artificial matrix generation process. In the feature image generation process, M, N, and nnz are normalized so that their values fall in the range from 0 to 1, by considering their maximum and minimum values.

Although the matrices in UFSMC are stored in the COO format, an SpMV implementation might assume another format. In such a case, the matrix needs to be converted to the assumed format before SpMV computation in practice. However, this paper focuses only on selecting the best SpMV implementation for a given matrix, and hence does not discuss the conversion time.

4.2 Training Process

The training process is monitored in terms of training and test data accuracies, and also of training and test data losses. The loss function used in this paper is *softmax loss* [13]. The changes of training and test data accuracies, as well as training and test data losses, are shown in Fig. 7. This figure shows that the test data accuracy increases with the training data accuracy, as the training proceeds. This means that,

Fig. 7 Changes of the training and test data accuracies and data losses.

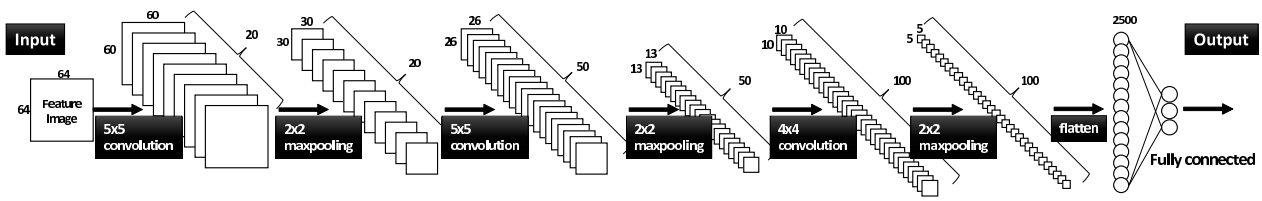


Fig. 6 CNN topology for SpMV code selection.

during the training process, the CNN discovers useful features of images, and thereby learns the underlying classification rules for SpMV code selection.

In Fig. 7, the test data accuracy increases quickly with the training data accuracy until the 100-th epoch, i.e., until every training data is presented to the CNN model 100 times. After the 100-th epoch, the test data accuracy slowly increases, and it reaches its maximum value at the 900-th epoch. The training and test data losses decrease drastically at the initial stage of the training process. Then, the test data loss becomes slightly increasing, while the training data loss still keeps gradually decreasing. It can be observed that a smaller training/test data loss does not necessarily mean a better test data accuracy. Since the best test data accuracy is achieved after the 900-th epoch of this process, the model obtained at the 1000-th epoch is used to discuss the benefits of the proposed mechanism in Sect. 4.3.

4.3 Performance of the Proposed Mechanism

4.3.1 Prediction Accuracy

During the training process, the accuracy of exactly selecting the best SpMV implementation reaches 85.1%. In order to discuss its stability, the training and test data are reorganized three times, and the same training process is executed using the CNN in Fig. 6. The classification accuracies are 84.61% and 84.13%. Therefore, the classification accuracy of the proposed mechanism is stable. Even the worst-case accuracy is higher than that of the existing approach presented in [19]. Thus, the proposed mechanism outperforms the existing approach in terms of classification accuracy.

4.3.2 Expected Performance Gain

Figure 8 shows the performance difference in execution time among the best implementation, the worst implementation, and the implementation selected by the proposed mechanism. The matrices in the test set are ordered in ascending order according to the execution time of the best implementation. The vertical axis is logarithmic. These results show that, even if the mechanism cannot select the best implementation, the performance of the selected implementation is quite close to that of the best one in most cases.

For performance comparison among different SpMV code selection mechanisms, this paper uses the geometric mean of normalized performance [11] defined by

$$GM = \left(\prod_{i=1}^{N} Best_i / Selected_i \right)^{1/N}, \quad (1)$$

where N is the total number of test data, and $Best_i$ and $Selected_i$ are the execution times of the best implementation and the selected implementation, respectively, for the i-th test matrix.

