






Article

AAQAL: A Machine Learning-Based Tool for Performance Optimization of Parallel SPMV Computations Using Block CSR

Muhammad Ahmed ¹, Sardar Usman ^{2,†}, Nehad Ali Shah ³ , M. Usman Ashraf ^{4,*,†} ,
Ahmed Mohammed Alghamdi ⁵ , Adel A. Bahaddad ⁶  and Khalid Ali Almarhabi ⁷ 

¹ Department of Software Engineering, Islamia University Bahawalpur, Bahawalpur 63100, Pakistan; mahmad.riaz102@gmail.com

² Department of Computer Science and Software Engineering, Gran Asian University Sialkot, Sialkot 53310, Pakistan; sardar.usman@gaus.edu.pk

³ Department of Mechanical Engineering, Sejong University, Seoul 05006, Korea; nehadali199@sejong.ac.kr

⁴ Department of Computer Science, Government College Women University, Sialkot 53310, Pakistan

⁵ Department of Software Engineering, College of Computer Science and Engineering, University of Jeddah, Jeddah 21493, Saudi Arabia; amalghamdi@uj.edu.sa

⁶ Department of Information System, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia; dbabahaddad10@kau.edu.sa

⁷ Department of Computer Science, College of Computing in Al-Qunfudah, Umm Al-Qura University, Makkah 24381, Saudi Arabia; kamarhabbi@uqu.edu.sa

* Correspondence: usman.ashraf@gcwus.edu.pk

† These authors contributed equally to this work.



Citation: Ahmed, M.; Usman, S.; Shah, N.A.; Ashraf, M.U.; Alghamdi, A.M.; Bahaddad, A.A.; Almarhabi, K.A. AAQAL: A Machine Learning-Based Tool for Performance Optimization of Parallel SPMV Computations Using Block CSR. *Appl. Sci.* **2022**, *12*, 7073. <https://doi.org/10.3390/app12147073>

Academic Editor: Agostino Forestiero

Received: 5 May 2022

Accepted: 5 July 2022

Published: 13 July 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Abstract: The sparse matrix–vector product (SpMV), considered one of the seven dwarfs (numerical methods of significance), is essential in high-performance real-world scientific and analytical applications requiring solution of large sparse linear equation systems, where SpMV is a key computing operation. As the sparsity patterns of sparse matrices are unknown before runtime, we used machine learning-based performance optimization of the SpMV kernel by exploiting the structure of the sparse matrices using the Block Compressed Sparse Row (BCSR) storage format. As the structure of sparse matrices varies across application domains, optimizing the block size is important for reducing the overall execution time. Manual allocation of block sizes is error prone and time consuming. Thus, we propose AAQAL, a data-driven, machine learning-based tool that automates the process of data distribution and selection of near-optimal block sizes based on the structure of the matrix. We trained and tested the tool using different machine learning methods—decision tree, random forest, gradient boosting, ridge regressor, and AdaBoost—and nearly 700 real-world matrices from 43 application domains, including computer vision, robotics, and computational fluid dynamics. AAQAL achieved 93.47% of the maximum attainable performance with a substantial difference compared to in practice manual or random selection of block sizes. This is the first attempt at exploiting matrix structure using BCSR, to select optimal block sizes for the SpMV computations using machine learning techniques.

Keywords: sparse matrices; artificial intelligence; machine learning; compressed sparse row (CSR); block CSR; decision trees; random forest; gradient boosting; ada boost; high performance computing (HPC)

1. Introduction

The sparse matrix–vector product (SpMV) is considered to be the most important, time-consuming, and widely used scientific kernel [1] in various scientific disciplines such as computer graphics, computer vision [2–5], robotics [3], 3D or 2D problems [4], acoustic problems [5,6], thermal problems, healthcare, networking, operational research, and computational fluid dynamics (CFD).

Different features influence the efficiency of SpMV computations [7]. These include the specifications of storage formats, sparsity patterns, and implementation of software and hardware platforms [8]. Over the years SpMV optimization has revolved around

proposing new storage schemes (i.e., compressed sparse column (CSC), extended Block Compressed Sparse Row (BCSR) [9], compressed sparse row (CSR) [10], diagonal, hybrid coordinate format [9,10]), developing software libraries (i.e., Intel MKL, Trillions Project [11], cuSPARSE [7], and Cusp [12]) and efficiently exploiting DRAM bandwidth and cache hierarchies [13].

The main objective of wide range of proposed optimization strategies is to reduce the overall work set of algorithms and the main advantage of blocking methods is same, as they allow more room for optimizing the computation of the kernel.

By reducing the memory pressure, blocking methods leave more space for optimization, focusing on the computational chunk of the kernel, such as loop unrolling and vectorization, which can additionally improve execution.

Random selection and trial and error experimentation for choosing the optimal block size are time consuming and error prone. Owing to the irregular structure of matrices, random selection of the block size needs to be repeated several times for each matrix. Our proposed solution overcome above mentioned problems associated with random selection of block sizes by predicting near optimal block size against any sparse matrix by primarily utilizing the structure of matrix.

The contribution of this study revolves around the idea of proposing a machine learning-based performance optimization tool (AAQAL) for sparse matrix–vector products in shared memory architectures using BCSR format. For this purpose, we used different machine learning algorithms to predict the near-optimal block size and find the best solution for SpMV computation by exploiting the structure of the matrices.

The proposed research was conducted in six phases: matrix collection, matrix conversion to BCSR format, data set preparation (SpMV computation using BCSR), feature extraction based on the structure of sparse matrices, machine learning-based predictive models, and evaluations of the models using mean absolute error (MAE) and relative mean error (RME).

For our datasets, collections of matrices were sourced from the SuiteSparse Matrix Market Collection (UOF) [14]. We collected nearly 700 matrices from real-world problems targeting 43 different application domains. The sparse matrices are converted to B BCSR format, and different sparse matrix features are extracted. The minimum, maximum, and average execution times for different block sizes of a matrix have been attained. After preparation of our dataset, we used different base and ensemble machine learning models to predict the near-optimal block size for the best solution of SpMV by utilizing 80% of the dataset for training and the remaining 20% for testing.

The proposed methodology requires sparse matrix features to be extracted first, for arbitrary sparse matrix, before it can make predictions. Our proposed tool (AAQAL) extracts sparse matrix features and based on these features only, predicts the optimal block size at runtime before actual execution of SpMV computation. The SpMV computation will then be performed with the predicted block size with a manual or random selection of block sizes (in practice) is error prone and time consuming.

The intention to use BCSR storage format for computation is to reduce the sparsity structure and to combine the entries/elements block-wise containing certain information. As we are dealing with sparse matrices that hold most of the elements as zeros to avoid the problems created by sparse matrices, that is, time and space complexities, the BCSR format is used. The main aim of the proposed method is to allow users to automatically obtain the best configuration and performance for the implementation of SpMV calculations for any given sparse matrix. Based on the structure of the matrix, the AAQAL predicts the block size against any arbitrary sparse matrix. The suggested approach is primarily intended to enhance the performance optimization of existing contributions in this regard.

Generally, the proposed approach involves three main phases: data preparation, training, and testing. Data preparation mainly comprises the selection of raw sparse matrices, application of BCSR storage format, feature extraction, selection of block size with minimum, maximum, and average execution times. After collecting this dataset,

the dataset goes forward to the training phase. All matrices from different domains are collected through the SuiteSparse matrix market collection (UOF) [14] (see Figure 1).

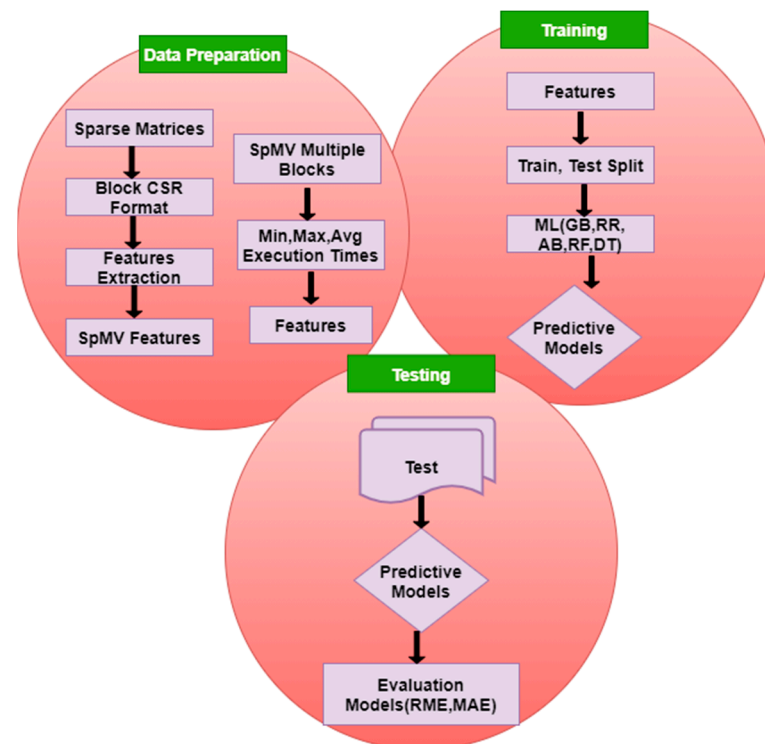


Figure 1. Data Preparation, training, and testing phases.

Matrices are converted from a matrix market format to BCSR. This conversion helped to remove the sparsity factor from the selected dataset. Subsequently, the sparse matrix–vector product computations were performed 2000 times with different block sizes to avoid any anomalies, and the minimum, maximum, and average execution times are recorded for each sparse matrix in the datasets.

Moreover, in the next phase, the selected features along with the optimal number of block sizes are fed to the training phase. The target variable is the block size and is selected along with the minimum execution time for SpMV computations. The training datasets are used to train different base and ensemble predictive models, which include random forest (RF), decision tree (DT), AdaBoost (AB), gradient boosting (GB), and ridge regressor (RR). These models were evaluated with the help of two evaluation metrics: mean absolute error (MAE) and relative mean error (RME).

This paper makes the following contributions.

- Propose, implement, and evaluate a machine learning-based tool (AAQAL) that helps users to choose an optimal block size for SpMV computation in a shared memory architecture.
- Train and test using nearly 700 real-world matrices obtained from 43 application domains, including linear programming, 2/3D problems, computer graphics, computer vision, and CFD.
- Perform in-depth analysis and performance evaluation using different base and ensemble machine learning techniques and visualization.

To the best of our knowledge, this work is the first to exploit the structure of the matrix to predict the near-optimal block size by using different base and ensemble machine learning predictive models. AAQAL is an Arabic word, which means “intelligent” or “smart”.

The remainder of this paper is organized as follows. A literature review and background information related to SpMV, BCSR, and machine learning algorithms are presented

in Section 2. Section 3 includes the proposed methodology and design. The detailed results and evaluation are presented in Section 4. Section 5 contains the conclusions of our research and suggests possible new directions for future research work. Table 1 lists the basic symbols used in this study.

Table 1. Symbol used in the paper.

Name	Symbols	Name	Symbols
Nnz	Number of nonzeros in the matrix.	A	$M \times N$ input matrix
M	Number of Rows	N	Number of Columns
X	$N \times 1$ dense vector	Y	$M \times 1$ Output vector
N	Number of matrices	F	Sparse Matrix features
SpMV	Sparse matrix–vector multiplication	CSR	Compressed Sparse Row
nnz _b	Number of non zeros per block	Nb	Dimension of each block
BCSR	Block Compressed Sparse Row		

2. Literature Review

2.1. Background

In this section, we first discuss the background on SpMV, BCSR storage format, and machine learning to give a brief overview.

2.1.1. Sparse Matrix–Vector Product (SpMV)

The SpMV kernel is one of the seven known categorized dwarfs of numerical methods, which have significant importance [15]. Sparse matrices are mostly populated by zero values. The SpMV kernel is formally denoted by:

$$y = Ax \quad (1)$$

where A is a multidimensional sparse matrix, x is the input dense vector, and Y is the resulting output dense vector (see Table 1). The dense and sparse matrix–vector products are shown in Figure 2.

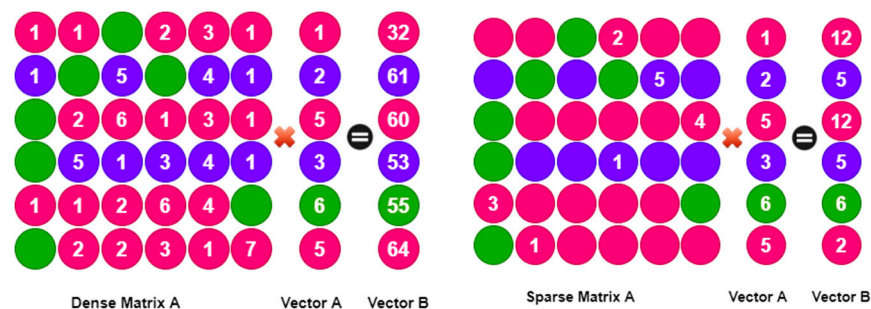


Figure 2. Dense vs. sparse matrix comparison.

