Performance Prediction and Ranking of SpMV Kernels on GPU Architectures

Christoph Lehnert[1], Rudolf Berrendorf[1](✉), Jan P. Ecker[1], and Florian Mannuss[2]

[1] Computer Science Department, Bonn-Rhein-Sieg University of Applied Sciences, Sankt Augustin, Germany
{christoph.lehnert,rudolf.berrendorf,jan.ecker}@h-brs.de

[2] EXPEC Advanced Research Center, Saudi Arabian Oil Company, Dhahran, Saudi Arabia
florian.mannuss@aramco.com

Abstract. Predicting the runtime of a sparse matrix-vector multiplication (SpMV) for different sparse matrix formats and thread mappings allows the dynamic selection of the most appropriate matrix format and thread mapping for a given matrix. This paper introduces two new generally applicable performance models for SpMV – for linear and non-linear relationships – based on machine learning techniques. This approach supersedes the common manual development of an explicit performance model for a new architecture or for a new format based on empirical data. The two new models are compared to an existing explicit performance model on different GPUs. Results show that the quality of performance prediction results, the ranking of the alternatives, and the adaptability to other formats/architectures of the two machine learning techniques is better than that of the explicit performance model.

Keywords: SpMV · Performance prediction · Linear regression · Gradient-boosting · KNN · Instance-based learning

1 Introduction

The sparse matrix-vector multiplication (SpMV) is the most time-consuming operation in iterative solvers [14]. Improving the efficiency of these operations is therefore important in many application fields [2], and many papers have been published on different sparse matrix formats and related SpMV implementations. Besides handling the sparsity as such, some formats try to utilize additional structural properties of a matrix. For example, work has been done on formats that do not rely on certain properties of a matrix and are therefore generally applicable, e.g., CSR [14], SELL-C-σ [8]. Additional research has been conducted on other formats that take advantage of the matrix structure (e.g., BCSR [7]) and/or a target architecture (e.g., supported formats in the Intel MKL and Nvidia cuSPARSE). Even if one format and target architecture is fixed, some formats/architectures have additional parameters, e.g., a slice size in the SELL-C-σ format or grid/block size on

P.-F. Dutot and D. Trystram (Eds.): Euro-Par 2016, LNCS 9833, pp. 90–102, 2016.
DOI: 10.1007/978-3-319-43659-3_7

Graphics Processor Units (GPUs) that need to be optimized for a given matrix or an architecture.

However no sparse matrix format performs best for all matrices and all processor architectures. Instead the formats differ significantly in their performance between formats for certain matrix structures or when switching between architectures. Choosing the right data format, parameter values, and possibly even the target architecture for a given matrix is therefore extremely important.

Simple and fast heuristics have been developed (see related work in Sect. 2) that try to predict which format or parameter values should be used for a given matrix based on few parameters than can be efficiently determined with low overhead. These heuristics have to be adapted to new formats, possibly to new sparsity patterns, and if possible to new architectures. Instead of manually developing new heuristics for new configurations, the question is whether it is possible to develop more general techniques to rank SpMV alternatives through runtime prediction. These techniques could then be used as a basis for deciding on a configuration for a given matrix.

This paper investigates three modeling techniques to predict the runtime of an SpMV operation on a GPU. Based on such predictions, a ranking of alternatives is possible that lets a program choose the best ranked alternative or, dependent on external parameters (prefer architecture X), choose the best ranked alternative after applying a filter at runtime.

One runtime prediction technique used in this work is based on the work of Guo et al. [6] and uses special benchmark matrices that determine two parameters in a simple linear model specific to some basic matrix formats. These performance models themselves are specific to a matrix format and are therefore not transferable to other formats. The other two newly developed models proposed in this paper use machine learning techniques and are more general. These two models differentiate between whether a mostly linear relationship exists between features and the SpMV runtime (using regression techniques) or whether non-linear relationships have a non-neglecting influence on the runtime (then using KNN). The three different approaches are evaluated using a set of sparse matrices and 5 different matrix formats: 3 basic ones that were used already in other papers (COO, CSR, ELL [14]) and 2 complex ones not used in such investigations before (SELL-C-σ [8], ELL-BRO [17]).

