EasyChair Preprint

№ 388

Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs

Israt Nisa, Charles Siegel, Aravind Sukumaran Rajam, Abhinav Vishnu and P Sadayappan

August 1, 2018

Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs

Israt Nisa*, Charles Siegel†, Aravind Sukumaran Rajam*, Abhinav Vishnu†, P. Sadayappan*
*Dept. of Computer Science and Engineering
The Ohio State University
Columbus, OH, USA
{nisa.1, sukumaranrajam.1, sadayappan.1}@osu.edu
†Pacific Northwest National Laboratory
Richland, WA, USA
{charles.siegel, Abhinav.Vishnu}@pnnl.gov

***Abstract*—Sparse Matrix-Vector multiplication (SpMV) is a key kernel for many applications in computational science and data analytics. Several efforts have addressed the optimization of SpMV on GPUs, and a number of compact sparse-matrix representations have been considered for it. It has been observed that the sparsity pattern of non-zero elements in a sparse matrix has a significant impact on achieving SpMV performance. Further, no single sparse-matrix format is consistently the best across the range of sparse matrices encountered in practice. In this paper, we perform a comprehensive study that explores the use of Machine Learning to answer two questions:**
1) Given an unseen sparse matrix, can we effectively predict the best format for SpMV on GPUs? 2) Can SpMV execution time for that matrix be predicted, for different matrix formats?

By identifying a small set of sparse matrix features to use in training the ML models, we demonstrate that efficient prediction of the best format with $\approx$ 88% accuracy can be achieved when selecting between six well known sparse-matrix formats on two GPU architectures (NVIDIA Pascal and Kepler), and $\approx$ 10% relative mean error (RME) with execution time prediction.

***Keywords*-Sparse matrix-vector multiplication (SpMV); GPU; Sparse Matrix format selection; XGBoost, Support Vector Machine (SVM); Multilayer Perceptron (MLP), Decision tree.**

I. Introduction

Sparse Matrix-Vector multiplication (SpMV) is a widely used kernel in scientific applications, graph analytics, and machine learning. The development of efficient SpMV implementations for a variety of architectures has been widely addressed, including multi-core systems, many-core systems and distributed memory systems comprising of multi-core/many-core architectures. In this paper, we focus on GPU implementations of SpMV. Over the last few years, significant research has been conducted on specialized formats, including but not limited to CSR, COO, ELL [1], HYB [2] and the recently proposed merge-based CSR [3], CSR5 [4], and yaSpMV [5]. A number of compact sparse-matrix representations have been evaluated for SpMV and it has been found that no single format is uniformly superior – the best performing format for one sparse matrix could achieve much lower performance than another format for a different sparse matrix. This variability leads to the format-selection problem: *Can an effective model be devised to quickly predict the best-performing format for a previously unseen sparse matrix?*

Since SpMV performance for different sparse matrices can vary considerably across various formats, a question of interest is: *Can the SpMV execution time for a previously unseen sparse matrix be effectively predicted for various representation formats?* Several prior efforts have considered performance modeling for the SpMV kernel [6], [7]. Usually, analytical model-based approaches are leveraged for performance modeling. However, analytical models assume linear independence among features – an assumption that is problematic for general performance modeling.

An alternative approach to analytical models is leveraging non-parametric approaches such as Machine Learning (ML). These approaches typically require a dataset with ground truth. For example, in format selection the ground truth is the best format for a given matrix; in performance modeling, it is the actual achieved FLOPS for each format under consideration. An advantage of ML algorithms is their ability to handle non-linearity in features – which becomes prevalent as the number of features describing the model increase.

Recently, format selection and performance modeling using ML techniques have generated significant interest. Simple ML algorithms such as decision tree and multi-class Support Vector Machine (SVM) classifier have been shown to achieve very good accuracy for format selection [8], [9], and [10]. Deep neural networks (DNNs) such as convolutional neural networks (CNNs) are also starting to be used for the classification problem [11], [12]. Zhao et al. [12] compress the sparse matrix as an image – and use CNN to solve the format selection problem.

Our objective in this paper is to assess the effectiveness

of a number of alternative ML-based approaches to format selection for SpMV, as well as performance modeling of SpMV for different formats. We also study the utility of different sparse matrix features for the format selection problem.

A. Contributions

We make the following contributions in this paper:

- We explore a set of base and ensemble ML algorithms and achieve a classification accuracy of 91% across basic storage formats like ELL, CSR, and HYB using XGBoost ensemble algorithm.
- For recently proposed advanced formats such as merged-based CSR and CSR5, we achieve a classification accuracy of 88% by using XGBoost algorithm.
- We achieve a low relative mean error (RME) in performance prediction over a large collection of over 2300 sparse matrices from the SuiteSparse sparse matrix collection.

We provide an in-depth analysis of results including metrics while using ensemble techniques; provide feature importance for performance modeling/classification; and performance penalty of mis-prediction.

The rest of the paper is organized as follows: Sec. II presents background information on sparse matrix formats and pertinent machine learning algorithms. Sec. III presents an analysis of properties of sparse matrices from the SuiteSparse collection. Details on feature selection for sparse matrices are described in Sec. IV. Sec. V reports the experimental evaluation to compare the effectiveness of several of ML techniques on the classification problem for effective format selection. Sec. VI presents experimental results for performance modeling of SpMV with the different formats. Section VII discusses related work on format classification and performance modeling and we conclude in section VIII.

II. Background

A. Sparse matrix storage formats:

Sparse Matrix-Vector multiplication (SpMV) is a key primitive for computational science and data science. Hence there has been substantial literature on numerous optimization techniques to enhance SpMV performance. The underlying structure of the sparse matrices can be highly irregular and has a direct impact on performance. SpMV performance on two matrices with similar size and number of non-zeros can be differ significantly, depending on the storage format and efficiency of implementation. Various machine learning techniques can be applied to select the best storage format for a sparse matrix. In this section, we describe various storage formats and machine learning algorithms.

1) COO format: The COO (coordinate) format is the simplest storage format. The sparse matrix is represented using three dense arrays corresponding to row indices, column indices, and the non-zero element values. The size of each array is equal to the number of non-zero elements in the matrix. Figure 1(a) shows the layout of the COO format. Bell and Garland [2] describe a GPU implementation of SpMV - first all product contributions are computed for the non-zero elements, followed by a segmented reduction across threads. Along with a simple representation, COO also has the advantage of very stable performance across matrices with very different sparsity.

