BestSF: A Sparse Meta-Format for Optimizing SpMV on GPU

AKREM BENATIA, WEIXING JI, YIZHUO WANG, and FENG SHI,

Beijing Institute of Technology, China

The Sparse Matrix-Vector Multiplication (SpMV) kernel dominates the computing cost in numerous scientific applications. Many implementations based on different sparse formats were proposed to improve this kernel on the recent GPU architectures. However, it has been widely observed that there is no “best-for-all” sparse format for the SpMV kernel on GPU. Indeed, serious performance degradation of an order of magnitude can be observed without a careful selection of the sparse format to use. To address this problem, we propose in this article BestSF (Best Sparse Format), a new learning-based sparse meta-format that automatically selects the most appropriate sparse format for a given input matrix. To do so, BestSF relies on a cost-sensitive classification system trained using Weighted Support Vector Machines (WSVMs) to predict the best sparse format for each input sparse matrix. Our experimental results on two different NVIDIA GPU architectures using a large number of real-world sparse matrices show that BestSF achieved a noticeable overall performance improvement over using a single sparse format. While BestSF is trained to select the best sparse format in terms of performance (GFLOPS), our further experimental investigations revealed that using BestSF also led, in most of the test cases, to the best energy efficiency (MFLOPS/W). To prove its practical effectiveness, we also evaluate the performance and energy efficiency improvement achieved when using BestSF as a building block in a GPU-based Preconditioned Conjugate Gradient (PCG) iterative solver.

CCS Concepts: • **Computing methodologies** → **Parallel algorithms**; • **Mathematics of computing** → Solvers;

Additional Key Words and Phrases: Sparse matrix-vector multiplication (SpMV), GPU computing, performance modeling, energy efficiency, iterative solvers

ACM Reference format:

Akrem Benatia, Weixing Ji, Yizhuo Wang, and Feng Shi. 2018. BestSF: A Sparse Meta-Format for Optimizing SpMV on GPU. *ACM Trans. Archit. Code Optim.* 15, 3, Article 29 (August 2018), 27 pages.

<https://doi.org/10.1145/3226228>

This article is an extension of the conference paper “Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU” that appeared in the proceedings of the 45th International Conference on Parallel Processing (ICPP 2016) [8]. We extended this work by (1) using pairwise weighted classification instead of the standard multiclass classification, (2) incorporating the BCSR and DIA formats in the selection system and extending our set of sparsity features, (3) studying the impact of using BestSF on the overall energy efficiency of the SpMV kernel on GPU, (4) discussing the overhead of the online selection of the best performing sparse format and proving the practical effectiveness of our work by evaluating the gain of using BestSF as a building block inside a GPU-based PCG iterative solver.

Part of the code used in our work is available at <https://github.com/sparse-bit/bestsf>.

This work is supported by the National Key R&D Program of China under Grant No. 2017YFB0202500 and the National Science Foundation of China (Grants No. 61300010 and No. 61300011).

Authors’ address: A. Benatia, W. Ji, Y. Wang, and F. Shi, Beijing Institute of Technology, 5 South Zhongguancun Street, Haidian District, Beijing, 100081, China; emails: akrem.benatia@yahoo.com, {jwx, frankwyz, bitsf}@bit.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

1544-3566/2018/08-ART29 \$15.00

<https://doi.org/10.1145/3226228>

1 INTRODUCTION

The Sparse Matrix-Vector Multiplication (SpMV) kernel is a non-trivial sparse Basic Linear Algebra Subprogram (BLAS) operation. It dominates the computing cost in many iterative methods for solving large-scale linear systems, eigenvalue problems, Krylov subspace methods, and other problems. Optimizing the SpMV kernel on modern hardware architectures is extremely beneficial for a wide range of application domains like large simulation systems, medical imaging, information retrieval, economic and climate change modeling, and many others [6, 21].

In the past few years, researchers have investigated many ways to improve the performance of the SpMV kernel on the recent hardware architectures such as many-core Graphics Processing Units (GPUs). Many sparse formats were proposed in the literature (COO, CSR, DIA, ELL, HYB, BCSR, etc.) [6, 14, 21, 59], and also different auto-tuning techniques that optimize the performance by tuning different parameters according to the sparsity structure of the input matrix [14, 23]. It has been widely observed that the GPU performance of the SpMV kernel is very sensitive to the used sparse matrix format. As shown in many studies [12, 19, 21, 33], using an “improper” sparse format can cause serious performance degradation of an order of magnitude or more. Thus, researchers and application designers usually face a challenging question: which sparse format should be used to minimize the SpMV kernel execution time on GPU?

The most widely adopted solution to such sparse format selection problem is to use the sparse format that offers the best average performance on a representative set of sparse matrices. To use the same terminology as in [56], this approach is known as the “winner-takes-all” approach. Its main drawback is that it ignores many sparse formats that are not competitive on average but offer very good performance on particular sparse matrices. The ideal solution, on the other hand, would be to use the sparse format with the best performance for each input sparse matrix. Unfortunately, given the rising complexity of the GPU architecture and the wide range of different sparsity patterns, it is extremely challenging to precisely determine which sparse format is the best for a given input matrix without running the SpMV kernel itself. Nevertheless, we can rely on machine learning techniques to build a classification system that predicts the best sparse format for a given input matrix. Specifically, in this work, we use cost-sensitive classification models trained using Weighted Support Vector Machines (WSVMs) [60] to predict the best sparse format for a given sparse matrix based on some easy to compute sparsity features. The resulting system can be seen as a new sparse meta-format that we call BestSF (Best Sparse Format).

In this article, we consider the sparse formats COO, CSR, BCSR, ELL, DIA, and HYB. These sparse formats are the basis for a lot of other simple and hybrid formats [6, 21]. We have considered some very simple sparsity features that affect the GPU performance of the SpMV kernel to characterize the sparse matrices. A dataset of 1,000 sparse matrices from the SuiteSparse matrix collection (formerly the University of Florida Sparse Matrix Collection) [20] was used in the learning and the testing process. Our experimental results on two different NVIDIA GPU architectures, Maxwell and Pascal, show that BestSF achieved more than 97% of the best performance possible with a perfect selection. We also studied the impact of selecting the best sparse format in terms of performance (GFLOPS) on the overall energy efficiency (MFLOPS/W) of the SpMV kernel on GPU. Our experimental results revealed that BestSF, despite being trained to predict the best sparse format in terms of performance (GFLOPS), was also able to achieve better overall energy efficiency than any of its constituent sparse formats. To prove the practical effectiveness of BestSF, we measured its overhead and evaluated the performance and energy efficiency gain of using it as a building block in a GPU-based Preconditioned Conjugate Gradient (PCG) iterative solver.

Automatically selecting the best performing sparse format was also addressed in [47], where a decision tree classifier was used. In our previous work [8], we demonstrated that using multiclass

Support Vector Machines (SVMs) with only very easy to compute sparsity features delivered better results. SVMs were also used in Nitro [40, 41] to select the best variant of different kernels (CG solvers, BFS, Histogram, Sort, and SpMV). To the best of our knowledge, all these existing studies of the problem of sparse format selection on GPU using a machine learning approach did not consider the difference between the performance of different sparse formats for a given matrix in the learning phase. In fact, to maximize the overall performance of the SpMV kernel, a good sparse format selection system should be trained to correctly classify the matrices with large performance differences between different sparse formats even at the expense of misclassifying sparse matrices with almost no performance difference. Also, the overhead of selecting the best performing sparse format has not been thoroughly discussed making the situation not very clear about which kind of applications can benefit from using such sparse format selection systems. Finally, while many recent studies suggest that energy efficiency should be considered on par with the performance in building both the hardware and the software [51, 55], we can notice that this aspect has been largely ignored when proposing, optimizing, or selecting between different sparse formats for the SpMV on GPU. This article comes as a continuation and extension of our previous work [8]. Its main new contributions are the following:

- Using a weighted pairwise cost-sensitive classification approach rather than the standard multiclass classification. We learn pairwise models between every pair of sparse formats (classes) and choose the format that was predicted most often by the pairwise models, and we also weight each learning point of the pairwise prediction problem by the SpMV performance difference between the two sparse formats.
- Complement our feature set with other easy to compute sparsity features. Also, BCSR and DIA formats are incorporated into our selection system.
- Evaluate the GPU energy efficiency of the SpMV kernel under BestSF. We do not claim to cover all the aspects affecting the energy efficiency for the SpMV kernel on GPU (like dynamic voltage and frequency scaling), but we provide an empirical study about the impact of selecting the best performing sparse format using BestSF on the overall energy efficiency of the SpMV kernel.
- Prove the practical effectiveness of BestSF by evaluating its overhead introduced by the online selection and translation between different sparse formats. We also evaluate the gain of using BestSF as a building block in a GPU-based PCG iterative solver.

The rest of this article is organized as follows: Section 2 gives a background about the sparse formats considered. Section 3 introduces the problem of sparse format selection. Section 4 gives an overview of BestSF’s workflow while Sections 5 and 6 cover all the details about the used sparsity features and the learning process. In Section 7, we evaluate both the performance and energy efficiency of BestSF. Section 8 is dedicated to evaluating the use of BestSF as a building block in a GPU-based PCG iterative solver. Finally, Section 9 discusses the related work, and Section 10 concludes our article.

2 SPARSE MATRIX FORMATS

The SpMV is a non-trivial level-2 BLAS operation that multiplies a sparse matrix $A(n \times m)$ by a dense vector $x(m)$; the result of this operation is a vector $y(n)$. Usual matrix formats (dense formats) are not efficient for implementing this kernel because of the memory wasted for storing zero values and the computing power lost in many multiplications by zero. The use of sparse formats, which reduces storing zero values, introduces many challenges. In the GPU context, over the memory-bounded nature of the SpMV [19], these sparse formats can lead, depending on the sparsity characteristics of the input matrix, to irregular and non-coalesced memory operations and

Fig. 1. The sparse matrix example in COO, CSR, BCSR, ELL, DIA, and HYB.

also to divergence among the threads in a single warp. In this work, we consider six major sparse formats: COO, CSR, BCSR, DIA, ELL, and HYB. For the simple sparse formats COO, CSR, ELL, DIA, and HYB, we used their implementations available in the CUDA-based open source CUSP library [18]. For the BCSR format, we used its implementation available in cuSPARSE library [45]. We will use the sparse matrix shown in Figure 1 to briefly introduce these formats. More details can be found in [6, 21].

