A Machine Learning-Based Model for Automated Selection of Task Granularity in SpMV Operations

Cong Xing SY2306336
Beijing University of Aeronautics and Astronautics
37 Xueyuan Road, Haidian District, Beijing City
congxing@buaa.edu.cn

Abstract

In this study, we conducted an in-depth discussion on the task granularity selection for Sparse Matrix-Vector Multiplication(SpMV kernel).Initially, we thoroughly analyzed the specific impact of different task granularities on performance in SpMV operations. Experimental observations revealed that varying task granularities significantly affect the computational performance of SpMV when processing certain sparse matrices. Furthermore, the research indicated that there is no universal task granularity setting that can adapt to the optimal performance needs of all SpMV kernels.

Based on this finding, we proposed a supervised learning-based method to select the optimal task granularity for SpMV. The training process of this model involved extracting features from sparse matrices to describe the characteristics of the input datasets. Additionally, we utilized optimal task granularity data from thousands of SpMV kernels executed on supercomputers for the model's supervised learning. This process ultimately produced a highly accurate prediction model.

The experimental results showed that, compared to traditional fixed task assignment methods, using our model to choose the appropriate task granularity for a specific sparse matrix can achieve a performance improvement of about 20%. This accomplishment not only demonstrates the potential of machine learning in optimizing high-performance computing tasks but also provides a new perspective for performance optimization of SpMV kernels.

1. Introduction

In the fields of scientific computing and artificial intelligence, the multiplication of sparse matrices with dense vectors (SpMV) is one of the core algorithms and a major consumer of software computation time. Unlike common dense matrices, most elements in sparse matrices used in practical applications are zero. Treating sparse matrices with the storage and computational methods of dense matrices would result in additional storage and computational burdens. This is because the numerous zero elements in sparse matrices do not affect the computational result, but storing and computing these zero elements brings unnecessary overhead, thereby reducing computational efficiency. To address this, previous researchers developed compressed storage methods based on the sparsity characteristics of matrices, storing only non-zero elements to avoid processing zero elements. This method effectively reduces the storage and access costs of matrices. Common sparse matrix storage formats include COO (Coordinate List), CSR (Compressed Sparse Row), ELL (ELLPACK), and HYB (Hybrid Format, i.e., ELL+COO) [?]. Although effective, the sparsity structure of the matrix (i.e., the distribution of non-zero elements) and the adopted storage format significantly influence the computational performance of SpMV.

In the field of scientific and engineering computation, optimizing the performance of sparse matrix operations remains a highly challenging task due to the uneven distribution of non-zero elements in sparse matrices and the complexity of the computer system's storage hierarchy. On the one hand, we can introduce new sparse matrix storage formats and corresponding SpMV computational algorithms to improve the layout of non-zero elements. This strategy aims to fully utilize the processor's cache and wide vector operation units to achieve a balance between upper-layer sparse matrix characteristics and lower-layer hardware architecture features. On the other hand, employing parallelization techniques to divide and distribute the computational tasks of sparse matrices across parallel computing systems is also an effective method. However, either approach introduces numerous configuration parameters, creating a vast optimization space. In this large space, using brute force to find the optimal configuration parameters is impractical. Therefore, developing automated performance optimization methods for sparse matrix operations to determine the best configuration parameters is of crucial importance. This not only enhances computational efficiency but also better adapts to various computing environments, thereby advancing scientific and engineering computation.

This study focuses on exploring how different task distribution strategies affect performance when performing sparse matrix-vector multiplication (SpMV) in parallel using MPI on multi-core, multi-node high-performance computers. We observed that task distribution strategies significantly impact the operational performance of SpMV, and there is no universally effective strategy that provides the best computational performance for all sparse matrices. Based on this finding, we propose a novel machine learning-based method for optimizing task distribution strategies. This method can train efficient models for specific datasets and target platforms. Through experimental validation, we found that compared to traditional fixed task granularity distribution methods, the task distribution strategy specified by our model can achieve an average performance improvement of about 20%. This result not only demonstrates the effectiveness of machine learning in optimizing task distribution strategies but also provides a new perspective for optimizing SpMV performance on multi-core processors.

The main research work of this article includes the following two aspects:

(1) Investigated the performance impact of different task distribution methods on sparse matrix and dense vector multiplication operations.

(2) Proposed a prediction method for selecting the best task distribution strategy based on a random forest classifier.