The paper is organized as follows. Section 2 gives an overview on related work. Section 3 introduces the three performance modeling methods. Then Sect. 4 discusses in detail the evaluation and reliability of the runtime prediction/ranking. The paper closes with a summary.

2 Related Work

Some papers [1,3,8,9,13] deal with simple explicit heuristics or provide empirical modeling for a Central Processing Unit (CPU) or GPU. This allows a program to choose at runtime which matrix format and/or which parameter values to use. This type of system is mainly used for autotuning. Those papers work with empirically derived fixed heuristics that work reasonable well for the supported simple matrix formats/architectures but need to be adapted for new and more complex matrix formats and/or architectures.

Xu et al. [18] claim that the SpMV is a memory-bounded problem. They suggest a prediction concept that estimates the memory access times needed to load and store the matrix and vector data for the SpMV operation. Although SpMV is memory bound, there are others factors that influence the runtime of this operation as well. Li et al. [10] published a probabilistic approach that takes into account the distribution of non-zero elements in each row of the sparse matrix. They use the model for the runtime prediction of the (simple) CSR, COO, ELL and HYB formats. Sedaghati et al. [15] use decision tree algorithms to select the most suitable format for a specific sparse matrix. Similar to our work, they use machine learning techniques, but with important differences. Their matrix formats are those supported by the vendor library and are fairly regular/simple. We used in addition more complex formats (SELL-C-σ, ELL-BRO), that have a more complex performance behavior. Additionally we allow a choice not only between the matrix formats but also between format/architecture parameter values for a specific format.

Guo et al. [6] uses profiled data for benchmark matrices and a simple 2-parameter linear model that is discussed and evaluated in detail in Sects. 3.1 and 4. Offline benchmark matrices and a heuristic performance model was also used in Lee et al. [9] for the CSR matrix format only.

3 Performance Modeling

This work aim to develop performance models that are not explicitly specific to a certain matrix format or a processor architecture but are more generally applicable and do not need to be reworked for new configurations. We describe in more detail below an existing benchmarking-based approach and then introduce a linear regression technique and the k-nearest neighbors approach.

Table 1 summarizes all features of the platform and matrices we found to be relevant for predicting performance in any of the described models on GPUs. Further features were analyzed, but no relevance for the runtime prediction could be identified. For some formats, there are several format specific parameters that may also influence the runtime of the SpMV operation. Although we have not included such format parameters, our work could be extended in that direction.

3.1 Benchmarking-Based Approach

The benchmarking approach we use in this paper is based on the work of Guo et al. [6]. Their prediction method consists of two major phases. In the instrumentation phase, platform information is gathered including the number of available streaming multiprocessors (SMs) and the maximum number of threads that can be processed by each SM at once. These are used to compute the so-called strip size. A strip is a maximum submatrix that can be computed by the

Table 1. Platform and matrix features that are relevant for the presented approaches.

Feature	Description
blocksize	The CUDA blocksize
nRows	The number of rows (the dimension of the square matrix)
nnz	The overall number of non zeros of the matrix
minNnz	The minimum number of non-zero elements per row among all rows
maxNnz	The maximum number of non-zero elements per row among all rows
modeNnz	The statistic mode of the number of non-zero elements per row among all rows
medianNnz	The median of the number of non-zero elements per row among all rows
minDist	The minimum distance between non-zero elements per row among all rows
maxDist	The maximum distance between non-zero elements per row among all rows
bandwidth	The maximum distance of non-zero elements from the diagonal
dispersion	The standard deviation of the numbers of non-zero elements among all rows
density	The fraction of the non-zero elements in the total number of array elements

GPU in one iteration if the full parallelism is used. The number of matrix rows that fit into a strip is therefore different for each of the used formats. According to Guo et al., one strip can be calculated in one step (simplified); therefore the number of strips a matrix consists of has a major influence on the runtime. Another important attribute in this model is the average number of non-zero elements in the matrix rows. The execution of synthetical special benchmark matrices is used to determine the relation between the number of strips and the average number of non-zero elements per row to the execution time. The SpMV operation is executed on multiple benchmark matrices:

- with a fixed size and an increasing number of non-zeros per row
- with a fixed number of non-zeros per row and an increasing number of strips
- multiple runs of matrices with (different) fixed numbers of non-zeroes per row and an increasing number of strips (ELL format only)

The information gathered from these SpMV executions is used to create a 2-parameter linear model [6]. In a program run, this model is parametrized with the information from the target matrix, and the actual runtime is predicted. This model is different for every matrix format and for a new matrix format an appropriate linear relationship has to be explicitly specified.