To discuss the performance gain of the proposed mechanism, five mechanisms are used in the following evaluation for comparison. The system uses five kinds of mechanisms to select the implementations in the implementation set. One is the perfect mechanism that can always predict the best implementation for each test matrix. During the execution period, every test matrix is mapped to its best implementation, and the system performance of the perfect mechanism is hence ideal. The second one is the proposed mechanism that uses a feature image mentioned in Sect. 3, and every test matrix is converted to a feature image for the prediction of its best implementation. In the prediction process, a feature image and a CNN model trained in advance are used. In the proposed mechanism, the implementation that is considered by the proposed mechanism to have the highest possibility is selected for each sparse matrix. The remaining three are simple selection mechanisms that will keep using a particular implementation in the implementation set.

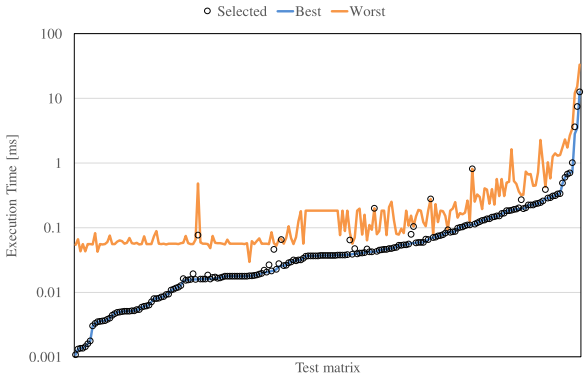


Fig. 8 Performance difference between the best and selected implementations.

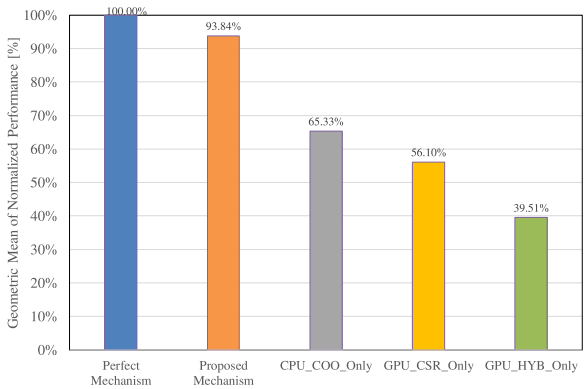


Fig. 9 The geometric mean of normalized performance of five different mechanisms.

Figure 9 shows the geometric means of normalized performances for the perfect mechanism, the proposed mechanism, and three implementations (GPU_CSR, CPU_COO and GPU_HYB). These results indicate that the expected performance of the proposed mechanism is 93.84% of that of the perfect mechanism. Thus, the performance of the proposed mechanism is close to the ideal one. Besides, it is obvious that the performance of the proposed mechanism

is expected to be much better than those of the other three mechanisms that are achieved when no selection mechanism is used. Accordingly, these results indicate that the proposed mechanism is able to improve the expected performance of the SpMV computing system. This means that underlying features of a sparse matrix for code selection are certainly discovered by deep learning, and the proposed mechanism has a potential to predict an appropriate implementation without predefined matrix features for the prediction.

In Fig. 8, it is observed that the proposed mechanism sometimes fails in predicting the best implementation of a middle-size matrix. A simple reason is that it is hard for the proposed mechanism to correctly predict the best implementation if the difference in performance between the best and worst implementations is small. If the matrix size is extremely small or large, the performance difference tends to be large. Therefore, most of the matrices mis-predicted by the proposed mechanisms are middle-size matrices.

5. Conclusions and Future Work

This paper has proposed a mechanism that uses a deep learning approach to solve the SpMV code selection problem. In the proposed mechanism, a sparse matrix is expressed by a feature image, and a deep leaning model is used to do the classification of the best implementation. The evaluation results show that a whole sparse matrix can be used to do the SpMV code selection, and the classification accuracy can be higher than those of existing approaches with predefined features.

This work has demonstrated that, only from the feature image of an input matrix, a deep leaning model can predict the best SpMV implementation among available ones. Hence, only little information about the original matrix size and a rough sketch of the non-zero element distribution is needed for the prediction. This is an important contribution of this work that had not been reported so far.