2) CSR format: The compressed sparse row format (CSR) is the most commonly used format to represent sparse matrices. Like COO, CSR also uses 3 arrays to represent the sparse matrix. While the column indices and values arrays of CSR are identical to COO, the third array (row-pointer) stores the starting index of each row, with all non-zero elements in each row being stored contiguously. Fig. 1(b) illustrated the CSR format. With the CSR format, SpMV can be parallelized by assigning a thread to process each row – the $scalar$ CSR kernel [2]. However, this approach has the disadvantage of non-coalesced access to the column-index and value arrays, which can be expensive for GPUs. Another approach to parallelization is to assign a WARP to process a row. With such a work distribution, the access to column indices and nonzero values do not suffer from uncoalesced access, but result in significant thread-divergence for rows with few non-zeros.

3) ELLPACK format: In ELLPACK format, the rows with fewer non-zero elements than max_nnz are left shifted and padded with zeroes. One matrix presents the column indices and the other presents the values of nonzero elements. As shown in Fig. 1(c), ELL stores the matrix in a column-major format and each row is processed by a single thread [2]. Good load balance across threads and avoidance of thread divergence are achieved, but the memory access pattern to the input vector x might not be contiguous.

4) HYB format: The HYB format combines the benefits of COO and ELL. ELL performs well when the standard deviation of non-zero elements per row is low. On the other hand, the COO format is insensitive to the underlying structure. The HYB format uses a combination of COO and ELL, storing some matrix rows in COO format and the others in ELL format. A threshold parameter is used to decide whether a row should be represented in COO or ELL. If the number of non-zero values in a row is greater than the threshold, then an ELL format is used; otherwise the COO format is used. The threshold is often computed as max(4096, number of rows/3) [2]. The average number of non-zeros per row (nnz_mu) can also be used as a threshold metric, which we use in this paper.

5) CSR5 format: Liu et al. [4] proposed CSR5 – which is an extension of the CSR format. It uses five arrays to

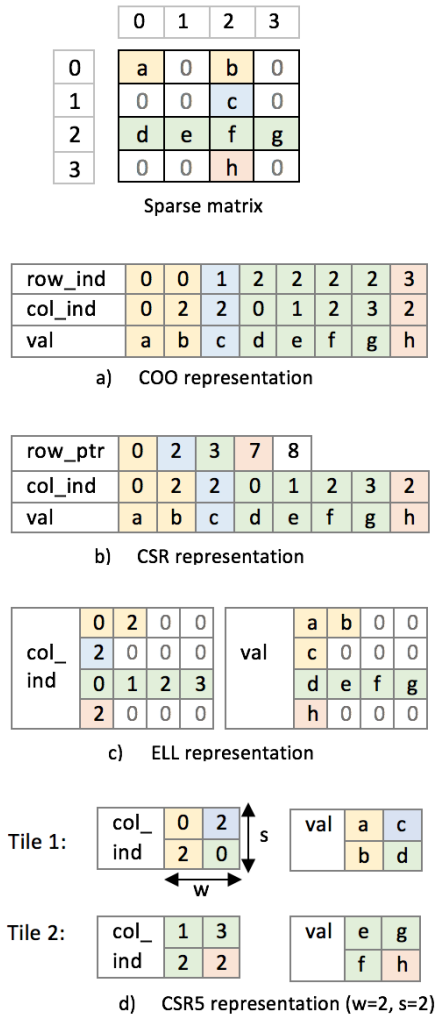

Figure 1: Sparse matrix representation using COO, ELL, CSR and CSR5 formats

store the matrix, adding two more arrays to the three arrays used with CSR. A sparse matrix is partitioned into equally sized small 2D tiles to achieve good load balancing. The two additional arrays are used to store a tile transposition order (tile_ptr) and information about the tiles (tile_desc). Figure 1(d) illustrates the CSR5 format.

6) Merge-based CSR format: The merge-based CSR SpMV algorithm for GPUs [3] also focuses on enhancing load balancing among threads despite the irregularity of the underlying sparse matrix structure. It works with the standard CSR sparse matrix representation. The standard SpMV implementations using CSR suffer from load imbalance for matrices with high variance in the number of non-zeros in different rows of the matrix. Merge-based CSR aims to alleviate this load balancing problem. It logically partitions the work of processing a group of contiguous rows into equal sized sub-problems that are distributed across threads.

B. Machine Learning Algorithms

1) Decision Trees: A popular machine learning approach to classification is the use of Decision Trees (DT). Its popularity is in part due to easy interpretability. The output after training is a binary tree, with each node incorporating a rule to decide whether to proceed to its left or right child. The leaves of the tree encode the final predictions of the model.

2) Support Vector Machines: A Support Vector Machine (SVM) is a model that constructs a hyperplane (or set of hyperplanes) in a high dimensional space containing data. It does so by determining *support vectors* from the data set which are closest to the boundary between classes of samples. It then places a hyperplane such that the distance between it and support vectors on either side is equal. SVMs suffer from being harder to interpret than decision trees, having a higher computational complexity, and only being directly applicable to binary classification. However, SVMs can often achieve higher accuracy than decision trees.

3) Multi-Layer Perceptrons: The multi-layer perceptron (MLP) is a class of neural network models. MLPs model functions by building iterated compositions that approximate them. Each *layer* of an MLP consists of several *neurons*, which each take input from all neurons in the previous layer, multiply them by *weights* and then return the output from a nonlinear function at this value. Often, several MLP models are combined by averaging their predictions into an *ensemble* to improve accuracy. It is known that MLPs can approximate any function arbitrarily closely, and do so more efficiently when utilizing the hierarchical structure [13].

4) XGBoost: XGBoost (for Extreme Gradient Boosting) is an ensemble ML technique for transforming a collection of weak models into a strong model. Usually applied to decision trees, XGBoost trains a sequence of trees in such a way that each one is most effective on those samples that are poorly classified by all previous trees. This can be interpreted as performing gradient descent in *function space* [14] rather than on the parameters of a static function. Along with ensembles of neural networks, XGBoost has been extremely successful in competitive data science, winning several Kaggle competitions [15].