2.1 Coordinate Storage (COO)

COO uses three arrays: one to store the nonzero elements of the matrix (*data*), and the two others to store the coordinates (*row_index* and *col_index*). It is the most intuitive storage scheme for sparse matrices (Figure 1). The memory space needed for this format is $size_{value} \times nnz + 2 \times size_{index} \times nnz$.

2.2 Compressed Sparse Row (CSR)

CSR uses three arrays: *data*, *col_index*, and *ptr*. The array *data* stores the nonzero elements. The integer array *col_index* is for storing the column indices of the nonzero elements of the sparse matrix. The last array, *ptr*, is used to store row pointers to the offset of each row in *data* (Figure 1). Due to its memory space efficiency ($size_{value} \times nnz + size_{index} \times (nnz + m + 1)$), CSR is the most popular format for storing sparse matrices.

2.3 Blocked Compressed Sparse Row (BCSR)

BCSR divides the input matrix into blocks of $r \times c$ elements, and stores each non-empty block similar to the CSR format. If $r = c = 1$, the BCSR format is equivalent to the CSR format. An example of 2×2 block size is shown in Figure 1. BCSR format stores only one column index and one row pointer per block. Thus, the memory requirement is $size_{value} \times (r \times c \times k) + size_{index} \times ((n/r + 1) + k)$, where k is the number of blocks. BCSR format can store fewer row pointers and column indices than CSR, but at the possible cost of filling in explicit zeros (padding).

The best block size that leads to the best performance depends on the sparsity of the input matrix (problem size, block density, etc.) [14]. To find the best block size for each input matrix, we considered a similar approach to [52, 53] based on the fill ratio estimation algorithm proposed in [54] which accurately measures the ratio of the number of stored values (including explicit zeros) to the number of non-zeros by sampling only a small number of rows of the matrix.

2.4 ELLPACK (ELL)

ELL uses two arrays: *data* and *col_index*. The array *data* stores the nonzero elements. The integer array *col_index* stores the column indices of each nonzero element (Figure 1). If the dimension of the sparse matrix is $n \times m$, each array has a size of $n \times \max$ with $\max$ the maximum number of nonzero elements in a single row of the sparse matrix; thus, the memory space needed for this format is $\text{size}_{\text{value}} \times n \times \max + \text{size}_{\text{index}} \times n \times \max$. Each row of *data* stores the nonzero elements of the corresponding row of the sparse matrix, and each row from *col_index* stores the column indices of nonzero elements from the corresponding row of the sparse matrix. Zeros are added to every row of *data* and *col_index* with the number of nonzero elements less than $\max$ (zero padding). A given matrix fails to convert to ELL if the memory space needed after the zero padding exceeds the total global memory of the used GPU.

2.5 Diagonal (DIA)

DIA uses two arrays: *data* to store the nonzero elements, and *offsets* to store the offset of each diagonal from the central main diagonal which corresponds to the offset 0. DIA format is designed mainly for the sparse matrices for which most of the nonzero values are stored along diagonals. If some weak diagonals have very few nonzero elements, the DIA format may waste a large amount of memory space. In the example of the matrix A (Figure 1), DIA introduces a significant amount of zero padding, but it can be very suitable for regular sparse matrices arising from some regular discretization methods [6]. If *dia* is the number of diagonals of the input sparse matrix with at least one nonzero element, then the memory needed for the DIA format is $\text{size}_{\text{value}} \times \min(n, m) \times \text{dia} + \text{size}_{\text{index}} \times \text{dia}$. Similar to ELL, a given matrix fails to convert to DIA if the memory space needed after the zero padding exceeds the total global memory of the used GPU.

2.6 Hybrid ELL/COO (HYB)

HYB partitions the matrix into two parts, a “dense” part to be stored using ELL and a “very sparse” part to be stored using COO. Given an input sparse matrix, CUSP implementation computes a histogram of row sizes to find a threshold value k , where other implementations just take, as a threshold, the value of nonzero elements per row [47]. ELL is used for all the nonzero elements of the columns on the left side of k , whereas COO is used for the rest of the nonzero elements on the right of k (Figure 1). The memory space needed for this format is $\text{size}_{\text{value}} \times (n \times k + X) + \text{size}_{\text{index}} \times (n \times k + 2X)$ where X is the number of nonzero values handled by the COO format.

3 THE IMPORTANCE OF SPARSE FORMAT SELECTION ON GPU

In this section, we introduce the problem of sparse format selection based on our experimental observations on two different NVIDIA GPU architectures (Table 1). Note that in all our experiments, the GPU cache was activated but not the texture memory, and the CUDA code was compiled using *nvcc8* with the computing capability supported by each device. The nonzero elements of the matrices are coded using double precision.

Figure 2 shows the performance (GFLOPS), on two NVIDIA GPUs (Table 1), of the SpMV kernel under the sparse formats COO, CSR, BCSR, ELL, DIA, and HYB for five sparse matrices from the SuiteSparse matrix collection [20]. We can make two main observations:

- The best performing sparse format (the sparse format with the highest GFLOPS value) varies depending on the sparsity of the input matrix.
- The best performing sparse format for the same input matrix can vary from one GPU to another as we can see for the matrix 1 in Figure 2(a) and (b).

Table 1. GPUs Used in Our Experiments

	GPU1	GPU2
GPU model	GTX Titan X	GTX 1080
Architecture	Maxwell	Pascal
Compute capability	5.2	6.1
GPU cores	3,072	2,560
Shared memory/SM (Bytes)	49,152	49,152
L2 cache (Bytes)	3,145,728	2,097,152
Total global memory (MB)	12,288	8,113
TDP (W)	250	180

Fig. 2. Performance of the SpMV kernel under COO, CSR, BCSR, ELL, DIA, and HYB for five sparse matrices on two different GPUs. Note that the matrices 3, 4, 5 were not convertible to DIA format because of the memory space limitation of the used GPUs.

It is evident from this example (Figure 2) that the best performing sparse format for a given matrix depends both on the sparsity structure of the matrix and the used GPU. *Matching between each input sparse matrix and its best performing format on a given GPU is known as the problem of sparse format selection.*

3.1 Observations on the GPU Performance of the SpMV Kernel

To prove the importance of this problem, we show in the following the impact of selecting the best performing format on the overall performance of the SpMV kernel on GPU. To do so, we measured the performance of the SpMV kernel under the six considered sparse formats on a dataset¹ of 1,000 real-world sparse matrices from the SuiteSparse matrix collection [20]. All these sparse matrices fit into the global memory of the two GPUs used in our work (Table 1).

The pie chart in Figure 3 displays the distribution of the best performing sparse format on the 1,000 sparse matrices of our dataset. We can observe that, on the two GPUs, CSR covers an important percentage of the dataset (34% on Maxwell and 45% on Pascal). On the Maxwell GPU, BCSR comes in the second place followed by ELL and HYB. On the Pascal GPU, ELL comes in the second place followed by BCSR and HYB. On both GPUs, COO and DIA cover only a small percentage (less than 7%) of our dataset. Table 2 shows the average performance loss, on our

¹The list of the sparse matrices used in our work is included as a supplementary file to this manuscript.

Fig. 3. Distribution of the best performing sparse format for the SpMV kernel on a dataset of 1,000 sparse matrices using two NVIDIA GPUs.

Fig. 4. Distribution of the matrices for which CSR is not the best performing sparse format on the intervals of performance loss of using CSR instead of the best performing sparse format for each matrix.

Table 2. The Average Performance Loss of Using a Single Sparse Format for All the Inputs Instead of Using the Best Performing Format for Each Input Matrix

GPU	COO	CSR	BCSR	DIA	ELL	HYB
Maxwell	45.9%	21.3%	35.8%	94.2%	51.0%	26.2%
Pascal	40.8%	16.0%	47.0%	92.9%	47.7%	20.9%

dataset, if one of the sparse formats COO, CSR, BCSR, ELL, DIA, or HYB is used instead of the best performing format for each input matrix of our dataset. For example, the use of the HYB format introduces an average performance loss of more than 26% on the Maxwell GPU and 20% on the Pascal GPU. DIA and ELL introduce an important performance loss on the two GPUs. Note that given the zero padding needed in the construction of the sparse formats DIA and ELL, some sparse matrices fail to convert to these two sparse representations introducing 100% of performance loss. We can also notice that CSR has the best overall performance with less than 21.3% of performance loss on both GPUs which qualifies CSR as the “winner-takes-all” sparse format. However, as shown in Figure 4, it introduces a significant performance loss of more than 20% on an important number of sparse matrices of our dataset. All the previous observations clearly indicate that *selecting the best format for each input matrix is very important for optimizing the overall performance of the SpMV kernel on GPU*.

3.2 Observations on the GPU Energy Efficiency of the SpMV Kernel

The metric used in our study to characterize the energy efficiency is performance per watt (FLOPS/W) or the number of floating point operations per joule (FLOP/Joule) as $FLOPS/W = \frac{FLOP/Second}{Joules/Second} = FLOP/Joule$ [51]. For the SpMV kernel, the number of floating point operations is proportional to the number of nonzero elements of the sparse matrix, but the total energy (Joules) must be first measured in order to compute the performance per watt after that. We utilize the onboard GPU power sensors to obtain the instant power consumption of our kernels using the NVIDIA Management Library (NVML) interface which returns the power readings in milliwatts [43]. We followed a similar methodology as in [10, 39] to compute the total GPU energy consumed by our kernels.

We measured the energy efficiency of the SpMV kernel under the considered six sparse formats on our two GPUs for all the sparse matrices of our dataset. After that, we found the best sparse format in terms of energy efficiency for each one. Similar to Table 2, Table 3 shows the average energy efficiency loss, on our dataset, if one of the sparse formats COO, CSR, BCSR, ELL, DIA, or

Table 3. The Average Energy Efficiency Loss of Using One Format for All the Inputs Instead of Using the Most Energy Efficient Format for Each Input Matrix

GPU	COO	CSR	BCSR	DIA	ELL	HYB
Maxwell	45.4%	24.6%	40.9%	94.4%	52.2%	28.8%
Pascal	39.7%	22.0%	47.9%	93.2%	47.5%	19.7%

Fig. 5. Distribution of the matrices of our dataset on the intervals of energy efficiency loss introduced by using the best performing sparse format instead of the most energy efficient sparse format.

