Mohammed, Thaha; Albeshri, Aiiad; Katib, Iyad ; Mehmood, Rashid

DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems

Published in:
Journal of Supercomputing

DOI:
10.1007/s11227-020-03489-3

Published: 01/06/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Mohammed, T., Albeshri, A., Katib, I., & Mehmood, R. (2021). DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems. *Journal of Supercomputing*, *77*(6), 6313–6355. https://doi.org/10.1007/s11227-020-03489-3

DIESEL: A Novel Deep Learning based Tool for SpMV Computations and Solving Sparse Linear Equation Systems

Thaha Mohammed
Department of Computer Science
Aalto University
Espoo , 02150, Finland
thaha.mohammed@aalto.fi

Aiiad Albeshri
Department of Computer Science
King Abdulzaziz University
Jeddah 21589, Saudi Arabia
aaalbeshri@kau.edu.sa

Iyad Katib
Department of Computer Science
King Abdulzaziz University
Jeddah 21589, Saudi Arabia
iakatib@kau.edu.sa

Rashid Mehmood
High Performance Computing Center
King Abdulzaziz University
Jeddah 21589, Saudi Arabia
RMehmood@kau.edu.sa

Accepted: 2020

Abstract

Sparse linear algebra is central to many areas of engineering, science and business. The community has done considerable work on proposing new methods for sparse matrix-vector multiplication (SpMV) computations and iterative sparse solvers on graphical processing units (GPUs). Due to vast variations in matrix features, no single method performs well across all sparse matrices. A few tools on automatic prediction of best-performing SpMV kernels have emerged recently and require many more efforts to fully utilize their potential. The utilization of a GPU by the existing SpMV kernels is far from its full capacity. Moreover, the development and performance analysis of SpMV techniques on GPUs have not been studied in sufficient depth. This paper proposes DIESEL, a deep learning-based tool that predicts and executes the best performing SpMV kernel for a given matrix using a feature set carefully devised by us through rigorous empirical and mathematical instruments. The dataset comprises 1056 matrices from 26 different real-life application domains including computational fluid dynamics, materials, electromagnetics, economics, and more. We propose a range of new metrics and methods for performance analysis, visualization and comparison of SpMV tools. DIESEL provides better performance with its accuracy 88.2%, workload accuracy 91.96%, and average relative loss 4.4%, compared to 85.9%, 85.31%, and 7.65% by the next best performing artificial Intelligence (AI) based SpMV tool. The extensive results and analyses presented in this paper provide several key insights into the performance of the SpMV tools and how these relate to the matrix datasets and the performance metrics, allowing the community to further improve and compare basic and AI-based SpMV tools in the future.

Keywords Sparse Linear Algebra · Sparse Linear Equation Systems · Sparse Matrix Vector Product (SpMV) · Iterative Solvers · Graphics Processing Units (GPUs) · Artificial Intelligence · Deep Learning

1 Introduction

Linear algebra is central to many areas of engineering, science, economics, business, and social sciences, particularly sparse linear algebra that is included by the Berkeley scientists in their set of motifs, the seven dwarfs [1]. Sparse linear systems have a significantly large proportion of zeroes in the system due to reasons such as system symmetry, autonomous nature of its subsystems, etc. Sparse linear algebra exploits these system properties and uses specialized

storage schemes and algorithms so as to efficiently store, access, and compute sparse matrices. Among the sparse algebra methods, iterative solution of linear systems are of prime importance due to its application in many important areas including partial differential equation (PDEs), high accuracy surface modelling, Markov Chains and Markov Decision Processes (MDPs), web ranking, queuing systems, fault modelling, weather forecasting, telecommunication systems, sensor networks, natural language modelling, computational biology, computational physics, computational chemistry, big data, smart cities, deep learning, and many more. We will see later that the 1056 matrices used in this paper belong to 26 application domains of sparse linear algebra and this clearly manifests the significance of this problem [2].

Sparse linear systems are generally of the form $Ax = y$, where A is a sparse matrix and y is a dense vector and x is the solution of the system that needs to be obtained. The system $Ax = y$ can be solved by direct methods or iterative methods. Direct methods suffer from the fill-in problem as they require modifying the matrix multiple times during the solution process, and, hence, iterative methods are usually preferred. Sparse Matrix-Vector multiplication (SpMV) is a key operation in iterative solution methods for linear equation systems. Iterative solvers such as Jacobi and Gauss-Seidel mainly require multiple iterations to converge, each iteration comprising a single SpMV computation whereas a single iteration in Krylov iterative methods includes multiple SpMV operations. SpMV is a Level-2 Basic Linear Algebra Subprogram (BLAS) operation between a sparse matrix and a dense vector and can be represented mathematically as $y = A \times x$. See e.g. [3, 4] for details.

Graphics Processing Units (GPUs) have emerged as a revolutionary technology and have enhanced the performance of many science, engineering, entertainment, and other applications. The challenges lie in adapting the existing algorithms and designing new ones, to fully utilize massively parallel architectures of GPUs. GPU is designed to provide high throughput and utilizing its potential fully requires identifying fine-grained parallelism in the problem at hand and keeping the device busy. Computations should be structured in a manner that there is sufficient regularity in the execution path and the memory access. Nonzeros are a significantly small, sparsely populated, proportion of a large sparse matrix. Keeping track of the locations of these nonzeros, i.e. the sparsity structure, which varies greatly from one matrix to another, and supplying them to meet the full capacity of the massively parallel cores in GPUs require delicacy. Consequently, SpMV researchers have been faced with challenges including non-coalesced memory access to both the sparse matrix A and the vector x, load imbalance among the array of threads and warps, and thread divergence among a warp of threads. See e.g. [5, 6, 7].

We have done an extensive literature review of SpMV techniques on GPUs (see Section 3). The literature review has revealed that the SpMV community has done considerable work on proposing new storage schemes and kernels for SpMV computations. However, due to vast variations in sparsity structures of matrices, any of the proposed schemes do not perform well across sparse matrices. Moreover, the utilization of the GPU devices by the existing SpMV kernels is far from its full capacity. We have observed two major research gaps and addressing these gaps is expected to make far-reaching advances in this area. Firstly, the literature review has clearly established the need for multi-scheme and multi-kernel works where the best performing SpMV scheme and/or kernel can be selected automatically based on a matrix sparsity structure (see Section 4.1, where we empirically validate the need for scheme prediction). In these directions, we have found five sets of works [8, 9, 5, 10, 11] (see Section 3.2). The research is in its infancy and requires many more efforts to make advances in these directions. Secondly, the development of SpMV techniques on GPUs and their performance analysis have not been studied in sufficient depth. We attempt to address these two research gaps in this work as follows.

We develop a tool called DIESEL that predicts and executes the best performing SpMV kernel for a given matrix (see Figure 1). DIESEL stands for Deep learning based IterativE Solver tool for large sparsE Linear equation systems. Each kernel utilizes a specific sparse storage scheme and is selected based on a carefully devised set of features. We have used fully connected deep learning neural networks to train and predict the best performing storage scheme and kernel. This is the first work where deep learning has been used for SpMV kernel prediction. The benefits of using deep learning will be discussed later in Section 4 and are also manifested through the higher performance that our tool delivers compared to the existing ones. The core of the deep learning engine is a feature set that we have developed in this work (see Section 4.5). The feature set comprises 14 carefully selected features through rigorous empirical and mathematical instruments including extensive experiments involving various features, correlation matrix, t-SNE (t-Distributed Stochastic Neighbor Embedding is a machine learning method for dimensionality reduction), and 2D and 3D cross-feature performance visualizations. The dataset that we have used to train, test and evaluate DIESEL comprises 1056 matrices from 26 different real-life application domains including computational fluid dynamics, circuits, materials, electromagnetics, and economics (from the SuiteSparse collection: see Section 4.4). Moreover, in addition to the existing performance metrics, we use a range of new metrics and methods for performance analysis, visualization and comparison of the DIESEL tool. These include results involving (a) performance of each of the 1056 matrices in the dataset; (b) performance aggregated over the application domains and over the whole dataset; (c) performance behavior of SpMV kernels against matrix dimensions, *nnz* (nonzeros in the matrix), and *npr variance* (variance of nonzeros per row in a matrix); (d) performance behavior in relation to the dataset characteristics; (e) prediction accuracy, workload

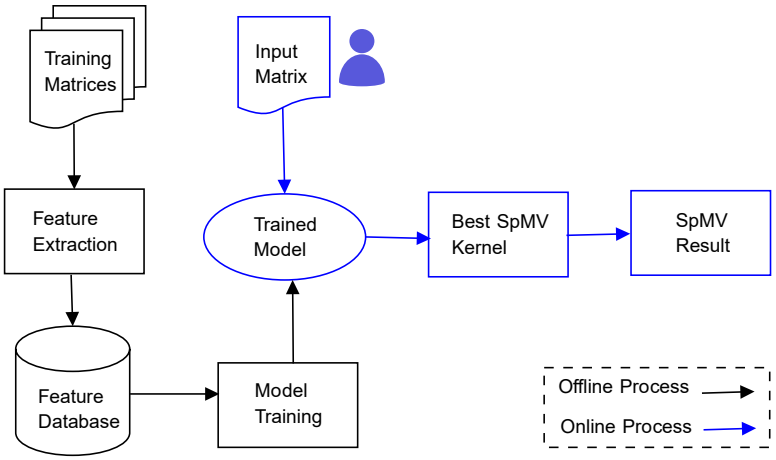


Figure 1: Overview of DIESEL

accuracy, and relative performance loss and gain; (f) comparison with other cutting-edge artificial intelligence (AI) based tools; and (g) comparison with widely-used commercial library Paralution.

Our work on the feature set and performance analysis methods mentioned above attempts to address the second research gap. We believe that the instruments and methods for SpMV tool development and analysis presented in this paper will help the community in developing better SpMV tools. These tools are just the beginning and we have just scratched the surface. There is so much more that can be done to improve our proposed feature set development instruments and performance analysis methods, and thereby to optimize SpMV performance on GPUs (and other architectures). In the past, developing analytical models to understand the performance issues and optimizing the performance was nearly impossible. However, due to the recent developments in AI, it is possible to understand performance bottlenecks and let AI select the best strategies to address those bottlenecks.

Comparing the performance of DIESEL with its SpMV kernels utilizing a specific sparse storage scheme among Coordinate (COO), Diagonal (DIA), Compressed Sparse Row (CSR), Ellpack (ELL), and Hybrid (HYB); see Section 2, DIESEL provides highest and average giga floating point operations per second (GFLOPS) values of 54.39 and 8.5, compared to 37.11 and 6.0 by the next best kernel, delivering speedups of $\times$1.47 and $\times$1.42, respectively (see Section 5.1). Comparing DIESEL with the other cutting-edge AI tools, DIESEL provides better performance with its accuracy of 88.2%, workload accuracy of 91.96%, and average relative loss of 4.4%, compared to the respective values of 85.9%, 85.31%, and 7.65% by the next best tool (see Section 5.6). Comparing DIESEL with Paralution, DIESEL outperform Paralution in 80% of the test cases (see Section 5.7). The details of these and the other results is given in Section 5. The extensive results and analyses presented in this paper provide several key insights into the performance of the SpMV tools and how these relate to the matrix dataset and the selected performance metrics. These insights could be used by the community to further improve and compare SpMV tools in the future.

In summary, this paper makes the following contributions (for details, refer to the paragraphs above).

- We proposes DIESEL, a deep learning-based tool that predicts and executes the best performing SpMV kernel for a given matrix and outperforms the cutting-edge academic (see Section 5.6) and commercial (see Section 5.7) SpMV tools.
- We devise a feature set through rigorous empirical and mathematical instruments. Both the feature set and its design process are our contributions.
- We propose a range of new metrics and methods for performance analysis, visualization, and comparison of SpMV tools to be used by the SpMV community in the future.

The organization of this paper is as follows. Section 2 briefly introduces the background topics with pointers to the relevant sources. Section 3 reviews the related works and provides motivations for this work. Section 4 describe the methodology and design of the proposed tool DIESEL. The tool is evaluated and compared with other sparse storage kernels and tools in Section 5. Section 6 concludes with future directions. The table of contents is provided below for the convenience of reviewers and readers. It can be removed as advised.

2 Background Material

Researchers have developed various sparse storage schemes to efficiently store sparse matrices and improve SpMV performance. We have used five storage schemes that are most commonly used by SpMV researchers. These are COO (Coordinate), CSR (Compressed Sparse Row), DIA (Diagonal), ELL (ELLPACK), and HYB (Hybrid ELL/COO) formats. Further information about these formats and implementation details can be found, e.g., in our earlier publications [7, 12].

Graphics Processing Units (GPUs) provide high computational throughput through massive parallelism. Further details are available on the Nvidia website and in many books and papers, see e.g. [13]. Deep learning has also been the subject of many books, see e.g. [14].

3 Literature Review: SpMV Computations and Iterative Solvers on GPUs

We have done an extensive literature review of SpMV techniques on GPUs and other architectures. A great deal of SpMV literature exists on CPU, FPGAs, and Many Integrated Core (MIC) architectures, see e.g. [15, 16, 17]. New directions are also emerging that combine high performance computing with big data, which are likely to affect the future direction of high-performance SpMV computations and iterative solvers and these are discussed in [18, 19, 15]. In this section, we review notable literature related to the focus of this paper, i.e., SpMV computations and iterative solution of large sparse linear equation systems on GPUs. The works on SpMV computations and iterative solvers can be classified into four categories: (a) improvements in sparse storage formats, (b) enhancing the SpMV computation algorithms and implementations, (c) analytical modeling and performance analysis, and (d) using Artificial Intelligence (AI) to improve performance. This classification is not specific to GPUs and applies also to other architectures. The works that fall under the first category include [20, 21, 22]. The work in the second category include [23, 24, 7]. The works related to these first two categories have already been reviewed in our earlier works [12, 7]. We review here works related to the last two categories in the following two subsections.

3.1 Analytical Modeling and Performance Analysis

Many researchers have developed analytical models that predict the performance of the various sparse storage schemes and SpMV kernels for GPUs. Guo et al. [25] propose an analytical and profile based performance modeling technique to predict SpMV execution times for COO, CSR, ELL, and HYB kernels. Based on the performance modeling, they design an auto-selection algorithm to report the best storage format for the sparse matrix automatically. The optimization schemes that are based on analytical modeling are hard to build and extend as it is necessary to build a specific model for each sparse format under consideration. Analytical models consist of many assumptions to simplify the underlying hardware complexities. Li et al. [26] proposed an optimization scheme based on calculating the probability mass function (PMF) to analyze the sparsity structure of the matrices. They also propose probabilistic models for predicting the performance of COO, CSR, ELL, and HYB formats. It then computes and compares the performance of specific formats and pre-defined formats based on mathematical expressions to select a format. The calculation of the performance model requires knowledge of the GPU architectural parameters. Alahmadi et al. [27] explore the performance of SpMV computations and Jacobi iterative method on GPUs and analyze the performance bottlenecks and address the limitations of the existing approaches. They discuss various performance characteristics of SpMV on GPU such as active warps (resource usage), occupancy, memory throughput, memory access, and instruction throughput. They also discuss various optimization strategies such as coalesced memory access, reduce thread divergence, changing thread/block configurations, and increasing occupancy. They compare the performance of six sparse SpMV techniques namely COO, CSR, ELL, HYB, DIA, and CSR5 on GPUs.

3.2 Artificial Intelligence based Techniques

There is no sparse storage scheme that gives the best performance across all sparse matrices and application domains (see Section 4.1). The performance of SpMV kernels varies for a given storage scheme based on the matrix sparsity structure. This has been addressed using AI in three different groups of works. The first group of works has used Decision Trees and an ensemble of machine learning methods [8, 10], and the second has used SVM (Support Vector Machine) [9, 5]. The third work has used C5.0 Decision Trees. The first work in this direction is by Sedghati et al. [8], who proposed a classification system based on decision trees. They had selected two feature sets. The first one consisted of the matrix characteristics, density, mean, and standard deviation. The other feature set consist of advanced features that are compute-intensive to calculate and create a large overhead to the feature extraction process. These, among others, are the number of nonzeros per block and the mean and standard deviation of these blocks.