III. Sparse Matrix Characteristics

To motivate the need for classification and performance modeling, we tabulate a few important characteristics of the matrices in the SuiteSparse collection. These are shown in Table I. The columns associated with avg. rows and avg. cols present the average number of rows and columns of the matrices corresponding to the same nnz range. Similarly, avg. density, avg. nnz_mu and avg. nnz_sigma show the average density of the matrix, average nnz per row and average standard deviation of nnz per row, respectively. In this dataset, the nnz range is between 3 to 96M elements. A significant fraction (greater than 50%) of the matrices have

nnz range	no of matrices	avg. rows	avg. cols	avg. density	avg. nnz_mu	avg. nnz_sigma
0∼10,000	747	639	759	4.62	7	4.5
10K ∼ 50K	508	3,590	4,248	1.29	15	18
50K ∼ 100K	209	8,881	10,974	1.03	34	31
100K ∼ 500K	362	24,695	30,714	.69	69	50
500K ∼ 1M	147	70,669	92,925	.75	155	128
1M ∼ 5M	208	173,473	205,277	.61	214	72
5M ∼ 50M	109	1,290,926	1,302,773	.43	852	42
>50M	9	8,101,908	8,101,908	.002	29	5

Table I: Feature analysis of the SuiteSparse repository

non-zero elements less than 50K. We also observe that the average density decreases with increasing number of rows, columns and nnz. However, no clear pattern is observed in standard deviation of average nnz per row.

A few storage formats are less sensitive to the irregular matrix structure. For example, in COO format the parallelism is achieved by computing over each non zero element – resulting in excellent load balancing. However, the replication of row entries adversely affects the performance of the COO format. The ELL uses zero padding to fill up the rows that have lesser than maximum number of non zero elements in a row. Therefore, matrices with high variance in the row length are adversely affected due to work load imbalance by each thread. A similar trend is also observed for the CSR format. Since CSR conducts parallelism over rows of a matrix, similar workload at each row shows better performance using CSR format. Recent studies such as CSR5 [4], merged-based CSR [3], yaspmv [5] aim to achieve a stable performance despite the irregularity in the underlying structure of a matrix. Especially, merged-based CSR shows consistent performance as a function of non-zeros on a large number of matrices.

Yet, there are a substantial number of matrices where performance is far lower than the matrices of similar nnz. Two such matrices, rgg_n_2_19_sO and auto are shown in Figure 2. Both matrices have ≈ 6.5M non-zero elements and are square matrices of dimensions 524K and 450K, respectively. However, the achieved GFLOPS by using CSR5 on rgg_n_2_19_sO and auto are 22 and 18, respectively. Similarly, merged-based CSR provides 21 and 15 GFLOPS for these two matrices. Figure 2 provides an indication of the complexity associated with performance modeling and format selection problem, since these matrices are similar in macro structure, but significantly different in actual performance.

The performance of SpMV is sensitive to both architectural configuration as well as sparsity structure of the matrix. GPU specialized formats focus on higher parallelism as well as reduced data movement and lower thread divergence. The trade-off between the reduction across the non-zeros is often compromised with atomic operations. In Figure 3, we show the achieved GFLOPS for various matrices with different formats. We conclude from the figure that no single format

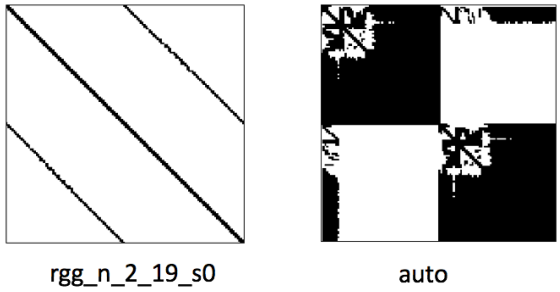

Figure 2: Matrices with similar number of rows, columns and non-zero elements but with different GFLOPS for CSR5 and merged-based CSR formats

is a consistent winner across all the matrices. We observe that for the same matrix, the performance difference across formats can be significant. Hence, format determination problem requires effective feature extraction (such that the sparsity structure can be captured) and modeling techniques such as machine learning that can handle the non-linearity between extracted features.

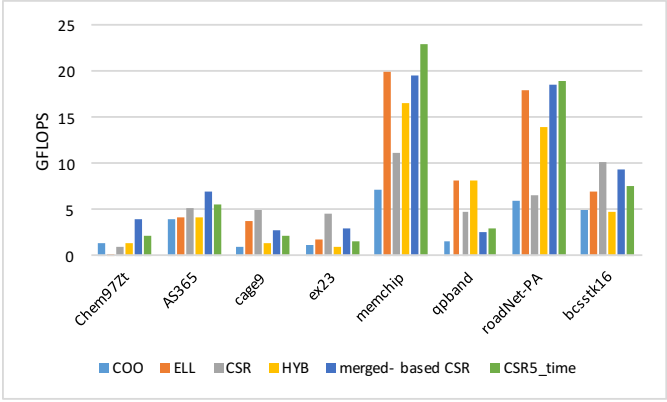

Figure 3: GFLOPS comparisons across different matrix storage format using Tesla k80c GPU and single precision data type

In this paper, we extract a total of seventeen (simple and advanced combined) features to capture the structure of the matrices. This is followed by leveraging state of the art

set	feature	description
1	rows, cols	number of rows and columns
	nnz	number of non zero elements
	nnz_mu	average nnz per row
	density	density of the matrix
2	nnz_max	maximum number of nnz in a row
	nnz_sigma	standard dev. of nnz per row
	row_block_count_*	avg. and std. deviation of the number of continuous nnz chunk per row
	row_block_size_*	avg. and std. deviation of the size of continuous nnz chunks in a row
3	block_count	total number of the continuous nnz chunks
	row_block_count_*	min and max of the number of continuous nnz chunks in a row
	row_block_size_*	min and max of the size of continuous nnz chunks in a row

Table II: Description of features used in this paper. Feature set "1" is considered basic and sets "2" and "3" are advanced feature sets.

machine learning techniques on the features to select the best storage format for a given matrix. We also demonstrate that the similar set of features can be used for performance modeling of execution time of a given sparse matrix.

IV. Sparse matrix features for machine learning

In this section, we present our design and evaluation methodology for sparse matrix format selection and performance modeling. We present design choices for feature selection, methodology for label collection, the datasets which are used in this paper, and the machine learning models which are used for format selection and performance modeling. We begin with a brief description of the feature selection metholodology.