2. Research Background and Motivation

2.1. Sparse Matrix and Dense Vector Multiplication Operation

A sparse matrix refers to a matrix where the number of non-zero elements is much smaller than the total number of matrix elements. Typically, we use the sparsity formula $\beta = \frac{nnz}{m \times n}$ to quantify the degree of sparsity of a matrix, where nnz represents the number of non-zero elements in the matrix, and m and n respectively represent the number of rows and columns of the matrix. When the density of a matrix is less than 0.05, it is usually classified as a sparse matrix.

Sparse Matrix and Dense Vector Multiplication

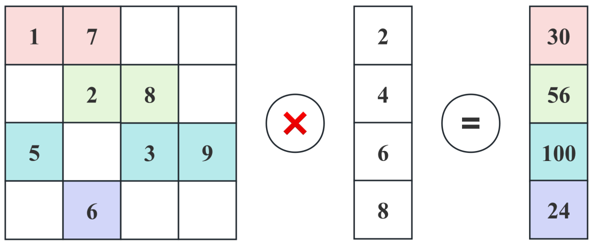


Figure 1. Sample of sparse matrix-vector multiplication

(SpMV) refers to the process of computing the product of a matrix stored in a sparse format and a dense vector, which is a key computational kernel in many scientific and engineering applications. Specifically, the form of SpMV can be represented as $y = Ax$, where the input matrix $A(m \times n)$ is sparse, while the input vector $x(n \times 1)$ and output vector $y(m \times 1)$ are dense. For example, in the following figure (Figure ??), a simple SpMV example is shown, where both m and n are 4, and nnz is 8. This type of operation is particularly suitable for large datasets and high-performance computing scenarios, where efficient processing of sparse matrices is crucial for computation time and resource utilization.

To enhance the computational efficiency of sparse matrices, a compressed storage method is often used to store them. Currently, common sparse matrix storage formats include COO (Coordinate List), CSR (Compressed Sparse Row), BCSR (Block Compressed Sparse Row), ELL (ELLPACK), SELL (Sliced ELLPACK), and CSR5, among others [?]. Among these, the CSR format is widely used due to its storage efficiency and processing speed. In the CSR format, the sparse matrix is stored using three arrays: the row offset array (row_offset), column index array (col_idx), and value array (val). The size of the row offset array is $m + 1$ (where m is the number of rows of the matrix), while the column index and value arrays are both of size nnz, which is the number of non-zero elements in the matrix. As shown in Figure ??, the CSR format reduces the need for storage space by compressing the repeatedly accessed row indices in the row offset array, and it also improves the efficiency of accessing the row indices of non-zero elements. The CSR format is particularly suited to sparse matrices where the distribution of non-zero elements is uneven across rows, as it can effectively utilize storage space while providing fast data access. Algorithm ?? provides an example of an SpMV implementation based on the CSR storage format, demonstrating how to effectively utilize this storage format to optimize the computational perfor-

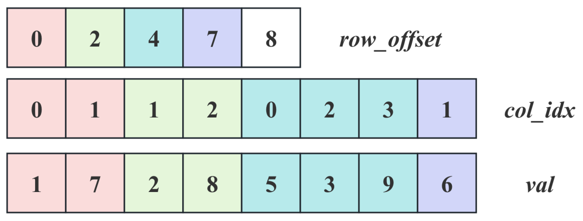


Figure 2. Storage format of CSR

mance of sparse matrices.

Algorithm 1 Sparse Matrix-Vector Multiplication

```
Require: Array csr_val[ ], Array row_offset[ ]
Require: Array col_idx[ ], Vector X
Ensure: Vector Y
  for $i = 0$ to $m - 1$ do
    $sum = 0$
    for $j =$ row_offset[$i$] to row_offset[$i + 1$] do
      $sum = sum +$ csr_val[$j$] $\times$ X[col_idx[$j$]]
    end for
    Y[$i$] $= sum$
  end for
```

2.2. Task Allocation Strategy and Performance Impact

Since there is no dependency between the calculation results of two adjacent rows in Sparse Matrix-Vector Multiplication (SpMV), the cumulative calculation of each row can be considered an independent subtask, making SpMV easily scalable [?]. Assuming there are t processes on each node, under default circumstances, the m rows of the sparse matrix are divided into t subtasks, with each process responsible for one subtask, making the task granularity $g = m/t$. When the task granularity is $g = m/(2t)$, each process is responsible for 2 tasks; when it's $g = m/(4t)$, each process handles 4 tasks; and so forth, such that when the task granularity is $g = m/(Kt)$, each process is responsible for K tasks.