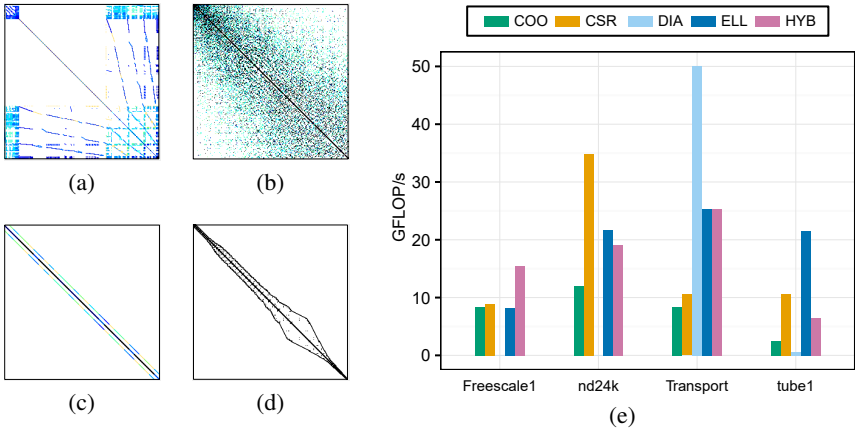


Figure 2: Sparsity structure of four sparse matrices and their GFLOPS performance for five different kernels (**a**) Freescale1; (**b**) nd24k (72000); (**c**) Transport (160211); (**d**) tube1 (21498); and (**e**) Performance (GFLOPS).

Benatia *et al.* [9] proposed a method for selecting a storage format to perform SpMV operation on GPUs. The SpMV itself was performed on two GPUs, GTX 580 and a GTX 980Ti. Their proposed features include the number of rows and columns, the total number of nonzeros, *npr*, standard deviation and *npr variance*, the density of the sparse matrix, and the difference between the maximum and average *npr*. The authors extended their work further in [5] where they used a weighted SVM with the help of pairwise classification approach (instead of multi-class SVM). The features used are similar to their previous work but in this case the feature combinations are selected based on the pair of storage formats for creating each pairwise model.

Israt et al. [10] extended their earlier work ([8]) and compared the performance of the machine learning techniques, specifically the SVM and decision-tree based models, discussed by Benatia et al. [9] and Sedaghati et al. [8], along with multi-layer perceptrons (MLP), gradient boosting based models (XG-Boost) and ensembles of MLP for the prediction of the best sparse storage format for the SpMV computations of a matrix. The feature set used is similar to their earlier work ([8]) where the basic feature set is extended by adding the *density* parameter. The second feature set is also the same as in [8] containing features at the matrix block level. Tan et al. [11] propose an auto-tuning SpMV library called SMATER that creates a learning model using data mining and analytical models. They use the C5.0 decision tree algorithm to generate a rule-set pattern with confidence data. A scorecard is used to select the best performing kernel for the predicted storage format.

We will provide a detailed comparison of the relevant AI-based tools with DIESEL in Section 5.6.

3.3 The Case for Research

The literature review has revealed that the SpMV community has done extensive works on proposing new storage schemes and kernels for SpMV computations. However, the utilization of the GPU devices by the existing SpMV kernels is far from its full capacity. We have observed two major research gaps and addressing these gaps could make far-reaching advances in this area. Firstly, the literature review has clearly established the need for multi-scheme and multi-kernel works where the best performing SpMV scheme and/or kernel can be selected automatically based on a matrix sparsity structure (see Section 4.1, where we empirically validate the need for scheme prediction). In these directions, we have found only three sets of works that have already been discussed in the previous subsection. The research, therefore, is in its infancy and requires many more efforts to make advances in these directions. Secondly, the development and performance analysis of SpMV techniques on GPUs have not been studied in sufficient depth (this applies to all four categories of works discussed earlier in this section). The reader may verify our statements and is encouraged to consult the most recent review of SpMV computations on GPUs [6]. This paper attempts to address these two research gaps. We propose our deep learning-based SpMV tool, DIESEL, and provide an extensive analysis of its design, development, and evaluation using a range of existing and new methods and metrics (a summary of how DIESEL attempts to address the two research gaps is given in Introduction (Section 1)).

4 DIESEL: Methodology and Design

The DIESEL tool provides multiple GPU implementations of SpMV operation and an iterative solver for the solution of sparse linear equation systems. Currently, it uses the Jacobi iterative method but we plan to extend it with additional iterative methods, storage schemes, and SpMV kernels. It takes a sparse linear equation system, or a matrix and a vector, as an input and uses a GPU kernel that provides the best performance for the SpMV or the iterative solution. The best performing kernel is predicted by the tool using deep learning applied to a set of empirically and analytically selected matrix performance features. A fairly large dataset comprising real-life applications is selected to make the training, prediction, and evaluation of the DIESEL tool as diversely realistic as possible.

This section provides details of the methodology and design of the DIESEL tool as follows. Section 4.1 provides empirical motivations for selecting the best performing SpMV kernel depending on the matrix properties. Section 4.2 describes the deep learning network architecture and motivates the need for deep learning in developing the SpMV kernel prediction models. Section 4.3 provides an overview of DIESEL. Section 4.4 provides the methodology and details of the dataset used in this paper. Finally, Section 4.5 details the feature selection methodology and the development process.

4.1 Need for Automatically Selecting the Best SpMV Storage Scheme and Kernel

The matrix sparsity structure varies from matrix to matrix, and application to application. Figures 2(a), (b), (c), (d) show the structure of four sparse matrices, each from a different application domain, acquired from the University of Florida Sparse Matrix collection; these are taken from circuit simulation problems contributed by Freescale Semiconductor, 3D mesh problems, 3D finite element flow and transport problems, and tube modeling problems in structural mechanics, respectively. The dimensions for these square matrices can be found in the respective captions. Figure 2(e) depicts the performance (GFLOPS) of five different SpMV kernels for the four matrices, each kernel uses a different sparse storage format (COO, CSR, ELL, HYB, DIA). Note in the figure that the ELL and HYB kernels provide better results on average compared to the other kernels. The DIA kernel provides best results for the "Transport" matrix due to the diagonal structure of the matrix but gives poor performance for all other matrix structures; no results are given for "Freescale1" and "nd24k" because these matrices require more memory to store the matrices than is available with the K20 GPU. The ELL kernel provides better results for all the matrices (marginally better than HYB for the "Transport") except for "Freescale1" due to a big difference in this case between the minimum and maximum *npr* (see [28] for details on ELL, and [20] for HYB, also see [7] for a discussion on relative performance of these scheme). The HYB kernel performs best for the "Freescale1" matrix because HYB is designed especially for such sparsity structures where large amount of zero padding is needed to compensate for a large variation in the minimum and maximum number of non-zeros per row. The CSR kernel provides the best results for the "nd24k" matrix because its nonzero elements are spread across the whole matrix space. The science of SpMV performance in relation to the matrix sparsity structure will become clearer as we discuss our feature selection methodology in Section 4.5. The figure substantiates the view that performance of an SpMV kernel depends on the sparsity structure of the underlying matrices. The GPU architecture used for the SpMV computations, the thread-block-warp configuration of the SpMV kernel, and the storage scheme used for the sparse matrix storage are among the factors that affect the performance of SpMV kernels on GPUs.

4.2 Deep Learning Neural Networks for Predicting Performance

We have designed a deep architecture that has six layers, an input layer, four hidden layers, and an output layer. The input layer has 14 neurons corresponding to the number of input features. The second layer has 100 neurons, the third layer has 50 neurons, the fourth layer has 25 neurons, the fifth layer consists of 10 neurons and the last layer (the output layer) has five neurons (corresponding to the number of classification classes). All four hidden layers use ReLU as the activation function. The cost function of the deep network is minimized using Adam optimizer [29]. The learning rates utilized by the Adam optimizer are set to the default values of 0.9 and 0.999 for the optimizer learning rate.

We used three techniques to prevent overfitting in DIESEL. These are regularization, dropout, and early stopping. *Regularization* is a technique utilized to reduce the complexity of the model by adding a penalty term to the loss function [30]. Based on the value added to the loss function the technique can be classified as L1 or L2 regularization. L1 penalty minimizes the absolute value of the DNN weights while L2 minimizes the squared value of the DNN weights. L2 can learn more complex patterns than L1. We used L2 regularization in DIESEL. We also utilized in DIESEL the *dropout* technique [31], which is also a regularization technique that avoids overfitting by randomly dropping neurons from the DNN. This results in a new neural architecture that reduces overfitting. Moreover, we have used another technique in DIESEL called *early stopping* [30] where the training process is stopped when a certain number of iterations have passed with no progress in the loss function.

We tried various network configurations in terms of the number of various layers and neurons and these configurations provided us best prediction accuracies. It is widely accepted that selecting the number of hidden layers and neurons — i.e., an ideal deep learning network — is still an art lacking robust theoretical foundations [14]. The impact and utilization of recent state-of-art optimizers as well as tuning these optimizers with appropriate hyper-parameters such as learning rates for SpMV prediction needs to be extensively studied. Dynamic selection of the hyper-parameters and its impact on the kernel selection with varying sparse matrix structure have not been studied. We will look further into these design aspects of the DIESEL tool in the future.

Algorithm 1: DIESEL_Master: The Master Algorithm

Input : Matrix $\boldsymbol{A}$, vector $\boldsymbol{y}$, *feat_train*
Output : Vector $\boldsymbol{x}$

Function MAIN($\boldsymbol{A}$, $\boldsymbol{y}$, *feat_train, op*):
  feat ← EXTRACTFEATURES($\boldsymbol{A}$)
  nfeat ← $\log_{10}$(*feat*)
  if Network not trained
    θ, b ← TRAINING(*feat_train*)
  SpMV_GKernel ← DEEPPRED(*nfeat*)
  if *op* == JACOBI
    x ←JACOBI($\boldsymbol{A}, \boldsymbol{y}$, *SpMV_GKernel*)
  if *op* == SPMV
    x ←SPMV($\boldsymbol{A}, \boldsymbol{y}$, *SpMV_GKernel*)
  return x

Algorithm 2: DIESEL_Train: The DNN Training Algorithm

Input : *feat_train*
Output : $\{\theta_1 ....\theta_L\} \in R^2$, $\{b_1 ..., b_L \in R\}$

Function TRAINING(*feat_train*):
  Init: $b \leftarrow 1, h_{(1)} \leftarrow nfeat, \theta \in [-1, 1]$
  while $cost > minimumcost$
    for l in $1 \leftarrow L$
      $h_{(l+1)} \leftarrow \texttt{Activation}\,(\theta_{(l)} h_l + b_{(l)})$
    // L is the total layers
    Calculate Loss function derivative as
      $\partial_{(n_l)} \leftarrow -(y - h_{(n_l)}) \bullet \texttt{Activation}\,(u_{(n_l)})$
    for Layer $l = L - 1$ to 2
      $\partial_{(l)} \leftarrow \theta_{(l)} \partial_{(l+1)}, \bullet \texttt{Activation}\,(u^{(l)})$
    Compute the gradient for each l as:
      $\nabla_{\theta_{(l)}} \leftarrow \partial_{(l+1)} (h_{(l)})^T$
    Find optimal weights using ADAM optimizer
    Update the current cost
  return x

Algorithm 3: DIESEL_Prediction: The Prediction Algorithm

Input : *nfeat*, θ, y
Output : SpMV_GKernel

Function DEEPPRED(*nfeat*):
  Init: $h_{(1)} \leftarrow nfeat$
  for l in $1 \leftarrow L$
    $h_{(l+1)} \leftarrow \texttt{Activation}\,(\theta_{(l)} h^l + b^{(l)})$
  return $h_{(L)}$

Algorithm 4: DIESEL_Solve: The Jacobi Iterative Solver

Input : $\boldsymbol{A}$, $\boldsymbol{y}$, $\boldsymbol{x}$, SpMV_GKernel
Output : $\boldsymbol{x}$

1 **Function** JACOBI($\boldsymbol{A}$, $\boldsymbol{y}$, *SpMV_GKernel*):
  Init: $error \leftarrow 1.0$, $\epsilon \leftarrow 10^{-6}$, $k \leftarrow 0$
  $\boldsymbol{x} \leftarrow (1/size(\boldsymbol{A}))$, $\overline{\boldsymbol{x}} \leftarrow 0$
2 Move $\boldsymbol{A}$, $\boldsymbol{y}$, $\boldsymbol{x}$, and $\overline{\boldsymbol{x}}$ to *gpu*
3 **while** $error > \epsilon$
4 $\overline{\boldsymbol{x}} \leftarrow$ SPMV(A, x, SpMV_GKernel)
5 $\overline{\boldsymbol{x}} \leftarrow \frac{\boldsymbol{y}-\overline{\boldsymbol{x}}}{D(\boldsymbol{A})}$
6 Compute *error*
7 $\boldsymbol{x} \leftarrow \overline{\boldsymbol{x}}$
8 Get the result $\boldsymbol{x}$ from *gpu*
9 **return** $\boldsymbol{x}$

Algorithm 5: DIESEL_SpMV: The SpMV GPU Kernels

Input : $\boldsymbol{A}$, $\boldsymbol{x}$, SpMV_GKernel
Output : $\overline{\boldsymbol{x}}$

1 **Function** SPMV($\boldsymbol{A}$, $\boldsymbol{x}$, SpMV_GKernel):
2 Move $\boldsymbol{A}$ and $\boldsymbol{x}$ to *gpu*
3 **switch** SpMV_GKernel **do**
4 **case** 0 **do**
5 $\overline{\boldsymbol{x}} \leftarrow$ SPMVCOO <<< gs, bs >>> $(\boldsymbol{A}, \boldsymbol{x})$
6 **case** 1 **do**
7 $\overline{\boldsymbol{x}} \leftarrow$ SPMVCSR<<< gs, bs >>> $(\boldsymbol{A}, \boldsymbol{x})$
8 **case** 2 **do**
9 $\overline{\boldsymbol{x}} \leftarrow$ SPMVHYB<<< gs, bs >>> $(\boldsymbol{A}, \boldsymbol{x})$
10 **case** 3 **do**
11 $\overline{\boldsymbol{x}} \leftarrow$ SPMVELL<<< gs, bs >>> $(\boldsymbol{A}, \boldsymbol{x})$
12 **case** 4 **do**
13 $\overline{\boldsymbol{x}} \leftarrow$ SPMVDIA<<< gs, bs >>> $(\boldsymbol{A}, \boldsymbol{x})$
14 Get the result $\overline{\boldsymbol{x}}$ from gpu
15 **return** $\overline{\boldsymbol{x}}$

4.3 DIESEL Overview

The DIESEL tool comprises five main components. The algorithms for the five components are given in Algorithms 1 to 5. The DIESEL Master component is delineated in Algorithm 1. Initially, we train the model (if it has not been trained before) using the DIESEL Train component provided in Algorithm 2 (called from Algorithm 1: line 5). The model is trained using a feature set devised by us comprising 14 features. The features are extracted before the training by the Master component (Algorithm 1: line 3). We shall discuss this feature set and the dataset in subsequent subsections. Adam optimization [29] and Rectified Linear Units (ReLU) [14] are used in the training process (Algorithm 2: line 2 – line 10). If the model has already been trained then we use the DIESEL Prediction component (Algorithm 3) to determine (predict) an appropriate Jacobi GPU kernel (`Sparse_GKernel`) based on the matrix features using the trained model obtained from the Train component (Algorithm 2). If the desired operation, op, is Jacobi iterative solution, the computed value of the `Sparse_GKernel` is passed to the DIESEL Solve component (Algorithm 4) that in turn calls the SpMV component (Algorithm 5). The SpMV component executes a specific SpMV kernel based on one of the five storage schemes selected using `Sparse_GKernel` and returns the vector x. If the desired operation, op, is SpMV, the Master component calls the SpMV function (Algorithm 5) directly and finally returns the vector x.

4.4 Dataset Formation

We created a dataset for training and testing of the deep learning prediction model utilizing the University of Florida repository of sparse matrices (called SuiteSparse) [2]. We analyzed the SuiteSparse collection and selected 1056

Table 1: The application domains used in our experiments

ID	Name	Count	ID	Name	Count
1	2D/3D	94	14	Materials	25
2	Acoustics	6	15	Model Reduction	26
3	CPS	50	16	Optimization	18
4	Circuit	195	17	Power	20
5	Counter	5	18	QCD	46
6	Combinatorial	13	19	Undirected Graphs	3
7	Economics	26	20	Unweighted Graphs	15
8	Eigenvalue	31	21	Semi-Conductor	33
9	Electromagnetics	24	22	Statistics	4
10	Finite Elements	1	23	Thermal	25
11	FDC	4	24	Structural	212
12	Least Square	18	25	CFD	140
13	LP	20	26	Graphics/Vision	2

matrices of different sizes and sparsity structure from 26 application domains. Table 1 lists the 26 application domains and the number of matrices for each of them. The criteria for the selection of the matrices is listed below.