A. Feature selection

Table II shows the complete list of extracted features used in this paper. The feature set 1 (shown as multi-row with "1" in the table) consists of features that can be computed in $\mathcal{O}(1)$ time. This includes features such as the number of rows, columns, non-zero elements, average non-zero per row. However, this feature set is not sufficient in capturing the spatial patterns associated with sparse matrices. As an example, previous studies [8], [9] have addressed this issue by introducing additional features that are shown in feature set 2. The feature set 2 and 3 requires a scan on the entire matrix resulting in a time complexity of $\mathcal{O}(nnz)$.

One of the bottlenecks of SpMV operation is the non-uniform memory access pattern of the vector. The access pattern is determined by the sparsity structure of the matrix. For an unstructured sparse matrix, it is highly likely that the accesses to the vector are uncoalesced, which increases the cost of accessing the vector elements. In addition to the uncoalesced accesses, the probability of cache hits is also reduced. On the other hand, for certain sparsity structures – such as banded diagonal matrix – the vector accesses are more coalesced (less cost) and the probability of the cache hits are higher.

Hence, we collect additional features (feature set "3") that capture this pattern. As an example, we count the number of blocks that have continuous non-zeros in a row (tot_blocks). Although prior efforts [9] have mentioned the use of similar features, they were unable to demonstrate the importance of these features in their ML based model.

Resource	Details
GPU 1	Tesla K40c: 13 Kepler SMs, 192 cores/MP, 12 GB Global Memory, 824 MHz, 1.5MB L2 cache
GPU 2	Tesla P100: 56 Pascal SMs, 64 cores/MP, 16 GB Global Memory, 1328 MHz, 4MB L2cache

Table III: Description of testbeds used in this paper

B. Methodology for label collection

We use the execution time of a matrix as the label for the matrix. The execution time is collected by executing each matrix multiple (50) times and then averaging the execution time. We label the format with minimum execution time (maximum GFLOPS) as the best format for the format selection problem. For the classification problem, the input features and the best format is used as the input. For performance prediction, the input features and average execution time is used as the input. For both classification and performance modeling, we use 5-fold cross validation and 80-20 splits of the train-test dataset.

C. Dataset and architectures explored

The SuiteSparse collection contains $\approx$ 2700 matrices from real applications in various domains. It covers a wide variety of regularity, from structural engineering and computational fluid dynamics to networks and graphs. In this work, we evaluate 2300 matrices out of 2700 matrices. We were unable to use the remaining 400 matrices since they either did not fit in the GPU memory or failed to execute for one or more storage formats.

We used two machines to evaluate the performance of the dataset. The first machine contains an NVIDIA k80c (Kepler) GPU and is coupled with Intel Xeon(R) CPU E5-2680(28 core). The second machine consists of an NVIDIA P100 GPU with similar CPU configuration. Table III shows the details of these machines.

D. Machine learning models

We use a wide set of ML algorithms for both format selection and performance modeling. We use SVM and decision-tree based models, as suggested by Benatia et al. [8], and Sedaghati et al. [9]. We supplement these ML models with multi-layer perceptrons, gradient boosting based models (XGBoost) and ensembles of MLP.

The ML models have have their own set of hyper-parameters and the performance of each scheme is sensitive

to the tuning of the hyperparameters. For XGBoost, we explore parameters like n_estimators: number of trees used, max_depth: maximum tree depth and learning rate [16]. We tune these parameters using $GridSearchCV$, which performs an exhaustive search over a range of supplied parameters and finds the best parameter set. In our set of experiments, the range of n_estimator is {50, 100, 200, 500} , max_depth is {32, 64, 128} and learning rate is {.1, .01}. In SVM, like XGBoost using $GridSearchCV$ the best parameter set is chosen over the range of C {100, 1000, 10000} and gamma of {.1, .01, .001}. C keeps a balance of correctness of training examples and simplicity of the decision surface and gamma defines the importance of training examples [17]. The MLP model we use in this work consists of 3 hidden layer of 96, 48 and 16 neurons respectively with a mini-batch size of 16.

V. Results: SpMV format selection

In this section, we present an evaluation of machine learning on the format selection problem. We begin with a discussion of format selection, especially the COO format.

A. Discussion on COO format

We observed that the COO format outperforms other formats (ELL, CSR, and HYB) in $\approx 10\%$ cases. Similarly, when using 6 formats together, we observe that the COO format rarely outperforms other formats. For double precision in both K80c and P100 machine, there are no such cases and for single precision, there is one such case. We also observed that when COO format is the best, the performance of at least one of the other formats is similar. Hence, the performance loss by excluding the COO format is minimal. *Hence, we remove the cases where COO format is the best format*. Similar observation is made by Benatia et al. [8], where they avoid the COO format in their study.

B. Comparison of ML models on ELL, CSR and HYB formats

To begin with, we compare the results using basic storage formats like ELL, CSR, and HYB. Tables IV, V and VI present the classification accuracy on 3 basic formats over 2300 matrices from the SuiteSparse repository. We observe that XGBoost consistently outperforms decision tree, MLP and SVM on K80c and P100 for both single and double precision data types. By using 11 and 17 features, the XGBoost achieves up to 91% classification accuracy. We also observe that by reducing the number of features – such as shown in Table IV – MLP outperforms SVM in all cases, whereas with higher number of features like in Table V and VI, for single precision cases, SVM achieves equal or better accuracy than MLP across both machines. Table VI shows that using extra features to capture the detailed vector access pattern does not provide additional benefits.

Machine	precision	decs. tree	SVM	MLP	XGBST
K80c	single	**69%**	62%	68%	**69%**
k80c	double	69%	62%	68%	**70%**
P100	single	72%	72%	75%	**75%**
P100	double	72%	69%	73%	**74%**

Table IV: Classification accuracy on basic 3 formats: ELL, CSR, HYB using feature set 1 consisting of 5 features which are computed in $\mathcal{O}(1)$ time. The best format(s) is shown in bold.