HYB is used instead of the most energy efficient format for each input matrix of our dataset. It is clear from Table 3 that there is no best for all sparse format in terms of energy efficiency. For example, using the CSR sparse format for all the matrices will introduce more than 22% of energy efficiency loss. Also, we noticed that, for 10% of the matrices of our dataset in Maxwell and almost 24% in Pascal, the most energy efficient sparse format is *different* from the best performing one. This observation joins what has been noticed in [16] and confirms that changing the program implementation (in our case the SpMV implementation using different sparse formats) may affect the performance and energy efficiency in a different way. However, we can notice in Figure 5, which shows the distribution of the matrices of our dataset on intervals of energy efficiency loss if the best sparse format in terms of performance is used instead of the most energy efficient one, that we have a significant energy efficiency loss (of more than 10%) only for a relatively small number of cases. All the previous observations suggest that *using the best performing sparse format (GFLOPS) will also ensure the best energy efficiency (FLOPS/W) for a large number of sparse matrices.*

4 BestSF: A LEARNING-BASED SPARSE META-FORMAT

In this section, we present a high-level overview of BestSF's architecture regardless of the considered sparse formats and the used machine learning algorithm. All the details about the used sparsity features and the learning process will follow up in Sections 5 and 6.

Given a set of sparse formats $f_i \in \mathcal{P}$ and a set of sparse matrices $A_i \in \mathcal{A}$, $P(A_i, f_j)$ is the performance (GFLOPS) of the SpMV kernel using the sparse matrix A_i under the sparse format f_j on a given GPU. The problem of sparse format selection consists in finding a mapping $s : \mathcal{A} \mapsto \mathcal{P}$ such that $\sum_{A_i \in \mathcal{A}} P(A_i, s(A_i))$ is maximized. Almost all contemporary approaches to this kind of algorithm and data structure selection problem use machine learning techniques to build predictors of the best algorithm as a function of some features that characterize the performance of the candidate algorithms [9]. Two alternatives have been widely used:

- Standard multiclass classification: In this approach, we can train a classifier that discriminates between different sparse formats based on some sparsity features. The main downside

Fig. 6. BestSF workflow.

of this approach is that the model can be very complex to learn as it considers all the sparse formats together.

- Regression: In this approach, we can train regression models for the performance of the SpMV kernel under each sparse format on a given GPU and select the sparse format with the best-predicted performance. Unlike the multiclass classification, this approach learns the performance model of each class individually which is less complex than learning a multiclass classification model.

Different from these two alternatives, BestSF relies on a pairwise weighted classification approach. This approach has been shown to be effective with the recent versions of SATzilla, a portfolio-based approach for SAT [57]. It learns pairwise models between every pair of sparse formats and chooses the sparse format that was predicted most often by the pairwise models. We also weight each training point of the pairwise prediction problem by the SpMV performance difference between the two sparse formats. The lower a training point's weight, the less significant that training point in creating the model. This is motivated by the fact that we care most about getting predictions with large performance differences correct even at the expense of misclassifying matrices with almost no performance difference.

As shown in Figure 6, BestSF includes two stages: an offline training stage and an online decision-making stage. The offline stage consists in training the pairwise models between every pair of sparse formats f_i and f_j . If we consider p sparse formats, then the number of pairwise models to train is $p \times (p - 1)/2$. Given an input sparse matrix, its best sparse format is predicted in the online decision-making stage.

4.1 The Offline Training Stage

As shown in Figure 6, the offline training of the pairwise models includes the following steps:

- (1) Constitute a dataset of sparse matrices that will be used in the learning and the testing of the classification system.
- (2) Extract some relevant sparsity features of the matrices that can capture at best the performance patterns of the considered sparse formats on GPU.
- (3) For all the matrices of the dataset, compute these features and run the SpMV kernel under each sparse format on the GPU to determine the real class of each matrix.

Table 4. Sparsity Features

Features	
<u>Dimensional Features:</u>	
n	The number of rows of the sparse matrix.
m	The number of columns of the sparse matrix.
nnz	The number of nonzero elements of the sparse matrix.
<u>Features related to the distribution of nonzero elements:</u>	
mu	The average number of nonzero elements per row, $mu = \frac{nnz}{n}$.
sd	The standard deviation on nonzero elements per row. If r_i is the number of nonzero elements of the row i , then $sd = \sqrt{\frac{1}{n} \sum_{i=1}^n (r_i - mu)^2}.$
d	The density of nonzero elements in the sparse matrix, $d = \frac{nnz}{n \times m}$.
cv	The coefficient of variation of nonzero elements per row, $cv = \frac{sd}{mu}$.
max	The maximum number of nonzero elements in a single row of the matrix, $max = MAX(r_1, r_2, \dots, r_n)$.
$max - mu$	The difference between max and mu .
dia	The number of diagonals with at least one nonzero element in the matrix.
dis	Calculated as $dis = \frac{1}{n} \sum_{i=1}^n dis_i$ where dis_i is the average distance between each pair of continuous nonzero elements in the row i of the matrix.

- (4) Select the most important features to use for training each pairwise model (f_i, f_j) and construct the pairwise models using a suitable machine learning algorithm. When training each model (f_i, f_j) , each training point is weighted according to the performance difference between the sparse formats f_i and f_j .

4.2 The Online Decision-Making Stage

Given an input sparse matrix in a default sparse format $f^{(d)}$, the following steps are performed to predict its best sparse format:

- (1) Compute the sparsity features.
- (2) Use these features as inputs to the trained pairwise models. The predicted best sparse format, $f^{(p)}$, is then the one that has been predicted most often by the pairwise models.
- (3) Convert the considered sparse matrix to the predicted best sparse format ($f^{(d)} \rightarrow f^{(p)}$).

In our work, we used the implementation of the SpMV kernel under COO, CSR, DIA, ELL, and HYB available in the open source CUDA-based CUSP library [18]. Concerning the BCSR format, cuSPARSE [45] library is used. The dataset used includes 1,000 sparse matrices from the SuiteSparse matrix collection [20]. In the following sections, we provide all the details about the sparsity features used in our work, the learning process, and the evaluation of BestSF.

5 SPARSITY FEATURES

Table 4 summarizes the used features in our study. We can notice that all the considered features are very easy to compute (computationally not expensive) for a given input matrix. Given the rising complexity of the GPU architecture and the wide range of different sparsity patterns, it is very hard to isolate and explain the exact effect of each one of these sparsity features on the

performance of the SpMV kernel under the considered sparse formats on GPU. After all, the lack of such thorough understanding is behind the motivation of using a machine learning approach. However, we summarize in the following points some general observations about the performance of different sparse formats and the considered features:

- The number of rows n , the number of columns m , and the number of nonzero elements nnz are known as the dimensional features of the problem as they characterize the volume of work needed to compute the SpMV kernel.
- The performance of the SpMV on GPU, depending on the used sparse format, is sensitive to the distribution of the nonzero elements on the rows of the sparse matrix. To characterize this sensitivity, we included mu , sd , d , cv , max , and $max - mu$ (Table 4). CSR and BCSR kernels give good performance for matrices with regular distribution of the nonzero elements, characterized by large values of mu and relatively small values of cv , for which these kernels do not suffer from unbalanced distribution among the threads. Also, BCSR favors sparse matrices with relatively high values of d . For the matrices with a highly unbalanced distribution of the nonzero elements, thus characterized by large values of cv , the use of HYB may have better performance as it breaks down the matrix to two sub-matrices with more regular distributions. The ELL format is expected to give good performance for the matrices with relatively small values of $max - mu$ because, in this case, ELL will less suffer from the problem of excessive zero padding that introduces more computation than needed.
- The feature dia is mainly related to the DIA format. The performance of the SpMV kernel under the DIA format increases with the decreasing values of dia .
- The feature dis is used to naively characterize the random access to the multiplied vector and the dispersion level of the nonzero elements of the sparse matrix. Also, small values of dis is a good indicator of the existence of many blocks of nonzero elements in the matrix which favors the BCSR format.

6 LEARNING THE PAIRWISE MODELS

6.1 Learning Algorithm

We chose to use SVMs as a training algorithm in building BestSF. SVMs, in general, have been identified in [30] as a promising machine learning method for the search algorithm selection problem in comparison with other machine learning methods. Also, SVMs deliver a good accuracy when the decision boundaries between different classes are highly nonlinear [17, 49], which we intuitively expect to be the case in the context of sparse format selection. Basically known as a binary classifier, several methods exist to extend SVMs for multiclass classification [27]:

- One-against-all approach: in the learning phase, it constructs p SVM models where p is the number of classes. The n th model is trained with all of the examples in the n th class with positive labels, and all other examples with negative labels. To find the predicted class for a new example, the decision functions of the previous models are evaluated and the new example is affected to the class with the highest value of the decision function.
- One-against-one approach: in the learning phase, this method performs a pairwise classification by constructing $p(p - 1)/2$ classifiers where each one is trained on data from two classes. Given a new example, it is affected to the class that has been voted most often by the previously trained modes. It has been shown in [27] that the one-against-one approach is more suitable for practical use than the one-against-all. This approach was used in our previous work [8].