Unlike dense matrices, sparse matrices require an explicit representation of the coordinates of non-zero elements and their manipulation in the sparse matrix–vector product demand memory bandwidth. The loop structure of the sparse matrix–vector product is more irregular than that of the dense matrix, which may lead to a less optimized compiler-generated code. The different data structures resulted in many indirect memory accesses that add performance bottlenecks by having additional load operations and poor compiler optimization. An efficient matrix–vector product involves multiple aspects, that is, the matrix pattern, to determine if the number of elements in each row varies strongly from row to row, if the sparse matrix is near a dense matrix or not. Knowing the matrix pattern, it is easier to find the most suitable storage scheme. Another important factor is the effective utilization of cache. The data distribution strongly depends on the matrix pattern. The use of pointers to exploit the sparsity of the matrix is unavoidable, as they lack spatial locality

and lead to poor cache utilization. An irregular sparse structure leads to a number of cache misses and memory indirection, or the use of pointers requires extra load operations.

2.1.2. BCSR

The efficient storage format for sparse matrix–vector product is highly hardware-dependent, which can be problematic for heterogeneous systems [16] and also has a variety of sparse matrix storage formats to choose the most optimal one. An efficient matrix–vector product involves multiple aspects, that is, the matrix pattern, to determine if the number of elements in each row varies strongly from row to row, if the sparse matrix is near a dense matrix or not. Knowing the matrix pattern, it is easier to find the most suitable storage scheme. The performance of the SpMV is affected by the temporal locality (reuse data that brought it into the memory) and spatial locality (use of every data element brought into the memory) of an application. Temporal locality can be improved with BCSR to maximize the data access before it is replaced in the cache.

If the sparse matrix X is made up of dense blocks of non-zeros with a regular pattern, we can change the CSR format to BCSR for the efficient exploitation of those block patterns. Block matrices usually result from the scattering of partial differential equations with certain degrees of freedom associated with grid points. The matrix is then divided into smaller blocks equal in size to the number of degrees of freedom, and each block is treated as a dense matrix, even if there are zeros. For example, if we have $n \times n$ matrix having n_b as the dimension of block and nnz_b as the nonzero per block (see Table 1), then we require total memory given by:

$$nnz = nnz_b \times n^2_b \quad (2)$$

The dimension of block n_d of matrix X is defined by

$$n_d = \frac{n}{n_b} \quad (3)$$

2.1.3. Machine Learning

With the advancement in computer technology, storage capabilities and sensor technologies have driven us to the age of big data; the term big data has become more prominent with a large amount of data availability. Machine learning algorithms are used most frequently in these applications to extract the desired information from such a tremendous amount of data. Machine learning is commonly used to build a model from prevailing datasets to predict new sample data under testing. For example, many URLs are labeled with their known reputation using analysis. This analysis is used by machine learning algorithms to train their model to classify new URLs according to their reputation.

Machine learning sample data contain a set of characteristics and labels of the data. In reputation analysis examples of URLs, a characteristic could be whether there is an inclusion of unusual text in URLs such as amazon.com. In machine learning, the characteristics and labels of the data set are organized in an oriented column format; hence, each characteristic is represented by a separate column. Therefore, the operation and analysis of the characteristics are assessed only for a specific column associated with it. The same principle was replicated using a row-oriented format in machine learning [17–20]. In this study, we focus on five different algorithms: decision tree (DT), random forest (RF), gradient boosting (GB), AdaBoost, and ridge regressor to predict the near-optimal block size for the sparse matrix.

2.2. Related Work

Over the years, different optimization techniques have been proposed, primarily focusing on register/cache blocking [21,22], reordering of rows/columns [19], compression [23,24], and introduction of new storage schemes. The storage formats are generic in nature and thus can be used to store any structure of sparse matrix, but with certain performance penalty. On the other hand, most storage schemes target the optimization of matrices with

specific patterns. Numerous blocking optimization techniques have been proposed but primarily focused on proposing new storage scheme. There is very little research conducted on the use of machine learning for the optimization of SpMV computation, and most of the efforts are dedicated only to automated format selection and execution time prediction based on sparse matrix features.

In BCSR matrix is divided into blocks of fixed sizes and optimal block size is dependent on structure of the matrix and underlying platform. BCSR provides many advantages including loop unrolling per block, compressing indexing and other low-level optimizations but also requires excessive padding to fill the blocks with explicit zero values [24]. Bitmap based sparse matrix storage format [25] uses blocks of fixed size and only non-zero elements need to be stored in values array. Bitmaps need to be stored that encode the non-zero structure using Bruijn sequences. Reduce memory usage resulted in the additional computation time. Similarly Block Compressed Common Coordinate (BCCOO) storage format [26] uses bit flags to point to start of the row. The use of bit array achieves high compression but ultimately requires additional array to be executed in parallel with SpMV Computation. Other optimization based on BCSR [27] applicable to sparse matrices with multiple blocks appear in recurring pattern. Variable Block Length (VBL) format [28] does not have fixed size blocks and requires execution of additional loop in SpMV computation to proceed through each block. The Compressed sparse eXtended format (CSX) [29] reduces the memory footprints of a sparse matrix by compressing index information using Compressed Sparse Row Delta Unit (CSR-DU). Index reduction is achieved but on the expense of high complexity in determining the substructures with considerable overhead. Other optimizations in the literature include Compressed Sparse block (CSB) format [30] and Recursive Sparse Blocks (RSB) [31], while pattern based representation (PBR) [32] aims to reduce storage overhead and keeping locality but with additional computation overhead. All these optimization techniques required alteration at algorithmic level on the expense of additional data structure or extra computation. With fixed size blocking techniques, selection of optimal block size is still not addressed in the literature. Trial and error based manual selection of blocks size is error prone and time consuming. Even after considerable efforts we may settle up with the block size which is not optimal. The proposed methodology does not change the SpMV algorithm but instead use the matrix structure only to select the optimal block size before actual execution of SpMV computation.

Recently, researchers started using machine/deep-learning techniques to optimize the performance of the SpMV kernel in both shared and distributed memory environments [33–35]. To estimate the optimal number of processes for SpMV computations of an arbitrary sparse matrix on a distributed memory computer system, in our previous work we [36] proposed a data-driven, machine learning method and tool known as ZAKI for SpMV computations. Experimental data from 2000 real-world matrices were collected from 45 heterogeneous application domains. The matrix sparsity structure was used to predict the optimum number of processes in distributed memory environments for a given matrix using various base and ensemble machine learning approaches [36]. In [37], the author proposed a machine learning approach to predict the optimal storage format for the SpMV kernel on a GPU (COO, CSR, ELL, and HYB). Support vector regression (SVR) and multilayer perceptron neural network (MLP) were used, and the performance results of the experiment on two separate GPUs (Fermi GTX 512 and Maxwell GTX 980 Ti) show that the average prediction error ranges between 7% and 14% [38]. In [39], a machine learning-based tool was proposed to automatically predict the best process mapping strategy and data distribution based on the structure of the matrix. Barreda et al. [37] proposed high level abstraction of the structure of sparse matrix based on fitting blocks of non-zero elements to convolutional neural network (CNN). The results demonstrate the robustness of CNN model and predicted the performance of CNN on intel HASWELL core and ARM A57 core. Barreda et al. [38] proposed CNN based model to estimate the execution time of SpMV and energy consumption of DRAM at variable processor frequencies with over all relative error ranges between 0.5% to 14%. Eberhardt et al. [40] proposed Block CSR based optimization

of SpMV for sparse matrix with dense block substructures. They proposed algorithms for both CPU intel many integrated core and GPU architectures and demonstrated that the performance of SpMV up to 4X faster than NVIDIA and cuSPARSE, 3x faster than MKL, and 147X times faster than intel math kernel library. The machine learning based optimizations for SpMV computation mostly focused on selection of optimal storage format, execution time or performance prediction and some efforts dedicated to energy consumption.

Achieving higher performance usually requires carefully choosing the sparse matrix storage format and fully utilizing the underlying system architecture [41–43]. Over the years, SpMV's optimization has revolved around ideas of improving memory bandwidth, matrix representation, instruction throughput, matrix reordering, cache, and register blocking, etc. The structure of the matrix is an unknown entity before runtime which motivates the idea of using machine learning based approach to predict the optimal block size before actual execution of SpMV computation using BCSR storage format. Block sizes are selected randomly and manual selection is time consuming and error prone. Even after trying different combinations of block sizes (random and manual selection in practice), we may settle with the block size which is not optimal.

To the best of our knowledge, our work is the first attempt at using machine learning techniques to optimize the performance of the SpMV kernel using the BCSR storage format and predict the near-optimal block size based on the structure of the matrix.

3. Materials and Methods

Our proposed methodology consists of three main phases: data preparation, feature extraction, and predictions using different machine learning algorithms. For data preparation, we collected nearly 700 matrices from multi-disciplinary application domains. The features were extracted by exploiting the structure of the sparse matrix. The training set comprises different features along with the block size for which the minimum execution time is obtained. The predictive model predicts the block size and is validated against arbitrary matrices. Selecting the correct set of features is important for the effective implementation of a predictive model. The overfitting issue arises when the number of features is excessively large for the training set. Therefore, dimensionality reduction and feature selection are very important to avoid overfitting.

Let n_b be the number of blocks and T_i represent the execution time of the i_{th} matrix where $0 < i < N$ and N is the total number of matrices in our dataset. Execution time of each matrix is recorded against different number of blocks; minimum and maximum time is recorded accordingly.

$$t_{imax} = \max(t_1, t_2, \dots, t_{nb}) \quad (4)$$

$$t_{imin} = \min(t_1, t_2, \dots, t_{nb}) \quad (5)$$

where nb is the Total number of blocks.

$$t_{iavg} = \sum_{i=1}^{i=nb} \frac{t_i}{nb} \quad (6)$$

t_{iavg} is calculated time when ever random block size is chosen for the execution.

For the predictions, related to performance optimization of SpMV in shared memory system via machine learning models, we followed the subsequent key steps:

- Step 1. Collection of real-world matrices from multi-disciplinary domains, sourced from Suit Sparse matrix market Collection (UOF) [35].
- Step 2. Conversion of sparse matrices from matrix market format to block CSR and SpMV computation using different block sizes. The SpMV computations are performed 2000 times for each sparse matrix using different block sizes.
- Step 3. We prepared sparse matrices dataset by recording SpMV execution time and their corresponding block sizes (Step 4). Furthermore, we utilized them in the training and testing process (Steps 5 and 6).

- Step 4. We have extracted set of SpMV features by utilizing the structure of the matrix along with a block size for which the minimum execution time is obtained.
- Step 5. We used different machine learning models to find the near optimal block size.
- Step 6. We evaluated the accuracy of the predictive models by using two metrics, mean absolute error and relative mean error.

SpMV computations, features extraction and training are represented in the following proposed Algorithm 1.

Algorithm 1: Algorithm of SpMV using Block CSR

```

1      A : is the input matrix
2      f : is the output features
3      n: is the number of matrices
4      A': is matrix in block CSR Format
5      nb: is the number of blocks
6      Function feature SpMV(A::in, f::out, timin ::out, nb ::out)
7      for J = 1 to J <= n do
8          convert matrix to Block CSR
9          A'j ← Aj
10         extract required features
11         fj ← extract features of A'j
12     end for
13     for k = 1 to k <= nb do
14         for i = 1 to i = 2k do
15             cal_exe_time[i] ← Call SPMV(A'j, x)
16         end for
17     end for
18     Where x is the dense vector

```

3.1. Creating Dataset

After collection of sparse matrices from multiple domains and their conversion from the matrix market format to BCSR format, SpMV computation is performed on each matrix with multiple block sizes and their execution time is recorded. SpMV operation is performed 2k times to avoid any anomalies (execution time variations of low dimensional matrices) and used the average execution time of 2k iterations. With 2k iterations the execution time remained very much consistent. The dataset was labeled with block size associated with the minimum execution time recorded against each matrix. The average execution time was computed by taking the average time associated with multiple block sizes. The maximum execution time is the worst-case scenario.

Our dataset comprises 43 application domains, as listed in Table 2. The column name shows the largest matrix from each domain along with its dimensions (rows and columns). The number of non-zero entries in the largest matrix for each domain is also listed. These matrices are collected from multiple domains to diversify the dataset and thus avoid the situation of being biased towards specific domains or targeting specific structure of matrices.

Table 2. Application domains.