Machine	precision	decs. tree	SVM	MLP	XGBST
K80c	single	89%	88%	88%	**91%**
k80c	double	86%	87%	88%	**89%**
P100	single	85%	**89%**	87%	88%
P100	double	86%	87%	88%	**89%**

Table V: Classification accuracy on basic 3 formats: ELL, CSR, HYB using feature sets 1 and 2 consisting of 11 features used in Sedaghati et el. [9]

Machine	precision	decs. tree	SVM	MLP	XGBST
K80c	single	87%	88%	87%	**91%**
k80c	double	84%	87%	86%	**89%**
P100	single	86%	**88%**	86%	**88%**
P100	double	87%	87%	**89%**	**89%**

Table VI: Classification accuracy on basic 3 formats: ELL, CSR, HYB using feature sets 1, 2 and 3 consisting of 17 features

Machine	precision	decs. tree	SVM	MLP	XGBST
K80c	single	60%	62%	62%	**67%**
k80c	double	64%	63%	64%	**68%**
P100	single	65%	65%	67%	**69%**
P100	double	63%	65%	67%	**69%**

Table VII: Classification accuracy on 6 formats: COO, ELL, CSR, HYB, CSR5 and merged-based CSR using feature set 1 consisting of 5 features which are computed in $\mathcal{O}(1)$ time

Machine	precision	decs. tree	SVM	MLP	XGBST
K80c	single	81%	83%	83%	**85%**
k80c	double	81%	85%	85%	**88%**
P100	single	79%	83%	82%	**84%**
P100	double	81%	83%	84%	**86%**

Table VIII: Classification accuracy on 6 formats: COO, ELL, CSR, HYB, CSR5 and merged-based CSR using feature sets 1 and 2 consisting of 11 features used in Sedaghati et el. [9]

Machine	precision	decs. tree	SVM	MLP	XGBST
K80c	single	78%	83%	83%	**85%**
k80c	double	82%	85%	85%	**88%**
P100	single	79%	83%	82%	**84%**
P100	double	79%	83%	83%	**85%**

Table IX: Classification accuracy on 6 formats: COO, ELL, CSR, HYB, CSR5 and merged-based CSR using feature sets 1, 2 and 3 consisting of 17 features

Machine	precision	decs. tree	SVM	MLP	XGBST
K80c	single	79%	**85**%	83%	**85**%
k80c	double	83%	87%	86%	**88**%
P100	single	77%	83%	83%	**84**%
P100	double	79%	84%	85%	**86**%

Table X: Classification accuracy on 6 formats: COO, ELL, CSR, HYB, CSR5 and merged-based CSR using the top 7 (imp.) features according to XGBoost feature importance

C. Comparison of results on combined basic and advanced formats

In table VII, VIII, and IX, we present classification results by using six formats including advanced formats like CSR5 and merged-based CSR. We also add the COO format to give a comprehensive view of the consistency of our models. We observe that XGBoost consistently provides the best (or similar to one or more ML algorithms) classification accuracy across both machines and both data types. We also observe that by using additional 6 features, no improvement in accuracy is observed as shown in Table IX.

D. Insight

Figures 4 and 5 show the importance of each 17 features from the combined feature set 1, 2 and 3. This data is collected for the XGBoost classifier. We observe that on both machines and for both precisions (single and double), although the order of feature importance is different, the top 7 important features are same. We refer to these features as *imp. features* for rest of the paper. More surprisingly, an advanced feature like nnzb_tot is one of them which is crucial for the vector access pattern by a matrix and belongs to feature set 3.

We use these imp. features to examine the accuracy of our models. In Table X, we notice the accuracy achieved by using imp. features is equal or better than the previously best reported accuracy. This observation holds for XGBoost and other ML models. An exception is a single case for decision tree on P100 for single precision data type. In both 11 features and 7 features cases, the best achievable accuracy is 88% across all formats.

An important metric of a classification model is the average performance slowdown caused by mispredicted formats. An ideal model delivers a low number of cases with large slowdown as well as high classification accuracy. Tables XII, XI, XIII present the slowdown caused by mispredicted formats by using SVM, MLP and XGBoost, respectively. We observe that XGBoost outperforms all other formats in this case too.

VI. Results: SpMV performance modeling

In this section, we present an evaluation of ML based performance modeling for SpMV. Recent studies such as [18] have explored the concept of using ML techniques such as multi-layer perceptron (MLP) and support vector regression (SVR) to model SpMV performance. We draw inspiration from these studies and extend them by using an ensemble of MLP for the performance modeling task. We use an MLP ensemble to predict the performance of all 6 formats: COO, ELL, CSR, HYB, CSR5 and merge-based CSR and compare with base MLP based prediction. We use relative mean error (RME) to compare the performance different performance models. The RME metric can be defined as an average of relative error between predicted and measured performance:

$$RME = 1/n \sum_{i=1}^{n} \frac{|pred_i - measured_i|}{measured_i}$$

A. RME of combined models

In Figure 6, we show the RME of different ML methods using double precision on Tesla K80c and P100 GPU. We observe that the MLP ensemble is able to achieve an RME of 7% for CSR5 and merge based CSR. The result of using different ML methods using double precision on Tesla K80c and P100 GPU is shown in Figure 6.

We further tune the MLP and MLP ensemble regressor and achieve an average RME as low as 12% and 10% on P100 and K80c over 2300 matrices, respectively. A similar pattern is observed for the single precision evaluation. The MLP ensemble regressor achieves 5.4% improvement over regular MLP regressor on P100 for the double precision data type. On average across both machines and datatype, the MLP ensemble achieves an improvement of 3.5% in the overall RME.

B. RME of individual models

We previously used an MLP and an MLP ensemble for performance prediction on all 6 formats together. In this section, we evaluate the models when the individual formats are trained separately. We notice that the RME obtained by each format – including the advanced format like CSR5 and merge-based CSR – is low for both cases. This can be explained as both formats are insensitive to the irregularity of the matrix structure. CSR5 has a RME ranging from 11 - 13% for a double precision data type for Tesla K80c and P100 machine, respectively. For merge-based CSR the range is from 9 - 11%, respectively. The most widely used format in real applications, CSR, achieves an RME of 8% and 11% on Tesla P100 and K80c machine by using MLP ensemble regressor for double precision data type. Figure 7 shows the average RME of each format for double precision data type on Tesla K80c and P100 by using MLP ensemble regressor. We observe that the MLP ensemble regressor outperforms MLP in all the cases.