The main drawback of the previous methods based on standard SVMs is that the training points are all given equal importance in the training phase and does not allow relative importance of training points to be considered. To solve this problem, we used the binary WSVM [60] to build $p \times (p - 1)/2$ WSVM pairwise models (p is the number of sparse formats). For training the model (f_i, f_j) , we attribute a weight, w_s , to each training point (s) according to the performance difference between the corresponding pair of sparse formats as shown in Equation (1). $P(A_s, f_i)$ and $P(A_s, f_j)$ in Equation (1) represent the performance of the SpMV kernel using the matrix A_s under the sparse formats f_i, f_j respectively. $w_s(f_i, f_j)$ depends linearly on the performance improvement between the two sparse formats f_i and f_j . Given a new example (sparse matrix in our case), we used a similar voting strategy to the one used in the standard one-against-one strategy: if the model (f_i, f_j) selects f_i , then the score of f_i is increased by one. Otherwise, the score of f_j is increased by one. We repeat this process with all $p \times (p - 1)/2$ pairwise models. Then, the new example is affected to the class with the largest score.

$$w_s(f_i, f_j) = \alpha \times \frac{\max(P(A_s, f_i), P(A_s, f_j)) - \min(P(A_s, f_i), P(A_s, f_j))}{\max(P(A_s, f_i), P(A_s, f_j))}. \quad (1)$$

We used the implementation of WSVM available in the open source LibSVM library [13]. We kept using the standard C-SVM algorithm and selected the Gaussian Radial Basis Function (RBF) kernel. We used the cross-validation technique as suggested in [26] for selecting the two parameters C and γ . The parameter α in Equation (1) is set to $\alpha = 100.0$ (we achieved our best results with this value). A very important step is scaling the data points before the training/testing process. LibSVM comes with a default linear scaling scheme that linearly scales each attribute to the range $[-1, 1]$ or $[0, 1]$. However, we found that using a logarithmic transformation ($\log_{10}(x)$) of our data points instead of the linear scaling improves both the accuracy of classification and the performance of BestSF.

6.2 Wrapper Feature Selection

Our feature set includes 11 sparsity features of the sparse matrices as shown in Table 4. Besides the three-dimensional features n , m , and nnz , that characterize the volume of work needed to compute the SpMV kernel, not all of the other features are important for training each pairwise model. In other words, we have to select the most important features for each pairwise model. To do so, we used a wrapper-based approach [28] that runs the learning process with different feature subsets of our initial feature set (our 11 features) to determine the best subset to train each pairwise model. Given that our initial feature set includes only 11 features, we used the exhaustive search as a base for our wrapper feature selection, but other search strategies can be used if more features are included and the search space becomes very huge. The features subsets in Table 5 are obtained using our dataset on our GPUs. These features subsets may slightly change if we use other GPUs or other datasets. This is why BestSF includes the feature selection as part of the learning phase as shown in Figure 6.

6.3 Training and Testing

First, we run the SpMV kernel under the considered sparse formats for all the 1,000 sparse matrices of our dataset, and each matrix is attributed to its class according to its best sparse format. Then, we randomly select 80% of our dataset (800 sparse matrices) to constitute the training set that will be used to train each of the pairwise models. The remaining 20% (200 sparse matrices) are used as a testing set. This 80-20 splitting of our dataset is repeated five times to constitute five different learning/testing sets. We run the training process to train all the pairwise models using WSVN as

Table 5. Sparsity Features Used to Train Each Pairwise Model

Classifier	Feature subset	Classifier	Feature subset
(COO,CSR)	$\{n, m, nnz, mu, sd, max\}$	(CSR,BCSR)	$\{n, m, nnz, mu, d, dis\}$
(COO,ELL)	$\{n, m, nnz, d, max\}$	(DIA,ELL)	$\{n, m, nnz, d, cv, max, dia\}$
(COO,DIA)	$\{n, m, nnz, sd, dia\}$	(DIA,HYB)	$\{n, m, nnz, sd, max - mu, dia\}$
(COO,HYB)	$\{n, m, nnz, mu, cv\}$	(DIA,BCSR)	$\{n, m, nnz, d, dia, dis\}$
(COO,BCSR)	$\{n, m, nnz, d, dis\}$	(ELL,HYB)	$\{n, m, nnz, d, cv, max - mu, dis\}$
(CSR,ELL)	$\{n, m, nnz, cv, max\}$	(ELL,BCSR)	$\{n, m, nnz, d, max - mu, dis\}$
(CSR,DIA)	$\{n, m, nnz, dis, dia\}$	(HYB,BCSR)	$\{n, m, nnz, d, cv, dis\}$
(CSR,HYB)	$\{n, m, nnz, mu, max, max - mu\}$		

Table 6. Accuracy and Overall Performance of BestSF

GPU	Accuracy	PLUB	PGO (COO)	PGO (CSR)	PGO (BCSR)	PGO (HYB)
Maxwell	85.3%	1.6%	94.3%	52.4%	71.6%	63.3%
Pascal	83.4%	2.1%	91.9%	41.0%	82.3%	46.3%

explained above. Then, the testing set is used to evaluate the accuracy, performance, and energy efficiency of BestSF.

7 EVALUATION

In this section, we evaluate the accuracy of BestSF as well as its overall performance and energy efficiency. We also compared BestSF, which is based on weighted pairwise classification, with the standard multiclass SVM classification used in [8, 40, 41] and also the decision trees used in [47]. The results presented for each GPU are the average of the results obtained with each one of the five different random learning/testing divisions prepared previously. A comparison of the performance of BestSF with CSR5 [35] and Merge-based CSR [36] is provided and the overhead introduced by the online decision-making phase (Figure 6) is discussed. In our experiments, we considered the CSR format as being the default sparse format ($f^{(d)} = \text{CSR}$) as it is the “winner-takes-all” sparse format.

7.1 Accuracy of Classification

Given a testing set of N unseen sparse matrices ($N = 200$ for our case) and if N_+ is the number of matrices correctly classified, then the accuracy is calculated as $100 \times N_+/N$. Table 6 shows that BestSF achieved from 83.4% to 85.3% of accuracy between the two used GPUs.

7.2 The Overall Performance of BestSF

We calculate the average Performance Loss Under Best (PLUB). This metric measures how far the performance of BestSF is from the performance of an ideal classifier (with 100% of accuracy). If we consider a testing set with N sparse matrices, $f_k^{(r)}$ is the real best performing sparse format for each sparse matrix A_k from the testing set, then PLUB is given in Equation (2). We also calculate $\text{PGO}(f_i)$, the average Performance Gain of using BestSF Over using the sparse format f_i for all the testing matrices as shown in Equation (3). Table 6 shows the PLUB values obtained on our two GPUs and also the values of $\text{PGO}(\text{COO})$, $\text{PGO}(\text{CSR})$, $\text{PGO}(\text{BCSR})$, and $\text{PGO}(\text{HYB})$. Note that we did not report the values of $\text{PGO}(\text{ELL})$ and $\text{PGO}(\text{DIA})$ because some testing matrices fail to convert to

Table 7. Overall Energy Efficiency of BestSF

GPU	ELUB	EGO (COO)	EGO (CSR)	EGO (BCSR)	EGO (HYB)
Maxwell	2.4%	96.2%	68.4%	85.2%	79.7%
Pascal	3.8%	90.2%	46.1%	89.4%	56.3%

the ELL and DIA formats, which makes Equation (3) not applicable for all the testing points.

$$PLUB (\%) = 100 \times \frac{1}{N} \sum_{k=1}^N \frac{P(A_k, f_k^{(r)}) - P(A_k, f_k^{(p)})}{P(A_k, f_k^{(r)})}, \quad (2)$$

$$PGO(f_i) (\%) = 100 \times \frac{1}{N} \sum_{k=1}^N \frac{P(A_k, f_k^{(p)}) - P(A_k, f_i)}{P(A_k, f_i)}. \quad (3)$$

We summarize our observations in the following points:

- On both GPUs, the PLUB is less than 3%, which means that by using BestSF to select the best performing sparse format we reached more than 97% of the maximum performance possible. By checking the misclassified testing points, we found that most of the misclassifications occur when BestSF selects the wrong sparse format but with a very close performance to the optimal one. This explains why we have a near-optimal performance even if the accuracy of classification did not exceed 86% on both GPUs.
- Compared with just using one format (COO, CSR, BCSR, or HYB) for all the testing sparse matrices, BestSF registered an average performance improvement ranging from 41% to 94.3% confirming that using BestSF is better, in terms of average performance, than using just one sparse format for all the sparse matrices of our testing sets.

7.3 The Overall Energy Efficiency of BestSF

We recall that BestSF is trained to select the best performing (GFLOPS) sparse format and not the most energy efficient format. We used the metric ELUB to measure the average energy efficiency loss introduced by using BestSF under using the most energy efficient sparse format for all the inputs. If we consider N testing matrices, $EE(A_k, f_i)$ is the GPU energy efficiency of the SpMV kernel using the sparse format f_i , $f_k^{(e)}$ is the most energy efficient sparse format for each sparse matrix A_k of the testing set, then ELUB is given in Equation (4). We recall that $f_k^{(p)}$ is the predicted sparse format for the matrix A_k . We also calculate EGO(f_i), the average Energy efficiency Gain of using BestSF Over using the sparse format f_i for all the testing matrices as shown in Equation (5). Table 7 shows the ELUB values obtained on our two GPUs and also the values of EGO(COO), EGO(CSR), EGO(BCSR), and EGO(HYB).

$$ELUB (\%) = 100 \times \frac{1}{N} \sum_{k=1}^N \frac{EE(A_k, f_k^{(e)}) - EE(A_k, f_k^{(p)})}{EE(A_k, f_k^{(e)})}, \quad (4)$$

$$EGO(f_i) (\%) = 100 \times \frac{1}{N} \sum_{k=1}^N \frac{EE(A_k, f_k^{(p)}) - EE(A_k, f_i)}{EE(A_k, f_i)}. \quad (5)$$

We can make the following observations:

- On the two used GPUs, the ELUB is less than 4% meaning that using BestSF achieved more than 96% of the optimal energy efficiency possible.

Table 8. Accuracy, PLUB, and ELUB Obtained with Using the Standard One-vs-One Multiclass SVMs and Decision Trees (BFTree) with Different Feature Sets

Feature sets	GPU	multiclass SVMs			Decision Trees (BFTree)		
		Acc. (%)	PLUB (%)	ELUB (%)	Acc. (%)	PLUB (%)	ELUB (%)
Simple features used in [47]	Maxwell	73.6	6.3	9.3	70.9	6.5	9.8
$\{d, mu, sd\}$	Pascal	71.3	6.7	10.4	70.0	7.1	10.9
Sparsity features used in Nitro [40, 41]	Maxwell	76.1	5.6	8.1	74.2	6.1	9.1
$\{mu, sd, maxdev, ell_fill, dia_fill\}$	Pascal	73.2	6.0	10.1	71.0	6.8	10.3
Our sparsity features (Table 4)	Maxwell	82.0	2.8	3.6	77.3	3.2	5.3
	Pascal	80.7	3.4	4.6	75.4	4.2	6.9

— Compared with just using one sparse format (COO, CSR, BCSR, or HYB) for all the testing sparse matrices, BestSF registered an average energy efficiency improvement ranging from 46.1% to 96.2% confirming that using BestSF is better, in terms of average energy efficiency, than using just one sparse format for all the input matrices.