3.2 Linear Regression Techniques

Algorithms based on machine learning techniques are used in various scientific fields. One such machine learning approach is linear regression. It is a technique that relates a response vector of training instances, for example the measured SpMV runtimes of several matrices, to the features of the matrix by assuming that a linear relationship exists. The goal is to determine the relevance of these predictors and apply linear coefficients to each of them so that the best approximation of the responses for all (training) instances can be obtained. This goal can be achieved by iteratively refining the linear regression model to minimize the residual error between the modeled values and the actual responses [12].

Basic Model. In this step, the linear relation for the first training data record (feature values and SpMV runtime of the first matrix) is established. The second record is added, and the coefficients that have been identified in the first step are adjusted until the squared error between the estimations of both instances (by using the common coefficients) and the actual response is minimal. This process is repeated for all training data records. The result of this training phase is a linear model that leads to the best approximated responses for all training instances. Its particular coefficients ω_i can be utilized to predict the result T_{test} for all new test instances with feature values α_i. This prediction is obtained by using Eq. (1) for an instance with m features. A linear intercept ω_0 also results from the procedure described in [12].

$$T_{\text{test}} = \omega_0 + \omega_1 \times \alpha_1 + \omega_2 \times \alpha_2 + ... + \omega_m \times \alpha_m \tag{1}$$

Model Enhancements. This regression technique is applicable if linear relations exist between the features. For a SpMV operation, many but not all features have such properties. An open question at the beginning of the research was whether the linear features dominate the runtime or whether a linear model is not suitable because the influence of non-linear features is too high.

The relationships between the matrix formats are certainly not linear, and the thread mapping also behaves non-linearly. Therefore in the training phase, distinct models are generated for different matrix formats and thread mappings. First investigations have also shown that selecting proper features per format (and only those) is an essential step. Table 2 presents the selected subset of features for each format. Other features did not positively influence the quality of the predictions. In a next step in the training phase, the feature values were logarithmized. These steps had proved to be sufficient to accurately predict SpMV runtimes for the simple formats COO, CSR and ELL.

Gradient-Boosting. For the more complex matrix formats SELL-C-σ and ELL-BRO, the prediction quality was less good. To be able to predict with a high accuracy the SpMV runtimes even in cases where (1) some non-linear relationship exists between the matrix features and SpMV execution times for a specific

Table 2. Features selection for the approaches linear regression/gradient-boosting (LR) and k-Nearest Neighbors (KNN) and the selected formats.

Feature	COO		CSR		ELL		SELL-C-σ		ELL-BRO	
	LR	KNN	LR	KNN	LR	KNN	LR	KNN	LR	KNN
blocksize	X	X	X	X	X	X	X	X	X	X
nRows	X		X	X	X					
nnz	X	X		X	X	X	X	X	X	X
minNnz	X		X			X	X	X	X	X
maxNnz	X				X	X	X		X	
modeNnz	X		X				X		X	
medianNnz	X		X	X	X		X		X	
minDist	X					X	X		X	
maxDist			X		X	X				
bandwidth	X		X		X		X			
dispersion	X				X	X	X	X	X	
density			X	X			X		X	

format or (2) important features for this format have not been identified and can thus not be represented in the modeling process, a gradient-boosting technique was chosen as an alternative to the linear regression model. This technique was then used for the formats SELL-C-σ and ELL-BRO.

Gradient-boosting [5] is also a regression technique. To determine the responses of completely new instances, a general function is approximated using all available training samples. Here, the search is for the concrete function that leads to a minimal estimated error among all training instances. As explained in detail in [5], gradient-boosting approaches start with a simple and often weak approximating model that only fits a small number of the training instances and iteratively refine it by applying the same *base learner* to the previous intermediate results. In our approach, the base learner is a regression tree, parametrized using several factors including its depth and the splitting rules at each non-terminal node. The model is built by roughly fitting the training instances by initially using a simple tree. The result is iteratively refined by applying further regression trees on the particular residual errors of the prior iteration and combining the intermediate results to a final complex model.