Typically, all subtasks are allocated to processes in a round-robin fashion, so using different task granularities will result in different task allocation methods. Suppose a node has two cores, meaning it can execute two processes simultaneously ($t = 2$), denoted as p_0 and p_1. When $g = 2$, the task allocation method for SpMV as shown in Figure ?? is as illustrated in Figure ??, with each process responsible for the computation of two consecutive rows. When $g = 1$, the task allocation is as shown in Figure ??. In the allocation

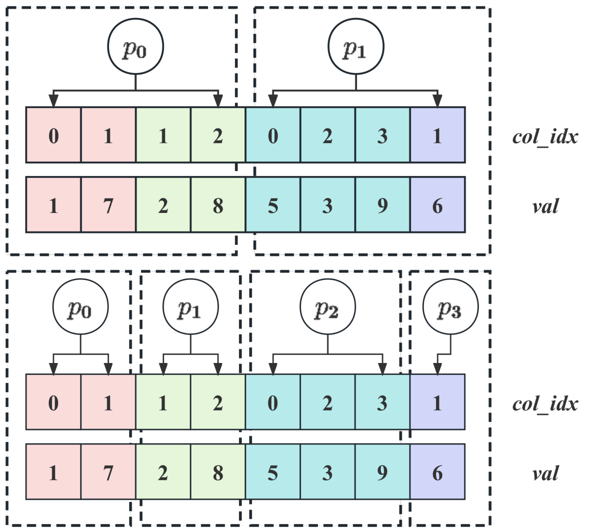


Figure 3. Task allocation of SpMV

method shown in Figure ??, each process is responsible for the computation involving 4 non-zero elements, hence the computational load is balanced. In the allocation method shown in Figure ??, p_0 is responsible for the computation of 5 non-zero elements, while p_1 handles 3 non-zero elements, leading to an imbalance in the load.

As shown in Figure ??, from the University of Florida dataset, 8 different sparse matrix datasets were randomly selected to examine the relationship between 13 different task granularities (1, 2, 4, 6, 8, 10, 12, 16, 24, 32, 48, 64, 128) and the obtained Sparse Matrix-Vector Multiplication (SpMV) performance. The horizontal axis represents the number of processes executed in parallel, and the vertical axis represents the performance of SpMV (in GFlops) under different task granularities, i.e., with different numbers of parallel processing processes. It can be seen that the optimal task granularity for achieving the best performance varies across different sparse matrix datasets. For instance, with a task granularity of 16, SpMV achieves its best performance on the matrix dataset Ip_ken_13, but its performance is relatively poor on other matrix datasets. When the task granularity is 4, SpMV performs best on the matrix dataset Ishp_577, and its performance on the matrix set ex9 is comparable to that with the optimal task granularity, but again, it performs poorly on other matrix datasets. Therefore, for a given dataset and hardware platform, it is necessary to build a model to predict the best task granularity. This helps to balance the computational and memory access load among multiple processes, allowing for the best match between the sparse matrix dataset, SpMV computation, and the hardware platform, thereby maximizing the computational potential of multicore processors.

The selection of the optimal task granularity for SpMV depends on the input dataset and the underlying hardware platform. Current methods often leave the determination of task granularity to programmers to accomplish through a trial-and-error approach. Although analytical modeling methods can predict appropriate task granularity, this approach heavily relies on architecture experts to build platform-specific models, which need to be remodeled when the platform changes. In light of this, this paper proposes to use machine learning methods to automatically construct an optimal task granularity prediction model, thereby freeing programmers from this task and overcoming the shortcomings of existing methods.

3. Automated Method

3.1. Model Introduction

The process of building a model to predict the optimal task granularity is illustrated in Figure ??. For a given sparse matrix dataset, on one hand, a Python script is used to read the predefined sparse matrix features; on the other hand, the performance of thousands of sparse matrices at each task granularity for SpMV is measured using the Dawn supercomputer platform, and the task granularity corresponding to the optimal performance is marked. Subsequently, the matrix feature values and optimal task granularity are used as inputs for machine learning algorithms to train the optimal task granularity prediction model. In building the model, the scikit-learn machine learning library available in Python is used. This library offers a variety of machine learning classifiers and models, including Random Forest Classifier (RFC), K-Nearest Neighbors Classifier (KNC), Decision Tree Classifier (DTC), Gaussian Naive Bayes Classifier (GNC), Logistic Regression Classifier (LRC), Multilayer Perceptron Classifier (MLP), Support Vector Machine Classifier (SVC), and Gradient Boosting Classifier (GBC). The paper implements and compares the performance of all eight methods mentioned above for building the optimal task granularity prediction model and finds that using the Decision Tree Classifier yields the best prediction results (best average SpMV performance).