1. The matrices are *real-valued*, not *complex*.
2. The matrix are rectangular or square.
3. The matrix are asymmetric or symmetric.
4. The sparse matrix is able to fit within the 80% of the global memory of GPU. The size of the sparse matrix is calculated in the COO format ($16 \times nnz$ bytes). The reason being that SuiteSparse stores the matrices originally in the COO format. Note that the ELL format can use excessively high memory if the matrix structure is highly irregular. Therefore, we were unable to execute the ELL kernel for 350 of the 1056 matrices in our dataset. Similarly, DIA is only efficient in storage and computations for banded matrices and we were unable to execute the DIA kernel for 807 of the 1056 matrices.
5. The total number of rows in a matrix should at least equals to the warp concurrency, which is the ratio of the total threads across all the streaming multiprocessors (SM) to the warp-size, i.e. $warp\ concurrency = (Threads\ per\ SM * SM)/warpsize$. This should not enforce any limitations on the tool in terms of large-scale computing because performance is not a concern for small matrices on GPU.

Table 2: Selected Features From Sparse Matrices

Feature	Description	Complexity
m	The number of rows	$O(1)$
n	The number of columns	$O(1)$
nnz	The total number of nonzero values in the matrix	$O(1)$
density	The density of the matrix, $nnz/(m * n)$	$O(1)$
mean	The mean number of nonzero values per row (*npr*)	$O(1)$
sd	The standard deviation of *npr* in a matrix	$O(2m)$
var	The *variance* of *npr* in a matrix	$O(1)$
maxnnz	The maximum of *npr* in a matrix	$O(m)$
maxavg	The difference between *maxnnz* and *mean*	$O(1)$
distavg	The mean distance between first and last nonzero values in each row	$O(m)$
clusteravg	The mean of the number of distinct consecutive *npr*	$O(nnz)$
ndiag	The number of matrix diagonals with one or more nonzero values	$O(nnz)$
diagfill	The diagonal fill-in ratio for matrices, $(ndiag \times m)/nnz$	$O(1)$
fill	The fill-in ratio for matrices, $(m \times maxnnz)/nnz$	$O(1)$

4.5 Feature Selection Methodology and Development Process

We have elaborated on the fact that the performance of SpMV kernels depends on the sparsity structure of the matrices (see Section 3.3). Therefore, we collected a total of 22 features of sparse matrices to characterize them and used those features in devising a performance prediction instrument. Some of these features exist in the literature while others were devised by us. A total of 14 sparsity features were selected finally after a detailed empirical analysis of the matrix feature set to be used in our tool. We shall elaborate on this empirical analysis in the rest of this subsection. Table 2 lists our final feature set. These 14 features enable the performance prediction instrument to meticulously characterize the sparsity structure of a matrix and predict the GPU kernel that would provide the best performance for that matrix. To the best of our knowledge, a maximum of eleven features have been used in the literature to characterize matrix sparsity structure and predict performance. This matrix feature set is among the contributions of this work.

4.5.1 Feature Set: Computational Complexity

All of our features are simple to calculate: m, n, nnz, *density*, *mean*, and *variance* all have a time complexity of $O(1)$. m, n, and nnz are usually given as an input to the solver. The mean number of nonzero elements per row (*mean*) is easily calculated from m, n, and nnz. Standard Deviation (sd) has a complexity of $O(2n)$. Variance can be calculated with a complexity of $O(1)$ once we have *mean* and sd. For finding the maximum number of nonzero elements in a row (*maxnnz*), we need to traverse each row once. The average distance between the first and the last nonzero value per row (*distavg*) can be calculated at the same time when calculating *maxnnz*. The feature *clusteravg* represents the average number of consecutive nonzero elements per row. This is computed by finding the largest cluster of consecutive nonzero elements for each row and then taking an average of the largest consecutive nonzero clusters for all the rows in the matrix. *ndiag* is the number of diagonals in the matrix and *diagfill* is the product of the number of diagonals to the size of the diagonals. *ndiag* and *diagfill* were added to improve the conversion rate of the nonconvertible matrices to DIA sparse storage. The feature *fill* is used to determine whether a matrix can be feasibly stored using the ELL storage format.

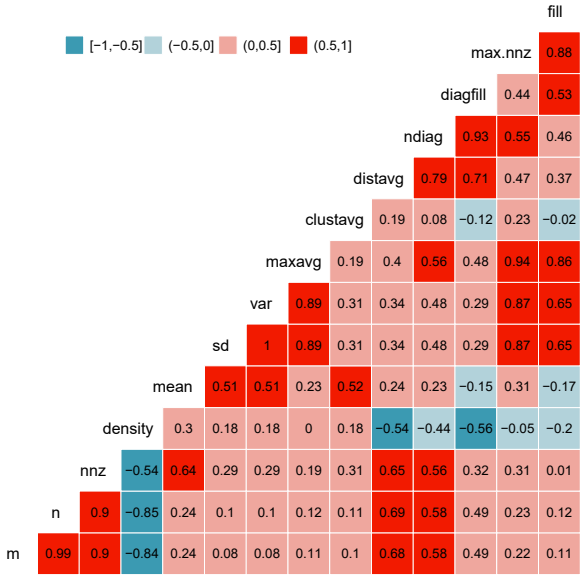


Figure 3: The correlation matrix for our selected features

4.5.2 Correlations among Features

Figure 3 shows the correlation matrix for the selected feature set given in Table 2. The positive correlations are depicted in red and pink colors while the negative correlations are in the dark and light blue colors. The red and darker blue colors depict higher values of the correlations. A low correlation between the features is desirable because it implies that the features are relatively distinct and provide a valuable contribution to the feature set in improving the prediction accuracy. It is possible to include more features in the set but a larger number of features increase the computational time for predictions.

Note that the overall correlation between the features in our selected feature set is mostly positive and relatively low (more blocks with the pink color), and the negative correlations mostly are small except a few. Looking further, for

example, for the feature *nnz*, there are only five features that have a correlation above 0.5. These are *n* (0.9), *m* (0.9), *distavg* (0.65), *mean* (0.64), and *ndiag* (0.56). These correlations are due to the linear nature of *nnz*, e.g. when *m* and *n* increase, *nnz* also increases. Similarly, a higher value of *nnz* would also usually imply higher values for *mean*, *distavg*, and *ndiag*. Note that the respective correlation values of the five features with the feature *nnz* are as one would expect (e.g. *n* and *m* have very high correlation value, 0.9). Note also that *n* and *m* show the highest correlation between them because we have a large number of square matrices in our dataset.

Table 3: Summary of Our Feature Set

Features	Min	Max	Average
m	838	8.34M	138k
n	292	8.34M	142k
nnz	1074	229M	194k
density	9e-07	0.12	3.9e-03
mean	0.59	424	27.13
sd	0.00	3594	40.3
var	0	12M	29669
maxnnz	1	2M	9094
maxavg	0	2.3M	9067
distavg	0	4M	34724
clusteravg	0	126.64	6.585
ndiag	1	10M	79643
diagfill	1	978k	9236
fill	1	259k	972

The feature *density* has the highest number and values of negative correlations with other features. The absolute values of three of the negative correlations are below 0.5, two of them are very low, near zero (*fill* and *maxnnz*). All the positive correlations are very small except *maxavg* which is zero, indicating no correlation. The highest negative correlations are with *m* and *n*. This shows that *density* of the sparse matrices decreases, largely, with the increase in the number of rows and columns (i.e., the size of the matrices). This is typically true with sparse matrices. The features *m* and *n* are so fundamental to matrices that their high correlations with other features could be ignored unless a feature has strong correlations with most of the features in the set. The absolute value of the correlation with *nnz* is slightly above 0.5 and is not that high. The correlation indeed shows that our data set is balanced in that *density* does not increase with the size of the matrices, and rather it decreases, however, not to a great extent (considering all the matrices in the data set). This inverse correlation is expected because larger *m* and *n* lead to higher denominator values. Figure 4 plots the $\log_{10}$ values of *density* against the log_{10} values of *nnz* for the whole dataset. The features *distavg* and *diagfill* are also negatively correlated to *density* to a similar degree as with *nnz* and therefore same explanation applies to them. Moreover, the two features are needed for two widely-used sparse storage schemes (*distavg* for ELL and HYB, and *diagfill* for DIA).

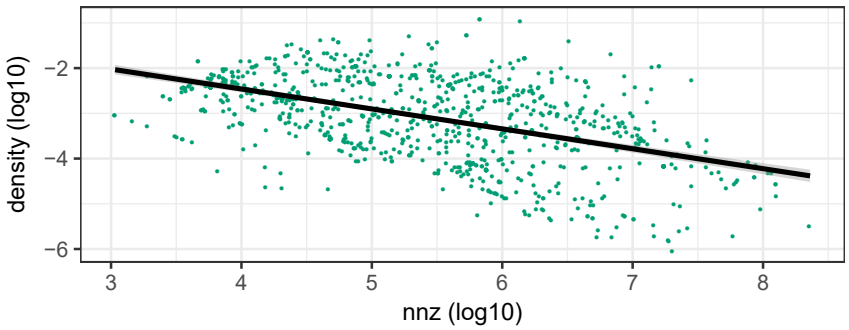


Figure 4: The log_{10} values of *density* against the log_{10} values of *nnz*

In summary, our feature set depicts, largely, a low correlation between its member features. There are some high-correlation features but these features have low correlations, mostly, with other features. The feature set shows that our data set is fairly balanced. The overall low correlation (light colors) of the feature set also implies that the member features contribute distinct information for the prediction model. Table 3 gives a quantitative summary of the features of

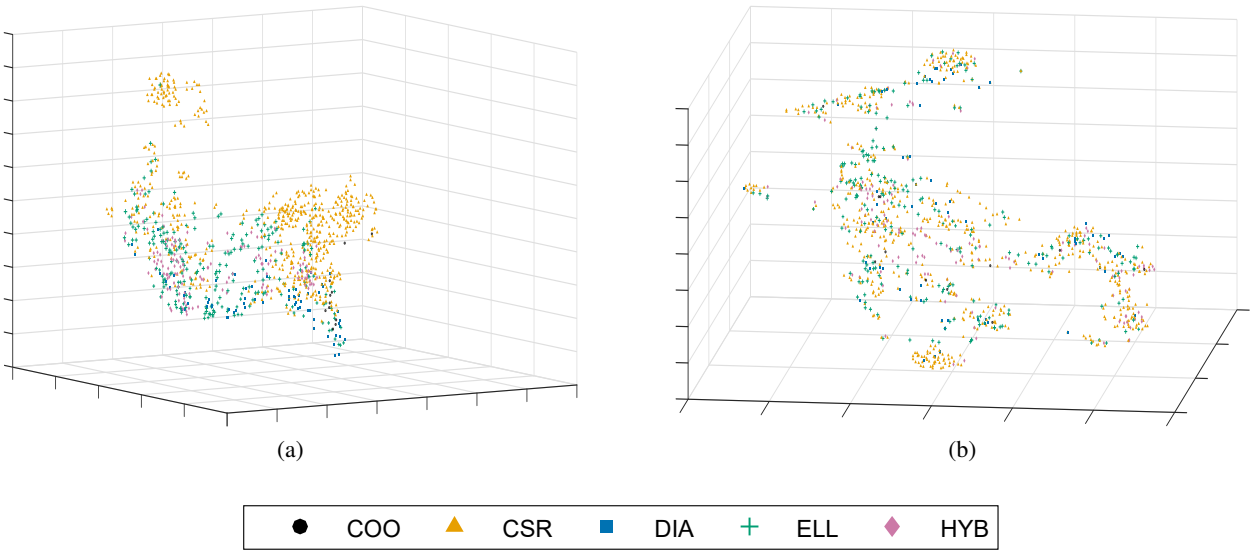


Figure 5: T-SNE profiles of the five GPU SpMV kernels for (a) 14 sparsity features, (b) 22 sparsity features

the matrices in our data set. In the next subsection, we study the process of identifying the feature set and how it has enabled the DIESEL tool to perform better than the other tools.

4.5.3 Feature Set Development

We now explain the process that we have followed in developing the feature set for the DIESEL tool. Specifically, we study the relationship between the sparsity features and the resulting performance of GPU SpMV kernels, in order to explain the rationale for the selection of the 14 features. We had originally selected a total of 22 features. Eight of them were dropped after the feature selection process due to the lack of any comprehensible relationship between them and the SpMV kernel performance, hence affecting the accuracy of predicting best performing kernel.

The depiction, analysis, and organization of data with high dimensions is challenging. We have 14 features or dimensions of our data. Therefore, to analyze our feature set, we have used the t-Distributed Stochastic Neighbor Embedding (t-SNE) [32, 33] technique which allows dimensionality reduction of highly non-linear spaces and is widely used for visualizing high-dimensional datasets (see e.g., [34]). A high-dimensional object is modelled by a 2D or 3D point such that similar and dissimilar objects are represented by nearby and distant points respectively. The reduced 3D space generated by the t-SNE technique for our final feature set comprising 14 features is depicted in Figure 5(a). The figure depicts the performance profiles in GFLOPS of the five SpMV kernels for our final feature set in reduced 3D space. The SpMV kernels are the same as in Figure 2; COO, CSR, HYB, ELL, and HYB. The data points that relate to each of the 1056 matrices are colored according to the kernels that produced the best performance for a given matrix. They indicate the distribution of sparse kernel performance in reduced space.

The t-SNE reduced space mappings can be analysed by the feature separability and overall geometric characteristics of the reduced spaces that they produce. A good feature set could typically be characterized by visuals where similar data points are clustered closely and different data points are spread across the reduced space [33]. The characteristics of a good feature set are evident in Figure 5(a) where similar points are mostly clustered together. For example, the CSR kernel (yellow triangles) can be seen distinctly clustered mainly in two locations: the right and the top left sides of the plot. A few very small clusters of the CSR kernel can be seen in other location across the plot. Small clusters of other kernels, ELL, HYB, and DIA, can also be seen spread across the 3D space, although with some overlap. The COO kernel is present sparsely on the bottom right side of the graph. Figure 5(b) shows the 3D reduced space for our original feature set comprising 22 features. In contrast to Figure 5(a), note that the similar data points are spread across the reduced space in a cluttered manner with, relatively, a much higher overlap between the kernels indicating a poor feature set. The CSR kernel can be seen spread throughout the graph showing significant overlap with other kernels and the sizes of its distinctly separable kernels are very small.

To further illustrate the relationship between the feature set and the best kernel prediction, we have plotted the best performing kernels for specific features in 3D and 2D spaces. Figure 6 depicts the performance profiles in a 3D space

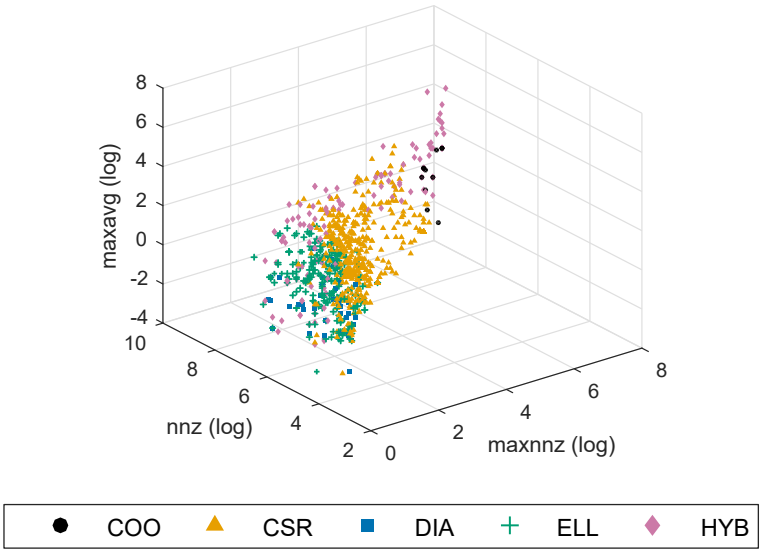


Figure 6: Performance (GFLOPS) profiles of the five GPU SpMV kernels for selected sparsity features

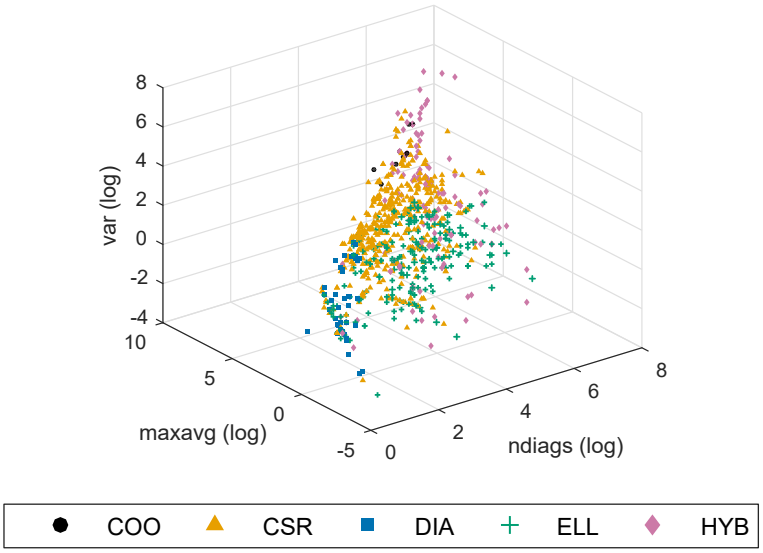


Figure 7: Performance (GFLOPS) profiles of the five GPU SpMV kernels for selected sparsity features

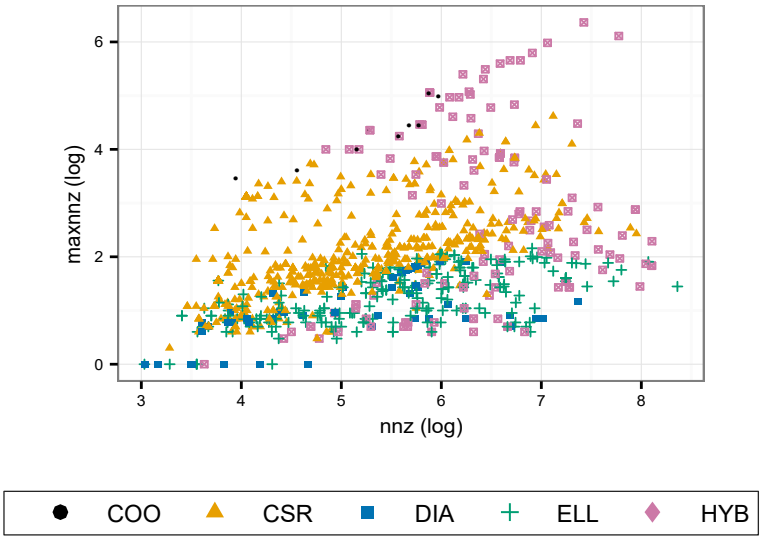


Figure 8: Performance (GFLOPS) profiles of the five SpMV kernels (*maxnnz* vs. *nnz*)

comprising three features: *maxavg*, *nnz* and *maxnnz*. Note that the HYB kernel dominates the upper-right area of the plot which represents highest values for the triple (*maxavg*, *nnz*, *maxnnz*). This is expected because HYB performs better than any other kernel for a higher *maxavg* due to the fact that a larger value for the maximum number of nonzero elements per row (*maxnnz*) typically indicates a larger value also for *maxavg* and *nnz*. As mentioned earlier, the HYB format is the best performing for large matrices and was introduced by Nvidia to overcome the limitations of CSR and ELL [20].