3.3 k-Nearest Neighbors

The k-Nearest Neighbors (KNN) [11] approach belongs to the instance-based learning methods that are suitable for both regression and classification tasks. The general idea behind such techniques is not to determine and store a concrete calculated function but to compare a new instance to all training records or a subset of them. The desired response value for the new instance is then retrieved by identifying similar training instances and accounting for their individual responses.

The KNN algorithm compares the feature values of the new instance with those of the training instances. A distance measure is chosen to select the k training records that are closest to the new instance, presuming that they are the most similar ones among all training instances. For a regression problem, the response value is (for example) retrieved by computing the mean of the k neighbors responses [11].

To successfully use the KNN algorithm, an appropriate distance function as well as a proper k-value have to be accommodated to different problem areas. Further enhancements of the KNN technique are the distance-weighted and the feature-weighted KNN approach. When using the first one, the calculated distances to the neighbors of a new instance are weighted. While neighbors with a low distance have a high impact on the calculated value, the impact decreases the greater the distances are [11]. The second technique applies weights to each feature.

Model Specifics. We achieved the most accurate runtime predictions by using k-values of 4 for the COO, CSR, ELL and BRO-ELL formats and 2 for the SELL-C-σ format. These differences are caused by the provided training sets per kernel. Using a k-value of 4 for the SELL-C-σ format leads to higher inaccuracies for a few test matrices, because only few neighbors for this kernel exists for those matrices. Taking more neighbors into account diminishes the quality of the runtime estimations. This problem can be dealt with by providing a bigger set of training instances with a higher variety of features. As a distance function, the euclidean distance was used. Manually derived weights were used for the preselected features shown in Table 2. The KNN approach has been shown to be much more sensitive to the correct selection of features compared to the linear regression approach. Furthermore, the feature values have been 0-1-normalized since the data ranges of the features are too different. The distance weighting was realized by using the inverse of the euclidean distance for computing the neighbors contribution to the calculation of the response variables, i.e., the estimated runtimes.

4 Evaluation

In this section, all three approaches are evaluated. First the evaluation methodology will be explained, followed by the evaluation of the runtime prediction quality and the ranking quality.

4.1 Evaluation Methodology

The training and evaluation of the prediction approaches requires the measurement of the runtimes of the different SpMV kernels on a GPU. The runtimes do not include the data transfer times to the GPU. The measurements are performed on GPUs with two different architectures (Nvidia M2050 and K80). Only one of the two GPUs of the K80 is used. To minimize measurement inaccuracies and startup overhead, all SpMV operations are executed 200 times and the median is always used as the actual runtime.

Table 3. Set of used test matrices with some additional structure information.

#	Matrix	Rows/columns	nnz	Bandwidth	nnz per row			
					min	max	mode	med
1	`Ga41As41H72`	268096	18488476	22519	18	702	37	37
2	`PR02R`	161070	8185136	84250	1	92	66	66
3	`Si34H36`	97569	5156379	18908	17	494	37	37
4	`crankseg_2`	63838	14148858	61047	48	3423	195	195
5	`nd6k`	18000	6897316	16766	130	514	468	416
6	`TSOPF_RS_b2383`	38120	16171169	33353	2	983	4	6
7	`tmt_sym`	726713	5080961	1921	3	9	7	7
8	`af_1_k101`	503625	17550675	859	15	35	35	35
9	`af_shell1`	504855	17588875	4909	20	40	35	35
10	`gsm_106857`	589446	21758924	588744	12	106	32	32
11	`matrix_spe1Ref_a`	900000	18612000	17999	12	21	21	21
12	`boneS10`	914898	55468422	8969	12	81	81	66
13	`atmosmodd`	1270432	8814880	21904	4	7	7	7
14	`kkt_power`	2063494	14612663	2046911	1	96	3	3
15	`memchip`	2707524	14810202	1647939	2	27	4	5
16	`Flan_1565`	1564794	117406044	20702	24	81	81	81
17	`circuit5M_dc`	3523317	19194193	2832158	1	27	4	5
18	`matrix_spe10_dpdp_a`	3506080	50928264	3378961	2	16	16	16