No.	Name	Non-Zero	Rows	Columns	Domains
1	1138_bus	4054	1138	1138	Power Network Problems
2	abb313	1557	313	176	Least Squares Problem
3	arc130	1037	130	130	Materials Problem
4	bcsstk13	83,883	2003	2003	Computational Fluid Dynamics Problem
5	bcsstk12	34,241	1473	1473	Duplicate Structural Problem
6	bcsstk31	1,181,416	35,588	35,588	Structural Problem
7	bcsstk32	2,014,701	44,609	44,609	Structural Problem
8	jagmesh4	9504	1440	1440	2D/3D Problem
9	saylr4	22,316	3564	3564	Computational fluid Dynamics Problem

Table 2. Cont.

No.	Name	Non-Zero	Rows	Columns	Domains
10	rbas480	17,088	480	480	Robotics Problem
11	Hardesty2	4,020,731	929,901	303,645	Computer Graphics/Vision Problem
12	nv2	37,475,646	1,453,908	1,453,908	Semiconductor Device Problem
13	hangGlider_3	149,532	15,561	15,561	Optimal Control Problem
14	vsp_model1_crew1_cr42_south31	379,952	45,101	45,101	Random Unweighted Graph
15	NLR	24,975,952	4,163,763	4,163,763	Undirected Graph
16	FX_March2010	301,899	1319	9498	Term/Document Graph
17	CurlCurl_0	113,343	11,083	11,083	Model Reduction Problem
18	preferential Attachment	999,970	100,000	100,000	Random Undirected Graph
19	kron_g500-logn19	43,562,265	524,288	524,288	Undirected Multigraph
20	ccc	4,194,298	1,048,56	1,048,576	Undirected Graph Sequence
21	dielFilterV2clx	25,309,272	607,232	607,232	Electromagnetics Problem
22	S20PI_n	2881	1182	1182	Eigenvalue/Model Reduction Problem
23	bp_1200	4726	822	822	Optimization Problem Sequence
24	lp_bnl2	14,996	2324	4486	Linear Programming Problem
25	onetone2	222,596	36,057	36,057	Frequency Domain Circuit Simulation Problem
26	nemeth01	725,054	9506	9506	Quantum Chemistry Problem Sequence
27	nemeth02	394,808	9506	9506	Theoretical/Quantum Chemistry Problem
28	young1c	4089	841	841	Acoustics Problem
29	shl_200	1726	663	663	Subsequent Optimization Problem
30	gemat12	33,044	4929	4929	Subsequent Power Network Problem
31	fs_760_2	5739	760	760	Subsequent 2D/3D Problem
32	impcol_a	572	207	207	Chemical Process Simulation Problem
33	beacxc	50,409	497	506	Economic Problem
34	gyro_k	1,021,159	17,361	17,361	Duplicate Model Reduction Problem
35	flowmeter0	67,391	9669	9669	Model Reduction Problem
36	Pd	13036	8081	8081	Counter Example Problem
37	GD00_c	1041	638	638	Directed Multigraph
38	thermal2	8,580,313	1,228,045	1,228,045	Thermal Problem
39	bibd_18_9	1,750,320	153	48,620	Combinatorial Problem
40	mawi_201512012	38,040,320	18,571,154	18,571,154	Undirected Weighted Graph
41	circuit_1	35,823	2624	2624	Circuit Simulation Problem
42	fs_541_1	4282	541	541	2D/3D Problem Sequence
43	Cities	1342	55	46	Weighted Bipartite Graph

3.2. Feature Extraction

After collection of the sparse matrices and performing SpMV computations with random block sizes, we extracted some of the most relevant features that affect the performance of SpMV computation.

These features of the sparse matrix are listed in Table 3, along with their computation complexity, feature description, and formulae. The feature set is divided into two subsets, that is, basic features, which have low computational complexity, and other features, which require a full scan of the matrix and have certain associated computational complexity. The feature column shows the name of the addressed feature, along with their description and formula. The computational complexity of these features is listed in the last column of Table 3.

Table 3. Sparse matrices features with definition and complexity.

Set	Features	Description	Formula	Complexity
Basics Features	M	Number of rows.	M	$\mathcal{O}(1)$
	N	Number of columns.	N	$\mathcal{O}(1)$
	M + N	row + column	M + N	$\mathcal{O}(1)$
	nnz	Number of non zeros.	Nnz	$\mathcal{O}(1)$
	Density	$\frac{\text{nnz}}{M \times N}$	$\frac{\text{nnz}}{M \times N}$	$\mathcal{O}(1)$
High Complexity Features	nnz_min	Minimum nnz.	$\text{Min}(\text{nnz}_1, \dots, \text{nnz}_n)$	$\mathcal{O}(M)$
	nnz_max	Maximum nnz	$\text{Max}(\text{nnz}_1, \dots, \text{nnz}_n)$	$\mathcal{O}(M)$
	nnz_avg	Average nnz	$1/N \sum_{i=1}^N \text{nnz}_i$	$\mathcal{O}(M)$

Table 3. Cont.

Set	Features	Description	Formula	Complexity
nnz_sd		Standard deviation of nonzero elements per row.	$\sqrt{1/N \sum_{i=1}^N (nnz_i - nnz_avg)^2}$	$\mathcal{O}(2M)$
Bw_Avg		Average column distance between the first and the last nonzero element each row.	$1/N \sum_{i=1}^N Bw_i$	$\mathcal{O}(M)$
Bw_min		Minimum column distance between the first and the last nonzero element of each row.	$\text{Min}(Bw_1, \dots, Bw_n)$	$\mathcal{O}(M)$
Bw_max		Maximum column distance between the first and the last nonzero element of each row.	$\text{Max}(Bw_1, \dots, Bw_n)$	$\mathcal{O}(M)$
Bw_sd		Standard Deviation of column distances between the first and the last nonzero element of each row.	$\sqrt{1/N \sum_{i=1}^N (Bw_i - Bw_avg)^2}$	$\mathcal{O}(2M)$
Clustering	Clustering		$\sum_{i=0}^N i = 0 \text{ Clus}_i$	$\mathcal{O}(nnz)$

3.3. Training and Testing Phase

Initially, all the sparse matrices are converted into Block CSR format, sparse matrix features are extracted, and SpMV computation is performed 2000 times with different block sizes, and the dataset is labeled with the optimal block size based on the execution time. We selected 80 percent of the dataset for training and the remaining 20% for testing, as can be seen in Figure 3.

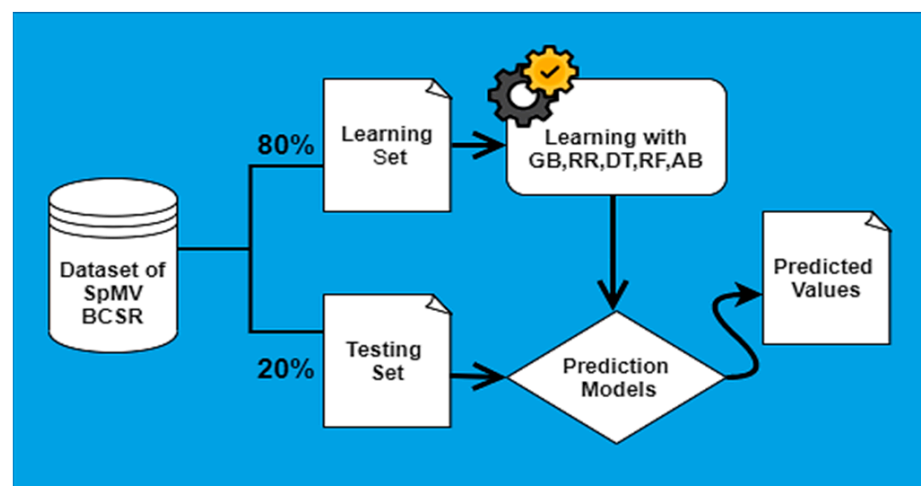


Figure 3. Training and testing process.

3.4. Model Evaluation Metrics

The model with the best prediction results is selected based on mean absolute error (MAE) and relative mean error (RME) evaluation metrics.

MAE

If y_i is the actual value of the i -th sample and \hat{y}_i is the predicted value, then the mean square error MAE can be estimated as

$$MAE(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| \quad (7)$$

RME

The relative mean error (RME) is one of the most commonly used evaluation metrics in machine learning and is the average of the percentage error of the forecast.

$$M = \frac{1}{n} \sum_{t=1}^n \left| \frac{A_t - F_t}{A_t} \right| \quad (8)$$

- n = the number of errors;
- Σ = symbol of summation;
- A_t is the actual value;
- F_t is measure of forecast value.

4. Results and Discussion

This section presents the results and discussion based on the methodology, implementation, and experiments shown in the previous section. Moreover, we have also presented the description of software specifications used while implementing the proposed solution. Execution time analysis of SpMV computation with multiple random block sizes along with average- and worst-case scenarios are presented in Section 4.2. Speedup achieved by using optimal block sizes and compared them with average- and worst-case scenarios. Detailed predictive analysis, performance gain, and limitations are presented in the subsequent sections.

4.1. Hardware and Software Specifications

Experiments are performed on Core i5 7th generation with 4 cores, 8 GT/s bus speed, 24 GB RAM, and 1TB storage. We used the following tools to perform SpMV computations and machine learning based predictive analysis. The software specifications are presented in Table 4.

Table 4. Software specifications.

Software	Tools/Library	Version
Operating System	Windows 10 pro	18,363.904
OpenMP	OpenMp	Openmp-5.1
Compiler	Visual Studio professional 2019	Visual studio 16.5
Python	Google Colab	Py 3.6.7.
Scikit-Learn	Sklearn	0.23.1

4.2. Execution Time Analysis of SpMV

We performed experiments on SpMV computations with multiple block sizes and calculated the minimum, maximum, and average execution times of SpMV, as shown in Equations (4)–(6), respectively.

Figure 4 shows the execution time comparison of the SpMV computation with various random multiple block sizes. The minimum, maximum, and average execution times of each matrix from the different application domains are shown in Figure 4. The dataset was sorted by nnz (minimum to maximum). Max time is a worst-case scenario in which the random block size is chosen with the maximum execution time. The average time is calculated as the aggregated execution time of the SpMV computation with multiple block sizes. The minimum execution time is the least execution time recorded against each matrix, and the associated block sizes are chosen as the target variable. The X-axis shows the different matrices from various application domains, and the Y-axis shows the execution time of SpMV computations on a logarithmic scale showing clear execution time difference between optimal and random selection of block sizes.

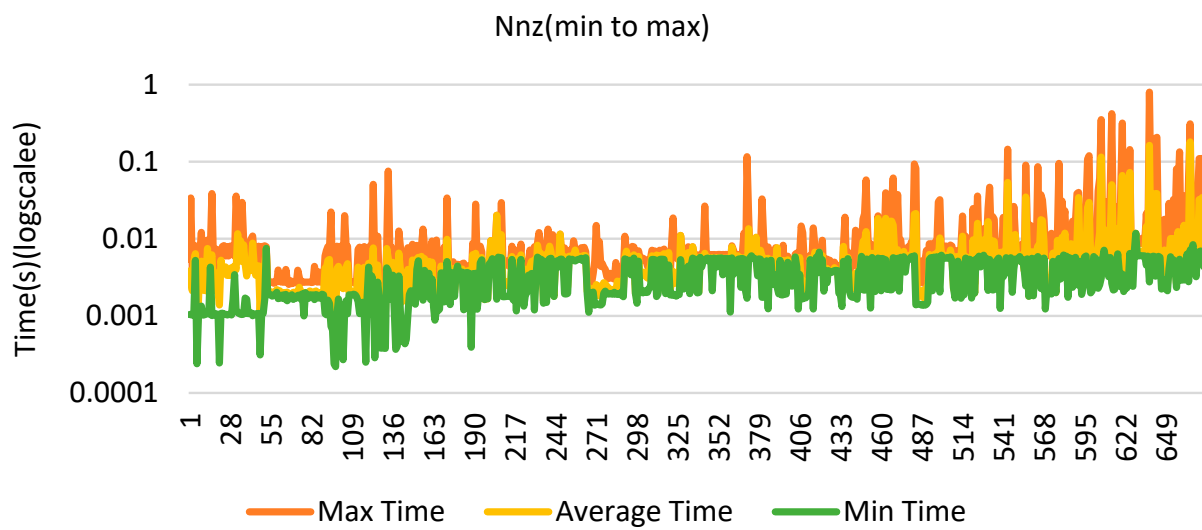


Figure 4. Execution time comparison of SpMV computation (dataset sorted by nnz -min to max).

Figure 5 shows the comparison of execution time of the SpMV computation on the row-sorted data set (minimum to maximum). The X-axis shows the different matrices from various application domains sorted by row (minimum to maximum), and the Y-axis shows the execution time of SpMV computations on a logarithmic scale.