In addition to the upper right corner of Figure 6, the HYB kernel also shows some presence on the lower back wall for higher values of *nnz* and relatively lower values of $maxavg$ implying that HYB performs better also in this space (*i.e.* high *nnz*, low-medium *maxavg*, and medium-high *maxnnz*). This continues from the top-right of the figure downwards, towards the bottom-left, up until we see a cluster of ELL presence (see the green cluster). This cluster indicates that ELL performs better for *relatively* low values of *maxavg* (note the log scale and hence the negative values), low to high values of *nnz*, and low to medium values of *maxnnz*. Note that all the three axes are in the log scale and therefore the values of the variables should be interpreted accordingly. These behaviours of ELL and HYB are expected because HYB was introduced due to the inability of ELL in providing performance for matrices with high variations in *npr*.

Looking further at Figure 6, the CSR kernel dominates the middle part of the 3D space except the second top plane in the left column where the CSR kernel extends from the left side of the plane due to its advantage in smaller matrices (lower value of row). Some sparse presence can be seen for the DIA kernel near the space with lowest values of $maxavg$ representing the possibility of diagonal matrices such as the Transport matrix in Figure 2. The COO kernel can also be seen towards the rightmost corner of the back wall indicating highly dense matrices with a higher number of *nnz* but relatively low variations in the value of $maxavg$. Figure 7 depicts the performance profiles for a 3D space comprising three features: var, $maxavg$, and $ndiags$. The figure can be explained similarly considering our explanations for Figure 6.

Figure 8 depicts the best performing kernels in a 2D space for features $maxnnz$ and *nnz*. Note that the HYB kernel dominates the upper-right area of the plot which represents the highest values for the pair (nnz, $maxnnz$). This is expected, as mentioned before, because HYB performs better than any other kernel for a higher $maxavg$. The HYB kernel also dominates, except for a few cases, the rightmost-lower corner because a higher value for *nnz* typically implies larger matrices with a higher value for $maxavg$; some exceptions to this are for the ELL kernel (in the rightmost-lower corner), which performs best for the highest *nnz* values, extending from leftmost of the horizontal lower part of the plane, indicating most of the *nnz* values with lowest values for $maxnnz$ (see the green stream on the bottom). The CSR kernel dominates the middle part of the graph. The 2D plane for the $ndiags$ and $maxavg$ features is depicted in Figure 9. Note that the DIA kernel shows its presence in the lower left corner, the CSR kernel dominates the middle part of the graph, while the HYB kernel performs best in the upper right corner. It shows that a lower, medium, and higher variation in $maxavg$ will provide higher GFLOPS for DIA, CSR, and HYB kernels, respectively. A medium value for $maxavg$ with a higher number of diagonals will provide better performance for ELL compared to the CSR kernel. Another observation from this graph is that a higher number of diagonals does not mean DIA would perform best (see Table 2).

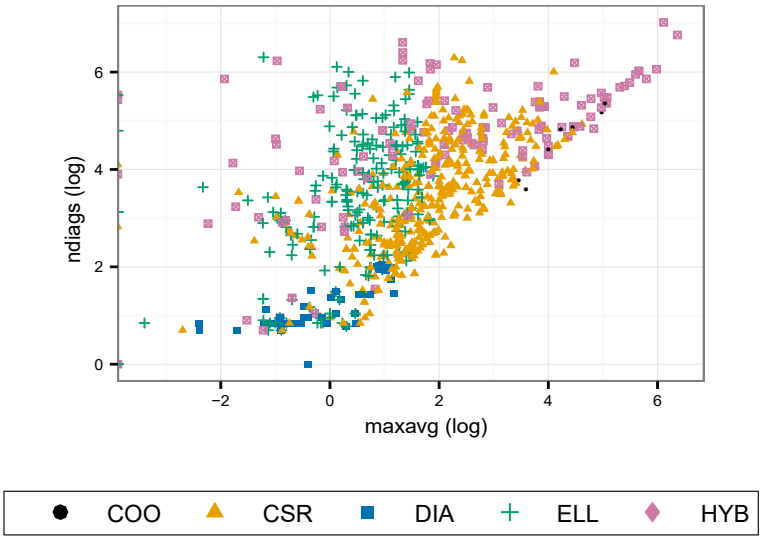


Figure 9: Performance (GFLOPS) profiles of the five SpMV kernels (*ndiag* vs. *maxavg*)

We conclude that the SpMV kernel which dominates a given area has higher performance gain compared to the other kernels in the same region. In the areas where a single kernel does not dominate, we find that the performance difference between the kernels is quite low. These empirical observations were used in devising our feature set. The feature set used in DIESEL enabled the deep learning tool to place decision boundaries in areas with clear performance gains to effectively predict the best performing kernel for a given matrix. The analysis presented in this section has been limited to a few features and depictions because a complete analysis of the whole feature set would require several pages and would appear a digression from the main topic of this paper.

5 DIESEL: Results and Analysis

This section evaluates the performance of our deep learning SpMV and iterative solution tool, DIESEL. The results reported are taken as average from the 10-fold cross-validation performed using the deep learning model. The tool performance is analyzed in terms of GFLOPS and execution times. Additional metrics are depicted and discussed to elaborate on the reasoning for particular performance behaviors of the SpMV kernels. The first category of analysis is where we have collected results for DIESEL and have compared it with five other kernels. In this category, Section 5.1 provides a comparison of the six kernels based on the performance for each of the 1056 matrices in the dataset. Section 5.2 compares the aggregated performance for the whole dataset. This essentially measures and compares the performance of the kernels at a coarser or overall level. Section 5.3 compares aggregated performance of the six kernels for each of the 26 application domains. Section 5.4 provides deeper investigations into the individual and relative performance of the six kernels and relates their performance to the various characteristics of our dataset, and *npr variance* and *nnz* of matrices. Section 5.5 looks at the accuracy of the DIESEL tool in predicting the best kernels for SpMV computations (this is the second category of results). We have defined and investigated the accuracy, workload accuracy, relative performance loss and gain of DIESEL. Section 5.6 compares DIESEL with the other cutting-edge tools that work on the principals of predicting the best SpMV storage formats or kernels using AI (this is the third category of results). Section 5.7 compares the performance of DIESEL for iteratively solving sparse linear equation systems with a widely-used commercial library Paralution (this is the fourth category of results). The platform we have used for the experiments consists of a node containing 2 Intel processors E5-2695v2 and an Nvidia K20 GPU. We used CUDA toolkit with C++ for the development of the DIESEL tool.

5.1 SpMV Performance Comparison: Each of 1056 Dataset Matrices

Figure 10 depicts as a boxplot the GFLOPS performance of the five kernels and DIESEL for all the 1056 matrices in our dataset. Boxplots are a standard way of giving information about the data distribution based on five statistical measurements, namely, the minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum. The median is the middle value among all the recorded GFLOPS values and is depicted by the thick horizontal line in the middle of the box plot. This is the 50th percentile of the GFLOPS values for each kernel. The first quartile (Q1/25th percentile) is

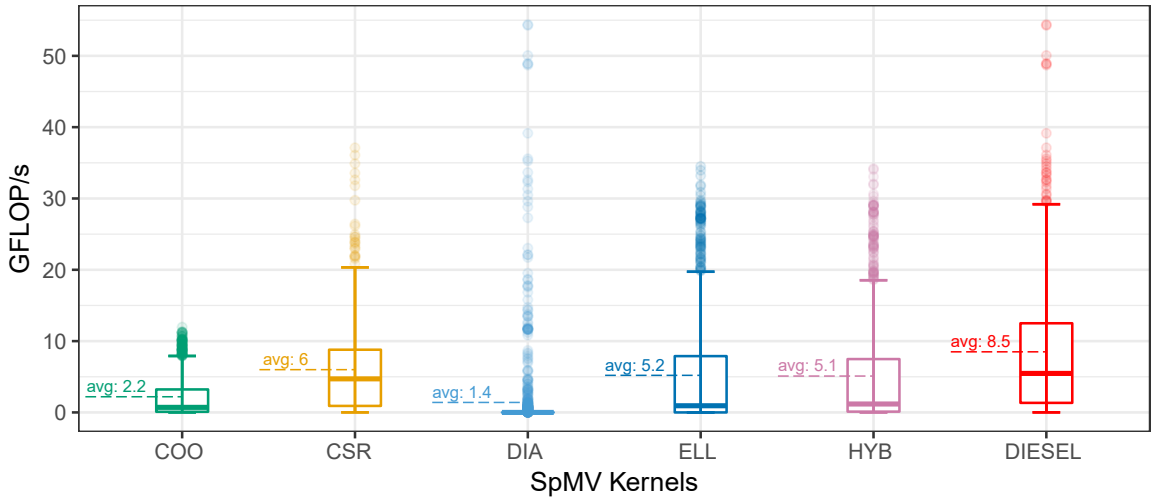


Figure 10: A Boxplot of GFLOPS performance of all six SpMV Kernels for each of 1056 Matrices in the Dataset

the median between the smallest number (not the minimum) and the median of the GFLOPS values for each kernel. The third quartile (Q3/75th Percentile) is the median between the median and the highest value (not the maximum) of the data for each kernel. The distance between the 25th to the 75th percentile is called the interquartile range (IQR). The maximum is calculated as Q3 + 1.5*IQR and the minimum as Q1 -1.5*IQR, and these are indicated in the boxplot by short horizontal lines at the top and bottom, respectively, of each boxplot in the figure (see e.g. the boxplot for DIESEL). Any value greater than maximum and minimum GFLOPS values is an outlier and is represented in the figure as circles beyond the minimum and maximum values. Moreover, we have added average GFLOPS values for each kernel along with the dashed lines indicating their positions for each boxplot.

Compare in Figure 10 the six statistics (minimum, Q1, Q2, Q3, maximum, and average) for all six boxplots and note that DIESEL provides the best all-round performance. DIESEL has the highest values for all six statistics. It provides the maximum GFLOPS value, 29.25 compared to 20.61 for the second-best kernel, CSR. DIESEL has the GFLOPS Q3 value, 12.5 compared to 8.79 for the second-best kernel, CSR. Q2 and Q1 are also the highest for DIESEL, 5.46 and 1.33, respectively, compared to 4.71 and 0.91, for the second-best kernel, CSR. DIESEL also has the highest minimum value among the kernels but this is not clearly visible in the figure. Clearly, DIESEL also outperforms other kernels in terms of the average values with its average GFLOPS value of 8.5. The highest values for all six kernels can also be compared to see that DIESEL provides the highest GFLOPS, 54.39, compared to 37.11 for the *overall* second-best kernel, CSR. Note that DIESEL selects DIA for that particular matrix and hence the highest GFLOPS for DIA is the same as for DIESEL.

Figure 10 can be used to calculate the average speedup of DIESEL against the other five kernels, which is 3.86, 1.42, 6.07, 1.63, and 1.66 for COO, CSR, DIA, ELL, HYB, respectively. To elaborate on the reasons for the various performance behaviors depicted by these six kernels, we will continue, hereon, to compare the performance of the six kernels using various metrics and visualizations.

5.2 SpMV Performance Comparison: Aggregated over entire Dataset

Figure 11 gives performance comparison of DIESEL with other kernels, however, this time aggregated over all the 1056 matrices in our dataset. Figure 11(a) compares the aggregate performance (GFLOPS) of DIESEL with the other five kernels (COO, CSR, DIA, ELL and HYB). The aggregate performance for each of the five kernels, as well as the DIESEL tool, are calculated by executing the SpMV operation for each of the 1056 matrices in our dataset; i.e., we executed the SpMV kernels five times using each of the storage schemes and another (sixth) time using the DIESEL tool that predictively used one of the five kernels based on the deep learning trained model. Two exceptions here are the ELL and DIA kernels because these were unable to execute for 350 and 807 matrices of the 1056 matrices, respectively, see Page 9. The GFLOPS and execution *time* for these missing matrix performance values for ELL and DIA kernels have been replaced with zeros. We will discuss this issue further later on using detailed performance analysis.

The aggregate performance reported here includes the losses of the DIESEL tool caused due to the prediction inaccuracy (see Section 5.5 for prediction accuracy evaluation of the DIESEL tool). The GFLOPS values for each matrix are

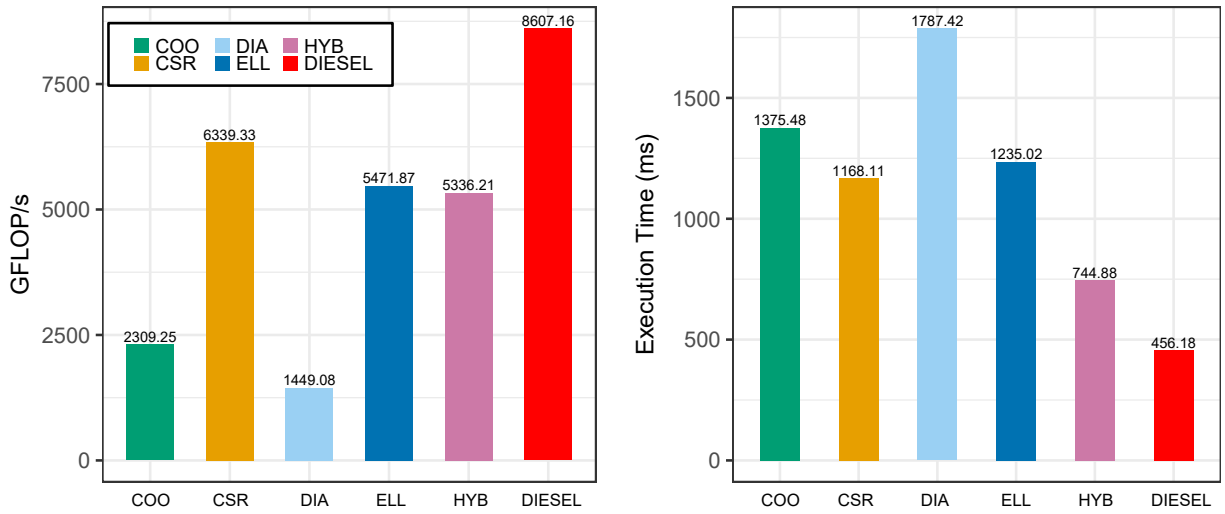


Figure 11: Performance comparison of DIESEL with other kernels (aggergated over all 1056 matrices). (**a**) The throughput in GFLOP/s; (**b**) Time taken in seconds for the execution

calculated as $2 \times nnz/executiontime$. Clearly, DIESEL (with its 8.61 GFLOPS) outperforms all the other five SpMV kernels by a fairly big margin (outperforms the next best, CSR, by 2.26 TFLOPS). The results presented are for a single SpMV computation for all matrices. The absolute gains depicted in the figure would be much higher for the complete solution of a linear equation system which typically involves many SpMV based iterations for the solution vector convergence. The CSR kernel provides the next best performance (6.34 TFLOPS), followed by ELL (5.47 TFLOPS), HYB (5.33 TFLOPS), COO (2.31 TFLOPS), and DIA (1.45 TFLOPS). The DIA kernel due to its unsuitability for general matrices provides the least performance. The COO format is not very efficient in storing the nonzero values and hence the COO kernel does not perform well compared to the other formats. The HYB would typically perform better than the CSR kernel for larger matrices but in this case the performance for CSR is better due to the performance aggregation over the 1056 matrices (see Section 5.4 for further discussion on this issue). It can be calculated from the GFLOPS values in Figure 11(a) that the DIESEL tool delivers 35.7%, 57.2%, and 61.2% higher performance (GFLOPS) over CSR, ELL, and HYB, respectively.