C. Indirect classification using regression

Previously, we observed that ensemble MLP technique achieves good performance prediction accuracy. This has an important implication on the possibility of using regression

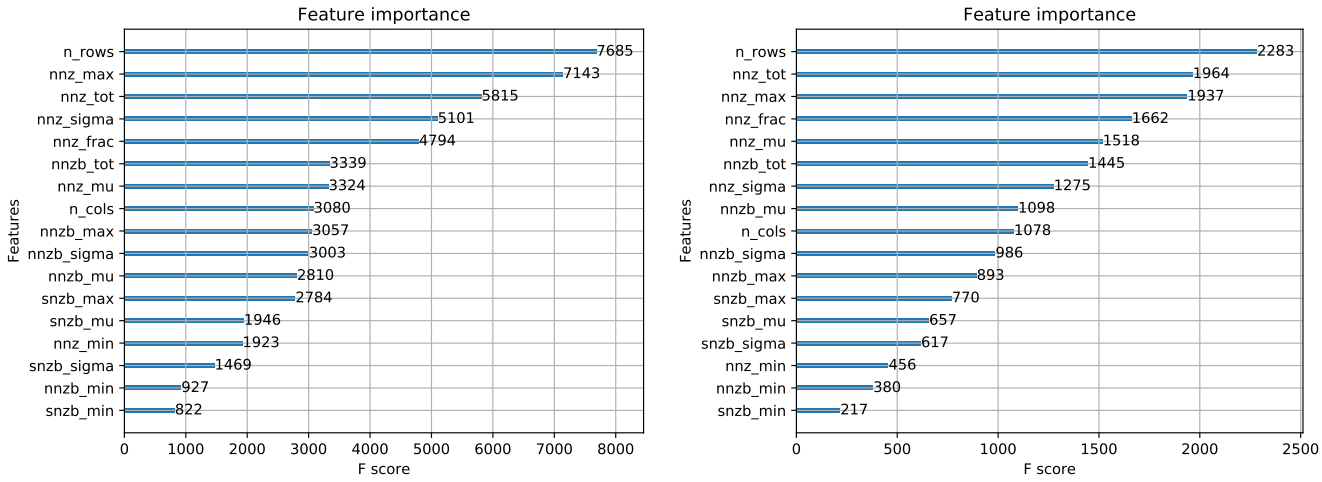

Figure 4: Importance of features using XGBoost on Tesla K80c and Tesla P100 machine with single precision

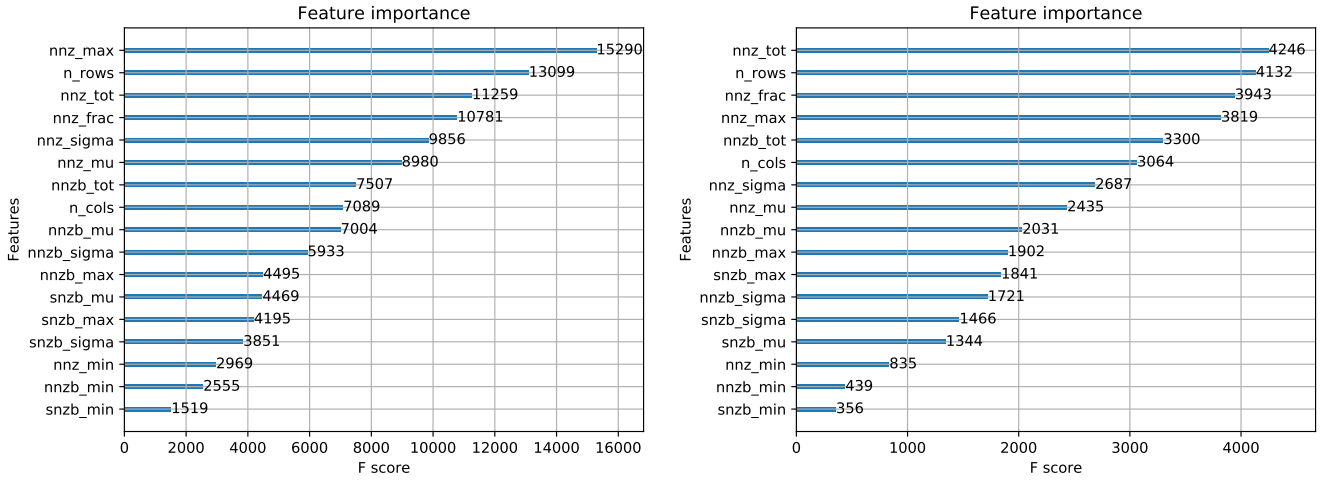

Figure 5: Importance of features using XGBoost on Tesla K80c and Tesla P100 machine with double precision

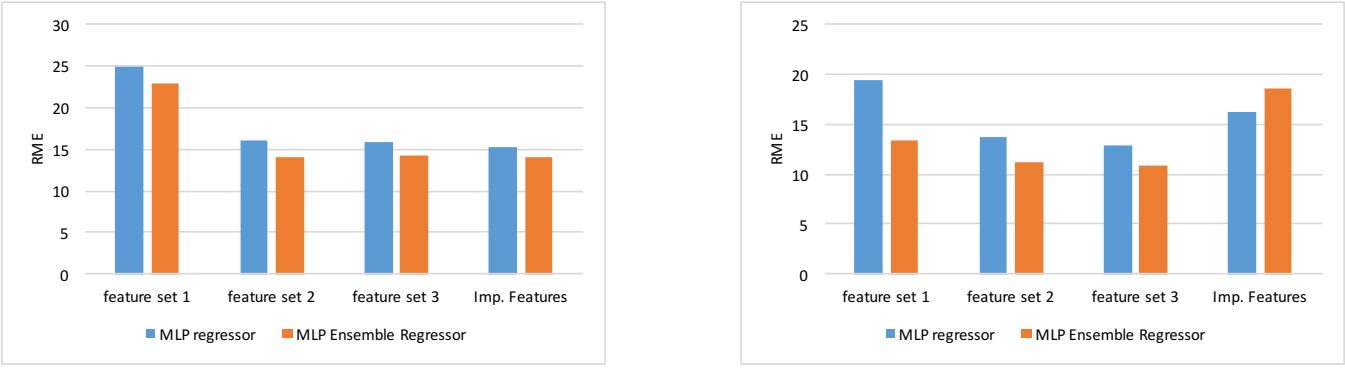

Figure 6: Average relative mean error (RME) of 6 formats using MLP and MLP ensemble regressor on Tesla K80c and Tesla P100 GPU using double precision data type

for format selection. Specifically, the format with the best predicted performance format can be selected as the best

feature set	no slowdown	>1x slowdown (cumulative)	>=1.2x Slowdown	>=1.5x Slowdown	>=2.0x Slowdown
1	285	175	89	61	25
2	444	16	12	3	1
3	447	13	10	2	1
Imp. Features	440	20	14	4	2