7.4 Comparison with the Previous Work

In the following, we first compare the pairwise weighted classification approach used in BestSF to two other learning approaches, standard multiclass classification and decision trees, previously used for solving the problem of sparse format selection [8, 40, 41, 47]. After that, we use a set of commonly evaluated large sparse matrices to compare the performance of BestSF with the performance of CSR5 [35] and merge-based CSR [36].

In [47], decision trees were used for training a classification system for selecting the best sparse format from CSR, ELL, and HYB. We showed in our previous work [8], that using multi-class SVMs with very simple to compute sparsity features allows better accuracy and selection quality. In [40, 41], the authors proposed Nitro to select the best variant of different kernels (CG solvers, BFS, Histogram, Sort, and SpMV) based on SVMs. Concerning the SpMV kernel, Nitro considers only the three sparse formats CSR, ELL, and DIA. To improve the performance of the SpMV kernel, BestSF includes six sparse formats COO, CSR, BCSR, ELL, DIA, and HYB. In addition, to improve the quality of selection, it relies on a pairwise weighted classification approach based on WSVM as explained previously. Another important aspect that makes the difference between BestSF and the previous solutions is the set of used sparsity features. Table 8 exposes this aspect and also shows the accuracy of classification, PLUB, and ELUB obtained with using the standard multi-class classification and decision trees (BFTree). Comparing to Table 6 and Table 7, we can notice that the weighted pairwise classification, used in BestSF, outperforms both the standard one-vs-one multiclass SVMs and BFTrees in terms of accuracy, average PLUB, and average ELUB values on the two used GPUs. Also, using our sparsity features improves all the previous metrics. In addition, Figure 7 shows the distribution of the misclassifications on intervals of PLUB values obtained with different learning approaches. We can see that using the weighted pairwise classification reduces the number of misclassifications with more than 10% of performance loss ($PLUB > 10\%$). All the previous observations confirm that the weighted pairwise classification based on WSVM, used in BestSF, is more suitable for the sparse format selection problem.

Figure 8 compares the performance and energy efficiency of the SpMV kernel under BestSF, across commonly used sparse matrices in evaluating different sparse formats [6, 35], to the performance obtained with using merge-based CSR [36] and CSR5 [35]. Unlike many sparse formats that perform very good on some sparse matrices and very bad on others, both CSR5 and merge-based

Fig. 7. Distribution of misclassifications on intervals of PLUB.

Fig. 8. Comparing BestSF to CSR5 [35] and merge-based CSR [36] on commonly evaluated large sparse matrices.

CSR have been designed to deliver consistently high performance on a wide range of sparsity patterns (structured and unstructured). The merge-based CSR is a new implementation of the SpMV kernel using the CSR format based on a fine-grained merge-based parallel decomposition to have a well-balanced execution. CSR5 is a new variation of the CSR format and the CSR5-based SpMV kernel is implemented based on a new low-overhead segmented sum algorithm. It showed a good performance in comparison to HYB, BRC (Blocked Row-Column [5]), and ACSR (Adaptive-CSR [4]) on both structured and unstructured matrices. We can observe in Figure 8 that BestSF, by selecting the right sparse format (from COO, CSR, ELL, DIA, HYB, and BCSR) for each sparse matrix, achieved an important performance improvement for most of the considered sparse matrices. We conclude that *selecting between different existing formats is a very promising approach to have a consistently high performance across different sparsity patterns*.

7.5 Overhead

As shown in Figure 6, BestSF includes an online decision-making phase in which given an input sparse matrix in its default sparse format $f^{(d)}$, the sparsity features are first calculated, the pairwise models are used to predict the best sparse format $f^{(p)}$, and finally the input sparse matrix is converted to the sparse format $f^{(p)}$ ($f^{(d)} \rightarrow f^{(p)}$). Thus, the practical effectiveness of BestSF depends

Table 9. The Sparse Matrices Used to Evaluate the Overhead Introduced by the Online Decision-Making Phase of BestSF

	Matrix	n	m	nnz
G1	<i>FEM_3D_thermal2</i>	147,900	147,900	3,489,300
	<i>TSOPF_RS_b2383</i>	38,120	38,120	16,171,169
	<i>wikipedia-20051105</i>	1,634,989	1,634,989	19,753,078
	<i>12month1</i>	12,471	872,622	22,624,727
	<i>relat9</i>	12,360,060	549,336	38,955,420
	<i>cage15</i>	5,154,859	5,154,859	99,199,551
G2	<i>nasasrb</i>	54,870	54,870	2,677,324
	<i>smt</i>	25,710	25,710	3,753,184
	<i>ecology2</i>	999,999	999,999	4,995,991
	<i>G3_circuit</i>	1,585,478	1,585,478	7,660,826
	<i>crankseg_2</i>	63,838	63,838	14,148,858
	<i>F1</i>	343,791	343,791	26,837,113
	<i>Fault_639</i>	638,802	638,802	28,614,564
	<i>inline_1</i>	503,712	503,712	36,816,342
	<i>boneS10</i>	914,898	914,898	55,468,422
	<i>Hook_1498</i>	1,498,023	1,498,023	60,917,445
	<i>audikw_1</i>	943,695	943,695	77,651,847
G3	<i>Bump_2911</i>	2,911,419	2,911,419	127,729,899
	<i>poisson192</i>	7,077,888	7,077,888	49,324,032
	<i>poisson256</i>	16,777,216	16,777,216	117,047,296

on the time overhead to perform this online decision-making phase. Note that all the steps of the online decision-making phase are executed on the CPU side (Intel i7-3770 running at 3.4GHz for our machine) using sequential implementations (we implemented our algorithms for computing the sparsity features, and used CUSP and cuSPARSE routines for the sparse format conversion).

Table 9 shows the sparse matrices used to evaluate the time overhead of BestSF. The sparse matrices of G1 and G2 are unstructured sparse matrices selected from the SuiteSparse matrix collection [20]. G1 matrices have been suggested in [32] and most of G2 matrices have been used in [11, 12]. The two sparse matrices of G3 are structured matrices resulting from a 3D finite difference discretization using the standard seven-point stencil. The matrices G2+G3 have the particularity of being square, symmetric, and positive-definite, and they are also used to evaluate the preconditioned conjugate gradient iterative solver based on BestSF in Section 8.

If the time needed for the prediction, including both features extraction and the runtime of the models, is p_time and the time needed for sparse format conversion is c_time , then the total time overhead is $tot_time = p_time + c_time$. If the predicted sparse format $f^{(p)}$ is the same as the default sparse format $f^{(d)}$ (CSR in our case), then there is no format conversion needed and c_time equals to zero in this case. Table 10 shows both p_time and c_time and also tot_time for all the sparse matrices of Table 9 for our two GPUs. As suggested in [32], the tot_time is also presented in terms of CSR-SpMV time on GPU. For example, an overhead of $30 \times$ CSR-SpMV means that the time overhead for the online decision-making phase takes about 30 times as long as the actual GPU SpMV time using the standard CSR representation. We refer to the CSR format as it is the most used format in different application domains. As shown in Table 10, the overhead in terms of CSR-SpMV, is ranging from 21 to 152 on the Maxwell GPU and from 17 to 144 on the Pascal

Table 10. Overhead Introduced by the Online Decision-Making Phase of BestSF

	Matrix	Maxwell GTX TITAN X						Pascal GTX 1080					
		$f^{(p)}$	p_time (ms)	c_time (ms)	tot_time (ms)	tot_time ($\times CSR-SpMV$)	$f^{(p)}$	p_time (ms)	c_time (ms)	tot_time (ms)	tot_time ($\times CSR-SpMV$)		
G1	<i>FEM_3D_thermal2</i>	ELL	16.02	30.02	46.04	97.06	ELL	16.03	30.02	46.05	116.23		
	<i>TSOPF_RS_b2383</i>	BCSR	27.02	95.76	122.78	152.03	BCSR	27.03	95.76	122.79	143.52		
	<i>wikipedia-20051105</i>	COO	74.05	125.09	199.14	30.31	HYB	74.06	173.12	247.18	25.68		
	<i>12month1</i>	CSR	42.03	0.00	42.03	21.05	CSR	42.04	0.00	42.04	17.32		
	<i>relat9</i>	HYB	149.06	349.23	498.29	87.13	ELL	149.06	322.21	471.28	91.95		
	<i>cage15</i>	HYB	218.14	853.57	1071.71	60.40	HYB	218.15	853.57	1071.72	29.12		
G2	<i>nasasrb</i>	BCSR	6.00	17.80	23.80	102.62	BCSR	6.01	17.80	23.81	108.96		
	<i>smt</i>	CSR	7.01	0.00	7.01	34.36	CSR	7.02	0.00	7.02	33.09		
	<i>ecology2</i>	DIA	14.02	46.03	60.05	146.87	DIA	14.03	46.03	60.06	113.18		
	<i>G3_circuit</i>	CSR	22.00	0.00	22.00	26.67	CSR	22.01	0.00	22.01	27.35		
	<i>crankseg_2</i>	CSR	25.01	0.00	25.01	37.32	CSR	25.02	0.00	25.02	31.77		
	<i>F1</i>	BCSR	52.04	156.85	208.90	115.45	BCSR	52.05	156.85	208.91	84.95		
G3	<i>Fault_639</i>	BCSR	58.03	243.16	301.19	133.97	BCSR	58.04	243.16	301.20	121.29		
	<i>inline_1</i>	BCSR	69.05	210.80	279.85	123.10	BCSR	69.05	210.80	279.86	101.21		
	<i>boneS10</i>	BCSR	106.03	318.70	424.73	122.04	BCSR	106.04	318.70	424.74	95.82		
	<i>Hook_1498</i>	BCSR	141.10	350.25	491.34	99.81	BCSR	141.10	350.25	491.35	74.28		
	<i>audikw_1</i>	BCSR	149.10	453.75	602.85	128.54	BCSR	149.11	453.75	602.86	86.42		
	<i>Bump_2911</i>	HYB	273.18	1095.73	1368.91	127.58	HYB	273.19	1095.73	1368.92	78.09		
G3	<i>poisson192</i>	DIA	138.06	466.31	604.37	104.20	DIA	138.07	466.31	604.38	89.77		
	<i>poisson256</i>	DIA	329.05	606.32	935.36	56.47	DIA	329.05	606.32	935.37	41.73		

$f^{(p)}$ is the predicted sparse format by BestSF.

p_time includes both sparsity features extraction and the runtime of the models. The measured runtime of the models is 0.048ms for the Maxwell GPU and 0.056ms for the Pascal GPU.

c_time represents the time needed for format conversion ($f^{(d)} \rightarrow f^{(p)}$).