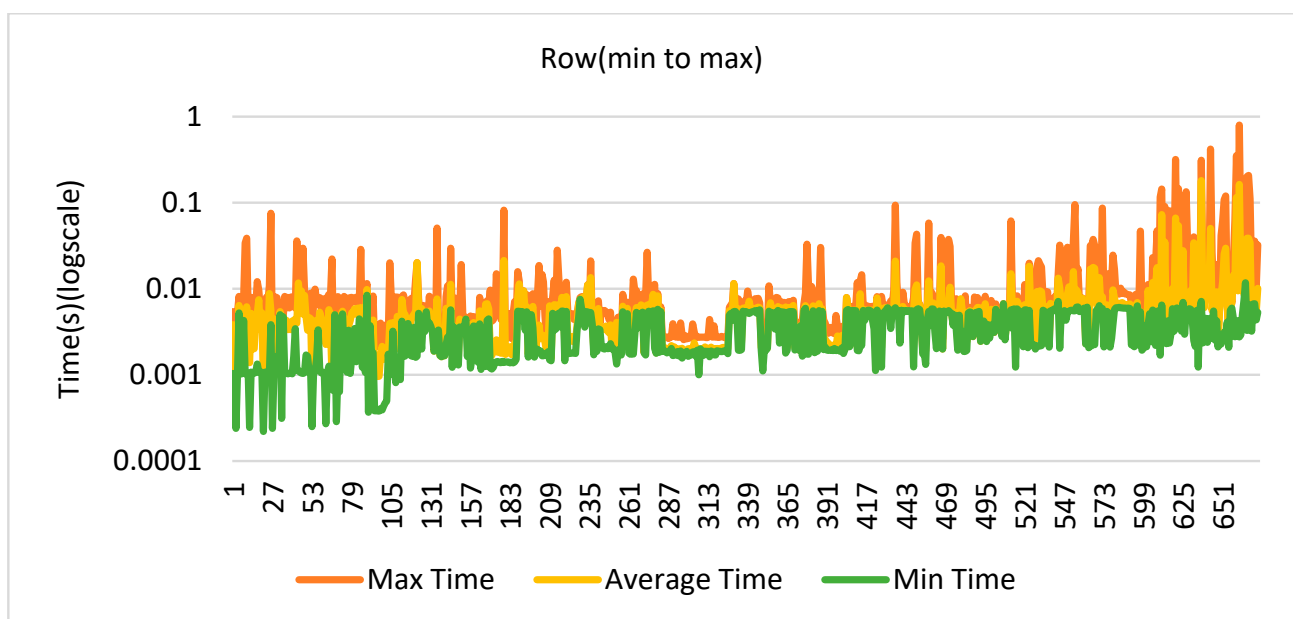


Figure 5. Execution time comparison of SpMV computation (dataset sorted by row-min to max).

The pre-processing or format conversion cost is highlighted in Figure 6, which shows execution time comparison with and without including the conversion time of CSR to BCSR format. The X-axis shows matrices in our dataset sorted on number of rows from minimum to maximum. Depending on the structure, CSR performs well for some matrices but overall BCSR (including conversion cost) shows better performance against matrices, included in our dataset.

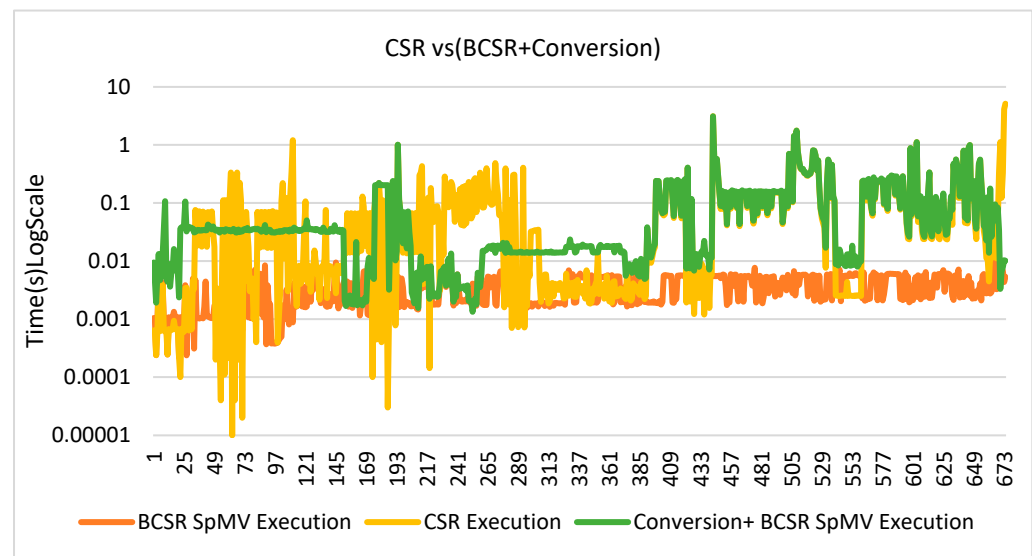


Figure 6. Execution time comparison of CSR and BCSR (with and without format conversion time).

4.3. Speedup

Figures 7 and 8 show the speed up of the entire dataset sorted on rows and the number of non-zeros, respectively. The speedup average and speedup maximum are calculated using Equations (9) and (10), respectively. The X-axis shows the number of matrices in our entire data set, and the Y-axis shows speedup against the average and worst-case scenarios.

$$\text{SpeedUp}_{\text{avg}} = \frac{t_{\text{avg}}}{t_{\text{min}}} \quad (9)$$

$$\text{SpeedUp}_{\text{max}} = \frac{t_{\text{max}}}{t_{\text{min}}} \quad (10)$$

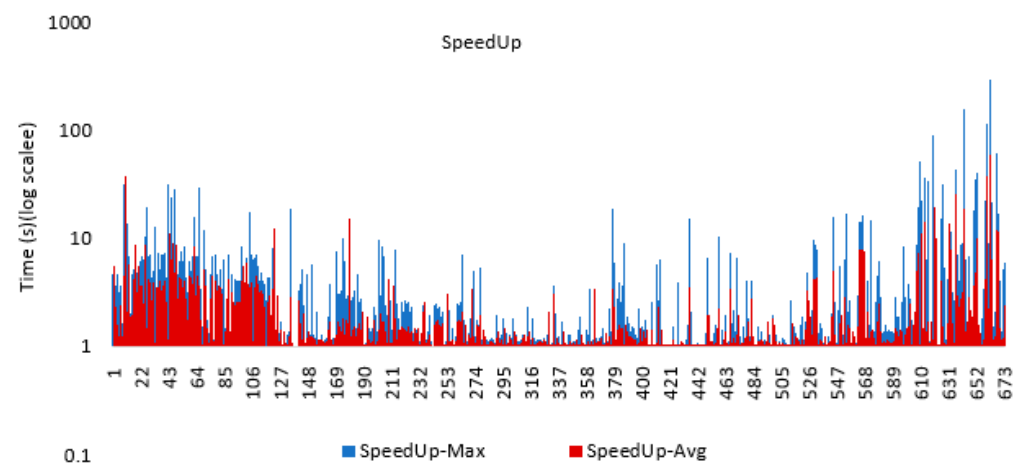


Figure 7. Speedup row sorted dataset (min to max).

The X-axis shows the application domains, while the Y-axis shows the execution time on a logarithmic scale. The execution time is calculated by taking the sum of all execution times of each domain recorded against multiple blocks. Figure 9a,c,e show the execution time comparison of different application domains in the same numerical order, as listed in Table 3. Similarly, Figure 9b,d,f show the speedup achieved using the entire dataset. Speedup against the worst-case scenario is represented in blue while speedup against the randomly chosen block size is shown as average speed up in red.

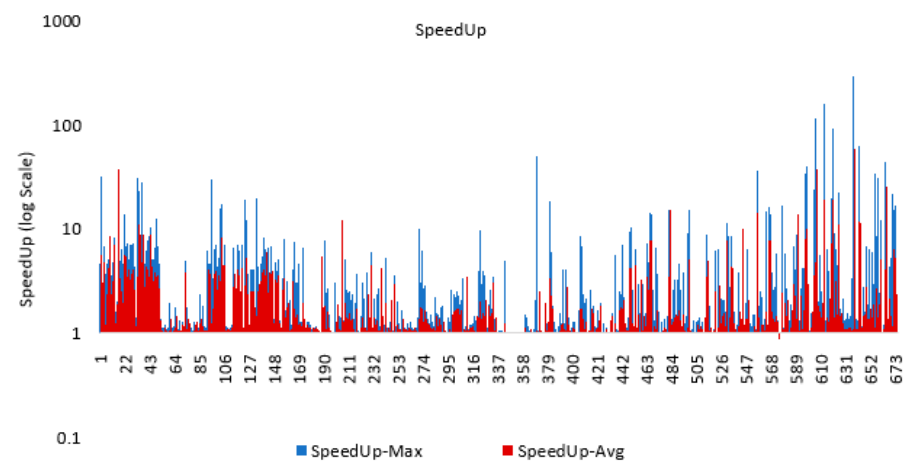


Figure 8. Speedup nnz sorted dataset (min to max).

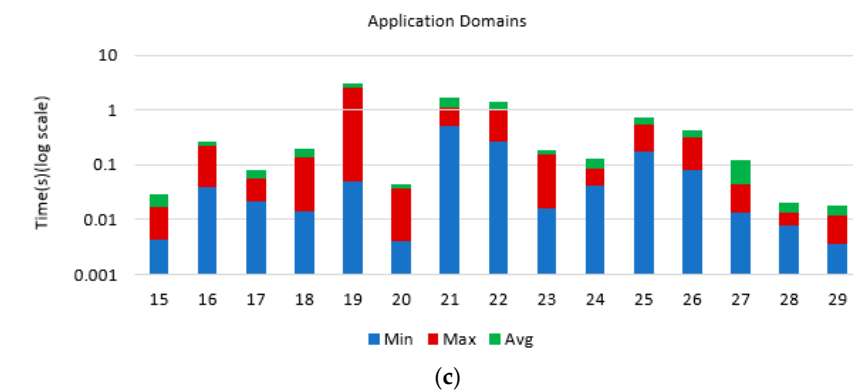
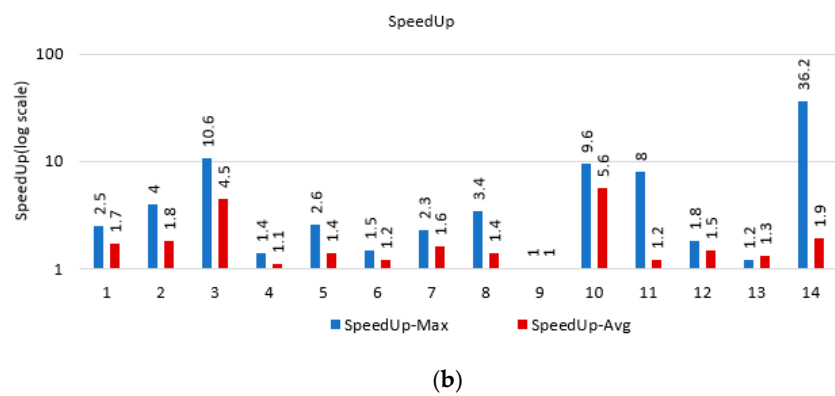
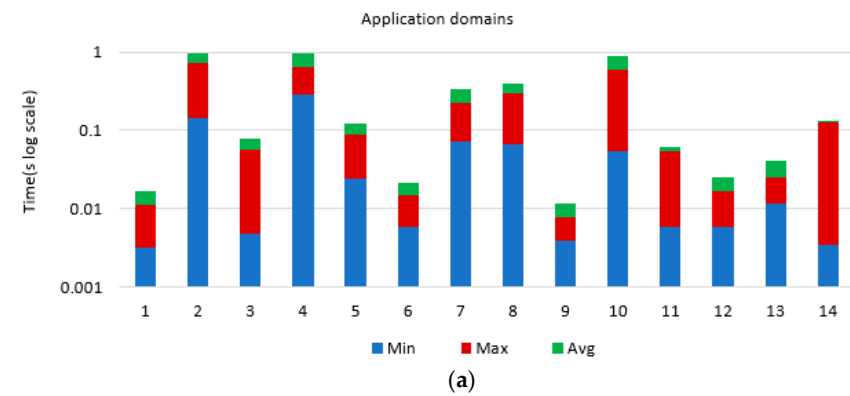


Figure 9. Cont.