Table XI: Number of slowdown cases by using SVM on Tesla P100 machine for double precision data

feature set	no slowdown	>1x slowdown (cumulative)	>=1.2x Slowdown	>=1.5x Slowdown	>=2.0x Slowdown
1	293	167	84	58	25
2	441	19	14	4	1
3	439	21	15	5	1
Imp. Features	446	14	10	3	1

Table XII: Number of slowdown cases by using MLP ensemble on Tesla P100 machine for double precision data

feature set	no slowdown	>1x slowdown (cumulative)	>=1.2x Slowdown	>=1.5x Slowdown	>=2.0x Slowdown
1	274	186	92	65	29
2	446	14	10	3	1
3	446	14	10	3	1
Imp. Features	445	15	11	3	1

Table XIII: Number of slowdown cases by using XGBoost on Tesla P100 machine for double precision data

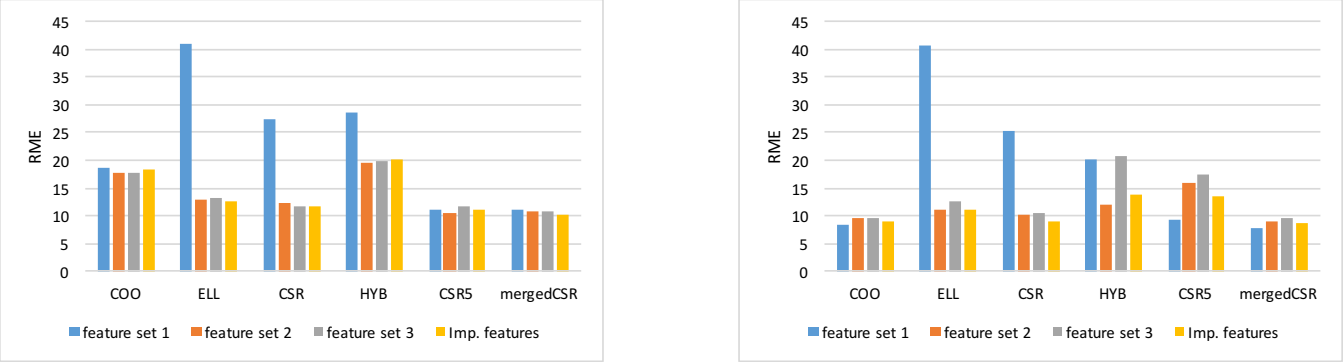

Figure 7: Relative mean error (RME) achieved by each 6 formats using MLP ensemble regressor on Tesla K80c and Tesla P100 GPU using double precision data type

storage format. The predicted format can then be compared with the actual best format (label) according to actual performance. We implement this approach and refer to it as the *indirect classification* method. Table XIV shows the result of using best direct classification technique – XGBoost – and indirect classification method.

For indirect classification, we define a *tolerance* parameter. Specifically, the regression predicts the SpMV execution time, which is a continuous variable. The tolerance parameter allows flexibility in selecting the best (or second/third best formats) as long as the prediction time of the other formats is within the tolerance. For example, under 5% tolerance, any format that has the performance within 5% of the best format would be considered as correctly classified. Naturally, a 0% tolerance is an extreme case.

In our evaluation, using 0% tolerance, we achieve a classification accuracy loss of 2% on K80c using double precision in comparison to the XGBoost based model. However, for single precision the classification accuracy decreases by 7%. On P100 machine, we observe a loss of 7% and 8% for single precision and double precision data, respectively. Using a tolerance of 5%, we can obtain a classification model equal or better than the direct classification method. In Table XIV, we see for K80c machine, we achieve the accuracy as 92% by using indirect classification.

VII. Related Work

Several researchers have proposed compression formats and associated algorithms with the compression formats. Our objective is to provide a brief overview of related research in selection of these formats and performance prediction of sparse matrices for these formats.

Machine	precision	XGBST	MLP ens.	MLP ens. 5% tol.
K80c	single	85%	78%	90%
k80c	double	88%	86%	92%
P100	single	84%	77%	89%
P100	double	86%	78%	87%

Table XIV: Classification accuracy achieved by XGBoost, MLP ensemble regressor and MLP ensemble regressor with 5% tolerance using 6 formats: COO, ELL, CSR, HYB, CSR and mergedCSR

Several storage formats like [4], [3], [19], [5], [20] have been proposed by the researchers. Benatia et al. [8] have proposed a classification technique using multi-class Support Vector Machine (SVM) classifier on manually extracted features. It achieves up to 88% classification accuracy on the four basic formats (COO, CSR, ELL, and HYB) with a dataset of 555 matrices from SuiteSparse [21] on NVIDIA Fermi and Maxwell GPUs. A popular machine learning techniques for classification - Decision Tree has been used in several recent studies [9], [10] etc. Like SVM classifier, Decision Tree also works on manually extracted features.

[9] used a dataset of 700 matrices and explored 5 storage formats including 4 basic formats and HYB_mu on NVIDIA Tesla k20c and k40c. It achieves 81% format classification accuracy. On the other hand, [10] uses more than 2000 matrices from the same ufl repository. On over 330 test matrices it achieves 85% and 82% classification accuracy respectively for single precision and double precision data with 1% accuracy gap. The model maintains a confidence value for each test sample. If the confidence factor is more than a threshold value, it executes the potential best formats and takes the final decision. Recently, Zhao et al. [12] and Cui et al. [11] proposed techniques using deep neural networks (DNNs) to solve the classification problem.