The process of building an optimal task granularity prediction model primarily consists of four basic steps:

1. Feature Selection;
2. Generation of Training Data;
3. Training the Model Using Machine Learning Algorithms;
4. Model Deployment.

3.2. Feature Extraction

The data required for training include the structural features of the sparse matrix and the optimal task granularity for the corresponding matrix. In selecting the matrix features, this work refers to the studies of Sedaghati and others [?], and also includes features identified during testing as having a significant impact on performance. To effectively characterize the input sparse matrices, this paper selects 7 matrix features, as specifically shown in Table ??.

Feature	Meaning
n_rows	Number of rows in sparse matrix
n_cols	Number of columns in sparse matrix
nnz_frac	Fraction of non-zero elements
nnz_min	Minimum value of nnz elements
nnz_max	Maximum value of nnz elements
nnz_avg	Average number of nnz elements
nnz_std	Standard deviation of nnz elements

Table 1. Sparse Matrix Features

For the matrix feature values in Table 1, a Python script is used to process the raw data files of the matrices and extract or calculate the corresponding feature values. Before training, to eliminate the influence of different scales among indicators, accelerate the speed of gradient descent to find the optimal solution, and improve the accuracy of the model, the feature values are standardized using the $StandardScaler()$ function. This standardization process centers the feature values around 0 with a variance of 1. Unlike $MinMaxScaler()$ normalization, StandardScaler standardization minimizes the interference of anomalous feature values.

3.3. Training the Model

3.3 Training the Model

To generate training data, it is necessary to run the SpMV program on each input dataset with different task granularities and record its performance. Taking the Dawning high-performance computing cluster hardware platform described in Section 4.1 as an example, 13 different values of K are chosen (1, 2, 4, 6, 8, 10, 12, 16, 24, 32, 48, 64, 128), to test the performance of SpMV for each value and determine the task granularity that yields the best performance. The feature values of the input matrices and the optimal task granularity together form the training dataset. The

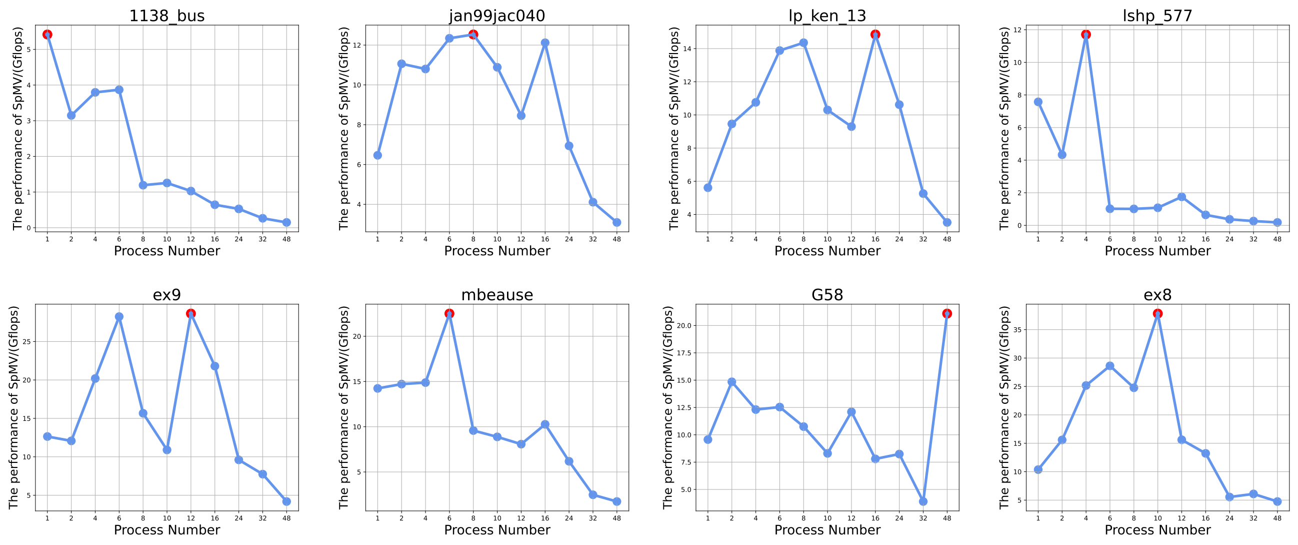


Figure 4. Effect of different task allocation granularities on the performance of SpMV