Figure 11(b) compares the aggregate execution times of DIESEL with the other five kernels similar to Figure 11(a) and manifest DIESEL's superior performance again over all other kernels. HYB provides the second best execution time (744ms) followed by CSR (1.1s), ELL (1.2s), COO (1.3s), and DIA (1.7s). It can be calculated from the execution times reported in Figure 11(b) that DIESEL delivers 1.63×, 2.56×, 2.70×, 3.01×, and 3.91× speedup over HYB, CSR, ELL, COO, and DIA, respectively. The performance of DIESEL and other kernels in terms of execution times mostly matches that of the GFLOPS performance in Figure 11(a). The exception is that the HYB kernel gives the second best performance in terms of the aggregate execution times. The ELL kernel holds 4th position this time as opposed to the 3rd in terms of GFLOPS, though the GFLOPS for the ELL kernel were only slightly higher than HYB. We will dig deeper into this behavior of the HYB, ELL and CSR kernels in Section 5.4 with additional visualization and analysis.

5.3 SpMV Performance Comparison: Aggregated over Application Domains

Figure 12 gives performance (GFLOPS and execution time) comparison of DIESEL with other kernels. In contrast to Figure 11, where the performance was aggregated for all the matrices (i.e., all the application domains), in this case, the performance is aggregated over matrices for each application domain. The results are compiled in two separate graphs (application domain groups 1 and 2) due to a large difference in the performance of some application domains over others (note the differences in the range of the y-axis scales). Note in Figure 12(a) that DIESEL (red bar) outperform other kernels by good margins. It outperform CSR by the highest margin for the "Structural" domain. This is because it has the highest number of mostly large matrices. DIESEL has slightly poorer performance for three application domains (out of 26), Graphics, Statistics and unweighted graphs. However, the three domains have a relatively small number of matrices (2, 4, and 15 matrices, respectively; see Table 1) and the average difference of performance for the three domains does not exceed 4 GFLOPS. Our future work will address this by improving the accuracy of the DIESEL

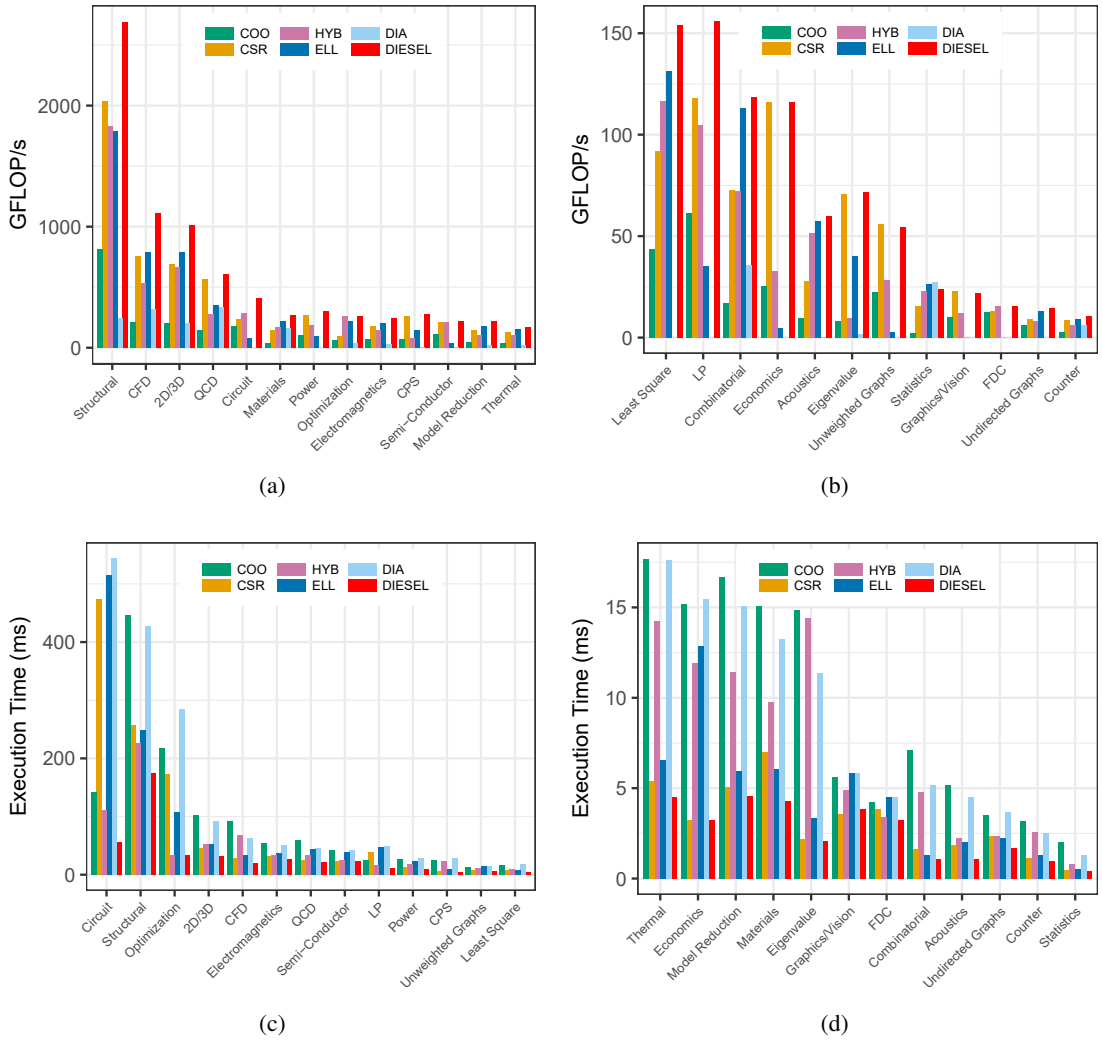


Figure 12: Performance comparison of DIESEL with other kernels (aggregated by the application domains). (**a**) GFLOPS (Domain Group 1); (**b**) GFLOPS (Domain Group 2); (**c**) Execution Time (Domain Group 1);(**d**) Execution Time (Domain Group 2)

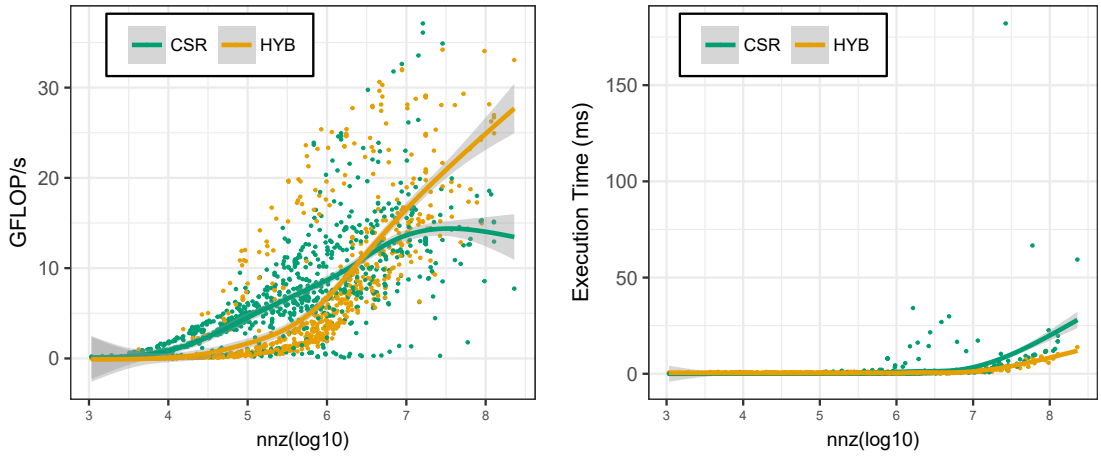
prediction component. Figure 12c and 12d compare the execution times in millisecond for the six kernels. DIESEL outperforms other kernels in most of the cases.

5.4 SpMV Performance Comparison: Deeper Investigations

5.4.1 Performance with respect to the Specific Dataset Characteristics

Figure 13a plots GFLOPS against *nnz* for each of the 1056 matrices in our data set for the CSR and HYB kernels. Note that the GFLOPS data points are dominantly visible for HYB on the upper side of the graph, implying that HYB, overall, provides the highest GFLOPS values compared to CSR. The trend lines in the figures are drawn from the GFLOPS (and execution times) data points using the Generalized Additive Model (GAM). The gray area above and below the trend lines show the confidence interval (with 0.95 probability). Note in the figures that the performance of the two kernels depends on the number of nonzero entries in the matrices.

The HYB format does not perform typically well compared to CSR for matrices with a relatively smaller number of *nnz* values (except, perhaps, for some peculiarly irregular matrices) [20, 6]. This behavior could also be seen in Figure 13a where the GFLOPS values for HYB are mostly lower than CSR on the left of the diagram, up until *nnz* values of approximately 3 million, where the two trend lines cross each and HYB begins to provide better GFLOPS. A large



(a) (b)

Figure 13: Performance comparison of CSR and HYB kernels (plotted against *nnz* for each of the 1056 matrices). (**a**) GFLOPS; (**b**) Execution Times

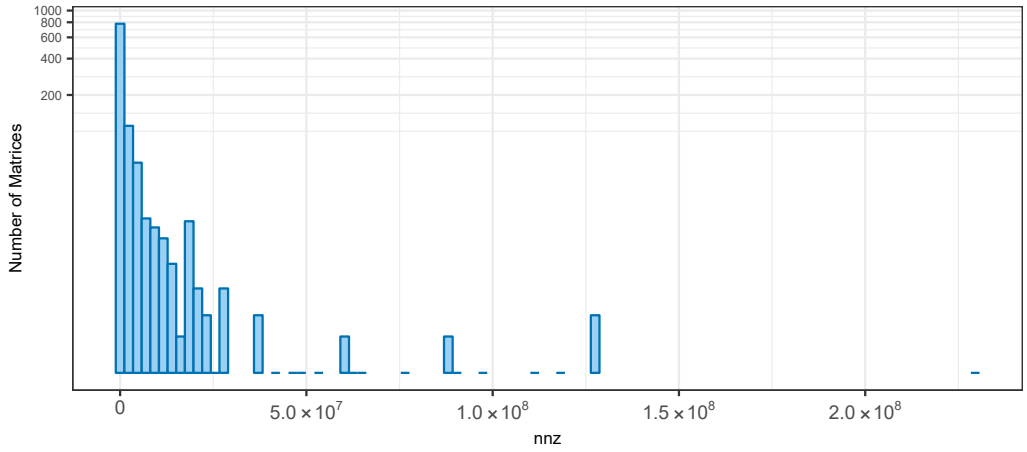
Table 4: Distributions of the Best Performing SpMV Kernels in our Dataset

	COO	CSR	DIA	ELL	HYB	Total Number
Matrices (%)	1%	57%	6.25%	22.25%	13.5%	1056
nnz (%)	0.13%	21.45%	2.72%	28.77%	46.93%	3, 684, 604, 669

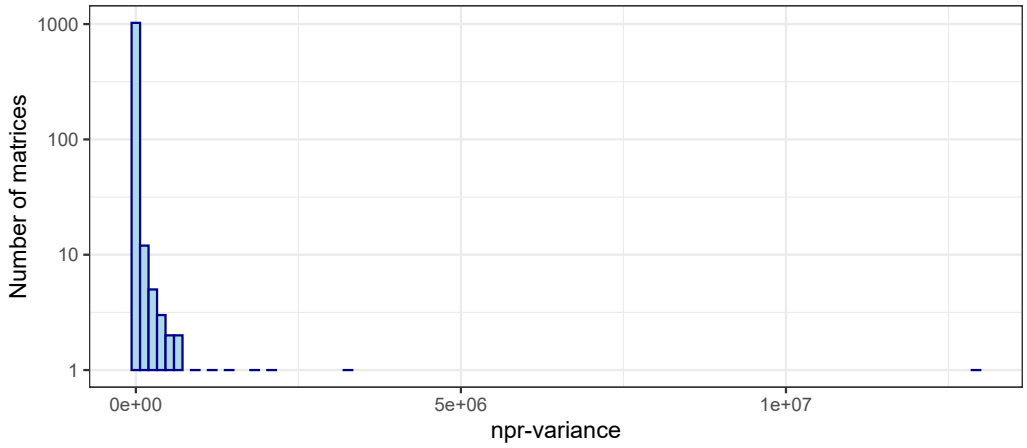
proportion of our dataset comprises matrices with relatively smaller values for *nnz* and *npr variance* (see Figure 14). Note in Figure 14 that most of the matrices in our dataset have $nnz \geq 25M$ while the maximum *nnz* in the dataset is 229M (the distribution of *npr variance* is also similar). This is also visible in Figure 13a where the scattered points on the left side of the intersection point are densely populated (implying a larger number of matrices with a relatively smaller *nnz*). For this reason, the aggregated performance of the HYB kernel for the whole data set is poorer than would be expected for if a majority of matrices had large *nnz* values. Therefore, in Figure 11, the aggregation of GFLOPS ($= 2 \times nnz/executiontime$) for HYB and CSR kernels does not keep the inversely proportional behavior with the execution time results (i.e., due to *nnz* in the equation).

To further elaborate this, Table. 4 provides the distributions of the best performing SpMV kernels in our dataset. The first row provides the distribution in terms of the percentage of 1056 matrices in the dataset that performed the best for each of the five kernels. The CSR kernel performed best for the largest number of matrices, 57%, equaling 602 matrices, as opposed to 13.5% for HYB. The second row gives the distribution of the best performing kernels in terms of *nnz* values that these kernels processed in computing SpMVs. In this case, the best performing kernel is HYB with its 46.93% share while the share of CSR has dropped to 21.45%. The total number of *nnz* in the dataset are 3, 684, 604, 669. Figure 13b plots execution *times* against *nnz* for each of the 1056 matrices shows similar behavior as for GFLOPS in Figure 13a though in an inverted manner. The details of the initial behaviors of HYB and CSR are not visible in the figure due to most of the $time$ values located near the $zero$ line compared to some relatively much higher values located near the top line ($200ms$). The number of visible points in Figure 13b are a few because most of the execution $time$ values are very small (near zero) and are clustered on top of each other under the two lines.

