Scalable Approach to Sparse Matrix Format Selection

Deepak Gupta
Dept. CSC, NCSU
dgupta22@ncsu.edu

Abstract

Sparse matrix vector multiplication (SpMV) is one of the most important kernel is many scientific computations and is often a major performance bottleneck. Performance of SpMV highly depends upon the format in which sparse matrix is represented. This project addresses this problem and focuses on selecting suitable SpMV format for a given sparse matrix by analysing its features. This prediction is done by using a prediction model created by XGBoost by using sample input matrices. For format selection seven most common sparse matrix formats are used from scipy sparse. The training data created by using 355 sample matrices shows an overall accuracy of 84.19% by using five fold cross validation.

Keywords SpMV, Program Optimization, Format Selection, Regression Model, Performance Modeling.

1 Motivation

Sparse matrix vector multiplication (SpMV) is one of the important kernel in many scientific computations, operations research, image processing, data mining, structural mechanics, and other fields [1,2]. SpMV kernel has extremely low arithmetic intensity and irregular memory access patterns. Therefore, it is one of the bottle necks in many of the applications and speeding up its performance is must.

Currently, this speedup is achieved by representing sparse matrices in different formats, which can take smaller space as compared to original formats and uses customized algorithms to perform SpMV calculations. It is observed that for better SpMV performance a proper format for storing sparse matrix must be selected. Various storage formats have been proposed for different applications and computer architectures. But no single storage format has been observed to perform well for all SpMV applications.

The proper storage format depends upon many factors like characteristics of sparse matrix, underlying hardware architecture, nature of application and so on. This project addresses this issue of selecting the suitable format for the given application. The project creates a scalable predictive model which analyses the characteristics of the given sparse matrix and selects a suitable format from a given set of formats which can deliver best performance for the given application.

2 Objectives

The project aims to predict a suitable format for sparse matrix representation from known formats by analyzing the input sparse matrix features. This prediction is done by using a pre-trained predictive model. The model shall return such a format which gives best execution time for the SpMV procedure executed on the given input matrix. Moreover, the model should return the selected format in a reasonable time and the time taken for converting the input sparse matrix from the original input format to the selected format shall not overcome the benefits of the reduced time execution achieved by the newly selected format. In other words, the time to predict and time taken to convert the original format to selected format shall be less than the speedup achieved by the newly selected format.

3 Background

There are various formats for representing sparse matrices. For example, as given in Figure 1, matrix A can be represented in various formats like coordinate format (COO), compressed sparse row format (CSR) or diagonal format (DIA).

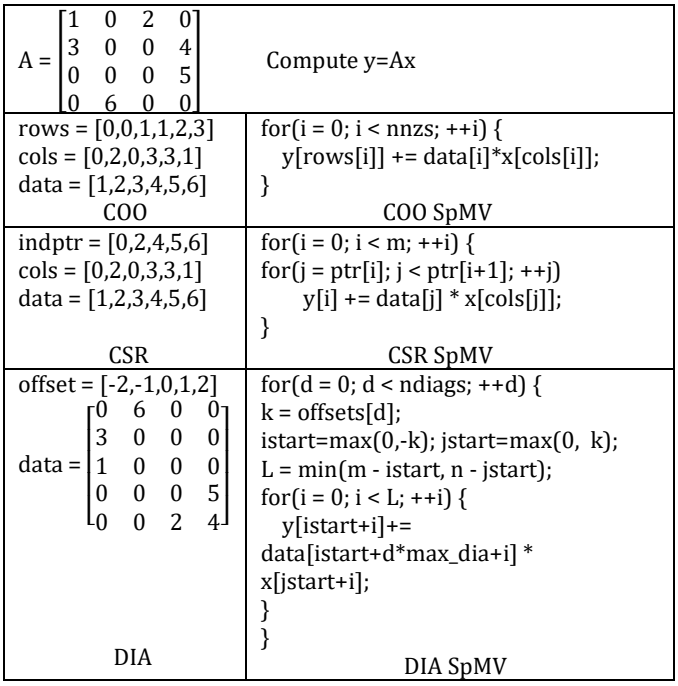

Figure 1. Sparse matrix storage format for COO, CSR and DIA format and their corresponding SpMV pseudo code [1]

This project uses seven different formats for representing sparse matrices:

1) BSR: Block Sparse Row format
2) COO: Coordinate format
3) CSR: Compressed Sparse Row format
4) CSC: Compressed Sparse Column format
5) DIA: Diagonal Storage format
6) DoK: Dictionary of Keys format
7) LiL: Linked List format

The implementation of these formats is defined in scipy sparse library of python as these are most commonly used formats.