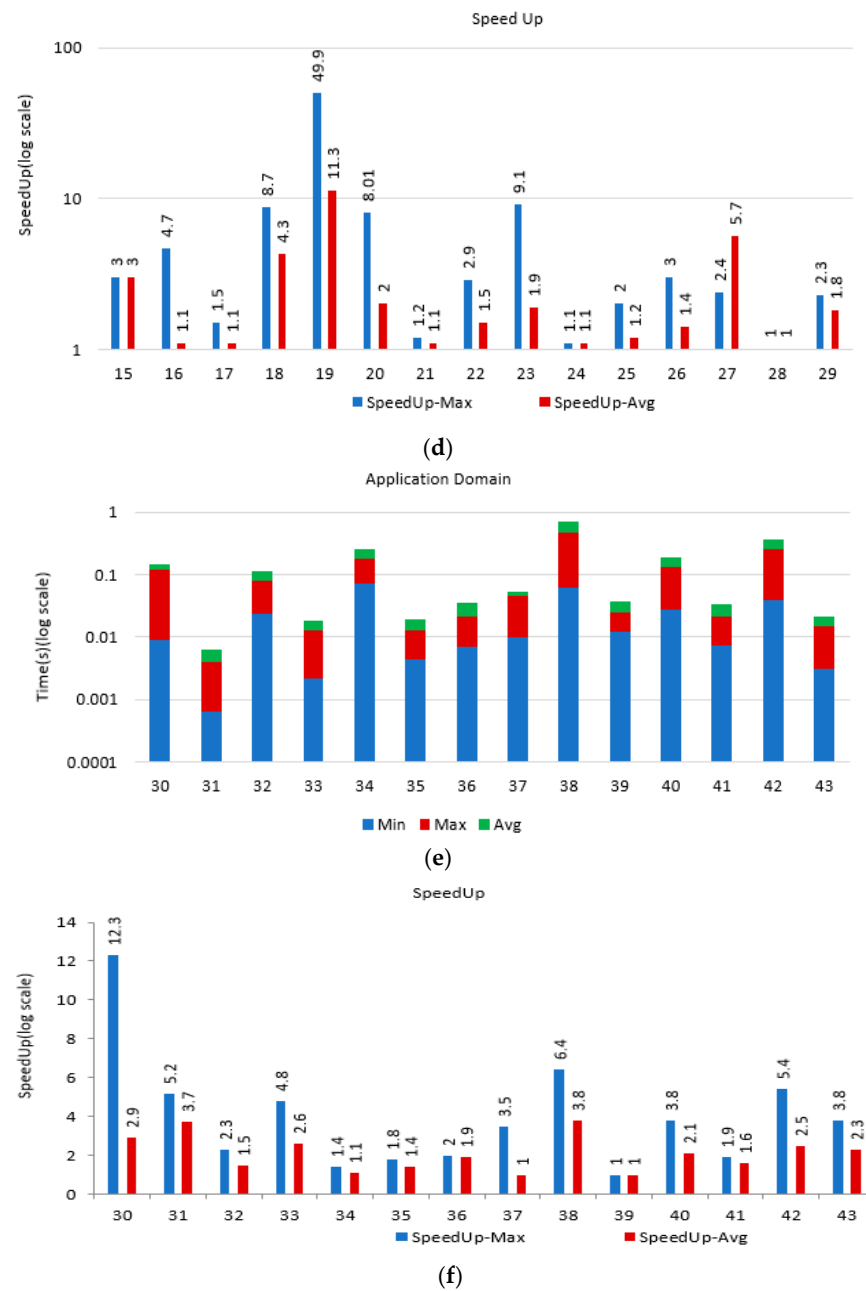


Figure 9. Domain wise execution time comparison (a–c) and speedup (d–f).

Figure 10 shows speedup of our proposed methodology against CSR (with and without) including format conversion time from CSR to BCSR). The speedup of BCSR execution (only) is represented in blue and red color represents the speedup of BCSR execution + format conversion time. Overall BCSR (including conversion time) showed speedup of 1.27 compared to CSR execution.

4.4. Predictive Analysis

We used five machine learning-based predictive models: decision tree (DT), random forest (RF), ridge regressor (RR), AdaBoost (AB), and gradient boosting (GB).

Our proposed method predicts the near-optimal block size based on the structure of the matrix, which is an unknown entity until runtime. Fourteen sparse matrix features are extracted and divided into three categories: full feature set (including all fourteen features), and basic and important feature sets. Basic features have the lowest computational complexity and do not require a full scan of the matrix. We calculated the feature score of

all machine learning models and labeled them as important features, as shown in Figure 11. Important features are selected using Scikit-Learn built-in feature importance metric and score is calculated against each machine learning algorithm used in this study. Common features with highest feature importance score are selected as important features.

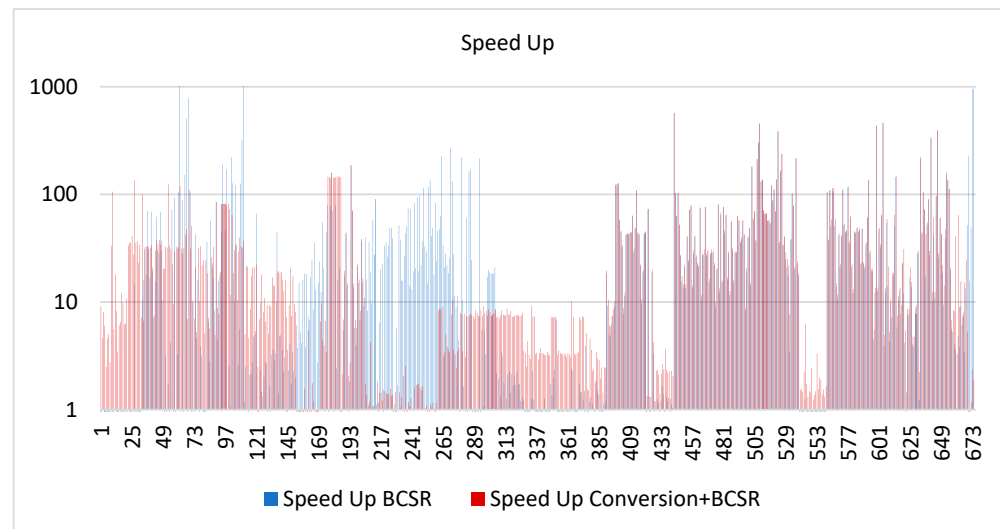


Figure 10. Speedup of BCSR (with and without format conversion time) against CSR.

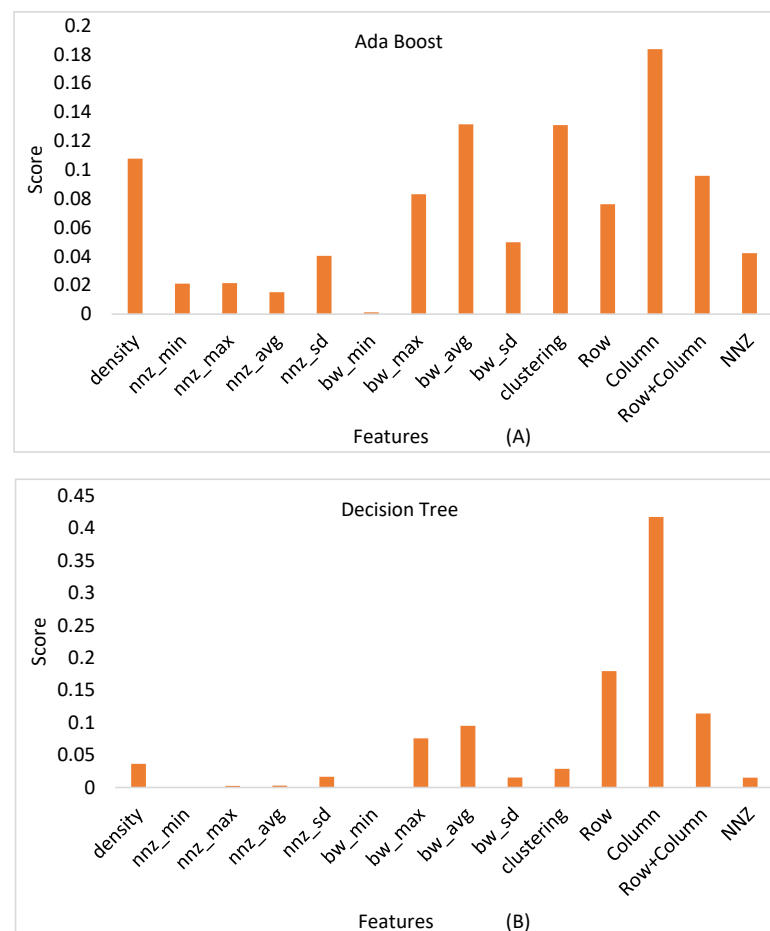


Figure 11. Cont.

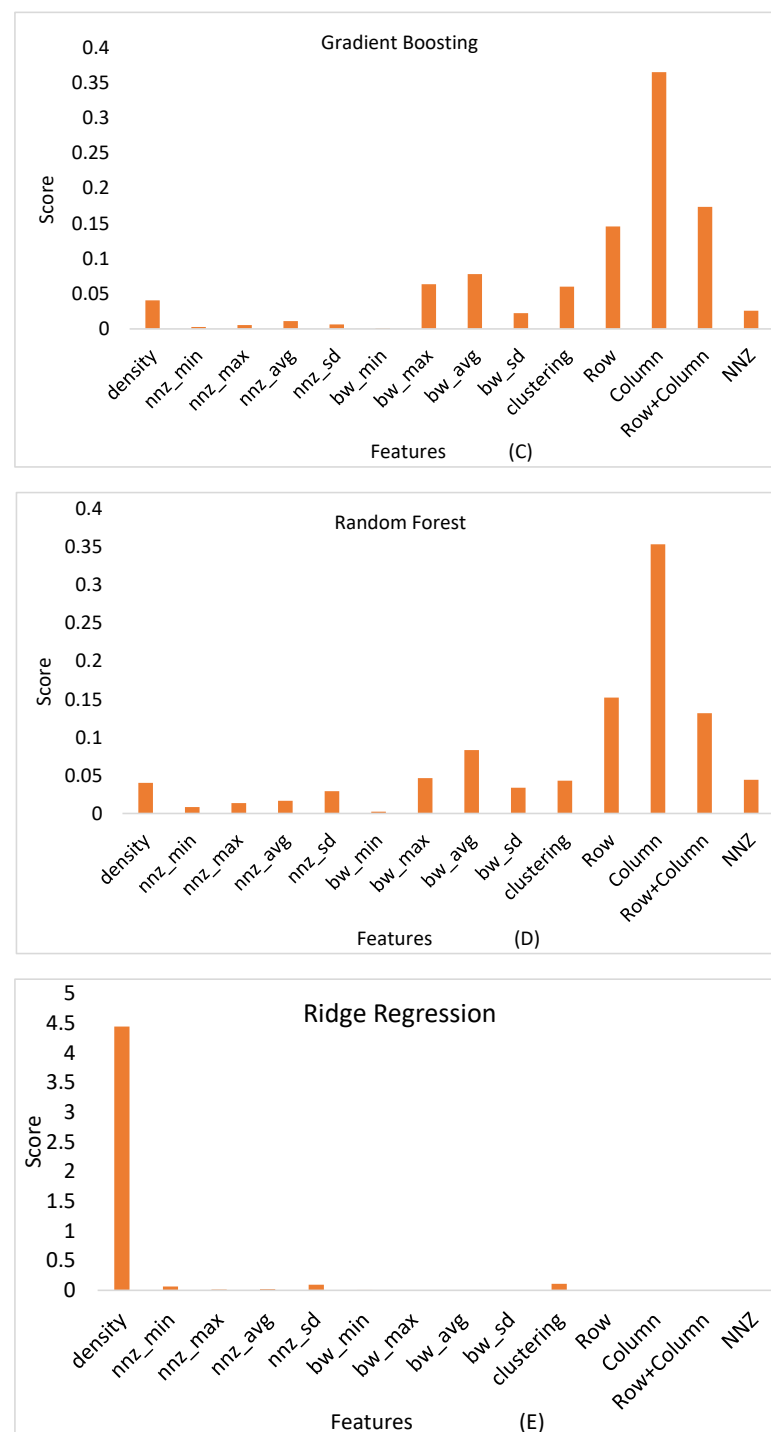


Figure 11. Feature's score (A) GB, (B) AB, (C) RF, (D) DT, and (E) RR.

We have drawn graphs for the actual versus predicted values (AAQAL). Actual values were obtained from the observed values in our data set. Figure 12A–E show the AAQAL versus actual observations marked as blue, while the AAQAL values are shown in red.