Figure 15 plots the GFLOPS performance in a manner similar to Figure 13a, but this time comparing HYB with the ELL kernel. Note a behavior similar to the HYB versus CSR plot except that HYB provides better performance than ELL with a smaller margin compared to CSR and that the crossing point for the GAM trend lines is much earlier ($nnz = 200,000$) than in the HYB versus CSR plot ($nnz = 3million$). Figure 16 plots the GFLOPS performance in a similar manner to Figure 13a and Figure 15, but this time for all the six kernels. Note that DIESEL (red data points) provide the highest GFLOPS values among the six kernels. Moreover, the GAM trend lines show that DIESEL provides the best performance among all the kernels and followed by HYB, CSR, ELL, COO, and DIA, respectively. The gray area above and below the trend lines show the confidence intervals (with 0.95 probability). Note the many data points for ELL and DIA near the zero line are indicating $zero$ GFLOPS. In this paper, we have taken the position that inability of a scheme to execute on a device (due to insufficient memory space) implies $zero$ GFLOPS (ELL and DIA



(a)



(b)

Figure 14: The histogram depicting the frequency of the matrices with respect to (a)*nnz* and (b)*npr variance*

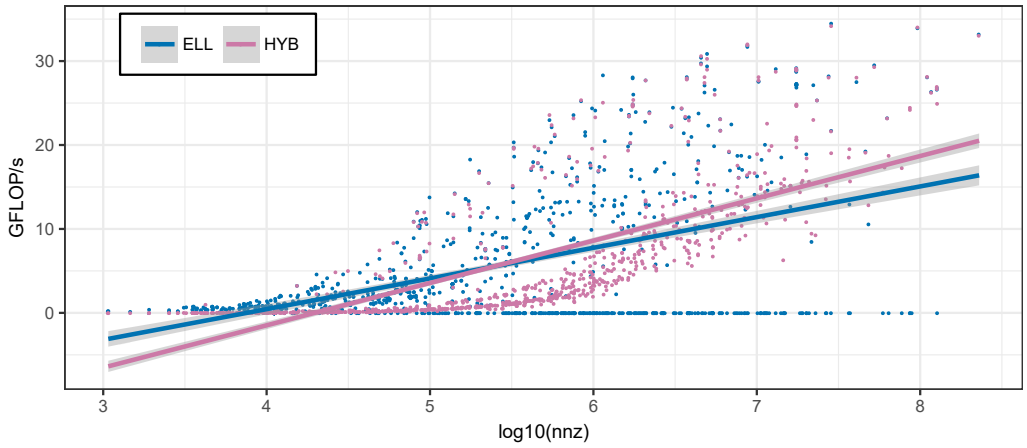


Figure 15: Performance comparison of ELL and HYB kernels (plotted against *nnz* for each of the 1056 matrices)

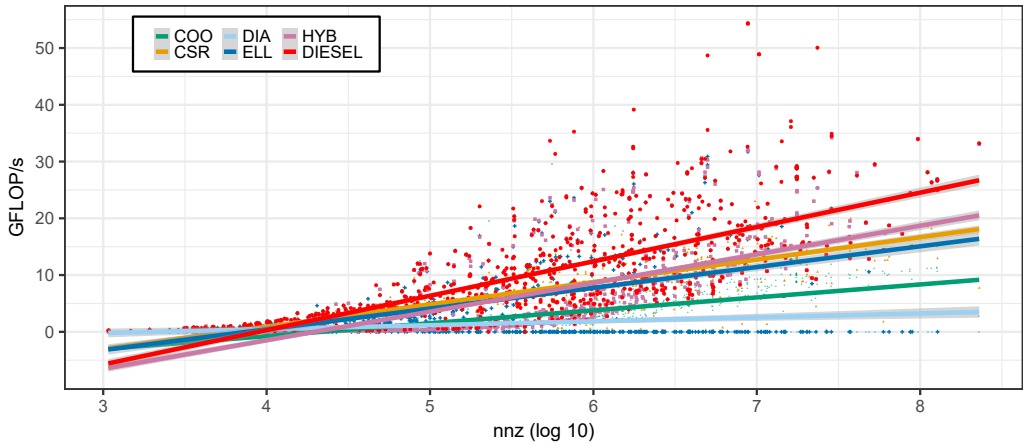


Figure 16: Performance comparison of all five storage formats (plotted against *nnz* for each of the 1056 matrices

were unable to execute for 350 and 807, respectively, of the 1056 matrices, see Page 9). This point may need further investigation and debate by the SpMV research community. Moreover, the community also needs to study the combined memory, execution time, and GFLOPS behavior of SpMV kernels because time performance can usually be traded-off by higher memory usage. We would like to clarify that the use of $zero$ GFLOPS values in this paper does not give any advantage to the DIESEL tool because it does not select any specific kernel, rather it selects the best kernel in terms of execution time.

5.4.2 Performance with respect to *nnz* and *npr variance* of All Matrices

A particular aspect of an SpMV kernel is its performance behavior against the increasing *npr variance* and *nnz* of matrices. This behavior is very much needed in SpMV tools because this is where most schemes are unable to deliver performance, causing poor utilization of massive parallelism offered by GPUs. To illustrate this, Figure 17 plots the GFLOPS performance of DIESEL and its sub-kernels against *npr variance* and *nnz* using a contour plot. A contour plot provides a two-dimensional view of a 3D space in which all points that have the same output (here GFLOPS) are connected to produce contour lines of constant responses. A contour plot consists of predictors on the x- and y-axes (the *npr variance* and *nnz*), contour lines that connect points that have the same response value (GFLOPS), and colored contour bands that represent ranges of the response values (GFLOPS). The color legend at the bottom of the figure depicts colors ranging from blue to green, yellow, orange and red, representing GFLOPS in increasing order from near zero to maximum, 54.39. More yellow, orange and red in the figure imply higher GFLOPS by a kernel and more blue and green shades means the kernel has overall low GFLOPS.

The *nnz* and *npr variance* values have been normalized between the range [0,1]. The maximum and minimum values of *nnz* and *npr variance* (*var*) are listed in Table 3. In our dataset, most of the matrices have relatively small values for *nnz* and *npr variance* compared to the maximum values of these two features (the maximum values for *nnz* and $nprv$ are 229M and 12M, respectively; see Figure 14). Secondly, the few matrices that have highest values for *nnz* may not have a high value for $nprv$, and vice versa. Moreover, the contour plots are 2D representations of 3D spaces and cannot be directly thought as plotting values in a 2D space. For these reasons, the plots in Figure 17 have lower-triangular shapes.

The worst overall performance is provided by DIA (dominantly blue color in Figure 17c) due to its inability to run for a large number of matrices in the dataset and poor performance for non-diagonal matrices. The second worst performance is by COO, evident in Figure 17a, by the blue and green shades that cover the majority of the figure except for a little yellow shade. The CSR kernel depicted in Figure 17b provides high GFLOPS (yellow) for low values of *npr variance* and *nnz* (bottom-left), however, provides poor performance overall for high *npr variance* and *nnz*. Particularly, its performance for matrices with high *npr variance* (the bottom-right) is poor. The ELL kernel in Figure 17d provides good performance for high *nnz* matrices but performs poorly for high *npr variance*. This behavior of ELL is well-known and is due to the inordinately large extra-padding required for matrices with high *npr*. HYB was proposed to address these deficiencies of ELL and Figure 17e verifies this by the yellow shade spread around most of the figure. The bottom of the figure is covered in light green which shows that it does not perform well for low *nnz* and high *npr variance*. DIESEL in Figure 17f inherits the good performance of all the kernels and provides the best perform visible in the largest yellow-shaded area among the six contour plots. Note that it inherits the good performance of CSR for low *nnz*

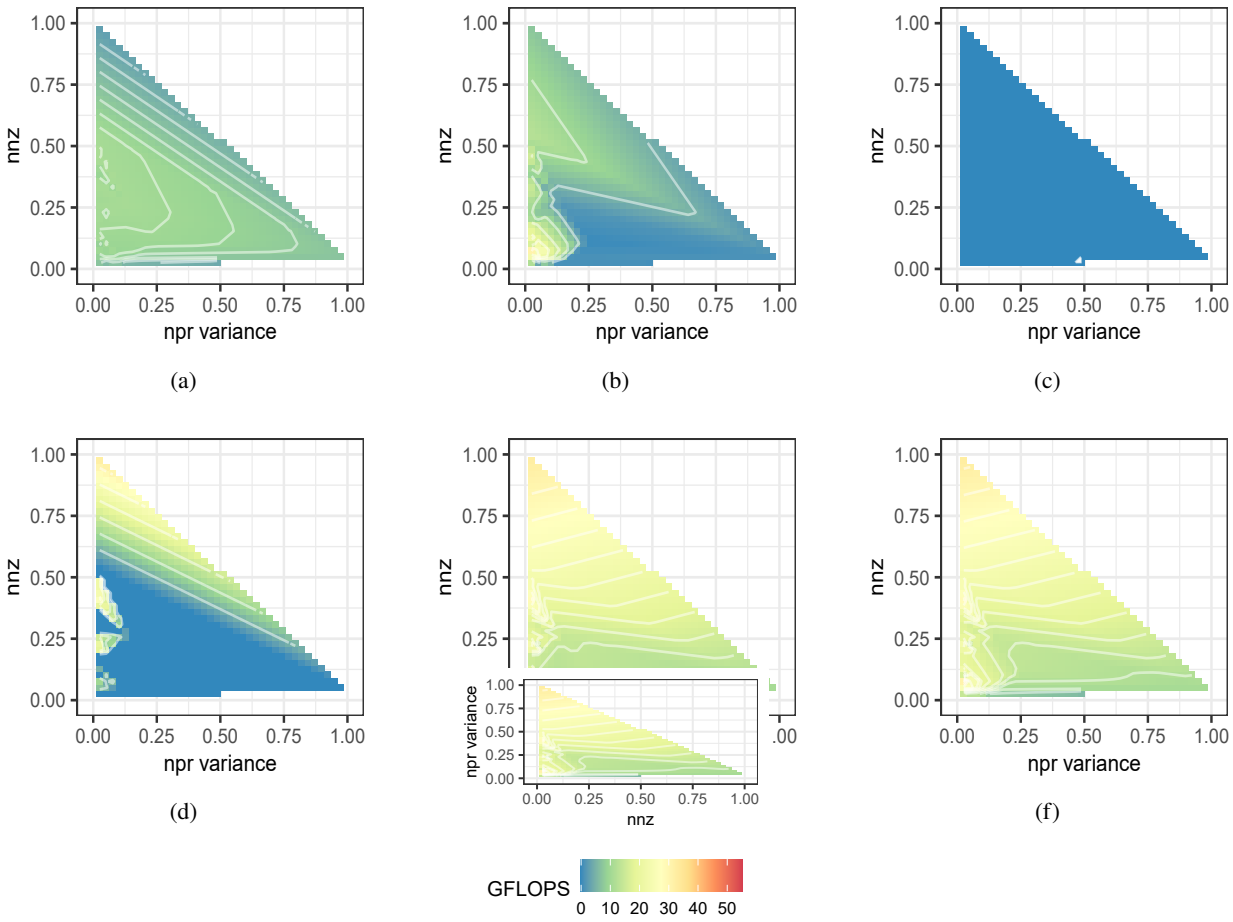


Figure 17: GFLOPS vs *npr variance* and *nnz* for (a) COO; (b) CSR; (c) DIA; (d) ELL; (e) HYB; and (f) DIESEL.

values (the bottom-left) where HYB fails. There is a small patch of blue shade at the very bottom of all six contour plots. These represent very small matrices and hence low GFLOPS are expected.

5.5 Prediction Accuracy, Performance Loss, and Performance Gain

To further understand the performance of the DIESEL prediction model, we calculated the accuracy of the classifier using Equation (1). The prediction *accuracy* is defined as the ratio of the number of correctly predicted sparse kernels to the total number of matrices in the dataset ($|dataset| = 1056$). B_i in the equation represents the GFLOPS value of the best kernel for the ith matrix and D_i is the GFLOPS value of the kernel predicted by DIESEL. DIESEL predicted 932 matrices correctly on average from the 10-fold cross-validation and achieved an accuracy of 88.2%. The numerator is the cardinality of the set which contains all i such that $B_i == D_i$, i.e., the numerator is 932.

$$accuracy(\%) = \frac{|\{i : \forall\ B_i == D_i\}|}{|dataset|} \times 100, \qquad \forall i \in [1, |dataset|] \tag{1}$$

The accuracy metric in Equation 1 gives the accuracy in terms of the number of correctly predicted kernel (or matrices). However, matrices range in terms of their computational workloads that depend on the number of nonzeros (*nnz*) to be multiplied and other matrix features. Consider the differences in two matrices, one with $nnz = 1,074$ and another with $nnz = 229M$; see Table 3. To include the workload aspects of a matrix for which a kernel has been correctly predicted, we define the term *workload accuracy* (or *effective accuracy*) as the ratio of the total number of *nnz* of the correctly predicted sparse kernels to the total number of *nnz* in the whole dataset. It can be calculated using Equation 2. This is an important metric, for instance, for comparing the performance of multiple predictive tools (see Section 5.6). Note that *nnz* is a good measure of the workload, however, for very sparse matrices m and n could become important too. The

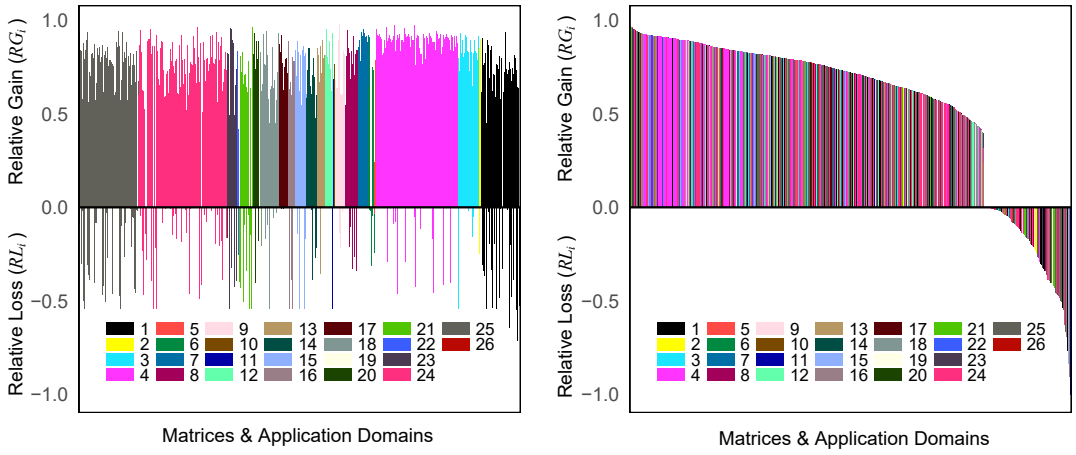


Figure 18: DIESEL: Relative Gains and Losses (Ratio of GFLOPS) for all 1056 Matrices (26 Application Domains)

workload itself will depend on the specific sparse storage format and kernel, and this will require further investigation and improvements by the community in the future.

$$workload\ accuracy\ (\%) = \frac{\sum\{nnz(i) : \forall\ B_i == D_i\}}{\sum nnz(i)} \times 100, \quad \forall i \in [1, |dataset|] \tag{2}$$

Another important performance metric is the average (performance) relative loss (*ARL*) for the dataset against the ideal performance incurred due to the inaccuracy of the DIESEL tool, calculated as in Equation (3) below. The *ARL* of DIESEL caused due to its inaccuracies is 4.4%. Equation (4) gives the relative (performance) loss, RL_i, for the ith matrix. Note that only 124 matices in the dataset are incorrectly predicted and the condition $B_i == D_i$ holds for the rest 932 of them implying a zero RL_i value for each of them.

$$ARL(\%) = \frac{100}{|dataset|} \sum RL_i, \quad \forall i \in [1, |dataset|] \tag{3}$$

$$RL_i = (B_i - D_i)/B_i, \quad \forall\ B_i > D_i \tag{4}$$

Similar to RL_i, Equation (5) defines the relative (performance) gain RG_i for the ith matrix. The gain only applies if it is not a loss, i.e., $B_i == D_i$. A_i in Equation (6) represents the average GFLOPS of all the schemes except the best one. DIESEL tries to predict the best kernel from the set of five kernels, COO, CSR, DIA, ELL, and HYB. The average is computed by adding the GFLOPS values of the four of the five kernels, excluding the best kernel, i.e., B_i (because DIESEL has predicted the best kernel).

$$RG_i = (D_i - A_i)/D_i, \quad \forall\ B_i == D_i \tag{5}$$

$$A_i = \frac{1}{4} \sum_{K_i \neq B_i} K_i, \quad \forall K \in \{coo, csr, dia, ell, hyb\} \tag{6}$$

Figure 18a shows the relative performance gains (RG_i) and losses ($-RL_i$) of the DIESEL tool over other kernels, for all the 1056 matrices, grouped by the 26 application domains (see Table 1). The positive y-axis indicates the gain (due to DIESEL providing higher GFLOPS, overall, over the other five kernels) and the negative y-axis indicates the loss against the *ideal* performance (due to the wrong predictions/ inaccuracies). The width of an application domain indicates the number of matrices in that domain. The colored area in the positive y-axis clearly shows that the gain of the DIESEL tool is much larger than the loss both in terms of the number of matrices (932 versus 124) and the length of the lines (gains are mostly touching the top value, 1.0, while most lines on the bottom are less than |0.5|). Figure 18b shows the

same data but in a descending gain and ascending loss order. Note the area of all the losses appearing on the right-hand side of the plot, reaffirming that the gain is much larger than the losses caused due to the tool inaccuracies.

The calculation of relative gain RG_i and A_i in Equations (5) and (6) needs further explanation. We had different options to compute the gain of DIESEL over the other kernels. Our premise for calculating the gain was that a user who does not have access to a tool such as DIESEL would usually use a kernel that is based on a single sparse storage format. This is because the user in most cases would not know the specific sparse scheme that would provide the best performance for each matrix in their matrix-set (except strictly banded and other structured matrices). In this context, one possibility was to compute the gain of DIESEL over the next best kernel in GFLOPS. Based on our various analyses reported earlier in this section, the overall next best kernel is CSR and we have already compared the performance of DIESEL and CSR using average speedup and other metrics. For the sake of argument, the next best kernel could be HYB, or ELL, or another kernel, and we have already compared DIESEL with all the five kernels. We have chosen here to compute the gain of DIESEL using average performance of other kernels (A_i). However, we do not believe that this (A_i) is the best or only way of such gain comparisons, and we plan to investigate this further in the future. We have devised a few new, deeper, ways of performance analyses and comparisons of SpMV computations in this work, to get feedback from the community, and hopefully, these deeper comparisons to evolve into widely accepted and standard metrics over time. Note that there are very few tools available similar to DIESEL and we have compared DIESEL with them in the next section.