The performance data of these formats is collected on various sparse matrices, collected from suite sparse. Since for newly selected format to actually deliver speed up the following equation must hold true:

$$T_{pred} + T_{conversion} + T_{SpMV_{new}} * N < T_{SpMV_{old}} * N$$

That is, the total of the time taken by the predictive model to predict the actual format, time taken for conversion from default format to new format and time taken by SpMV calculation for performing N iterations for newly selected format must be less than the time taken by SpMV calculation for performing N iterations for old or default format.
For the suite sparse matrix, the default format in which sparse matrices are read by scipy sparse is COO format. Therefore, for other formats to deliver better performance, the sum of their prediction time, time taken to convert from COO to the selected format and then time taken for SpMV calculation must be less than that taken by SpMV when matrix was in COO format. Therefore, the performance data on training data is calculated by considering the same.

4 Challenges

4.1 Default Formats

One of the biggest challenge is to implementation is that most of the sparse matrices are represented in a default format. For e.g. the matrices of suite sparse, which is used as a data set in the current project, are represented in COO (coordinate) format by default. This default format induces a bias towards the default format in the prediction. If we want our predictive model to be overhead conscious then the time taken for converting from default format to predictive format has to be taken into account. So, this means there is no overhead on case of default format. This introduces inherent bias in the prediction. This bias can be removed by not taking conversion time in to consideration. But then it will result in overhead oblivious predictive model. There is a tradeoff between the two and in current implementation this bias is neglected for making an overhead conscious predictive model.

4.2 Optimal Value of Number of Iterations of SpMV

The next challenge is to find an optimal value of N i.e. number of iterations of SpMV. Best approach is to use a predictor to predict the number of iterations, as is done in [1]. But if we are only focused in creating a predictive model which returns best suitable format for the given input matrix, then we will have to assume certain value of N. If this value of N is high, then it is almost equal to overhead oblivious predictor and if this value of N is very low then the bias of default format becomes significant. Therefore, an optimal value of N must be chosen. In context of this project this value is chosen to be 50.

4.3 Type of Predictive Model

Next challenge is to select the type of predictive model. For e.g. such prediction can be done with help of decision trees or neural network. That predictive model is selected which is best for such type of prediction and gives best accuracy. Generally, such problems are best suited for decision trees as there are no clearly defined features of input matrices. So, decision trees when used with ensemble techniques like boosting tends to remove the unimportant features and keeps the important ones. Therefore, XGBoost, which is an ensemble of boosted trees, is selected for making predictive model in this project.

4.4 Type of SpMV Formats

Another important challenge is to select the SpMV formats on which performance data is collected. Firstly, such a bag of formats must include all different types of formats which can perform well on diverse sparse matrices. Secondly, the chosen formats must be in line with new hardware advancements. Like the format can exploit the GPUs or FPGAs or any other hardware accelerator. The format can also exploit the parallelism provided by multicore processors. For this we can build two different predictors – one which focus on single thread performance and other which compares ones focusing hardware accelerators. And which predictor to use will depend upon the underlying hardware. But for the current project seven most common SpMV formats are used, as given above, whose implementation is defined in scipy sparse library of python.

5 Solution

The project is implemented in two modules:

1) Training data generator module – it takes some sparse matrices as input and generates training data from it. This training data includes some selected features of the input sparse matrices and performance data on selected SpMV formats.
2) Predictive model module – it takes the training data generated by the preprocessing module as input and trains a predictive model. When this predictive model is trained it can predict the suitable SpMV format for the given input matrix.

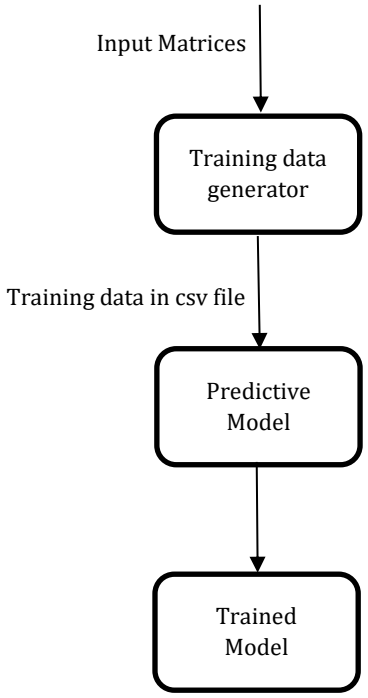

Figure 2. Overall architecture of the project

When a matrix is given as input to this trained model, it analyses the features of the input matrix and uses these features to predict the appropriate format which can produce the desired speedup.