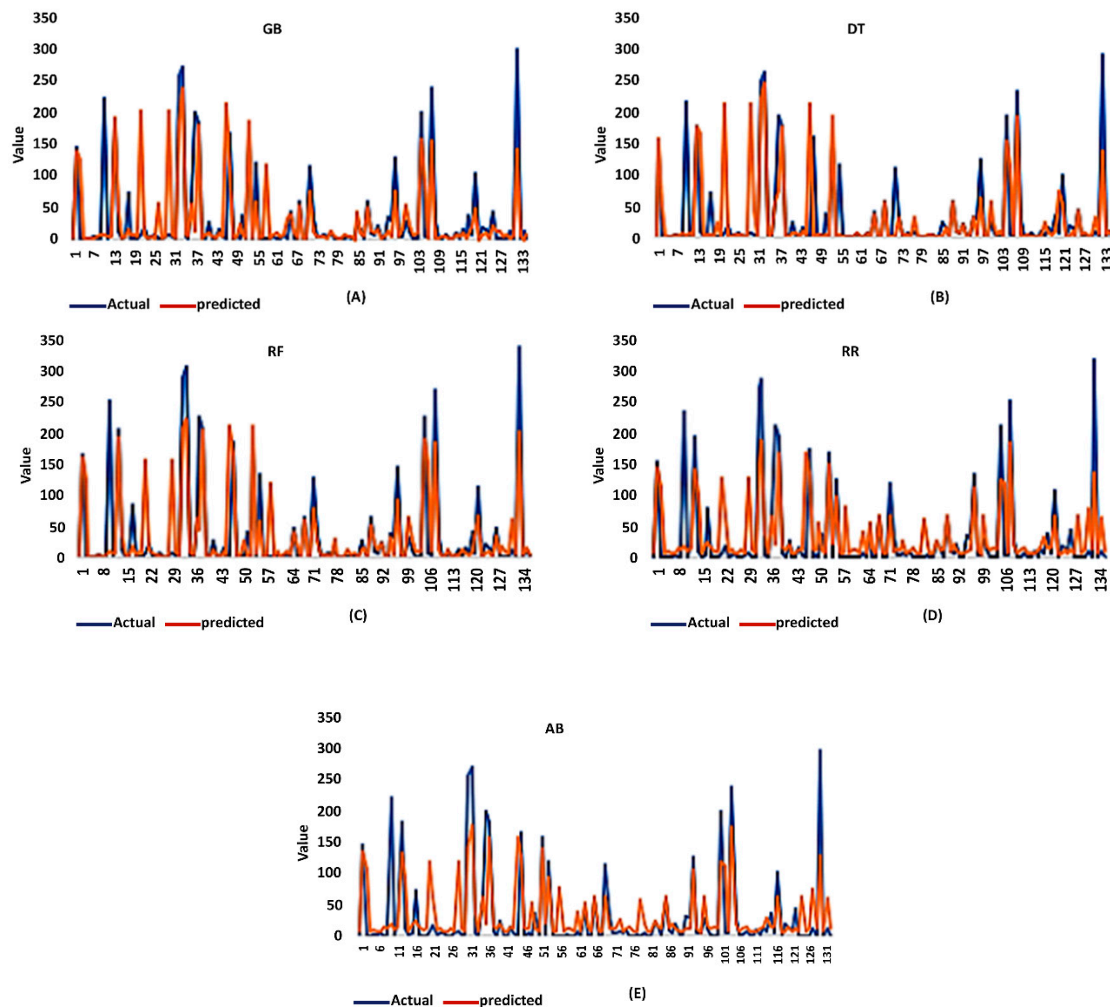


Figure 12. Actual vs. Predicted Values (A) GB, (B) DT, (C) RF, (D) RR, and (E) AB.

Figure 13A shows the mean absolute errors of different machine learning-based predictive models based on the full feature set. We calculated the mean absolute errors with basic features, important features, and all 14 sparse matrix features. Figure 13B shows the MAE comparison of different predictive models using basic features, which are calculated based on their feature importance graphs. Figure 13C shows the MEA comparison with important sparse matrix features. With the full feature set, DT outperformed the others with an error rate of 12.26. RF and GB both showed almost similar performance with error rates of 15.36 and 16.94, respectively. With the basic features set, RF performed better than all other models and showed an error rate of 14, while RR underperformed in all three cases. RF, DT, and GB show competitive performance with important features (based on the feature importance graph shown in Figure 11) and show error rates of 13.23, 14.14, and 14.98, respectively.

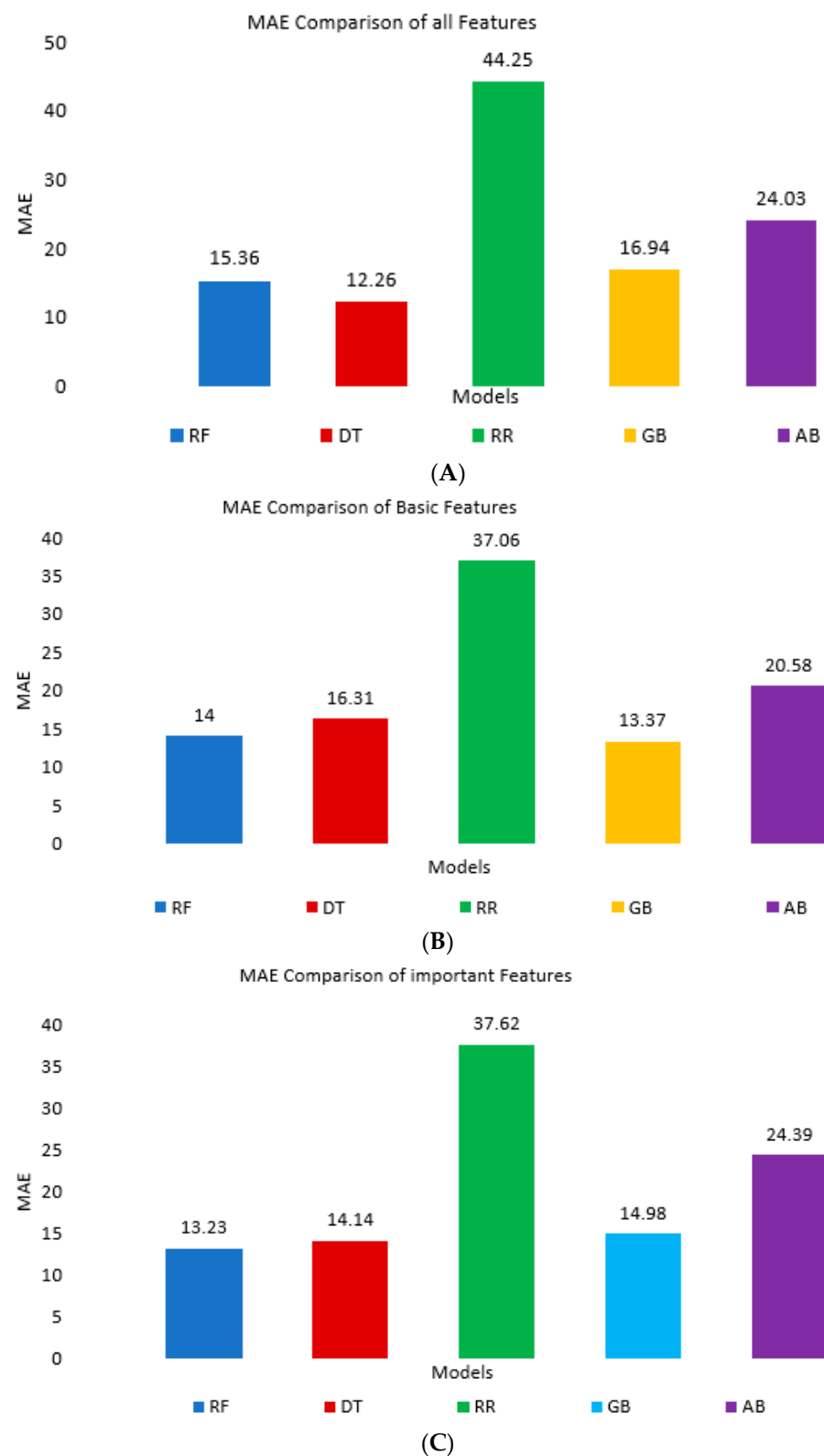


Figure 13. MAE Comparisons of (A) full, (B) basic, and (C) important sparse matrix features.

4.5. MAE and RME Comparison of Different Predictive Models

The RME comparison of the predictive models is shown in Figure 14. We calculated the relative mean errors with full features set, basic features, and important sparse matrix features. Figure 14A shows the RME of all 14 sparse matrix features, while Figures 12C and 14B show the basics and important features, respectively. With the full

and basic features set, DT, RF, and GB show almost similar performance, while RR underperforms in both cases.

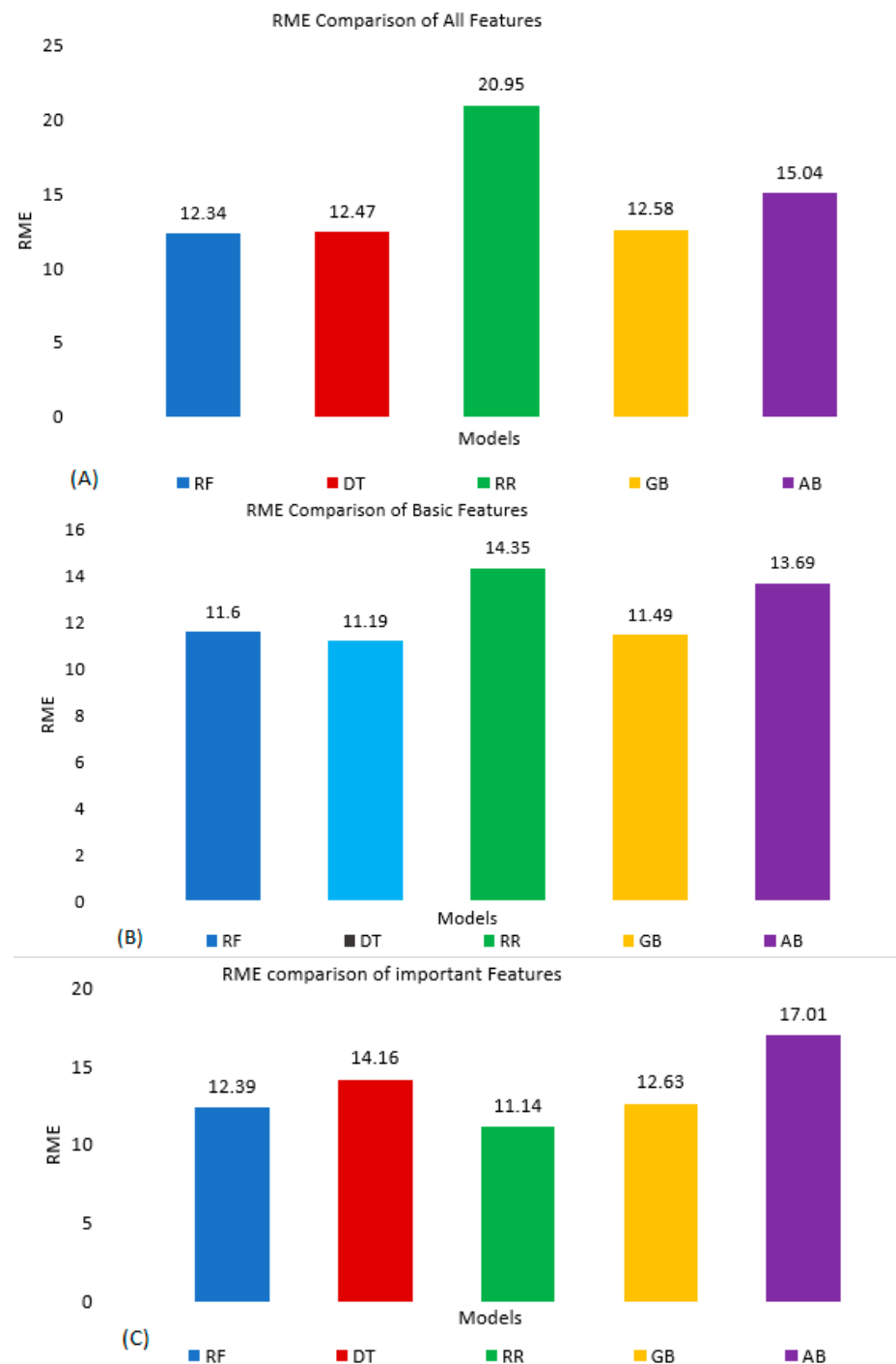


Figure 14. RME Comparisons of (A) all, (B) basic, and (C) important sparse matrix features.

4.6. Performance Gain

This section explains the performance gain and execution time of the proposed tool. Figure 15A shows the plot of the performance gain for all 14 sparse matrix features. The details of these sparse matrix features are listed in Table 3. Actual time is an idle case scenario and represents the least execution time associated with the target variable. The performance achieved is calculated against the best-case scenario, which is the actual time. The average represents the performance achieved when a random block size is selected

for each matrix and is calculated as the average execution time for different block sizes. The maximum is the worst-case scenario with the maximum execution time. Among all predictive models, DT performed better than the other models and achieved nearly 73% of the maximum possible performance gain.