$tot_time = p_time + c_time$.

GPU. We can notice that, except the matrices for which c_time is zero ($f^{(p)} = \text{CSR}$), the sparse format conversion is the main source of overhead. Note that the need for format conversion is not specific to only BestSF. For example, if we use ELL or HYB for all the input matrices (without using BestSF), putting a sparse matrix in these sparse formats usually needs to be done from the basic COO or CSR formats [21]. Note that p_time is largely dominated by the time needed for features extraction. Using the trained WSVM models is relatively fast because it mainly consists of simple dot products between the features vector (representing the input matrix) and the support vectors of the models. The measured runtime of the models is 0.048ms for the Maxwell GPU and 0.056ms for the Pascal GPU, which represents less than 1% of the total overhead for all the testing matrices (Table 10).

Given the overhead of the online decision-making phase, we conclude that BestSF is more suitable for applications in which the SpMV kernel is computed many times with the same sparse matrix. This is the case of a wide range of applications using iterative solvers for solving large linear systems ($Ax = b$) and eigenvalue problems ($Ax = \lambda x$) where the SpMV kernel is usually executed hundreds if not thousands of times with the same matrix A [6, 31].

8 APPLICATION: BESTSF AS A BUILDING BLOCK IN A GPU-BASED PCG SOLVER

In this section, we evaluate the performance and energy efficiency gain achieved when using BestSF as a building block in a GPU-based PCG iterative solver.

8.1 PCG Algorithm

Conjugate Gradient is one of the most popular iterative methods for solving large systems of linear equations $Ax = b$ where x is an unknown vector, b is a known vector, and A is a known, square, symmetric, positive-definite matrix [46, 48]. This method is generally applicable to large sparse systems for which direct methods are not efficient. This kind of sparse system arises very often when numerically solving partial differential equations and optimization problems. For unstructured sparse matrices, generally characterized by a large condition number (the ratio of the largest to smallest eigenvalue), the convergence of the CG method may need a large number of iterations. Preconditioning is usually used as a technique for improving the condition number of the input matrix and reducing the number of iterations required for convergence. However, an efficient preconditioner is not only the preconditioner that considerably reduces the number of iterations needed for the convergence but also the preconditioner for which the preconditioning matrix is easy to generate, and the preconditioning operation is easy to apply [33, 46]. Different preconditioners can be used with the PCG formulation shown in Figure 9. In this work, we consider the Jacobi (diagonal) preconditioner because of its embarrassingly parallel structure [22]. Other preconditioners, like the Symmetric Successive Over Relaxation (SSOR) and the Incomplete LU (ILU), offer better convergence rates, but applying these preconditioners comes at the price of expensive sparse triangular equations that can become a bottleneck on parallel architectures [1, 2, 34].

In this work, we use the formulation of the PCG algorithm shown in Figure 9. In each iteration (lines 8–17), the algorithm uses two dot product (DOT) operations (lines 9 and 14), two linear combinations of vectors (AXPY) in lines 10 and 11 and one AYPX operation in line 16 which consist of two successive operations: scaling a vector by a scalar (SCAL) and linear combination of vectors (AXPY). As we are considering the diagonal preconditioner, the preconditioning operation (line 12) consists of an element-wise multiplication (XMY) of the diagonal d of the matrix A ($M = d^{-1}$) by the vector r . The SpMV operation (line 8) is by far the most expensive operation among all the operations used by the Jacobi-based PCG algorithm.

```

1:  $i = 0$ 
2:  $r = b - Ax$ 
3:  $z = Mr$ 
4:  $p = z$ 
5:  $\alpha_{new} = r^T z$ 
6:  $\alpha_0 = \alpha_{new}$ 
7: while ( $i < i_{max}$  and  $\alpha_{new} > \epsilon^2 \alpha_0$ )
8:    $y = Ap$ 
9:    $\alpha = \frac{\alpha_{new}}{y^T p}$ 
10:   $x = x + \alpha y$ 
11:   $r = r - \alpha y$ 
12:   $z = Mr$ 
13:   $\alpha_{old} = \alpha_{new}$ 
14:   $\alpha_{new} = r^T z$ 
15:   $\beta = \frac{\alpha_{new}}{\alpha_{old}}$ 
16:   $p = z + \beta p$ 
17:   $i = i + 1$ 
18: end while

```

Fig. 9. Preconditioned Conjugate Gradient (PCG) algorithm.

In our implementation, the NVIDIA’s cuBLAS library [44] was used for the kernels DOT, AXPY, and AYPX (by combining SCAL and AXPY). For the XMY kernel, the thrust library [7] was used. Finally, for the SpMV kernel, we used the implementation of this kernel under different sparse formats COO, CSR, DIA, ELL, and HYB available in CUSP library and BCSR available in cuSPARSE.

To evaluate the gain of using BestSF for the SpMV kernel inside the PCG solver, we measured the total execution time (including the CPU-GPU data transfer) of the PCG algorithm with different sparse formats COO, CSR, BCSR, ELL, DIA, and HYB. We also used CSR5 and merge-based CSR to be able to compare BestSF with exterior sparse formats. We used 14 square, symmetric, positive-definite sparse matrices (Table 9, G2 and G3). We recall that the two matrices *poisson192* and *poisson256* (G3) are resulting from a 3D finite difference discretization using the standard seven-point stencil. They are banded and diagonally dominant structured with a relatively low condition number. We included them in the testing matrices to demonstrate the case of having a relatively small number of iterations. Note that all 14 testing matrices have not been included in the learning dataset used for training BestSF. In our experiments, the tolerance factor ϵ is set to $\epsilon = 10^{-6}$ and double precision is used.

8.2 Evaluation Results

Table 11 shows the execution time needed for the convergence for PCG-BestSF with our 14 sparse matrices and also the performance gain ($PGO(f_i)$) of using BestSF over using each of its constituent sparse formats (COO, CSR, BCSR, DIA, ELL, and HYB) and also over using merge-based CSR [36] and CSR5 [35]. Negative values of $PGO(f_i)$ indicate a performance loss under using the sparse format f_i . Note that the execution time reported in Table 11 represents the total execution time including the CPU-GPU data transfer, sparse format conversion, and format prediction time. For all 14 testing sparse matrices, we registered only one misclassification on each GPU (*G3_circuit* with 1.5% and 1.4% of performance loss on Maxwell and Pascal, respectively). BestSF was able to make the best choice for the matrices where there is a relatively important performance difference between different sparse formats like in *crankseg_2* and *Bump_2911*. Other than the performance loss introduced by a misclassification, there is a performance loss under the sparse format that has been selected for each matrix due to the time overhead discussed previously in Section 7.5. This performance loss does not exceed 3% on both GPUs even for the matrices with a relatively small number of iterations like *poisson192* and *poisson256*. An important performance improvement is also registered in comparison with CSR5 and merge-based CSR for most of the considered matrices.

Table 11. PCG Total Execution Time for BestSF and the Performance Gain Over Using its Constituent Sparse Formats (COO, CSR, BCSR, DIA, ELL, and HYB), and Also CSR5 and Merge-Based CSR (Noted MERGE)

Maxwell TITAN X										Pascal GTX 1080																	
Matrix	#iter	$f^{(r)}$	BestSF			PGO (%)					CSR5	MERGE	$f^{(r)}$	$f^{(p)}$	(s)	BestSF					PGO (%)					CSR5	MERGE
			$f^{(p)}$	(s)	COO	CSR	DIA	ELL	HYB	BCSR						COO	CSR	DIA	ELL	HYB	BCSR						
<i>nasasrb</i>	16,365	BCSR	BCSR	5.72	46.5	20.0	-	53.4	22.0	-0.1	14.0	20.6	BCSR	BCSR	4.51	33.5	10.9	-	53.1	10.9	-0.1	13.5	25.2				
<i>smt</i>	3,364	CSR	CSR	1.88	41.1	-0.5	-	39.9	20.8	16.6	6.8	7.3	CSR	CSR	1.42	31.3	-0.5	-	43.4	18.8	22.7	12.4	20.4				
<i>ecology2</i>	5,567	DIA	DIA	6.39	29.7	6.0	-0.3	1.9	2.6	10.9	6.4	5.3	DIA	DIA	6.44	34.8	5.8	-0.2	2.4	2.2	11.0	0.1	4.2				
<i>G3_circuit</i>	2,727	ELL	CSR	5.44	27.4	-0.9	-	-1.5	-0.1	26.0	-2.0	3.1	ELL	CSR	5.40	37.7	-0.24	-	-1.4	0.33	17.4	-7.1	0.7				
<i>crankseg_2</i>	787	CSR	CSR	1.73	41.6	-1.8	-	90.5	38.4	-0.7	6.1	9.4	CSR	CSR	1.35	30.3	-1.8	-	90.9	36.5	7.5	7.9	5.0				
<i>F1</i>	3,665	BCSR	BCSR	5.93	63.9	37.3	-	77.3	51.2	-1.2	25.4	18.2	BCSR	BCSR	7.46	59.8	37.1	-	77.2	39.9	-0.7	28.6	8.5				
<i>Fault_639</i>	7,146	BCSR	BCSR	19.17	42.9	11.3	-	63.6	12.4	-0.5	-10.5	-9.3	BCSR	BCSR	18.99	53.7	12.5	-	67.9	9.6	-0.6	-16.6	-11.9				
<i>inline_1</i>	35,764	BCSR	BCSR	66.24	66.5	34.5	-	83.9	44.8	-0.1	26.6	25.0	BCSR	BCSR	76.28	67.0	35.0	-	35.5	35.5	-0.1	17.6	16.5				
<i>boneS10</i>	35,815	BCSR	BCSR	96.90	67.1	37.7	-	31.3	33.2	-0.1	24.7	27.5	BCSR	BCSR	111.82	69.5	40.7	-	37.7	38.1	-0.1	17.4	19.2				
<i>Hook_1498</i>	5,236	BCSR	BCSR	22.63	55.5	36.2	-	42.9	29.5	-0.7	6.9	8.8	BCSR	BCSR	19.21	70.8	52.3	-	60.1	47.7	-0.7	14.1	14.8				
<i>audikw_1</i>	6,819	BCSR	BCSR	25.20	68.6	38.5	-	72.9	49.3	-0.6	22.8	25.5	BCSR	BCSR	30.75	70.0	45.4	-	74.9	48.5	-0.5	15.4	14.8				
<i>Bump_2911</i>	7,551	HYB	HYB	76.26	49.3	25.4	-	54.4	-0.3	11.4	-18.2	3.1	HYB	HYB	89.67	54.1	38.3	-	-0.3	-0.4	-0.4	-19.1	31.8				
<i>poisson192</i>	393	DIA	DIA	5.05	35.5	18.6	-2.5	19.2	19.1	12.2	13.2	21.7	DIA	DIA	4.98	40.9	15.7	-2.5	16.3	17.0	8.9	15.1	22.9				
<i>poisson256</i>	527	DIA	DIA	13.82	38.9	26.8	-2.1	17.9	17.7	13.6	9.5	2.9	DIA	DIA	13.90	43.8	33.7	-2.2	15.7	16.2	10.5	10.4	9.1				