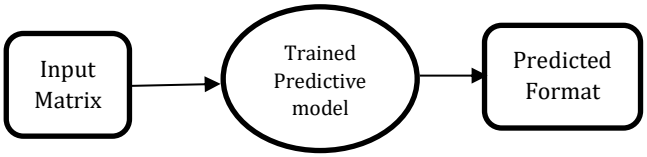

Figure 3. Workflow for predicting format for an input matrix

5.1 Training data generator module

This module takes the sparse matrices from the given dataset. The dataset for this project is taken from Suitesparse. It is a set of around 2800 sparse matrices that arise in real applications.
These sparse matrices are read one by one in a default COO format. Then some features peculiar to this read matrix are extracted from it. The extracted features are as given in the Figure 4.

Parameter	Meaning
M	number of rows
N	number of columns
NNZ	number of non-zeros (NZ)
Ndiags	number of diagonals
NTdiags ratio	the ratio of "true" diagonals to total diagonals, true diagonals represents one occupied mostly with NZ.
aver RD	average number of NZ per row
max RD	maximum number of NZ per row
min RD	minimum number of NZ per row
dev RD	the deviation of number of NZ per row
aver CD	average number of NZ per column
max CD	maximum number of NZ per column
min CD	minimum number of NZ per column
dev CD	the deviation of number of NZ per column
ER DIA	the ratio of non-zeros in DIA data structure
ER RD	the ratio of non-zeros in row-packed (ELL) structure
ER CD	the ratio of non-zeros in column-packed structure
row bounce	average difference between NNZs of adjacent rows
col bounce	average difference between NNZs of adjacent columns
d	density of NNZ in the matrix
cv	normalized variation of NNZ per row
max mu	max RD - aver RD
blocks	number of non-zero blocks
mean neighbor	average number of NZ neighbors of an element

Figure 4. Set of features extracted from input matrices [1]

These features help in understanding the nature of the sparse matrix and acts as a fingerprint for similar types of matrices.
Apart from these features, performance data is also collected on the input matrices from the dataset. The input matrix is converted from default format, i.e. COO in our case, to other selected formats like CSR, BSR etc. Time take to convert from default format to targeted format is noted. This helps in making our prediction overhead conscious. Then SpMV is conducted on the converted matrix for some fixed N number of times, 50 in our case. The time taken for performing this SpMV procedure N number of times is also noted and is added to the conversion time. So, for any input matrix total execution time is given as:

$$T_{total} = T_{conversion} + T_{SpMV} * N$$

The format whose T_{total} is least for the given input matrix is chosen as a label for that input matrix. This label is added to the training data. The above procedure is performed on all the matrices in the dataset to produce the training data. This produced training data is written in a csv file which is then used for training the predictive model and can also be reused in future.

5.2 Predictive model module

For prediction, XGBoost is used for making predictive model. XGBoost is an ensemble of trees which uses extreme gradient boosting for supervised learning problems. It uses the training data to predict the labels for input data.
For training XGBoost, a tuple of features, along with a suitable format as a label, is given as input to XGBoost. (Tuple = [$feature_1$, $feature_2$, $feature_3$, ……., $feature_m$, SpMV format])
XGBoost takes these tuples and makes an ensemble of boosted trees. This ensemble of trees can then be used to predict the suitable format from the features of an arbitrary input matrix.
For building any regression model, the main challenge is the identification of the important features of a sparse matrix. The selected features must be able to capture all the characteristics of the matrices. More features may increase the feature extraction overhead while less features may not be able to accurately characterize the matrix. So, an optimal balance is required between the two. The list of features given in Figure 4. ensures this balance [1].
The benefit of using an ensemble of boosted trees is that it automatically selects the most important features from the trained model. It calculates importance score of each feature and then ranks them. The features with low importance scores are then automatically pruned until a minimal set of features is obtained without sacrificing the prediction performance.
In order to further improve the performance, a five-fold cross validation is used. It helps in removing the overfitting problem and thereby improving the prediction accuracy.