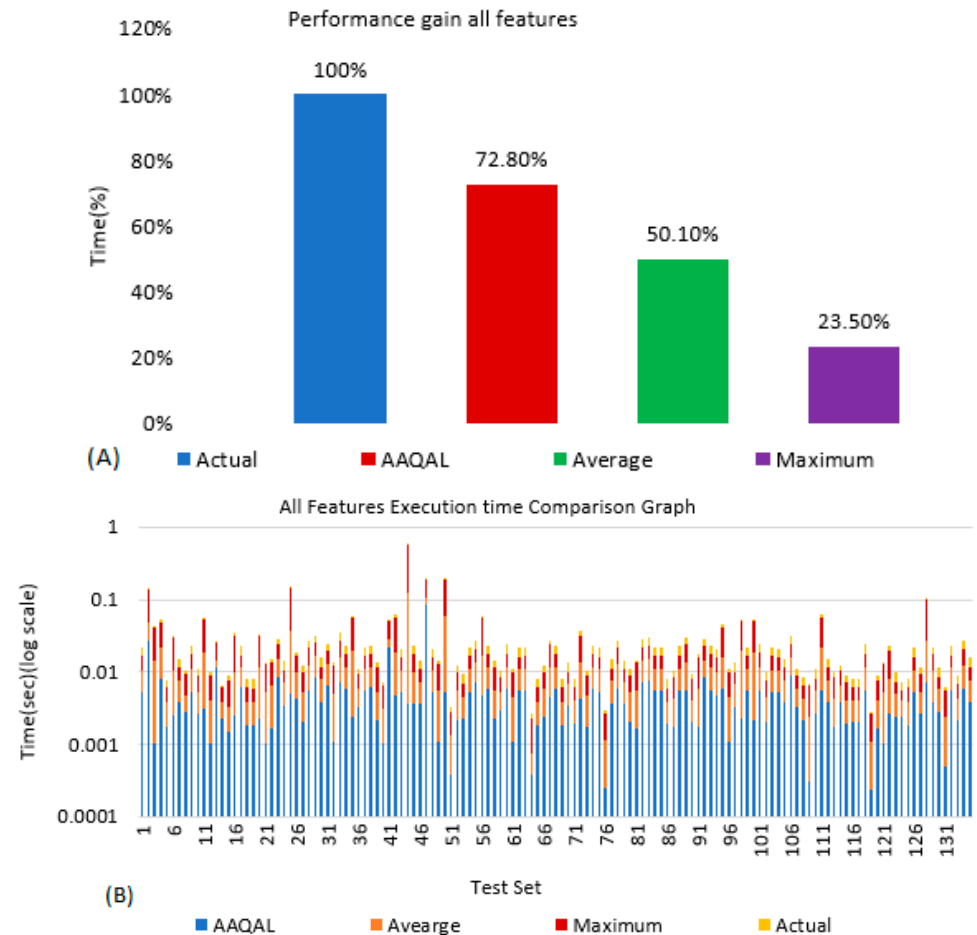


Figure 15. Performance gain (A) and execution time comparison (B) with all features (Test set).

Figure 15B shows the actual, maximum, average, and predicted time (AAQAL), which also includes cases in which our prediction is incorrect. For any incorrect prediction of block sizes, we used the execution time associated with the nearby block size. The Y-axis shows the execution time on the logarithmic scale, and the X-axis shows the sparse matrices in our dataset.

Figure 16A shows a graph of the performance gain with basic features. The details of the basic features are listed in Table 3. The basic features are those with low computational complexity. With basic features, AAQAL achieved almost 92% of the actual value, while random selection of block sizes could achieve only 51.88%. The performance difference between the random selection of block sizes and AAQAL is almost 40%.

Figure 16B shows the execution time comparison of the test set. The predicted time (AAQAL) was based on the basic feature set. The X-axis of the graph shows the execution time on a logarithmic scale, and the Y-axis shows different sparse matrices in our test set.

Figure 17A shows the plot of the performance gain with the set of important features. Important features are based on the feature importance graph and include nnz-max, rows and columns, rows, columns, clustering, and bw-max. Our proposed method achieved a performance of up to 93.47%. The performance difference between our proposed tool and the random selection of block sizes is almost 31.27%. The comparison of the execution time of our proposed tool based on an important feature set is shown in Figure 17B.

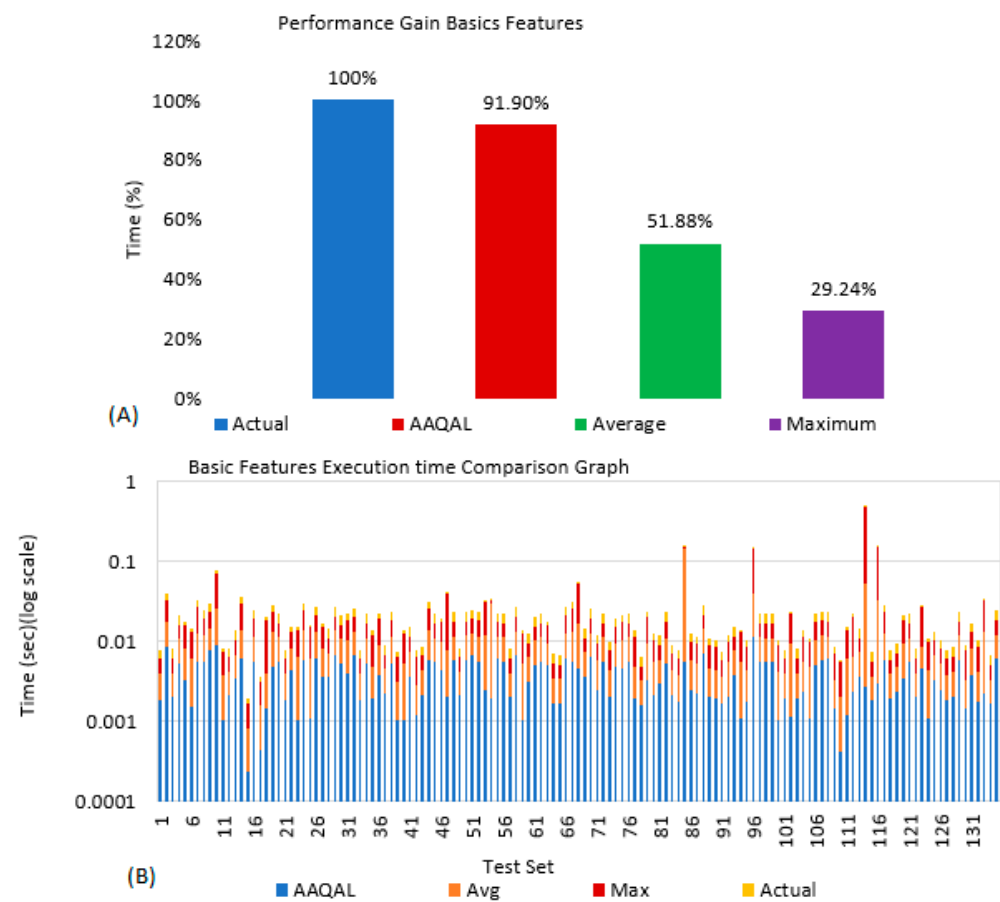


Figure 16. Performance gain with basic features (A) and execution time comparison (B) with basic features (test set).

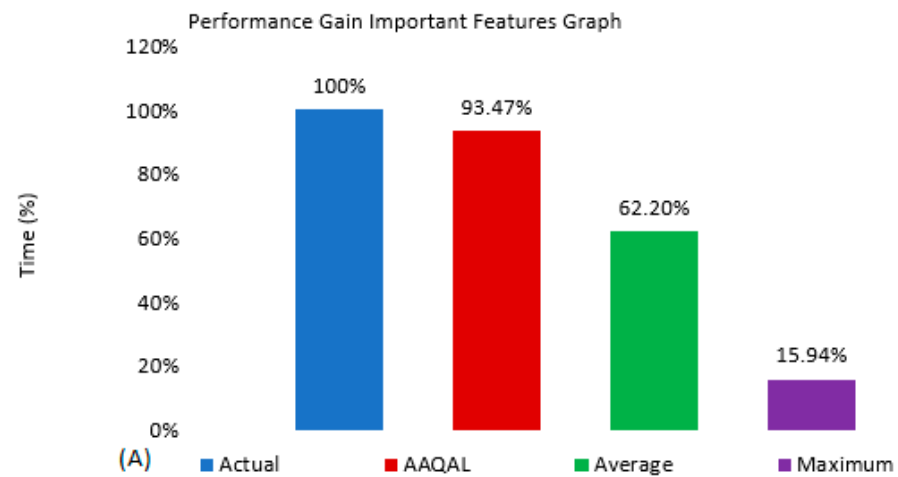


Figure 17. Cont.

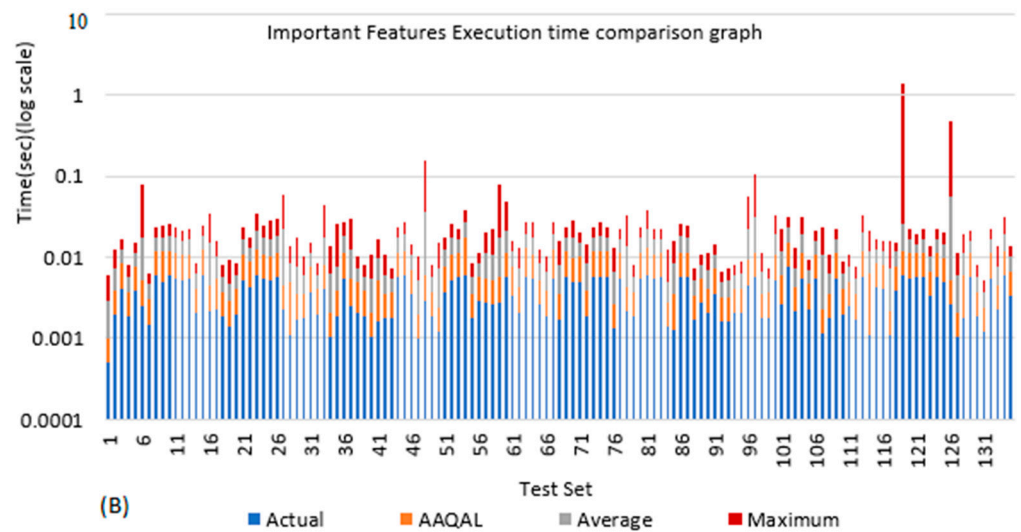


Figure 17. Performance gain random forest with important features (A) and execution time comparison (B) with important features (test Set).

4.7. Limitations

An efficient parallelization of SpMV requires optimal and balanced distribution of input matrix to processors with minimum inter-processor communication by exploiting the properties of sparse matrices and underlying machine architecture. As a result, different storage formats were proposed to efficiently utilized computation resources and to minimize overall communication and execution cost. Most of the storage schemes are customized to target the optimization of the sparse matrices with specific patterns. The overall work set of SpMV algorithm is reduced with blocking as these methods need to maintain single index per block. Selection of appropriate block size is not straight forward. Fixed size blocks require padding and variable size blocking requires additional data structure.

Manual process of trial-and-error block size is error prone and time consuming. Different sparse matrices have different structures which is an unknown entity before runtime, thus we used machine learning based optimization of SpMV kernel using BCSR. The cost associated with format conversion from CSR to BCSR increases with irregular patterns of non-zero elements which ultimately requires excessive padding. The variable blocking requires processing of additional data structures. The proposed methodology works with fixed size blocking and becomes more complex with variable block size and increasing number of cores and threads. We have experimented with, around 700 small to medium scale sparse matrices. The overall performance of the proposed methodology can be improved by increasing the data set size (sparse matrices of any dimension) and including more sparse features with low computation complexity. For the ease of implementation, we have considered only square sparse matrices and features are extracted from real world sparse matrices that require some computations to be performed to prepare dataset for training machine learning algorithms. Some of these features requires a full scan of sparse matrix and have computation complexity of $\Theta(M)$ and their standard deviation with complexity $\Theta(2M)$. There is a need to incorporate advance heuristics and performance models to accurately predict complex scenarios with variable block sizes that also adapt to modern multi-core architectures.

5. Conclusions

The multiplication of SpMV is ranked among the most significant and extensively utilized scientific kernels that arise in various scientific problems, such as computer graphics or computer vision, problems in robotics, 3D and 2D problems, acoustic problems, research in the surgical field, thermodynamics problems, medical care, and tissues. Several engineering, economic, social, and scientific projects require solutions to sparse linear equation

systems. Some common exemplary applications are thermodynamics problems, computer graphics, robotics, life sciences, research, smart phones, computer work, transportation, social media analytics, and autonomous vehicles. Therefore, SpMV is considered to be the most vital and time-taking kernel for iterative solutions of sparse linear equation systems. The performance of SpMV calculation is affected by a series of factors. Among these categories, some common factors are matrix properties, storage format, and operation of hardware and software platforms. Improvement in the application performance is always a challenge when dealing with multicore architectures because of the diversity and heterogeneity of these architectures. Over the years, numerous sparse matrix storage formats have been proposed and are highly dependent on the structure of the matrix. BCSR is considered to be the most promising one because it reduces the indexing structure to store sparse matrices.