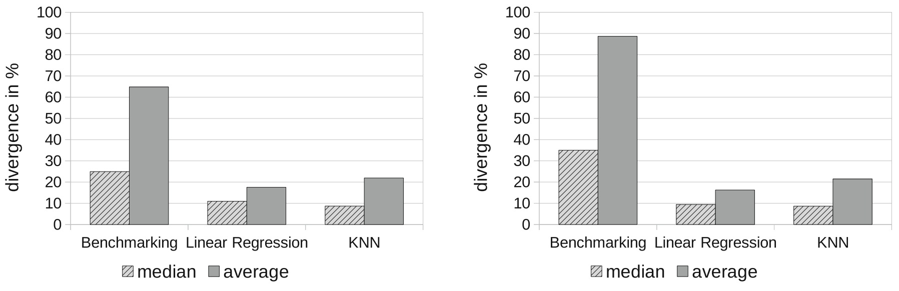
A set of 74 square matrices is used for the training and evaluation process. Table 3 shows the 18 test matrices that are used as target- or test matrices where the runtime must be predicted. The other matrices are the training matrices. The matrices show a wide variety of feature values. All matrices originate from the University of Florida Sparse Matrix Collection [4] or the SPE Comparative Solution Project [16]. For the SELL-C-σ format, a fixed C-value of 512 and σ-value of 2048 are used for all measurements. Likewise for the ELL-BRO format, a slice size of 256 and symbol size of 64 bit are used.

An exhaustive search on the available formats and valid thread mappings was performed for each matrix to determine the best achievable runtime for that matrix and the corresponding format and thread mapping. Only the three basic formats CSR, ELL and COO are used for the benchmarking-based approach, while all five formats are used for the two machine learning approaches.

4.2 Prediction Quality

The prediction quality is determined by comparing the predicted runtime with the actual measured runtime for the same format and same thread mapping.

This was done with all matrices of Table 3 using all supported formats and a large set of thread mappings, in total about 8600 test instances for the machine learning approaches and about 6100 ones for the benchmarking approach. Figures 1a and b present the divergence between the predicted and measured runtimes over all test instances of all three approaches on the M2050 and K80. They show that the median divergence of the machine learning approaches is very low at around 10 %. The average divergence of both these approaches is higher due to some inaccurate predictions.



(a) M2050 (b) K80

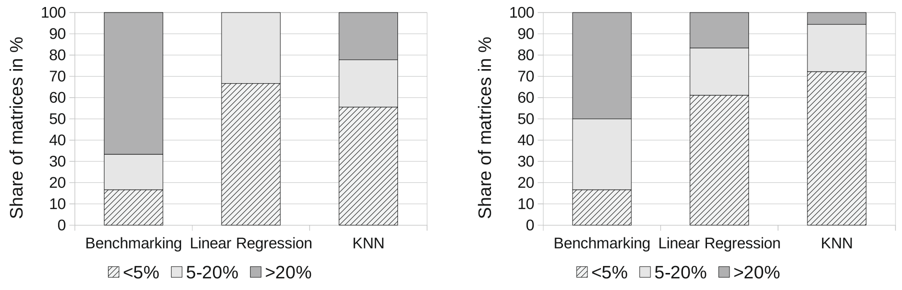
Fig. 1. Comparison of the predicted runtimes and the actual measured runtimes.

The median divergence of the linear regression approach is higher compared to the KNN approach, but its average is significantly better. This difference is caused by an overall smaller number of predictions with greater inaccuracy. The few high over- and underestimations of the KNN approach can be found in a small number of matrices. Examples are matrices `nd6k`, `circuit5M_dc` and `kkt_power`. They have in common that some of their features have the smallest values among all available training instances. The number of rows of matrix `nd6k`, for example, is much smaller than those of all training matrices. Since this value is crucial for estimating the CSR-runtime, training instances with a significantly higher number of rows may be chosen as the nearest neighbors, resulting in an imprecise estimation. This problem can be dealt with by extending the training set and including smaller matrices.

The figures also clearly show that the performance of the benchmarking-based approach is much worse. The median divergence is about 25 % and the average accuracy only reaches 65 %. On the newer architecture of the K80, the benchmarking-based approach performs even worse, which could indicate that the relatively simple model is less suitable for the newer and more complex Kepler architecture. In summary, both machine learning approaches deliver a very high prediction quality with median divergences of around 10 %.