6 Lessons and Experiences

6.1 Hardware Dependent SpMV Formats

One of the observation is that various SpMV formats are dependent on underlying architecture. So, the labels created for training data may differ from one architecture to another. Moreover, the format returned by the model may run fast on one architecture but at the same time may run slow on the

other. Therefore, the model trained for one architecture may not be useful for another architecture. Furthermore, as the performance of different formats can differ on multicore processors or GPUs or FPGAs, therefore, there is a need for training predictive models which can predict by considering the underlying hardware architecture and available resources [3,4].

6.2 No Comprehensive Library for SpMV Formats

While finding different SpMV formats and implementations it is observed that there is no comprehensive library which includes implementation of different formats. Some older formats are implemented in older languages like Fortran. Therefore, there is a great need for a comprehensive library in modern languages, like python and R, which includes different formats, including parallel implementations on GPUs and FPGAs.

6.3 Time Consuming Creation of Training Data

One of the great challenge while implementing the project was the creation of training data, which took several hours. Main time-consuming part is the extraction of several features. Although, feature extraction at time of creating training data can still be considered as low priority work, but such feature extraction at the time of predicting the format may reduce the benefits speedup given by the predicted format. Therefore, speeding and parallelizing the feature extraction can also be considered in future works.

6.4 Other Important Observations

a) Although, the current project uses XGBoost for the creation of predictive model. But use of other models like neural networks can also be considered.
b) The implementation gives a good insight into training and testing ensemble of boosted trees. Like the project uses five-fold cross validation procedure to calculate the accuracy of the predictive model.
c) Apart from giving the overall insight into how to use machine learning to optimized code by predicting a suitable format for sparse matrix vector multiplication, the project also gives an idea of using such techniques in various other fields and scenarios.

7 Results

On a sample of 355 sparse matrices the overall accuracy of the trained model by using five-fold cross validation is 84.19%

8 Remaining Issues and Solutions

8.1 SpMV Format Implementation

Currently, only seven sparse matrix formats are used for training data and model creation. For a more comprehensive coverage and better prediction, more sparse matrix formats need to be covered. However, no proper implementation and easily available algorithm of older formats makes this as a big task. Some of the older formats are implemented in older languages like Fortran, but their similar implementation in modern popular languages like Python and R is hard to find. Furthermore, lack of easily available algorithms for them makes it hard to convert them from Fortran to Python. The project is designed in such a way that implementations of new formats, as and when is ready, can be added to the project incrementally.

8.2 Parallel Implementations of SpMV formats

The implementation of various formats for GPUs or FPGAs or other parallel implementations are also to be covered to get a more comprehensive prediction [3,4]. Absence of comprehensive libraries for popular languages like Python and R makes the task even more tougher.

8.3 Hardware Dependent Formats

It is observed that, most of the formats for SpMV are hardware dependent. Therefore, there is a great need to come up with new predictive model design which can handle such scenarios and predict formats by considering the underlying hardware architecture.

8.4 Handling Default Formats

Different languages or platforms have different default formats. For example, Python's scipy reads the suite sparse matrices in COO format. If we want our predictive model to be overhead conscious then it needs to take conversion time into consideration. This introduces inherent bias towards default format. As all other formats have to be converted from the default format. However, it may fail in real scenarios where input matrix may be in different format other than the default format.

8.5 Number of Iterations of SpMV procedure

Another open issue is the at what value of N the model must be trained. If value of N is too low, then the model is biased towards default format. If the value of N is too high, then the model becomes overhead oblivious. Therefore, an optimal value of N is required. One of the solution can be to train different models for different ranges of N. Then number of iterations of SpMV can predicted, as given in [1]. After the number of iterations are predicted then the particular model which represents a particular range can be used to predict the suitable SpMV format.

8.6 Different Machine Learning Models

Current project uses XGBoost, which is an ensemble of boosted trees, to implement predictive model. Although, XGBoost suites best to the given problem. But, performance of other models, like neural networks, SVM, and so on, still needs some evaluation [3].

References

[1] Yue Zhao, Weijie Zhou, Xipeng Shen, and Graham Yin, "Overhead-Conscious Format Selection for SpMV-Based Applications," in IPDS, 2018.
[2] B.-Y. Su and K. Keutzer, "clSpMV: A cross-platform OpenCL SpMV framework on GPUs," in *Proc. ICS '12*. ACM, 2012, pp. 353–364.
[3] A. Benatia, W. Ji, Y. Wang and F. Shi, "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU," International Conference on Parallel Processing, 2016, pp. 496 –505.
[4] Neelima B, GRM Reddy and Prakash S., "Predicting an Optimal Sparse Matrix Format for SpMV Computation on GPU", IPDPS-14, PP. 1427-1436.