The general conclusion of this paper indicates the possibility of using deep learning for some other performance tuning problems. The combination of the SpMV kernels and the deep learning model of this paper is just one particular example in code selection problems. The proposed approach can be used for other SpMV implementations and matrix formats. Moreover, some other appropriate deep learning models can be used to solve other code selection problems. This will be further discussed in our future work.

Acknowledgments

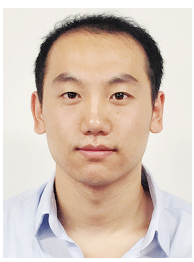
The authors would like to thank Associate Prof. Egawa and Prof. Okatani of Tohoku University for their meaningful discussions.

This research is partially supported by JST CREST "An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Heterogeneous Systems," Grant-in-Aid for Scientific Research (B) 16H02822, and Grant-in-Aid for Challenging Exploratory Research 15K12033.

References

[1] http://image-net.org/, 2014.
[2] http://opencv.org/, 2016.
[3] W. Abu-Sufah and A.A. Karim, "Auto-tuning of sparse matrix-vector multiplication on graphics processors," Supercomputing, Lecture Notes in Computer Science, vol.7905, pp.151–164, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
[4] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," Proc. Conference on High Performance Computing Networking, Storage and Analysis, Article No. 18, Nov. 2009.
[5] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, and D. Keyes, "Application of machine learning in selecting sparse linear solvers," International Journal of High Performance Computing Applications, 2006 (submitted).
[6] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural network with multitask learning," Proc. 25th International Conference on Machine Learning, pp.160–167, 2008.
[7] H. Cui, S. Hirasawa, H. Takizawa, and H. Kobayashi, "A code selection mechanism using deep learning," 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp.385–392, 2016.
[8] T.A. Davis and Y. Hu, "The university of Florida sparse matrix collection," ACM Trans. Mathematical Software, vol.38, no.1, Article No. 1, Nov. 2011.
[9] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," Proc. 28th International Conference on Machine Learning, pp.513–520, 2011.
[10] D. Grewe and A. Lokhmotov, "Automatically generating and tuning GPU code for sparse matrix-vector multiplication from a high-level representation," Proc. Fourth Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-4, Article No. 12, 2011.
[11] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results," Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15), Article No. 73, 2015.
[12] K. Kourtis, V. Karakasis, G. Goumas, and N. Koziris, "CSX: An extended compression format for SpMV on shared memory systems," Proc. 16th ACM Symposium on Principles and Practice of Parallel Programming, pp.247–256, 2011.
[13] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet classification with deep convolutional neural networks," Neural Information Processing Systems, pp.1097–1105, Dec. 2012.
[14] D. Langr and P. Tvrdík, "Evaluation criteria for sparse matrix storage formats," IEEE Trans. Parallel Distrib. Syst., vol.27, no.2, pp.428–440, 2016.
[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol.521, no.7553, pp.436–444, 2015.
[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol.86, pp.2278–2324, Nov. 1998.
[17] S. Lee and R. Eigenmann, "Adaptive runtime tuning of parallel sparse matrix-vector multiplication on distributed memory systems," Proc. 22nd Annual International Conference on Supercomputing, pp.195–204, 2008.
[18] J. Li, G. Tian, M. Chen, and N. Sun, "SMAT: An input adaptive auto-tuner for sparse matrix-vector multiplication," Proc. 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp.117–126, June 2013.
[19] S. Muralidharan, M. Shantharam, M. Hall, M. Garland, and B. Catanzaro, "Nitro: A framework for adaptive code variant tuning,"