5.6 Comparison with the Cutting-Edge AI-based SpMV Techniques

We now compare the SpMV performance of our DIESEL tool with other AI-based SpMV tools. We have found five works in the literature that are relevant to our work (these are from three different research groups with two sets of extended works). These works use AI techniques to automatically select storage formats to improve SpMV performance and have already been reviewed in Section 3. We compare DIESEL with four [8, 9, 5, 10] of these five works. The fifth work, the SMATER [11] library, is currently not available for use or development and hence we are unable to compare DIESEL with it. Moreover, we have compared DIESEL with other AI decision tree-based works [8, 10]. Therefore, SMATER has been somewhat covered in the comparative analysis presented in this paper.

The first work is by Sedaghati et al. [8], who proposed a decision tree based classifier for the prediction of the best sparse storage format for the SpMV computations. They used two sets of features. A set of 3 simple features namely *nnz*, *density*, and *mean*. The second feature set was a superset of the first feature set and comprised complex features including some features at the matrix block level. They concluded that the higher computational complexity of the complex feature set leads to a higher overhead but does not produce a corresponding improved performance. We, therefore, have focused on their basic feature set in implementing their prediction model and comparing it with DIESEL. The second work is by Benatia et al. [9] who have used multi-class SVM to predict the best storage formats for a matrix. They have used a new feature set (8 features, including the number of rows, the number of columns, *nnz*, *density*, *mean*, among others) for representing the sparse matrices on GPUs. The third work is also by Benatia et al. [5]. This is an extension of their earlier work discussed above [9]. In this extended work, they use a weighted SVM with the help of pairwise classification approach (instead of multi-class SVM). The features used are similar to their previous work but in this case the feature combinations are selected based on the pair of storage formats for creating each pair-wise model. We have used the same feature set (with 8 features) in implementing these two works while comparing with DIESEL. The fourth work is by Israt et al. [10] that implements three AI methods to predict the best sparse storage format. The methods are multi-layer perceptrons (MLP), gradient boosting based models (XG-Boost), and ensembles of MLP. They have used two feature sets in this work that are the same as were used by Sedaghati et al. in [8], except with some modifications such as inclusion of the feature *mean* in the feature sets. They compared their work with the decision-tree based models by Sedaghati et al. [8] (the first work discussed above) and the SVM based models proposed by Benatia et al. [9] (the second work). They concluded that the XGBoost ensemble performs better compared to the other two AI models they had used. Hence, we have implemented XGBoost for comparison with DIESEL using their extended feature set.

We have implemented all four of these works ([8, 9, 5, 10]) in order to compare our tool DIESEL with them. The Decision Tree and XGBoost models were implemented using the Weka software libraries. We used two implementations of the Decision Tree algorithm, SimpleCart and BFTree. For both these Decision Tree-based models, we set the parameters to the default values, minNum=2 and numFold=5. The Multiclass SVM model and the Weighted SVM model were implemented using the libSVM package [35]. We used the Gaussian Radial Basis Function (RBF) as the kernel. The two parameters *c* and *gamma* were selected using the cross-validation technique provided in the package. The software and parameters were used in accordance to the original works [8, 9, 5, 10], however, for fair comparison purposes, the results were collected using our dataset.

Table 5: Comparison of DIESEL with other AI-based works [8, 9, 5, 10] (based on our dataset and respective feature sets)

Technique	Features	*accuracy (%)*	*workload accuracy (%)*	*ARL* (%)
DT (Simple Cart) [8]	3	69.5	72.38	10.8
DT (BFTree) [8]	3	67.9	70.26	11.5
Multiclass SVM [9]	8	80	78.73	9.3
Pairwise Weighted SVM [5]	8*	84.7	85.31	7.65
XGBoost [10]	11	85.9	82.53	8.39
DIESEL	14	88.2	91.96	4.4

* This is the overall number of features. Pairwise, the selected features are from these 8 as per original work.

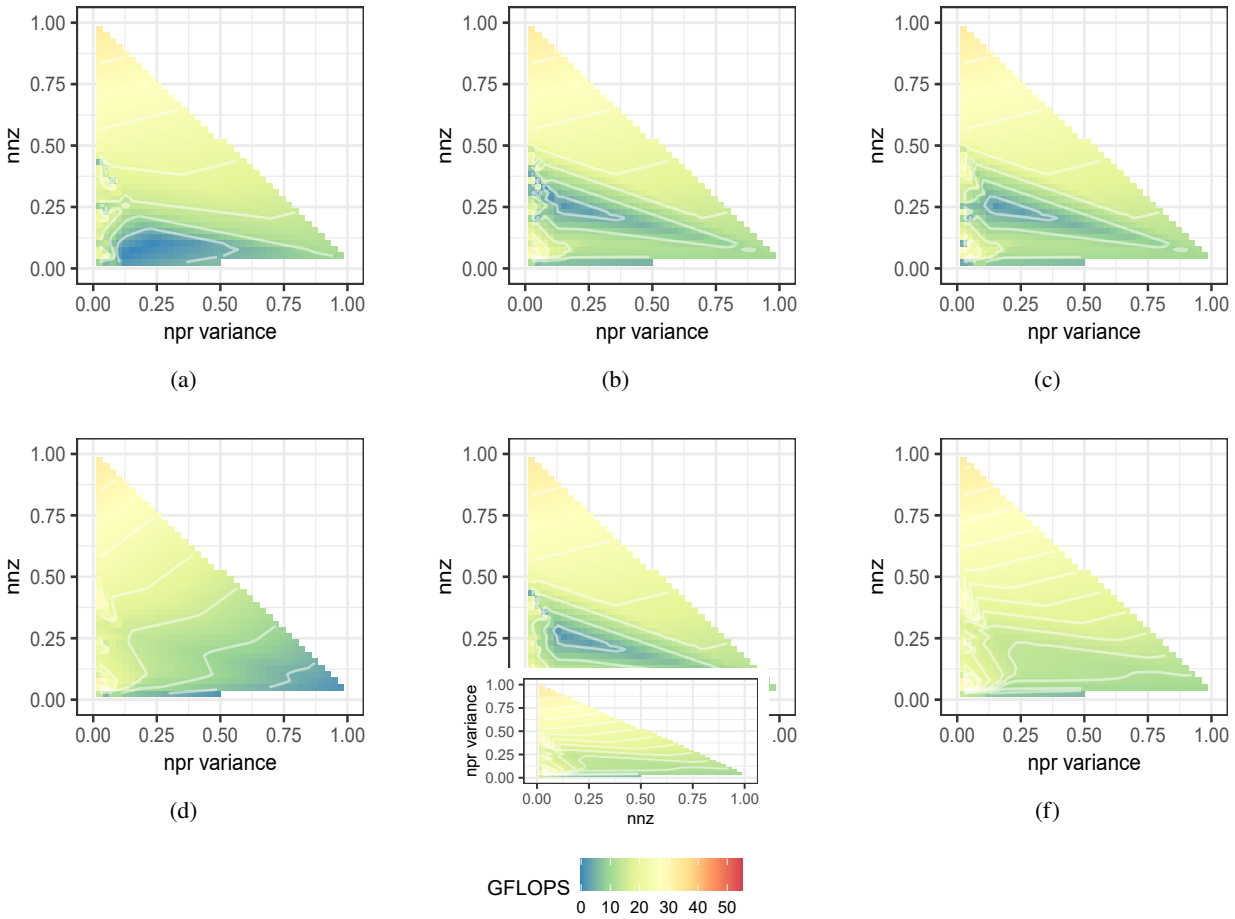


Figure 19: GFLOPS vs. *npr variance* and *nnz* for (a) Simple Cart; (b) BFTree; (c) Multiclass SVM; (d) Pairwise SVM; (e) XGBoost; and (f) DIESEL.

Table 5 provides a comparative performance of DIESEL and the other four works [8, 9, 5, 10]. The first column lists the compared techniques, the second column lists the respective number of features, the third column gives the *accuracy* (Equation(1)), the fourth column gives the *workload accuracy* (Equation(2)), and the fifth column gives the *ARL* (Equation(3)). Note in the table that DIESEL delivers the best performance in all three categories with a relatively good margin. The second-best *accuracy* is provided by XGBoost while Pairwise Weighted SVM provides the second best *workload accuracy* and *ARL* (smaller loss is better). Note also that the Decision Tree (DT) based classifier using Simple Cart provides better *workload accuracy* and *ARL* than BFTree. Moreover, both SVM based tools provide better performance than both DT based tools for all three metrics. See Section 5.5 for the details on the three performance metrics.

Figure 19 compares the behavior of DIESEL against *npr variance* and *nnz* in a similar fashion to Figure 17, however, this time for the six AI-based SpMV tools listed in Table 5. Note that DIESEL (Figure 19f) provides the best overall performance among all the kernels evident by the yellow shade that is spread around most of the plot area except the light green color around the bottom area. BFTree (Figure 19b), Multiclass SVM (Figure 19c), and XGBoost (Figure 19e) demonstrate similar GFLOPS behavior, with some differences, due to the prediction of similar kernels as the best-performing ones.

The time complexity of the feature extraction process has been discussed in Section 4.5.1 (see also Table 2). The space complexity of the feature calculation process for the entire dataset is $O(|N||f|)$, where $|N|$ is the number of matrices in the dataset and $|f|$ is the number of features. The space complexity of the feature extraction process for all the compared machine learning techniques in this section is also $O(|N||f|)$.

The space and time complexity of machine learning and deep learning algorithms can be found, for instance, in [36]. Accordingly, the complexity of the DIESEL training algorithm is $O(|L| \cdot n_{l_i}^2) \approx O(n_{l_i}^3)$, because in DIESEL, $|L| \approx n_{l_i}$, where $|L|$ is the number of layers in the deep neural network used in the DIESEL tool. The training complexity for decision trees and SVM-based techniques are $O(n \cdot |f| \cdot \log(|f|))$ and $O(n^2 \cdot |f|)$ respectively [36], where n is the number of training samples utilized.

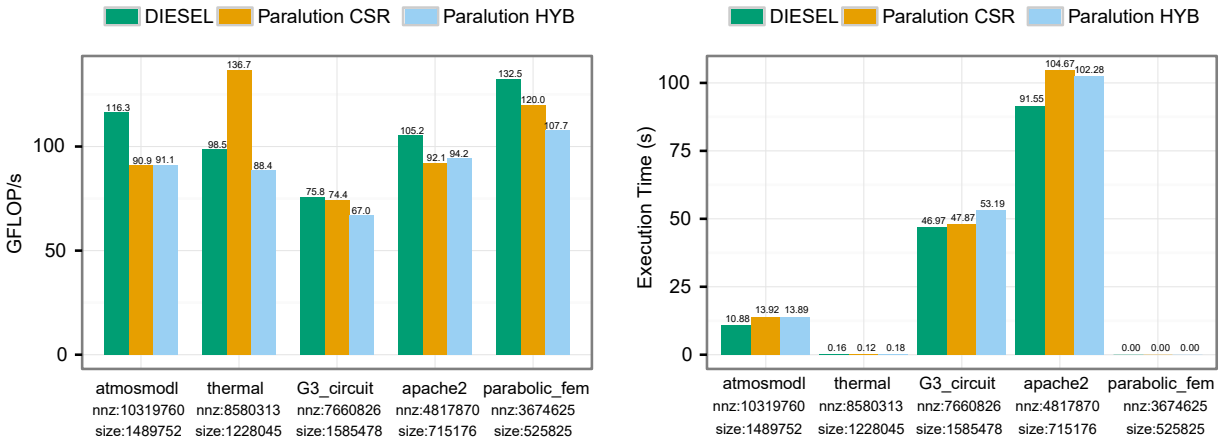
The time complexity of the classification process in DIESEL is $O(|f| \cdot n_{l_1} + n_{l_1} \cdot n_{l_2} + n_{l_2} \cdot n_{l_3} + \cdots + n_{l_{L-1}} \cdot n_{l_L} + n_{l_L} \cdot |S|) \approx O(n_{l_i}^2)$, where n_{l_i} denotes the number of neurons in the i^{th} layer and $|S|$ denotes the size of the output layer (the number of outputs). Comparatively, the decision tree-based algorithms [8, 9, 11] have a time complexity of $O(\log(|f|))$ and SVM-based algorithms [5, 9] have a time complexity of $O(n_{sv} \cdot |f|)$, where n_{sv} is the number of support vectors [36]. The decision tree-based techniques have a linear time complexity compared to DIESEL and SVM-based techniques that have quadratic complexity. However, the quadratic complexity of the machine learning process is compensated by the fact that the overall accuracy and the resulting performance of the SpMV kernels for SVM-based and DIESEL tools are better than the decision tree-based tools (see Table 5). Note that it is well-known [5, 11, 10] that the inference times are negligible compared to the total SpMV computation times within the iterative solutions and therefore the inference times can be ignored. The size (space) of the DIESEL deep model is $O(|L|)$ compared to $O(n)$ and $O(|f|)$ for decision tree and SVM-based techniques, respectively [36].

We did not compare empirically the training and inference times of DIESEL and the four AI-based tools because we have implemented the training and classification algorithms in parallel on GPUs while other AI-based tools use tools such as Weka that run sequentially on CPUs. Such a comparison would clearly show the strength of our tool due to the parallelisation of the processes, however, can be considered unfair by some due to the use of high performance computing (HPC) resources.

Comparing the usability and functionality of DIESEL with the other four works, we have integrated the deep learning-based techniques to our tool, whereas Sedaghati et al. [8] and Benatia *et al*. [9, 5] have used the tools Weka and libSVM for classification, respectively. We have developed an SpMV kernel, whereas Sedaghati *et al*. [8] use both CUSP and cuSPARSE libraries and Benatia *et al*. [9, 5] use the CUSP library for SpMV kernels. Israt *et al*. [10] also use Weka. DIESEL incorporates all storage formats used in these two works as well as the DIA format that has not been used by them. We extend the feature set and propose six new features, compared to their feature sets, and achieve better performance. A detailed analysis of the feature set has been provided which is not the case with the works compared here. We implement and perform analysis of a Jacobi iterative solver for sparse linear equation systems, which has not been considered by the others except Benatia et al. [5] who use cuBLAS to provide PCG (Preconditioned Conjugate Gradient) solver for the iterative solution. An important advantage of our work is that the DIESEL tool allows the integration of the training and prediction phases such that each new input matrix can be used to retrain the model if desired. The prediction part is inbuilt in DIESEL using software kernels in CUDA C++. Therefore, the DIESEL tool provides better usability since it does not require setting up multiple libraries and their environments.

5.7 Comparison with the Paralution Library for Jacobi Iterative Solution

We now compare the Jacobi iterative solver of the DIESEL tool with Paralution, which is a widely-used commercial library comprising various sparse iterative solvers and preconditioners on CPU, MIC and GPU devices. Specifically, we compare DIESEL with the CSR and HYB implementations of CUDA-based Jacobi iterative solver found in Paralution. The CSR and HYB are selected because they provide the best performance among all the available formats in Paralution. DIESEL, as we know, will predict a specific kernel to solve the sparse linear equations systems. We have selected Paralution for comparison with DIESEL because it has an implementation of the Jacobi iterative solver. CUSP is another library with a Jacobi implementation that we have compared with in our tests but did not present its results in this paper because it is widely known to have poor performance compared to Paralution [37]. Our experiments have also suggested the same that it performs poorly compared to Paralution and DIESEL. For benchmarking data, we have selected five



(a) (b)

Figure 20: DIESEL: Comparison with Paralution. (**a**) GFLOPS; (**b**) Total Solution Time

square matrices from various application areas, made available by the University of Florida Sparse Matrix repository [2]. We used these matrices instead of the dataset used for SpMV performance partly because these matrices have been widely used in the literature (see e.g., [38, 39, 40]) for performance comparison of sparse algebra methods. Moreover, a large proportion of matrices in the dataset do not converge with Jacobi because these are poorly conditioned.

Figure 20 compares performance of Jacobi iterative solvers in DIESEL and Paralution in terms of GFLOPS and total solution time for the five selected matrices. The matrix sizes are given at the bottom of the graphs. The matrices, G3_circuit, apache2, and parabolic_fem did not have the b vector, hence we computed a right-hand side corresponding to the unitary solution, i.e., $b = Ae$ with e the unitary vector as in [41]. The relative error for convergence was set to 10^{-7}. For the apache2 matrix, we collected results for 100,000 iterations because it did not converge earlier than 100,000 iterations. Figure 20(a) compares the GFLOPS performance. Note that DIESEL outperforms Paralution in 80% of the cases.