In this paper, we proposed a machine learning-based performance optimization of SpMV on the underlying shared memory architecture. We exploited the structure of matrices by extracting relevant features and predicting the optimal block size for SpMV computation. To achieve this, we utilized nearly 700 real-world matrices associated with 43 application domains. We used five different base and ensemble machine learning methods, including gradient boosting, decision tree, random forest, ridge regressor, and AdaBoost. DT showed the lowest MAE, while RF showed the lowest RME among all feature sets. Our proposed tool, AAQAL, achieved an ideally attainable performance of almost 93.5%. The difference between the random selection of block sizes and our proposed tool is approximately 40% against the ideally achievable performance. The manual process on hit and trial basis for the optimal block size is time consuming and error prone.

In the future, we will enhance the proposed approach by incorporating different learning models, adding more features with low computation complexity and increasing the dataset size, both in terms of size and dimensionality. The proposed methodology requires some modifications to cater more complex scenarios, i.e., variable size blocking, decomposition method (to avoid padding), etc. Currently we are working on improving the proposed methodology to minimize format conversion and features extraction cost by incorporating features that requires low preprocessing. We are also planning to extend our proposed methodology to a distributed memory environment with hybrid MPI/OpenMP programming models.

Author Contributions: Funding acquisition, A.M.A.; Investigation, K.A.A.; Methodology, M.A.; Project administration, M.U.A.; Resources, N.A.S.; Supervision, S.U.; Visualization, A.A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, Saudi Arabia, under Grant No. (RG-11-611-43).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, Saudi Arabia, under Grant No. (RG-11-611-43). The authors therefore, gratefully acknowledge DSR technical and financial support. We would like to thanks also the all the helping bodies from Leads, GCWUS, and Grand Asian University for assisting us to carry out this research work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Xie, K.; Lee, C.-R.; Liu, F.-Y. Performance Optimization of SpMV on Spark. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 689–694.
2. García, C.G.; Meana-Llorián, D.; G-Bustelo, B.C.P.; Lovelle, J.M.C.; Garcia-Fernandez, N. Midgar: Detection of people through computer vision in the Internet of Things scenarios to improve the security in Smart Cities, Smart Towns, and Smart Homes. *Future Gener. Comput. Syst.* **2017**, *76*, 301–313. [\[CrossRef\]](#)
3. Rahman, A.; Jin, J.; Cricenti, A.; Rahman, A.; Palaniswami, M.; Luo, T. Cloud-enhanced robotic system for smart city crowd control. *J. Sens. Actuator Netw.* **2016**, *5*, 20. [\[CrossRef\]](#)
4. Aliaga, D.G. 3D design and modeling of smart cities from a computer graphics perspective. *Int. Sch. Res. Not.* **2012**, *2012*, 728913. [\[CrossRef\]](#)
5. Zappatore, M.; Longo, A.; Bochicchio, M.A. Crowd-sensing our smart cities: A platform for noise monitoring and acoustic urban planning. *J. Commun. Softw. Syst.* **2017**, *13*, 53–67. [\[CrossRef\]](#)
6. Bello, J.P.; Mydlarz, C.; Salamon, J. Sound analysis in smart cities. In *Computational Analysis of Sound Scenes and Events*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 373–397.
7. NVIDIA. [Online]. 2016. Available online: <https://developer.nvidia.com/cuspars> (accessed on 4 May 2022).
8. Mehmood, R.; Crowcroft, J. *Parallel Iterative Solution Method for Large Sparse Linear Equation Systems*; University of Cambridge, Computer Laboratory: Cambridge, UK, 2005.
9. Asanovic, K.; Bodik, R.; Demmel, J.; Keaveny, T.; Keutzer, K.; Kubiawicz, J.; Morgan, N.; Patterson, D.; Sen, K.; Wawrzynek, J. A view of the parallel computing landscape. *Commun. ACM* **2009**, *52*, 56–67. [\[CrossRef\]](#)
10. Sun, H.; Gainaru, A.; Shantharam, M.; Raghavan, P. Selective Protection for Sparse Iterative Solvers to Reduce the Resilience Overhead. In Proceedings of the 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Porto, Portugal, 9–11 September 2020; pp. 141–148.
11. Zheng, C.; Gu, S.; Gu, T.-X.; Yang, B.; Liu, X.-P. BiELL: A bisection ELLPACK-based storage format for optimizing SpMV on GPUs. *J. Parallel Distrib. Comput.* **2014**, *74*, 2639–2647. [\[CrossRef\]](#)
12. Kourtis, K.; Goumas, G.; Koziris, N. Optimizing sparse matrix-vector multiplication using index and value compression. In Proceedings of the 5th Conference on Computing Frontiers, Ischia, Italy, 5–7 May 2008; pp. 87–96.
13. Grossman, M.; Thiele, C.; Araya-Polo, M.; Frank, F.; Alpak, F.O.; Sarkar, V. A survey of sparse matrix-vector multiplication performance on large matrices. *arXiv* **2016**, arXiv:1608.00636.
14. Pinar, A.; Heath, M.T. Improving performance of sparse matrix-vector multiplication. In Proceedings of the SC'99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, Portland, OR, USA, 14–19 November 1999; p. 30.
15. Kourtis, K.; Goumas, G.; Koziris, N. Improving the performance of multithreaded sparse matrix-vector multiplication using index and value compression. In Proceedings of the 2008 37th International Conference on Parallel Processing, Portland, OR, USA, 9–12 September 2008; pp. 511–519.
16. HSA: Heterogeneous System Architecture. Available online: <http://hsafoundation.com/> (accessed on 12 April 2022).
17. Siddiqui, N.; Yousaf, F.; Murtaza, F.; Ehatisham-ul-Haq, M.; Ashraf, M.U.; Alghamdi, A.M.; Alfakeeh, A.S. A highly nonlinear substitution-box (S-box) design using action of modular group on a projective line over a finite field. *PLoS ONE* **2020**, *15*, e0241890. [\[CrossRef\]](#)
18. Tariq, S.; Ahmad, N.; Ashraf, M.U.; Alghamdi, A.M.; Alfakeeh, A.S. Measuring the Impact of Scope Changes on Project Plan Using EVM. *IEEE Access* **2020**, *8*, 154589–154613. [\[CrossRef\]](#)
19. Manzoor, A.; Ahmad, W.; Ehatisham-ul-Haq, M.; Hannan, A.; Khan, M.A.; Ashraf, M.U.; Alghamdi, A.M.; Alfakeeh, A.S. Inferring Emotion Tags from Object Images Using Convolutional Neural Network. *Appl. Sci.* **2020**, *10*, 5333. [\[CrossRef\]](#)
20. Shinan, K.; Alsubhi, K.; Alzahrani, A.; Ashraf, M. Machine Learning-Based Botnet Detection in Software-Defined Network: A Systematic Review. *Symmetry* **2021**, *13*, 866. [\[CrossRef\]](#)
21. Im, E.-J.; Yelick, K.A. Optimizing Sparse Matrix Vector Multiplication on SMP. In Proceedings of the PPSC, San Antonio, TX, USA, 22–24 March 1999.
22. Im, E.-J.; Yelick, K. Optimizing sparse matrix computations for register reuse in SPARSITY. In Proceedings of the International Conference on Computational Science, San Francisco, CA, USA, 28–30 May 2001; pp. 127–136.
23. Willcock, J.; Lumsdaine, A. Accelerating sparse matrix computations via data compression. In Proceedings of the 20th annual International Conference on Supercomputing, Cairns, Australia, 28 June–1 July 2006; pp. 307–316.
24. Razzaq, J.; Berrendorf, R.; Hack, S.; Weierstall, M.; Mannuss, F. Fixed and variable sized block techniques for sparse matrix vector multiplication with general matrix structures. In Proceedings of the Tenth International Conference on Advanced Engineering Computing and Applications in Sciences, Venice, Italy, 9–13 October 2016.
25. Kannan, R. Efficient sparse matrix multiple-vector multiplication using a bitmapped format. In Proceedings of the 20th Annual International Conference on High Performance Computing, Bengaluru, India, 18–21 December 2013; pp. 286–294.
26. Yan, S.; Li, C.; Zhang, Y.; Zhou, H. yaSpMV: Yet another SpMV framework on GPUs. *ACM Sigplan Not.* **2014**, *49*, 107–118. [\[CrossRef\]](#)
27. Vuduc, R.W.; Moon, H.-J. Fast sparse matrix-vector multiplication by exploiting variable block structure. In Proceedings of the International Conference on High Performance Computing and Communications, Munich, Germany, 13–15 September 2006; pp. 807–816.

28. Karakasis, V.; Goumas, G.; Koziris, N. Performance models for blocked sparse matrix-vector multiplication kernels. In Proceedings of the 2009 International Conference on Parallel Processing, Vienna, Austria, 22–25 September 2009; pp. 356–364.
29. Karakasis, V.; Gkountouvas, T.; Kourtis, K.; Goumas, G.; Koziris, N. An extended compression format for the optimization of sparse matrix-vector multiplication. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *24*, 1930–1940. [\[CrossRef\]](#)
30. Buluç, A.; Fineman, J.T.; Frigo, M.; Gilbert, J.R.; Leiserson, C.E. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures, Calgary, AB, Canada, 11–13 August 2009; pp. 233–244.
31. Martone, M.; Filippone, S.; Tucci, S.; Gepner, P.; Paprzycki, M. Use of hybrid recursive csr/coo data structures in sparse matrix-vector multiplication. In Proceedings of the International Multiconference on Computer Science and Information Technology, Wisla, Poland, 18–20 October 2010; pp. 327–335.
32. Belgin, M.; Back, G.; Ribbens, C.J. Pattern-based sparse matrix representation for memory-efficient SMVM kernels. In Proceedings of the 23rd International Conference on Supercomputing, Yorktown Heights, NY, USA, 8–12 June 2009; pp. 100–109.
33. Hannan, A.; Hussain, F.; Ali, N.; Ehatisham-Ul-Haq, M.; Ashraf, M.U.; Alghamdi, A.M.; Alfakeeh, A.S. A decentralized hybrid computing consumer authentication framework for a reliable drone delivery as a service. *PLoS ONE* **2021**, *16*, e0250737. [\[CrossRef\]](#)
34. Fayyaz, S.; Sattar, M.K.; Waseem, M.; Ashraf, M.U.; Ahmad, A.; Hussain, H.A.; Alsubhi, K. Solution of combined economic emission dispatch problem using improved and chaotic population-based polar bear optimization algorithm. *IEEE Access* **2021**, *9*, 56152–56167. [\[CrossRef\]](#)
35. Hirra, I.; Ahmad, M.; Hussain, A.; Ashraf, M.U.; Saeed, I.A.; Qadri, S.F.; Alghamdi, A.M.; Alfakeeh, A.S. Breast Cancer Classification From Histopathological Images Using Patch-Based Deep Learning Modeling. *IEEE Access* **2021**, *9*, 24273–24287. [\[CrossRef\]](#)
36. Usman, S.; Mehmood, R.; Katib, I.; Albeshri, A.; Altowaijri, S.M. ZAKI: A smart method and tool for automatic performance optimization of parallel SpMV computations on distributed memory machines. *Mob. Netw. Appl.* **2019**, 1–20. [\[CrossRef\]](#)
37. Xiao, G.; Li, K.; Chen, Y.; He, W.; Zomaya, A.Y.; Li, T. CASpMV: A customized and accelerative SPMV framework for the sunway TaihuLight. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *32*, 131–146. [\[CrossRef\]](#)
38. Anzt, H.; Tsai, Y.M.; Abdelfattah, A.; Cojean, T.; Dongarra, J. Evaluating the Performance of NVIDIA's A100 Ampere GPU for Sparse and Batched Computations. In Proceedings of the 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), Atlanta, GA, USA, 12 November 2020; pp. 26–38.
39. Usman, S.; Mehmood, R.; Katib, I.; Albeshri, A. ZAKI+: A machine learning based process mapping tool for SpMV computations on distributed memory architectures. *IEEE Access* **2019**, *7*, 81279–81296. [\[CrossRef\]](#)
40. Davis, T.A.; Hu, Y. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw. (TOMS)* **2011**, *38*, 1–25. [\[CrossRef\]](#)
41. Ashraf, M.U.; Eassa, F.A.; Albeshri, A.A.; Algarni, A. Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems. *IEEE Access* **2018**, *6*, 23095–23107. [\[CrossRef\]](#)
42. Alsubhi, K.; Alsolami, F.; Algarni, A.; Albassam, E.; Khemakhem, M.; Eassa, F.; Jambi, K.; Ashraf, M.U. A Tool for Translating sequential source code to parallel code written in C++ and OpenACC. In Proceedings of the 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 3–7 November 2019; pp. 1–8.
43. Ashraf, M.U.; Eassa, F.A.; Osterweil, L.J.; Albeshri, A.A.; Algarni, A.; Ilyas, I. AAP4All: An Adaptive Auto Parallelization of Serial Code for HPC Systems. *Intell. Autom. Soft Comput.* **2021**, *29*, 615–639. [\[CrossRef\]](#)