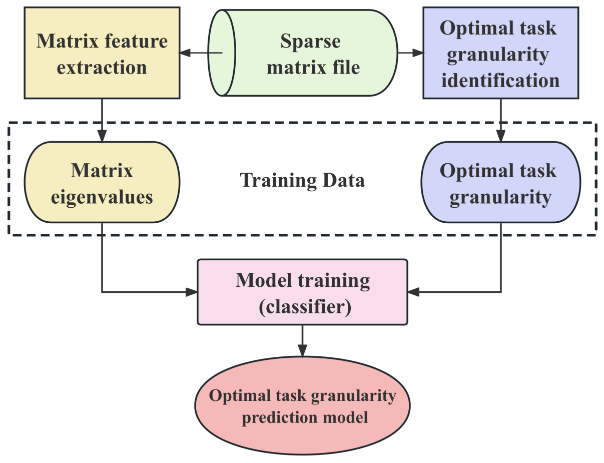

Matrix feature extraction
Sparse matrix file
Optimal task granularity identification
Matrix eigenvalues
Training Data
Optimal task granularity
Model training (classifier)
Optimal task granularity prediction model

Figure 5. Flow chart for implementing optimal prediction model

number of samples in the dataset equals the number of input matrices.

In practice, when building a model using machine learning, it's necessary to divide the dataset into a training set, a validation set, and a testing set. The training set is used for training the available models, the validation set is for selecting the model, and the testing set is for evaluating the performance of the trained model. Considering that insufficient training data may affect the accuracy of the prediction model, in this study, the data is randomly divided into a training set (80%) and a testing set (20%). Cross-validation is used within the training set for model tuning, and the trained model is then evaluated using the testing set. The specific process flow is illustrated in Figure ??.

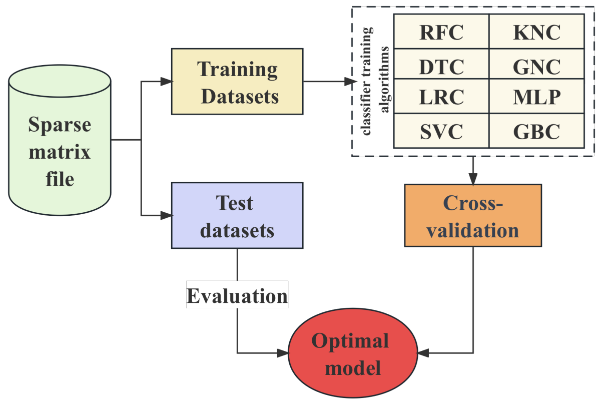


Figure 6. Flow chart of selection and evalution of model prediction

In the process, model tuning employs k-fold cross-validation. The principle of k-fold cross-validation can be summarized as follows: first, the dataset D is divided into k mutually exclusive subsets of similar size, that is, $D = D_1 \cup D_2 \cup ... \cup D_h, D_i \cap D_j = \varnothing$. Each subset D_i should maintain the consistency of data distribution as much as possible, being obtained from D through stratified sampling. Then, for each iteration, the union of $k - 1$ subsets is used as the training set, while the remaining subset serves as the testing set.

This results in k sets of training/validation sets, allowing for k rounds of training and validation. The final result is the average of these k test outcomes.

In practice, the most commonly used 10-fold cross-validation is implemented, which can be achieved by calling the $cross_val_score$ function. This function conducts cross-validation on the model and yields the average accuracy. Since this paper is more concerned with the impact of the trained model on SpMV performance, the $cross_val_predict$ function is used to obtain specific results from the cross-validation (optimal task granularity predicted for each matrix). These results are then correlated with the SpMV performance implemented by the model. Finally, the 20% testing set is used to evaluate the model that has been trained.

3.4. Model Deployment

The trained model is deployed in the form of a library for use by various processes responsible for executing SpMV. Specifically, during the initial stage of SpMV execution, based on the original input matrix being read, matrix feature values are computed. Next, these matrix feature values are input into the optimal task granularity prediction model to calculate the optimal task allocation granularity $K_{predict}$. This parameter is then fed into the MPI runtime system to guide the allocation of SpMV tasks to processes based on the task granularity.