4.3 Ranking Quality

The runtime prediction is used as a tool, for ranking SpMV alternatives for a given matrix. Even with a inaccurate prediction, correct ranking could still be possible, e.g., with a continuous over estimation as long as the ranking order is still correct. The evaluation of the quality of the ranking is done by comparing the measured runtime of the predicted first ranked configuration with the overall best measured runtime for that matrix over all configurations (formats and thread mappings).



(a) M2050 (b) K80

Fig. 2. Share of matrices where the divergence between the reached runtime and the overall best runtime is between certain values.

Figure 2a presents the share of predictions where the divergence is below 5 %, between 5 % and 20 %, and over 20 % for all approaches on the M2050. The figure shows that the machine learning approaches again deliver in total very good results, and much better results than the benchmarking-based approach. For the majority of matrices, the predictions of the machine learning approaches result in runtimes that are only less than 5 % slower than the best possible runtimes. There was also no prediction of the linear regression/gradient-boosting approach, which was more than 20 % slower than the optimal runtime. The real runtimes of the benchmarking approach are more than 20 % slower than the optimum in most of the cases.

Figure 2b presents the same comparison for the K80 GPU. The linear regression on the K80 performs slightly worse than that on the M2050 and the KNN, and the benchmarking approach perform slightly better than that on the M2050. A more detailed comparison of the data revealed more details not easily presentable in plots:

- The benchmarking approach delivers very inconsistent results, and the runtimes for the same matrix on different architectures vary greatly.
- The quality of the predictions for a specific matrix is very consistent for the linear regression and the KNN approaches and mostly independent of the used architecture.

– The KNN approach delivers very good predictions for the majority of matrices, but also some highly inaccurate predictions with up to around 160 % slower runtimes. The linear regression delivers better average predictions and no such extreme outliers.

4.4 Other Aspects

Table 4 shows the durations of the training-, modeling- and ranking-phases for all approaches. The ranking was done for the matrix `af_1_k101`. While the benchmarking-based and the pure linear regression and combined linear regression/gradient-boosting approach each have a quite extensive offline training phase, the online prediction of a runtime itself is simple and fast (evaluating a linear function with given coefficients). The KNN approach, however, has no training phase but a quite complex modeling phase for each single runtime prediction since the neighbors among all existing training matrices have to be identified and incorporated into the prediction value.

Table 4. Overhead (in msec). Results obtained by using *R*-tools are marked with *.

Phase	Benchmarking	LR	LR/Grad. boosting	KNN
Training (offline)	2,601,042	130.7*	4,078*	-
Modeling & ranking	0.001	0.001	0.013	153.4*

Table 5. SpMV-runtimes (in msec) per format of matrix `af_1_k101`

Format	BRO-ELL	SELL-C-σ	ELL	CSR	COO
Runtime (msec)	1.029	1.187	1.362	2.861	7.571

Table 5 shows the measured SpMV execution times for the same matrix using different formats. A comparison of the two tables shows that the modeling/ranking-phases for all approaches other than KNN are even faster than one SpMV execution with the most appropriate kernel for the given matrix. Besides predicting an adequate format, the approaches can also be used for selecting a proper thread mapping, simply by calculating the predictions for a set of reasonable thread mappings. Regarding the prediction overhead, this process is suitable for the benchmarking-based and linear regression techniques, whereas the overhead for the KNN approach is quite high.

5 Summary and Outlook

We developed two new general models for performance prediction and ranking of SpMV kernels on GPUs and compared these to a known explicit linear model. The two new models both show better prediction and ranking results than the simple explicit model. The linear regression model is most appropriate if linearity of parameters dominates. This was the case for simple/regular sparse matrix formats. The gradient-boosting regression technique could also handle the two more complex formats. If parameters show non-linearity, a KNN model is more suitable. This model delivers better results for formats with more complex performance behavior, but it has a higher runtime overhead. Based on the prediction, we were also able to use ranking to determine the most suitable format and architecture parameters for a given matrix.