The accuracy and performance of the DIESEL prediction engine were 100% because for each matrix it selected the best available kernel. For *atmosmodl*, DIESEL dynamically used the DIA kernel, and hence there is an improvement of 21% as compared to the Paralution CSR. DIESEL selected the CSR kernel for *apache*2, ELL for parabolic_fem and G3_circuit matrices, and the CSR kernel for the *thermal* matrix. The relative performance of DIESEL for four of the five matrices is better than Paralution, however, mostly with small margins except for *atmosmodl* where the margin is higher (21%). Paralution performed better for the *thermal* matrix with a good margin. Note that the CSR kernel in Paralution performs overall better than its HYB kernel. Though the Paralution HYB has slightly better GFLOPS for the *atmosmodl* and *apache*2 matrices compared to CSR (a difference of 0.2 and 2.1 GFLOPS, respectively), the difference is negligible. It appears that the CSR implementation in Paralution maybe better optimised than its HYB implementation. It may also be possible that the specific implementation of CSR in Paralution somehow favours certain matrix sparsity features over Paralution HYB, and vice versa.

Comparing DIESEL and Paralution CSR for the *thermal* matrix, note that Paralution CSR outperforms DIESEL (as well as Paralution HYB) by relatively considerable margins. Both DIESEL and Paralution CSR use the CSR kernel for the *thermal* matrix and this performance difference is due to the specific implementations of CSR and other kernels in DIESEL and Paralution that may create variations in performance due to the different and varying sparsity features across matrices. The implementations of a specific storage scheme can vary based on the thread-level and warp-level load balancing of the computations on the GPU and other factors. A specific implementation may have been optimised for a particular GPU and its execution on a different GPU device may not produce the desired performance due to the differences in the underlying architecture.

On the issue that DIESEL was only able to provide a marginally better performance, overall, compared to Paralution, note that our focus in this paper was not on optimizing GPU kernels because the DIESEL tool could include and use the best implementations available for any storage scheme from any SpMV tool. Moreover, it is possible to have multiple SpMV kernels of a specific scheme optimised for various sparsity features and train the DIESEL tool to select the best kernel from multiple kernels of a specific storage scheme (*e.g.* CSR) based on the sparsity features of the matrix. The

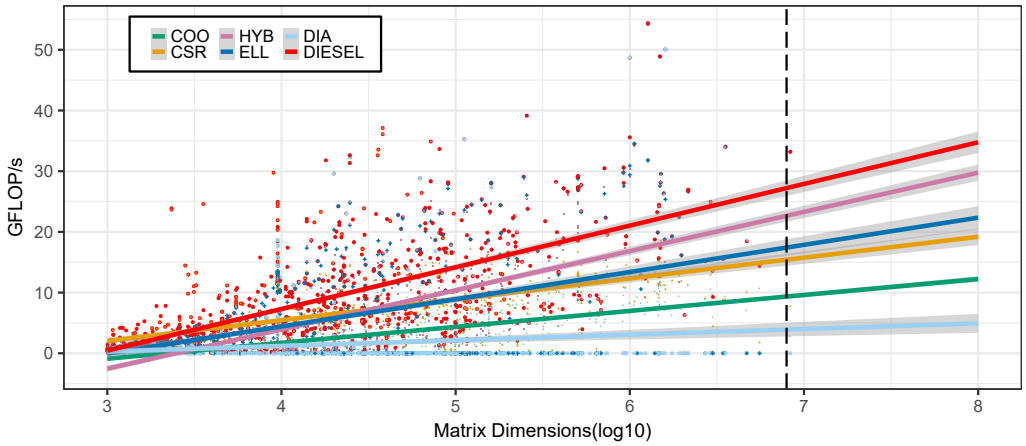


Figure 21: Extrapolation of performance for all six kernels (plotted against m for each of the 1056 matrices

important point to note here is that DIESEL had correctly selected the best kernel for execution. The implementation of highly optimized kernels in DIESEL would enable DIESEL to perform significantly better than Paralution due to its ability to select the best kernels. Future work will focus on optimizing the implementations of the DIESEL kernels.

Figure 20(b) compares the execution times for DIESEL and Paralution and depicts a pattern similar to Figure 20(a). The time taken for executing parabolic_fem is 0.000612 seconds, 0.000682 seconds, and 0.000555 seconds for Paralution CSR, Paralution HYB, and DIESEL, respectively. Since the values are small the text on the graph indicates them as zero.

Figure 21 shows the GFLOPS performance of DIESEL and 5 other kernels against the size of the matrices (m). Compare this with Figure 16 where we had plotted the same information but against *nnz*. We have currently tested our tool on the Nvidia K20 GPU. The figure extrapolates the GFLOPS of the five matrices to show the potential performance of our DIESEL tool on larger matrices that would be able to fit within the Nvidia Pascal and Volta GPUs. Note that the difference between the GFLOPS of DIESEL and other kernels increases with the increase in the matrix size.

6 Conclusions and Future Work

Sparse linear algebra is central to many areas of engineering, science, economics, business, and social sciences. This paper proposed DIESEL, a deep learning-based tool that predicts and executes the best performing SpMV kernel for a given matrix using a feature set carefully devised by us through rigorous empirical and mathematical instruments. The dataset comprises 1056 matrices from 26 different real-life application domains including computational fluid dynamics, materials, electromagnetics, economics, and more. We proposed a range of new metrics and methods for performance analysis, visualization and comparison of SpMV tools. DIESEL provided the best performance among all the kernels and tools by considerable margins in all aspects of the performance analysis. Comparing it with its SpMV kernels utilizing a specific sparse storage scheme, DIESEL provided the highest and average GFLOPS values of 54.39 and 8.5, compared to 37.11 and 6.0 by the next best kernel, delivering speedups of $\times$1.47 and $\times$1.42, respectively. Comparing DIESEL with the other cutting-edge AI tools, it provided the best performance with its accuracy of 88.2%, workload accuracy of 91.96%, and average relative loss of 4.4%, compared to the respective values of 85.9%, 85.31%, and 7.65% by the next best tool. DIESEL outperformed the commercial tool Paralution in 80% of the test cases. A particular aspect of the DIESEL performance among all kernels and tools is its considerably better behavior against the increasing *npr variance* of matrices. This behavior is very much needed in SpMV tools because this is where most schemes are unable to deliver performance, causing poor utilization of massive parallelism offered by GPUs.

The community has done considerable work on proposing new methods for SpMV computations and iterative sparse solvers on GPUs. Due to vast variations in matrix features, no single method performs well across all sparse matrices. A few tools on automatic prediction of best-performing SpMV kernels have emerged recently; the research in this direction is in its infancy and requires many more efforts to fully utilize the potential. The utilization of a GPU by the existing SpMV kernels is far from its full capacity. Moreover, the development and performance analysis of SpMV techniques on GPUs have not been studied in sufficient depth. The extensive results and analyses presented in this paper provide several key insights into the development and performance of the SpMV tools and how these relate to the matrix

datasets and the various performance metrics. This will allow the community to further improve and compare basic and computational intelligence based SpMV tools in the future.

In the future, the DIESEL tool will be extended in multiple dimensions. Currently, it uses the Jacobi iterative method for solving linear equation systems. We plan to extend it with additional iterative methods and SpMV kernels. We also plan to study the impact of varying hyper-parameters with respect to the sparsity structure of the matrices to dynamically determine the hyper-parameters and thereby improve the prediction accuracy and prediction time. We will continue to enhance the tool by extending the dataset, improving the feature sets, training procedures, better classification algorithms, and comparing it with other well-known commercial and free tools. We also plan to extend the tool to support multiple GPUs.

Acknowledgements

This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under grant number RG-6-611-40. The authors, therefore, acknowledge with thanks the DSR for their technical and financial support. This work is supported by the HPC Center at King Abdulaziz University, Jeddah, Saudi Arabia. The experiments reported in this paper were performed on the Aziz supercomputer at KAU. We are thankful to the anonymous reviewers whose comments have greatly improved the quality of this paper.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html.

[2] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38 (1):1:1–1:25, December 2011. ISSN 0098-3500. doi:10.1145/2049662.2049663. URL http://doi.acm.org/10.1145/2049662.2049663.

[3] Yousef Saad and Henk A. van der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1 - 2):1 – 33, 2000. ISSN 0377-0427. doi:http://dx.doi.org/10.1016/S0377-0427(00)00412-X. URL http://www.sciencedirect.com/science/article/pii/S037704270000412X. Numerical Analysis 2000. Vol. III: Linear Algebra.

[4] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[5] Akrem Benatia, Weixing Ji, Yizhuo Wang, and Feng Shi. Bestsf: A sparse meta-format for optimizing SpMV on GPU. *ACM Trans. Archit. Code Optim.*, 15(3), September 2018. ISSN 1544-3566. doi:10.1145/3226228. URL https://doi.org/10.1145/3226228.

[6] Salvatore Filippone, Valeria Cardellini, Davide Barbieri, and Alessandro Fanfarillo. Sparse matrix-vector multiplication on GPGPUs. *ACM Transactions on Mathematical Software*, 43(4):1–49, 2017. doi:10.1145/3017994.

[7] T. Muhammed, R. Mehmood, A. Albeshri, and I. Katib. SURAA: A novel method and tool for loadbalanced and coalesced SpMV computations on GPUs. *Appl. Sci.*, 9(5):947, 2019. doi:10.3390/app9050947.

[8] Naser Sedaghati, Te Mu, Louis-Noel Pouchet, Srinivasan Parthasarathy, and P. Sadayappan. Automatic selection of sparse matrix representation on GPUs. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, pages 99–108, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3559-1. doi:10.1145/2751205.2751244. URL http://doi.acm.org/10.1145/2751205.2751244.

[9] A. Benatia, W. Ji, Y. Wang, and F. Shi. Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 496–505, Aug 2016. doi:10.1109/ICPP.2016.64.

[10] I. Nisa, C. Siegel, A. S. Rajam, A. Vishnu, and P. Sadayappan. Effective machine learning based format selection and performance modeling for SpMV on GPUs. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1056–1065, May 2018. doi:10.1109/IPDPSW.2018.00164.

[11] Guangming Tan, Junhong Liu, and Jiajia Li. Design and implementation of adaptive SpMV library for multicore and many-core architecture. *ACM Trans. Math. Softw.*, 44(4), August 2018. ISSN 0098-3500. doi:10.1145/3218823. URL https://doi.org/10.1145/3218823.

[12] Thaha Mohammed. A novel deep learning based iterative solver for large sparse linear equation systems. Master's thesis, King Abdulaziz University, January 2017. URL https://kaupp.sa/Details/Thesis/133000.

[13] David B Kirk and W Hwu Wen-Mei. *Programming massively parallel processors: A hands-on approach*. Morgan kaufmann, 2016.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[15] S. Usman, R. Mehmood, I. Katib, and A. Albeshri. ZAKI+: A machine learning based process mapping tool for SpMV computations on distributed memory architectures. *IEEE Access*, 7:81279–81296, 2019. ISSN 2169-3536. doi:10.1109/ACCESS.2019.2923565.

[16] Hana Alyahya, Rashid Mehmood, and Iyad Katib. *Parallel Iterative Solution of Large Sparse Linear Equation Systems on the Intel MIC Architecture*, pages 377–407. Springer International Publishing, Cham, 2020. ISBN 978-3-030-13705-2. doi:10.1007/978-3-030-13705-2_16. URL https://doi.org/10.1007/978-3-030-13705-2_16.

[17] Rashid Mehmood and Jon Crowcroft. Parallel iterative solution method for large sparse linear equation systems. *Computer Laboratory: University of Cambridge*, 2005.

[18] Sardar Usman, Rashid Mehmood, and Iyad Katib. *Big Data and HPC Convergence for Smart Infrastructures: A Review and Proposed Architecture*, pages 561–586. Springer International Publishing, Cham, 2020. ISBN 978-3-030-13705-2. doi:10.1007/978-3-030-13705-2_23. URL https://doi.org/10.1007/978-3-030-13705-2_23.

[19] S. Usman, R. Mehmood, I. Katib, A. Albeshri, and S. Altowaijri. ZAKI: A smart method and tool for automatic performance optimization of parallel SpMV computations on distributed memory machines. *Mobile Networks and Applications*, 2019.

[20] Nathan Bell and Michael Garland. Efficient Sparse Matrix-Vector Multiplication on CUDA. Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.

[21] Jee W. Choi, Amik Singh, and Richard W. Vuduc. Model-driven autotuning of sparse matrix-vector multiply on GPUs. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, pages 115 – 126, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605588773. doi:10.1145/1693453.1693471. URL https://doi.org/10.1145/1693453.1693471.

[22] Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, Achim Basermann, and Alan R Bishop. Sparse matrix-vector multiplication on GPGPU clusters: A new storage format and a scalable implementation. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1696–1702. IEEE, 2012.

[23] Muthu Manikandan Baskaran and Rajesh Bordawekar. Optimizing sparse matrix-vector multiplication on GPUs. Technical Report RC24704 (W0812-047), IBM Research, 2009.

[24] Pantea Zardoshti, Farshad Khunjush, and Hamid Sarbazi-Azad. Adaptive sparse matrix representation for efficient matrix–vector multiplication. *The Journal of Supercomputing*, pages 1–21, 2015.

[25] P. Guo, L. Wang, and P. Chen. A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1112–1123, May 2014. ISSN 1045-9219. doi:10.1109/TPDS.2013.123.

[26] K. Li, W. Yang, and K. Li. Performance analysis and optimization for SpMV on GPU using probabilistic modeling. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):196–205, Jan 2015. ISSN 1045-9219. doi:10.1109/TPDS.2014.2308221.

[27] Sarah AlAhmadi, Thaha Muhammed, Rashid Mehmood, and Aiiad Albeshri. *Performance Characteristics for Sparse Matrix-Vector Multiplication on GPUs*, pages 409–426. Springer International Publishing, Cham, 2020. ISBN 978-3-030-13705-2. doi:10.1007/978-3-030-13705-2_17. URL https://doi.org/10.1007/978-3-030-13705-2_17.

[28] Roger G Grimes, David R Kincaid, and David M Young. *ITPACK 2.0 user's guide*. Center for Numerical Analysis, The University of Texas at Austin, 1979.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[30] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009. ISSN 1935-8237. doi:10.1561/2200000006. URL http://dx.doi.org/10.1561/2200000006.

[31] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, August 2013. ISSN 0162-8828. doi:10.1109/TPAMI.2013.50. URL http://dx.doi.org/10.1109/TPAMI.2013.50.

[32] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[33] Laurens van der Maaten and Geoffrey Hinton. Visualizing non-metric similarities in multiple maps. *Machine Learning*, 87(1):33–55, 2012. ISSN 1573-0565. doi:10.1007/s10994-011-5273-4. URL http://dx.doi.org/10.1007/s10994-011-5273-4.

[34] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, Feb 2017. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature21056. Letter.

[35] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[36] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. On-device machine learning: An algorithms and learning theory perspective, 2020.

[37] Max Grossman, Christopher Thiele, Mauricio Araya-Polo, Florian Frank, Faruk O. Alpak, and Vivek Sarkar. A survey of sparse matrix-vector multiplication performance on large matrices. *ArXiv*, abs/1608.00636, 2016.

[38] Carlo Janna, Massimiliano Ferronato, and Giuseppe Gambolati. The use of supernodes in factored sparse approximate inverse preconditioning. *SIAM Journal on Scientific Computing*, 37(1):C72–C94, 2015. doi:10.1137/140956026. URL http://dx.doi.org/10.1137/140956026.

[39] Ruipeng Li and Yousef Saad. GPU-accelerated preconditioned iterative linear solvers. *The Journal of Supercomputing*, 63(2):443–466, 2013. ISSN 1573-0484. doi:10.1007/s11227-012-0825-3. URL http://dx.doi.org/10.1007/s11227-012-0825-3.

[40] Mickeal Verschoor and Andrei C. Jalba. Analysis and performance estimation of the Conjugate Gradient method on multiple GPUs. *Parallel Computing*, 38(10 - 11):552 – 575, 2012. ISSN 0167-8191. doi:https://doi.org/10.1016/j.parco.2012.07.002. URL http://www.sciencedirect.com/science/article/pii/S0167819112000609.

[41] Massimo Bernaschi, Mauro Bisson, Carlo Fantozzi, and Carlo Janna. A factored sparse approximate inverse preconditioned conjugate gradient solver on graphics processing units. *SIAM Journal on Scientific Computing*, 38 (1):C53–C72, 2016. doi:10.1137/15M1027826. URL http://dx.doi.org/10.1137/15M1027826.