4. Performance Results and Analysis

4.1. Test Environment Description

The tests in this paper were conducted on the Sunway supercomputer platform, utilizing two nodes connected by a 100Gb wide-bandwidth link. Each node consists of 64 processor cores, with the core specifications being Intel (R) Xeon (R) Gold 7285 CPU @ 2.5GHz. Additionally, each node is equipped with 256GB DDR4 memory, utilizing 4 memory channels, and achieving a bandwidth of over 60 GB/s as determined by the stream benchmark test. The operating system on this test platform is Linux v4.15.0-46, Python version 3.10 was used, and process parallelism was implemented using MPI with mpi4py-3.1.4. MPI was also used to control the task granularity.

A total of 1627 sparse matrices used in the tests were sourced from the University of Florida Sparse Matrix Collection (https://sparse.tamu.edu). These matrices include both regular and irregular matrices that have been encountered in practical applications, spanning various fields such as computational fluid dynamics, electromagnetics, and computer graphics. They are widely used for testing sparse matrix algorithms. To obtain 13 sets of training data $K = (1, 2, 4, 6, 8, 10, 12, 16, 24, 32, 48, 64, 128)$ for each dataset, matrices with a number of rows (representing the total number of SpMV tasks) less than 100,000 were selected, totaling 1627 sample datasets. During model training, 80% of the samples comprised the training set, while the remaining 20% constituted the test set.

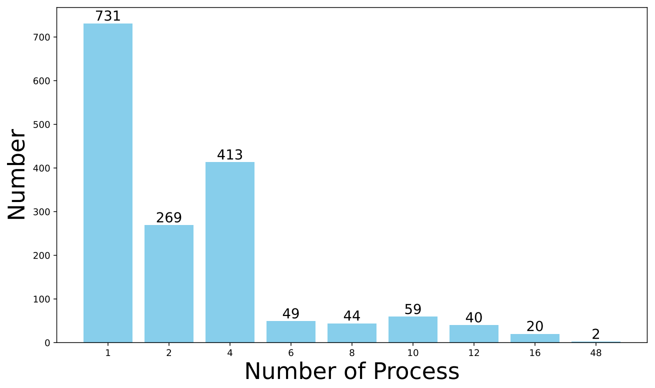
4.2. Performance Results

4.2.1 Overall Sparse Matrix Performance Across Different Task Granularities

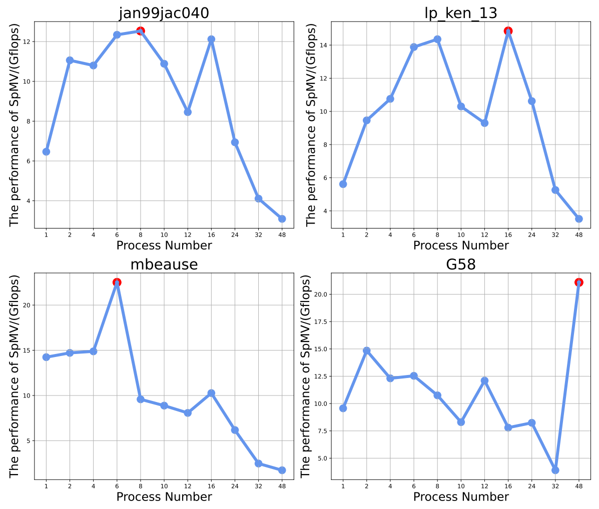
As shown in the graph below(figure ??), it displays the statistics of the number of task granularity categories corresponding to the minimum time achieved in all our samples in this experiment. It can be observed that the majority of optimal granularity assignments, yielding the best performance, fall between 1 and 4. This may be due to the presence of many sparse matrices with default values of 1 in the samples used in this experiment. In practical applications, this kind of situation is less likely to occur. Additionally, there are no optimal solutions beyond a task granularity of 64. This may be a result of the communication time between processes gradually exceeding the parallel computation time between processes. Therefore, when displaying relevant data, data beyond 48 are omitted, as shown in the graph. If the same task granularity allocation is applied to all these datasets, such as using $K = 4$ for all, then it would not achieve optimal efficiency for datasets with task granularities of 10, 12, 16, and 48. Moreover, it can be seen that SpMV does not exhibit a specific pattern with varying K for different input datasets, further emphasizing the necessity of constructing an optimal task granularity prediction model.