Our high quality results were only available after we fitted the general model more specifically to the concrete problems; this procedure is common when using these techniques. The procedure includes the proper selection of relevant features and weights and the separation of models with respect to formats and thread mapping. The approaches themselves are transferable to other formats and (GPU) architectures.

There are several opportunities to further improve our models. For the linear regression, we used a least-square/gradient-boosting approach. Here different regression techniques might deliver an even better quality. For the KNN approach, the weights could be determined by the system itself. Applying the models to a CPU-based systems would also be of interest.

Acknowledgements. We would like to thank the CMT team at Saudi Aramco EXPEC ARC for their support and input. Especially we want to thank Ali H. Dogru for making this research project possible. Additionally we appreciate the discussions on modeling with Marlis von der Hude and Peter Becker.

References

1. Ashari, A., Sedaghati, N., Eisenlohr, J., Sadayappan, P.: An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on GPUs. In: Proceedings of the 28th ACM International Conference on Supercomputing (ICS 2014), pp. 273–282. ACM (2014)
2. Berrendorf, R., Weierstall, M., Mannuss, F.: Program optimization strategies to improve the performance of SpMV-operations. In: Proceedings of the 8th International Conference on Future Computational Technologies and Applications, pp. 34–40 (2016)
3. Choi, J.W., Singh, A., Vuduc, R.W.: Model-driven autotuning of sparse matrix-vector multiply on GPUs. In: Proceedings of Principles and Practices of Parallel Programming (PPoPP 2010), pp. 115–125. ACM, January 2010
4. Davis, T.A., Hu, Y.: The University of Florida Sparse Matrix Collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2010)
5. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**, 1189–1232 (2000)

6. Guo, P., Wang, L., Chen, P.: A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on a GPUs. IEEE Trans. Parallel Distrib. Syst. **25**(5), 1112–1123 (2014)
7. Im, E.J., Yelick, K., Vuduc, R.: Sparsity: optimization framework for sparse matrix kernels. Int. J. High Perform. Comput. Appl. **18**(1), 135–158 (2004)
8. Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A.R.: A unified sparse matrix data format for efficient general sparse matrix-vector multiply on modern processors with wide SIMD units. SIAM J. Sci. Comput. **26**(5), C401–C423 (2014)
9. Lee, B.C., Vuduc, R.W., Demmel, J.W., Yelick, K.A.: Performance models for evaluation and automatic tuning of symmetric spare matrix-vector multiply. In: Proceedings of the International Conference on Parallel Processing, vol. 1, pp. 169–176. IEEE (2004)
10. Li, K., Yang, W., Li, K.: Performance analysis and optimization for SpMV on GPU using probalistic modeling. IEEE Trans. Parallel Distrib. Syst. **26**(1), 196–205 (2015)
11. Mitchell, T.M.: Machine Learning, vol. 1. McGraw-Hill, Singapore (1997)
12. Murphy, K.P.: Machine Learning: A Probabilistic Perspective, vol. 1. The MIT Press, Cambridge (2012)
13. Neelima, B., Reddy, G., Raghavendra, P.: Predicting an optimal sparse matrix format for SpMV computation on GPU. In: Proceedings of International Parallel & Distributed Processing Symposium Workshops (IPDPSW 2014), pp. 1427–1436. IEEE (2014)
14. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
15. Sedaghati, N., Mu, T., Pouchet, L.N., Parthasarathy, S., Sadayappan, P.: Automatic selection of sparse matrix representation on GPUs. In: Proceedings of the 25th International Conference on Supercomputing (ICS 2015). ACM (2015)
16. Society of Petroleum Engineers. http://www.spe.org/web/csp/: SPE Comparative Solution Project
17. Tang, W., Tan, W., Ray, R., Wong, Y., Chen, W., Kuo, S., Goh, R., Turner, S., Wong, W.: Accelerating sparse matrix-vector multiplication on GPUs using bit-representation-optimized schemes. In: Proceedings of Intrnational Conference on High Performance Computing, Networking, Storage and Analysis (SC 2013). ACM (2013) (article no. 26)
18. Xu, S., Xue, W., Lin, H.X.: Performance modeling and optimization of sparse matrix-vector multiplication on NVIDIA CUDA platform. J. Supercomput. **63**(3), 710–721 (2011)