Zhao et al. [12] used a dataset of 9200 matrices (400GB) combined with real-world matrices collected from SuiteSparse repository and synthetic matrices. The matrices are first converted to fixed size (128x128) image-like representation and then used as an input into a convolutional neural network (CNN) model. It provides so far the best accurate model with 93% classification accuracy on CPU platform and 90% on GPU platform. On CPU it uses 4 formats: COO, ELL, CSR, and DIA. On GPU, the model can handle more formats like CSR5, HYB, and BSR. While SVM and DT require identifying manual feature extraction, CNN infers the features automatically. However, when compared to SVM and DT, CNN incurs a high inference time.

Similar approaches using ML techniques have been explored in the area of performance modeling. Benatia et al. [18] proposed to use multi-layer perceptron (MLP) and support vector regression (SVR) to predict the performance number of a SpMV operation. On average, it achieves low prediction error of 7% to 14% on a dataset of 1800 matrices. However, they de-activate the texture memory, which is critical to GPU performance. In addition, they use NVIDIA CUSP [22] library which is known to be sub-optimal, as better libraries such as cuSPARSE [23] have become prevalent. A unique analytical model based on probability mass function (PMF) is proposed by [6] to calculate the non zero distribution pattern of the sparse matrix. Using basic formats like COO, ELL, CSR and HYB, an average relative mean error (RME) of less than 20% is achieved for over 80% cases.

Zhang et al. [7] proposed a microbenchmark-based performance model by rigorous analysis of the underlying architecture of the GPU. By using a model based on the GPU's native instruction set, it predicts the performance within 5%-15% error range for blocked ELLPACK format. Zardoshti et al. [24] developed an adaptive run-time approach to identify the best format among four basic formats. It executes a small portion of the input matrix and tune it with GPU architectural parameters and chooses the best performing matrix based on the portion. Guo et al. [25] developed a inter-architecture analytical performance modeling tool using 4 generation of GPUs. On a set of 14 matrices, the model can predict time within 3% to 7% error range on average. Several auto-tuning framework use performance model like [26] to guide the tuner for optimized performance.

VIII. Conclusions

In this paper, we have proposed Machine Learning (ML) based approaches for format selection and performance modeling of SpMV kernel. We have used SuiteSparse dataset for our evaluation and analyzed it under the metrics of classification accuracy, relative mean error (RME), average slowdown and provided insight on results and critical feature selection. We have used ML models such as SVM, decision tree, multi-layer perceptron and ensembles (XGBoost) for evaluating the classification and regression problems. We observe that: 1) ensembles provide upwards of 88% classification accuracy on state-of-the-art formats, 2) prediction RME is $\approx$ 10% which is highly attractive for capacity planning purposes, 3) only seven features are sufficient for performance modeling and classification, 4) the same ML based model is the best for format selection independent of GPU architecture and precision used, and 5) indirect classification (using regression for format selection) while providing a maximum of 5% slowdown provides 92% classification accuracy – similar to the accuracy reported by using a CNN based model. We conclude that using a small set of features and relatively inexpensive ML algorithms, it is possible to achieve state-of-the-art format selection accuracy, making our solution attractive for compute-constrained (such as edge devices) practical deployments in general.

ACKNOWLEDGMENT

This work was supported in part by the U.S. National Science Foundation (NSF) through awards 1629548 and 1513120,

REFERENCES

[1] R. G. Grimes, D. R. Kincaid, and D. M. Young, *ITPACK 2.0 user's guide*. Center for Numerical Analysis, Univ., 1979.

[2] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," ser. SC '09.

[3] D. Merrill and M. Garland, "Merge-based sparse matrix-vector multiplication (SpMV) using the CSR storage format," in *PP0PP, 2016*.

[4] W. Liu and B. Vinter, "CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication," in *ICS, 2015*.

[5] S. Yan, C. Li, Y. Zhang, and H. Zhou, "yaSpMV: Yet another SpMV framework on GPUs," in *PPoPP, 2014*.

[6] K. Li, W. Yang, and K. Li, "Performance analysis and optimization for SpMV on GPU using probabilistic modeling," *TPDS, 2015*.

[7] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *HPCA, 2011*.

[8] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Sparse matrix format selection with multiclass SVM for SpMV on GPU," in *ICPP, 2016*.

[9] N. Sedaghati, T. Mu, L.-N. Pouchet, S. Parthasarathy, and P. Sadayappan, "Automatic selection of sparse matrix representation on GPUs," in *ICS, 2015*.

[10] J. Li, G. Tan, M. Chen, and N. Sun, "SMAT: an input adaptive auto-tuner for sparse matrix-vector multiplication," in *ACM SIGPLAN Notices, 2013*.

[11] H. Cui, S. Hirasawa, H. Takizawa, and H. Kobayashi, "A code selection mechanism using deep learning," in *MCSoC, 2016*.

[12] Y. Zhao, C. Liao, J. Li, and X. Shen, "Bridging the gap between deep learning and sparse matrix format selection," in *PPoPP, 2018*.

[13] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE TNNLS, 2014*.

[14] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent in function space." NIPS, 1999.

[15] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[16] (2016) XGBoost documentation. http://xgboost.readthedocs.io/en/latest/python. Accessed: 2018-02-15.

[17] "SVM documentation," http://scikit-learn.org/stable/modules/svm.html, 2017, accessed: 2018-02-15.

[18] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Machine learning approach for the predicting performance of SpMV on GPU," in *ICPADS, 2016*.

[19] M. Steinberger, R. Zayer, and H.-P. Seidel, "Globally homogeneous, locally adaptive sparse matrix-vector multiplication on the GPU," in *ICS, 2017*.

[20] B.-Y. Su and K. Keutzer, "clspmv: A cross-platform OpenCL SpMV framework on GPUs," in *ICS, 2012*.

[21] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *TOMS, 2011*.

[22] S. Dalton, N. Bell, L. Olson, and M. Garland, "CUSP: Generic parallel algorithms for sparse matrix and graph computations," 2014, version 0.5.0. [Online]. Available: http://cusplibrary.github.io/

[23] C. NVIDIA, "Cusparse library," *NVIDIA Corporation, Santa Clara, California*, 2014.

[24] P. Zardoshti, F. Khunjush, and H. Sarbazi-Azad, "Adaptive sparse matrix representation for efficient matrix–vector multiplication," *JoS, 2016*.

[25] P. Guo and L. Wang, "Accurate cross–architecture performance modeling for sparse matrix–vector multiplication (SpMV) on GPUs," *CCPE, 2015*.

[26] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *ACM sigplan notices, 2010*.