4.2.2 Performance Evaluation of Prediction Methods

The graph (figure ??) below illustrates a comprehensive comparison between the task granularity predicted using our model and SpMV performance when using default block allocations. Default block sizes of 1, 2, 4, 6, 8, 10, 12, 16, 24, 32, and 48 were tested. It can be observed that dynamically allocating task granularity using the optimal model on the test set results in better performance than using any static task granularity allocation. When the task granularity is fixed at 1, it achieves the best performance metric in static task granularity, which aligns with the statistics showing the highest number of processes with a task granularity of 1 corresponding to the minimum time achieved in all samples, as shown in the previous graph. How-



(a) The Number of Max Gflops Processes



(b) The graph after omitting the relevant data

Figure 7. The overall performance of sparse matrices across different task granularities, along with the graph representation after simplifying the data

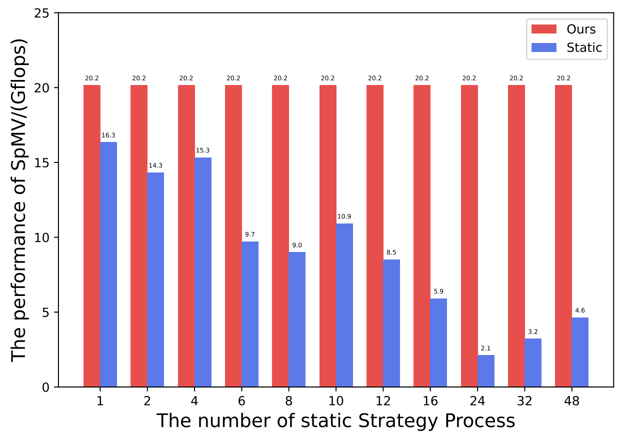


Figure 8. Ours Module VS Static Strategy

ever, in this paper, even if this static strategy is chosen, the performance is approximately 20% lower than the task granularity allocation performed automatically by our model. In other words, using our model for task granularity allocation on this dataset and platform can improve the SpMV computation efficiency by around 20%, and this improvement is even greater for task granularity allocations with different numbers of processes. In conclusion, on the one hand, the model trained in this paper achieves excellent performance results. On the other hand, adopting a fixed task granularity allocation and achieving high performance is not feasible.

4.2.3 Comparison of Different Machine Learning Algorithms' Performance Evaluation

Figure ?? displays the performance evaluation of eight different machine learning algorithms. The horizontal axis represents eight distinct machine learning algorithms, while the vertical axis represents the accuracy of the models trained. It can be observed that training with these eight machine learning algorithms yields nearly equivalent accuracy. In comparison, the Decision Tree Classifier (DTC) performs the best, achieving an accuracy of 87%. Therefore, it is chosen as the training algorithm for the model in this paper. However, it should be noted that there is still significant parameter tuning potential when training with these machine learning algorithms, which will be explored further in future work.

5. Related work

This section synthesizes research on Sparse Matrix-Vector Multiplication (SpMV) across two key platforms: multi-core CPUs and GPUs, while highlighting the uniqueness, limitations, and advancements of each study.

(1) SpMV Research on CPUs:

Pinar et al. [?] tackled low cache utilization in SpMV with Fixed-Size Blocking and Blocked CRS. Their reordering algorithm aimed to cluster non-zero matrix elements, improving efficiency significantly. However, the approach was NP-complete and relied on heuristic algorithms. Their method showed a performance increase of up to 33% on the Sun Enterprise 3000 pro-

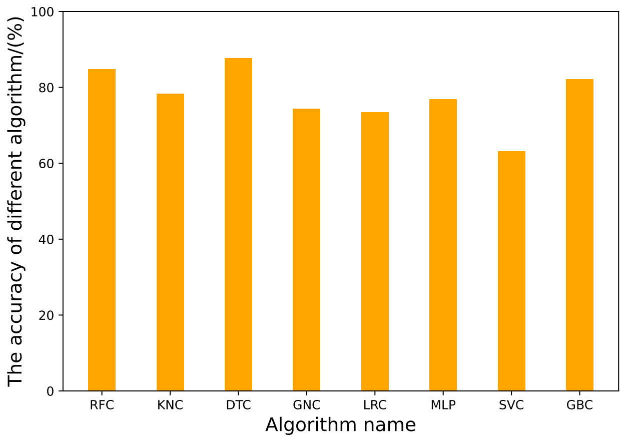


Figure 9. Comparison of Accuracy Among Different Models

cessor, yet it remained specific to certain processor architectures.

Liu et al. [?] focused on the Intel Xeon Phi processor, addressing SIMD inefficiency in SpMV with their ESB storage structure. Their load balancing strategies led to notable performance gains, achieving 1.85 times the speed of CSR format implementations, yet the study's scope was limited to the specific processor type.