$f^{(r)}$ represents the real best performing sparse format for each input matrix, and $f^{(p)}$ represents the predicted sparse format by BestSF.

Table 12 shows the GPU total energy consumption for PCG-BestSF with our 14 testing sparse matrices. Note that the energy consumption reported in Table 12 does not include the energy consumed by the CPU (needed for controlling the GPU). We also reported the energy gain ($\text{EGO}(f_i)$) of using BestSF over using its constituent sparse formats (COO, CSR, BCSR, DIA, ELL, and HYB) and also CSR5 and merge-based CSR. Negative values of $\text{EGO}(f_i)$ indicate an energy loss under using the sparse format f_i . $\text{EGO}(f_i)$ is reported 0 when the sparse format selected by BestSF ($f^{(p)}$) is also the most energy efficient sparse format ($f^{(e)}$). We observe that BestSF achieved an important energy efficiency gain over using the popular CSR format ($\text{EGO}(\text{CSR})$) for matrices like *F1_639* and *Bump_2911*. We can make similar observations concerning the other sparse formats. On both GPUs, we have only one matrix for which the predicted sparse format by BestSF is different from the most energy efficient format (*G3_circuit* with 10.6% and 15.7% of energy efficiency loss, respectively).

All the previous observations prove the practical effectiveness of BestSF, especially for unstructured sparse matrices for which the winner-takes-all approach to the format selection problem leads to an important overall performance and energy efficiency loss.

9 RELATED WORK

Given the importance of the SpMV kernel for a wide variety of application domains and the emerging of GPU computing, many GPU-based implementations of this kernel were proposed in the literature. Bell et al. proposed in [6] the implementation of several popular sparse formats (COO, CSR, ELL, DIA, etc.). These implementations are available in the open source CUSP library [18] used in our work. Many other sparse formats were proposed like some blocked formats that take advantage of the existence of many dense blocks in the structure of the sparse matrix (BCCOO [59], BCSR and BELLPACK [14], SELLPACK [38]). Also, many other optimization techniques for enhancing the reuse at the cache and the register level were proposed in the literature like in [42, 58]. A recent survey about different sparse formats for the SpMV kernel on GPGPUs can be found in [21]. Since the performance of the SpMV kernel depends on both the sparsity structure of the input matrix and the hardware characteristics, many recent techniques tried to optimize the overall performance by automatically tuning, partitioning, and combining between several sparse formats [14, 23, 24, 50]. Finally, we point out CSR5 [35] and merge-based CSR [36] that have been designed to have consistently high performance on a wide range of sparsity patterns (structured and unstructured matrices).

Less work has been done to study the energy efficiency of the SpMV kernel on GPU. In [39], the authors studied the performance per Watt of three different kernels, including the SpMV kernel, on two platforms: Intel Sandy Bridge CPU and NVIDIA Fermi GPU. They showed that in terms of GFLOPS/W, the SpMV kernel was better in the Intel Sandy Bridge than in the NVIDIA GPU, but they only considered the CSR format and presented the results with using only one type of sparse matrices (R-MAT). In [3], the authors unveiled some energy efficiency and performance frontiers for sparse computations on GPU-based supercomputers. LOBPCG (Locally Optimal Block Preconditioned Conjugate Gradient) was chosen as a benchmark as it combines between sparse and dense linear algebra operations including the SpMV kernel. We can find in the recent literature some studies concerning other scientific kernels like in [15]. Also, we can find in the work of Mitall et al. a recent survey about the methods of analyzing and improving the GPU energy efficiency [37].

Since the performance and the energy efficiency of different sparse formats depend both on the sparsity of the input matrix and on the hardware specifications, the question of automatically selecting the most suitable sparse format has been discussed in several recent studies. A similar performance of analytical models to the ones developed previously in [14, 23–25, 29] can be used to

Table 12. PCG Total GPU Energy Consumption for BestSF and the Energy Efficiency Gain Over Using Its Constituent Sparse Formats (COO, CSR, BCSR, DIA, ELL, and HYB), and Also CSR5 and Merge-Based CSR (Noted MERGE)

Maxwell TITAN X											Pascal GTX 1080												
Matrix	#iter	$f^{(e)}$	$f^{(p)}$	BestSF		EGO (%)						BestSF		EGO (%)									
				(J)		COO	CSR	DIA	ELL	HYB	BCSR	CSR5	MERGE	$f^{(e)}$	$f^{(p)}$	(J)		COO	CSR	DIA	ELL	HYB	BCSR
<i>nasasrb</i>	16,365	BCSR	BCSR	516.43	69.4	56.9	-	68.8	46.7	0.0	25.7	12.8	BCSR	BCSR	300.53	66.0	53.8	-	66.6	37.4	0.0	17.0	22.2
<i>smt</i>	3,364	CSR	CSR	256.88	45.8	0.0	-	26.6	10.1	8.9	-14.3	-21.2	CSR	CSR	143.29	42.0	0.0	-	28.1	0.7	7.3	-11.6	-19.4
<i>ecology2</i>	5,567	DIA	DIA	1038.24	34.7	17.4	0.0	0.3	0.4	11.8	-9.3	-25.2	DIA	DIA	627.98	41.0	19.2	0.0	3.6	3.7	6.0	1.0	-21.7
<i>G3_circuit</i>	2,727	ELL	CSR	825.45	36.5	0.0	-	-10.6	2.1	24.9	1.6	-7.8	ELL	CSR	628.40	30.7	0.0	-	-15.7	-14.2	-11.8	-12.7	-16.4
<i>crankseg_2</i>	787	CSR	CSR	187.80	54.0	0.0	-	78.5	21.8	14.3	20.9	11.4	CSR	CSR	120.09	46.9	0.0	-	80.3	15.1	4.8	12.3	20.1
<i>F1</i>	3,665	BCSR	BCSR	821.61	75.1	57.6	-	74.2	58.3	0.0	48.2	43.8	BCSR	BCSR	396.33	83.4	73.5	-	85.2	70.5	0.0	67.4	64.8
<i>Fault_639</i>	7,146	BCSR	BCSR	3,294.78	53.8	33.5	-	56.6	26.9	0.0	-2.8	-3.5	BCSR	BCSR	2,146.80	59.6	34.5	-	61.3	13.7	0.0	-6.0	-2.3
<i>inline_1</i>	35,764	BCSR	BCSR	10,245.46	76.4	57.1	-	85.2	57.4	0.0	46.5	46.2	BCSR	BCSR	5,901.33	81.7	66.5	-	-	62.0	0.0	53.8	58.2
<i>boneS10</i>	35,815	BCSR	BCSR	15,725.68	76.0	57.8	-	47.0	48.2	0.0	43.4	46.9	BCSR	BCSR	7,655.04	84.7	72.4	-	65.2	65.4	0.0	59.7	62.8
<i>Hook_1498</i>	5,236	BCSR	BCSR	4,222.17	61.9	43.1	-	36.4	27.2	0.0	21.0	25.5	BCSR	BCSR	1,598.35	81.1	72.2	-	68.1	61.4	0.0	48.6	54.6
<i>audikw_1</i>	6,819	BCSR	BCSR	3,828.21	78.2	57.5	-	73.3	60.0	0.0	43.8	51.8	BCSR	BCSR	2,961.71	78.1	60.7	-	74.6	57.7	0.0	36.6	47.5
<i>Bump_2911</i>	7,551	HYB	HYB	14,285.86	56.1	37.5	-	46.3	0.0	16.7	-6.7	22.2	HYB	HYB	1,0219.02	59.6	46.6	-	-	0.0	3.0	-22.9	49.6
<i>poisson192</i>	393	DIA	DIA	517.29	51.4	39.8	0.0	11.5	10.8	36.8	28.6	59.2	DIA	DIA	362.14	54.3	38.6	0.0	10.1	10.0	16.3	26.7	56.4
<i>poisson256</i>	527	DIA	DIA	1,603.72	51.2	43.5	0.0	10.9	10.7	45.1	18.3	42.6	DIA	DIA	1,118.67	53.3	46.3	0.0	10.0	10.2	22.3	17.1	42.1

$f^{(e)}$ represents the real most energy efficient sparse format for each input matrix, and $f^{(p)}$ represents the predicted sparse format by BestSF.

point out the best performing sparse format for a given input sparse matrix. However, the rising complexity of the GPU architecture and the wide range of different sparsity patterns make the analytical approach very challenging in practice, and if successfully used for a given hardware architecture, porting it to other architectures may be very difficult.