2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp.19–23, May 2014.
[20] M. Ozeki and T. Okatani, "Understanding convolutional neural networks in terms of category-level attributes," Computer Vision – ACCV 2014, Lecture Notes in Computer Science, vol.9004, pp.362–375, Springer International Publishing, Cham, 2015.
[21] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Technical Report, Stanford InfoLab, Nov. 1999.
[22] N. Reddy, R. Prakash, and R.M. Reddy, "New sparse matrix storage format to improve the performance of total SpMV time," Scalable Computing: Practice and Experience, vol.13, no.2, pp.159–171, 2012.
[23] K. Sakurada and T. Okatani, "Change detection from a street image pair using CNN features and superpixel segmentation," BMVC 2015, pp.61.1–61.12, Sept. 2015.
[24] Y. Shan, T. Wu, Y. Wang, B. Wang, Z. Wang, N. Xu, and H. Yang, "FPGA and GPU implementation of large scale SpMV," 2010 IEEE 8th Symposium on Application Specific Processors, pp.64–70, June 2010.
[25] B.-Y. Su and K. Keutzer, "clSpMV: A cross-platform OpenCL SpMV framework on GPUs," Proc. 26th ACM International Conference on Supercomputing, pp.353–364, June 2012.
[26] X. Sun, Y. Zhang, T. Wang, G. Long, X. Zhang, and Y. Li, "CRSD: Application specific auto-tuning of SpMV for diagonal sparse matrices," Euro-Par 2011 Parallel Processing, Lecture Notes in Computer Science, vol.6853, pp.316–327, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
[27] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N.M. Amato, and L. Rauchwerger, "A framework for adaptive algorithm selection in STAPL," Proc. Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp.277–288, 2005.
[28] V.N. Vapnik, Statistical Learning Theory, Wiley-Interscience, Sept. 1998.
[29] R. Vuduc, J.W. Demmel, and K.A. Yelick, "OSKI: A library of automatically tuned sparse matrix kernels," Journal of Physics: Conference Series, vol.16, no.1, pp.521–530, 2005.
[30] T. Wu, B. Wang, Y. Shan, F. Yan, Y. Wang, and N. Xu, "Efficient PageRank and SpMV computation on AMD GPUs," 39th International Conference on Parallel Processing, pp.81–89, 2010.
[31] J. Yangqing, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," Proc. 22nd ACM International Conference on Multimedia, pp.675–678, Nov. 2014.
[32] Y. Zhang, Y.H. Shalabi, R. Jain, K.K. Nagar, and J.D. Bakos, "FPGA vs. GPU for sparse matrix vector multiply," International Conference on Field-Programmable Technology, pp.255–262, 2009.
[33] D. Zou, Y. Dou, S. Guo, and S. Ni, "High performance sprase matrix-vector multiplication on FPGA," IEICE Electronics Express, vol.10, no.17, 20130529, 2013.



Hang Cui received the B.E. Degree from Dalian University of Technology in 2014, and the M.S. Degree in Information Science from Tohoku University in 2017. His research interests include HPC systems, machine learning and big data. He is presently with FANUC CORPORATION.

Shoichi Hirasawa is currently a researcher of National Institute of Informatics. His research interests include high-performance computing systems and programming languages processing. He received the B.S. Degree in School of Science of The University of Tokyo, and the M.S. Degree in Graduate School of Information Science and Technology of The University of Tokyo in 2000 and 2002 respectively. He is a member of IEEE CS, ACM and IPSJ.

Hiroaki Kobayashi is currently a Professor and Computer and Mathematical Science Department Chair of Graduate School of Information Sciences, Tohoku University. His research interests include high-performance/low-power computer architectures and memory subsystems, and their HPC applications. He received the B.E. Degree in Communication Engineering, and the M.E. and D.E. Degrees in Information Engineering from Tohoku University. He is a senior member of IEEE CS, and a member of ACM and IPSJ. He is also an associate member of Science Council of Japan.

Hiroyuki Takizawa is currently a professor of Cyberscience Center and Graduate School of Information Sciences, Tohoku University. His research interests include high-performance computing systems and their applications. He received the B.E. Degree in Mechanical Engineering, and the M.S. and Ph.D. Degrees in Information Sciences from Tohoku University in 1995, 1997 and 1999, respectively. He is a member of IEEE CS, ACM SIGHPC, and IPSJ.