Byun et al. [?] extended the OSKI model to pOSKI for multicore processors, optimizing data structures and tuning processes. They reported performance improvements up to 9.3 times compared to CSR. However, the model's reliance on heuristic performance models may limit its applicability in varying computational environments.

Kreutzer et al. [?] proposed the SELL-C-σ format to address the non-universality of mainstream sparse matrix storage formats. Their format showed better performance across different processors but highlighted the ongoing challenge of finding a universally optimal sparse matrix storage format.

(2) SpMV Research on GPUs:

Pichel et al. [?] were pioneers in optimizing SpMV on GPUs using reordering algorithms. Their findings underscored the significant impact of reordering techniques on performance but also revealed the absence of a universally superior storage format or reordering algorithm for GPUs.

Li et al. [?] introduced the PMF-based evaluation of SpMV on GPUs, offering a method to select the most efficient storage format based on matrix sparsity characteristics. While this approach provided a nuanced understanding of format efficiency, its practical applicability might be limited by the variability in matrix characteristics.

Yan et al. [?] developed the BC-COO format, reducing memory bandwidth requirements and improving cache hit rates. Their automatic framework for optimization parameter selection based on matrix characteristics showed substantial performance improvements but was specific to particular GPU models.

Liu et al. [?] proposed the CSR5 format, addressing load imbalance in irregular matrices. Its insensitivity to sparse structures made it broadly adaptable, yet the study mainly focused on performance comparisons without delving into potential limitations in diverse computational environments.

In contrast to these approaches, this paper introduces a machine learning-based method for optimal task distribution in SpMV. This novel approach leverages the characteristics of sparse matrices, offering a potentially more versatile and adaptable solution for varying datasets and platforms.

6. Conclusion

This paper discusses the impact of different task distribution strategies on performance for sparse matrix and dense vector multiplication operations. It was observed that task distribution strategies can significantly influence the performance of SpMV, and there is no fixed distribution strategy that achieves the best computational performance for all sparse matrices. Additionally, the paper proposes a machine learning-based method for predicting optimal task granularity, which can be trained for specific datasets and target platforms. Experiments show that, compared to the default block allocation method, using task granularity predicted by the model can achieve an average performance improvement of about 20%.

References

[1] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pages 1–11, 2009. 1

[2] A Chhabra, G Singh, and KS Kahlon. Multi-criteria hpc task scheduling on iaas cloud infrastructures using meta-heuristics. Cluster Computing, 24:885–918, 2021. 3

[3] Amit Chhabra, Gurvinder Singh, and Karanjeet Singh Kahlon. Qos-aware energy-efficient task scheduling on hpc cloud infrastructures using swarm-intelligence meta-heuristics. Comput. Mater. Contin, 64:813–834, 2020. 2

[4] Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R. Bishop. A unified sparse matrix data format for efficient general sparse matrix-vector multiply on modern processors with wide simd units, 2013. 7

[5] Kenli Li, Wangdong Yang, and Kuan-Ching Li. Performance analysis and optimization for spmv on gpu using probabilistic modeling. IEEE Transactions on Parallel and Distributed Systems, 26:196–205, 2015. 8

[6] S Li, C Hu, J Zhang, et al. Automatic tuning of sparse matrix-vector multiplication on multicore clusters. Science China Information Sciences, 58:1–14, 2015. 7

[7] Weifeng Liu and Brian Vinter. Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication. ACM, 2015. 8

[8] Xing Liu, Mikhail Smelyanskiy, Edmond Chow, and Pradeep Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. ACM, 2013. 7

[9] Juan C. Pichel, Francisco F. Rivera, Marcos Fernández, and Aurelio Rodríguez. Optimization of sparse matrix-vector multiplication using reordering techniques on gpus. Microprocess. Microsyst., 36(2):65–77, mar 2012. 7

[10] Ali Pinar and M. T. Heath. Improving performance of sparse matrix-vector multiplication. In Supercomputing, ACM/IEEE 1999 Conference, 1999. 7

[11] Naser Sedaghati, Te Mu, Louis-Noel Pouchet, Srinivasan Parthasarathy, and P. Sadayappan. Automatic selection of sparse matrix representation on gpus. In Proceedings of the 29th ACM on International Conference on Supercomputing, ICS '15, page 99–108, New York, NY, USA, 2015. Association for Computing Machinery. 4

[12] Shengen Yan, Chao Li, Yunquan Zhang, and Huiyang Zhou. Yaspmv: Yet another spmv framework on gpus. SIGPLAN Not., 49(8):107–118, feb 2014. 8