On the other hand, using a machine learning approach to build an algorithm or data structure selection system seems to be more attractive [9, 30]. For example, in [49], a binary SVM classifier was used to select the best algorithm from the two libraries MKL and CARMA for the dense matrix-matrix multiplication kernel. The decision tree classifier used in [47] demonstrated the use of machine learning techniques to select the best performing sparse representation for the SpMV on GPU. In their work, the authors proposed three different feature sets. The simple one consists of the number of nonzero elements and its distribution, but the best performance was obtained by using the other two advanced feature sets that consider the number and the size of blocks of nonzero elements of the matrices which requires more time overhead to be computed. Also, the decision trees may not be the most efficient classifier, especially if the decision boundaries between different classes are highly nonlinear which is expected to be the case for the SpMV on GPU. In our previous work [8], we demonstrated that using some other very simple to compute sparsity features with one-vs.-one multiclass SVM classifier enables one to reach better performance. SVMs were also used in Nitro [40, 41] to select the best variant of different kernels (CG solvers, BFS, Histogram, Sort, and SpMV).

To the best of our knowledge, all the existing work for solving the problem of sparse format selection on GPU using a machine learning approach did not thoroughly discuss the overhead of selecting the best performing sparse format and also did not study the impact of such selection on the energy efficiency of the SpMV kernel on GPU. Also, the difference between the performance of different sparse formats for a given matrix was not considered in the learning phase which can negatively impact the quality of the selection (the existence of misclassifications with an important performance loss). This article comes as a continuation and extension of our previous work [8]. We showed how using a weighted pairwise classification instead of the standard multiclass classification allows considering the performance difference between different sparse formats in the learning phase. We also studied the time overhead of the resulting system that we called BestSF (Best Sparse Format) and studied the impact of using it on the overall energy efficiency of the SpMV kernel on GPU. Finally, we measured the gain of using BestSF as a building block in a GPU-based Preconditioned Conjugate Gradient iterative solver.

10 CONCLUSION

In this article, we introduced BestSF (Best Sparse Format), a learning-based sparse meta-format for the SpMV kernel on GPU. Given an input sparse matrix, BestSF automatically selects the best performing sparse format from COO, CSR, BCSR, DIA, ELL, and HYB. It consists of two stages: an offline training stage and an online decision-making stage. In the first one, the pairwise models between every pair of sparse formats are trained using Weighted Support Vector Machines (WSVMs). In the second stage, the best performing sparse format for a given input matrix is predicted using the previously trained pairwise models. Our experimental results on two different NVIDIA GPU architectures, using a wide range of real-world sparse matrices, showed that BestSF achieved a noticeable performance (GFLOPS) and energy efficiency (MFLOPS/W) improvement over using a single sparse format for all the inputs. However, the overhead introduced by the online decision-making phase makes BestSF more suitable for applications where the SpMV kernel is executed many times with the same input matrix like in iterative methods for solving linear systems and eigenvalue problems.

SUPPLEMENTARY MATERIALS

The list of the 1,000 sparse matrices from the SuiteSparse matrix collection [20] used in our dataset is included as a supplementary file to this manuscript.

REFERENCES

- [1] Hartwig Anzt, Marc Baboulin, Jack Dongarra, Yvan Fournier, Frank Hulsemann, Amal Khabou, and Yushan Wang. 2016. Accelerating the conjugate gradient algorithm with GPU in CFD simulations. VECPAR.
- [2] Hartwig Anzt, Mark Gates, Jack Dongarra, Moritz Kreutzer, Gerhard Wellein, and Martin Köhler. 2017. Preconditioned Krylov solvers on GPUs. *Parallel Comput.* (2017).
- [3] Hartwig Anzt, Stanimire Tomov, and Jack Dongarra. 2015. Energy efficiency and performance frontiers for sparse computations on GPU supercomputers. In *Proceedings of the 6th International Workshop on Programming Models and Applications for Multicores and Manycores*. ACM, 1–10.
- [4] Arash Ashari, Naser Sedaghati, John Eisenlohr, Srinivasan Parthasarathy, and P. Sadayappan. 2014. Fast sparse matrix-vector multiplication on GPUs for graph applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 781–792.
- [5] Arash Ashari, Naser Sedaghati, John Eisenlohr, and P. Sadayappan. 2014. An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on GPUs. In *Proceedings of the 28th ACM International Conference on Supercomputing*. ACM, 273–282.
- [6] Nathan Bell and Michael Garland. 2008. *Efficient Sparse Matrix-vector Multiplication on CUDA*. Nvidia Technical Report NVR-2008-004, Nvidia Corporation.
- [7] Nathan Bell and Jared Hoberock. 2011. Thrust: A productivity-oriented library for CUDA. Retrieved June 2016 from <https://thrust.github.io/>.
- [8] A. Benatia, W. Ji, Y. Wang, and F. Shi. 2016. Sparse matrix format selection with multiclass SVM for SpMV on GPU. In *Proceedings of the 45th International Conference on Parallel Processing (ICPP'16)*. IEEE, 496–505.
- [9] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchet, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and others. 2016. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence* 237 (2016), 41–58.
- [10] Martin Burtcher, Ivan Zecena, and Ziliang Zong. 2014. Measuring GPU power with the K20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*. ACM, 28.
- [11] Ali Cevahir, Akira Nukada, and Satoshi Matsuoka. 2009. Fast conjugate gradients with multiple GPUs. In *International Conference on Computational Science*. Springer, 893–903.
- [12] Ali Cevahir, Akira Nukada, and Satoshi Matsuoka. 2010. High performance conjugate gradient solver on multi-GPU clusters using hypergraph partitioning. *Computer Science-Research and Development* 25, 1 (2010), 83–91.
- [13] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (2011), 27. <http://www.csie.ntu.edu.tw/~EIJcjl/lin/libsvm>.
- [14] Jee W. Choi, Amik Singh, and Richard W. Vuduc. 2010. Model-driven autotuning of sparse matrix-vector multiply on GPUs. In *ACM SIGPLAN Notices* 45 (2010), 115–126.
- [15] J. Coplin and M. Burtcher. 2014. Power characteristics of irregular GPGPU programs. In *Workshop on Green Programming, Computing, and Data Processing (GPCDP'14)*.
- [16] Jared Coplin and Martin Burtcher. 2016. Energy, power, and performance characterization of GPGPU benchmark programs. In *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops*. IEEE, 1190–1199.
- [17] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning* 20 (1995), 273–297.
- [18] Steven Dalton, Nathan Bell, Luke Olson, and Michael Garland. 2014. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. Retrieved September 2017 from <http://cusplibrary.github.io/>.
- [19] John D. Davis and Eric S. Chung. 2012. *SpMV: A Memory-Bound Application on the GPU Stuck Between a Rock and a Hard Place*. Technical Report, Microsoft Research Silicon Valley.
- [20] Timothy A. Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011). Retrieved September 2015 from <https://www.cise.ufl.edu/research/sparse/matrices/>.
- [21] Salvatore Filippone, Valeria Cardellini, Davide Barbieri, and Alessandro Fanfarillo. 2017. Sparse matrix-vector multiplication on GPGPUs. *ACM Transactions on Mathematical Software (TOMS)* 43, 4 (2017), 30.
- [22] Serban Georgescu and Hiroshi Okuda. 2010. Conjugate gradients on multiple GPUs. *International Journal for Numerical Methods in Fluids* 64, 10–12 (2010), 1254–1273.

- [23] Ping Guo, He Huang, Qichang Chen, Liqiang Wang, En-Jui Lee, and Po Chen. 2011. A model-driven partitioning and auto-tuning integrated framework for sparse matrix-vector multiplication on GPUs. In *Proceedings of the TeraGrid Conference: Extreme Digital Discovery*.
- [24] Ping Guo, Liqiang Wang, and Po Chen. 2014. A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs. *IEEE Transactions on Parallel and Distributed Systems* 25 (2014), 1112–1123.
- [25] Sunpyo Hong and Hyesoon Kim. 2009. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM, 152–163.
- [26] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2003. A practical guide to support vector classification. Retrieved October 2015 from <https://www.cs.sfu.ca/people/Faculty/teaching/726/spring11/svmguide.pdf>.
- [27] Chih-Wei Hsu and Chih-Jen Lin. 2002. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks* 13, 2 (2002), 415–425.
- [28] Ron Kohavi and George H. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 97, 1–2 (1997), 273–324.
- [29] Kishore Kothapalli, Rishabh Mukherjee, M. Suhail Rehman, Suryakant Patidar, P. J. Narayanan, and Kannan Srinathan. 2009. A performance prediction model for the CUDA GPGPU platform. In *Proceedings of the International Conference on High Performance Computing (HiPC'09)*. IEE, 463–472.
- [30] Lars Kotthoff, Ian P. Gent, and Ian Miguel. 2011. A preliminary evaluation of machine learning in algorithm selection for search problems. In *Proceedings of the 4th Annual Symposium on Combinatorial Search*.
- [31] Jakub Kurzak, David A. Bader, and Jack Dongarra. 2010. *Scientific Computing with Multicore and Accelerators*. CRC Press.
- [32] Daniel Langr and Pavel Tvrdik. 2016. Evaluation criteria for sparse matrix storage formats. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (2016), 428–440.
- [33] Ruipeng Li and Yousef Saad. 2013. GPU-accelerated preconditioned iterative linear solvers. *The Journal of Supercomputing* 63, 2 (2013), 443–466.
- [34] Shaozhong Lin and Zhiqiang Xie. 2017. A Jacobi_PCG solver for sparse linear systems on multi-GPU cluster. *The Journal of Supercomputing* 73, 1 (2017), 433–454.
- [35] Weifeng Liu and Brian Vinter. 2015. CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication. In *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 339–350.
- [36] Duane Merrill and Michael Garland. 2016. Merge-based parallel sparse matrix-vector multiplication. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 58.
- [37] Sparsh Mittal and Jeffrey S. Vetter. 2015. A survey of methods for analyzing and improving GPU energy efficiency. *ACM Computing Surveys (CSUR)* 47, 2 (2015), 19.
- [38] Alexander Monakov, Anton Lokhmotov, and Arutyun Avetisyan. 2010. Automatically tuning sparse matrix-vector multiplication for GPU architectures. In *International Conference on High-Performance Embedded Architectures and Compilers*. Springer, 111–125.
- [39] Julia S. Mullen, Michael M. Wolf, and Anna Klein. 2013. Pakck: Performance and power analysis of key computational kernels on CPUs and GPUs. In *IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–6.
- [40] Saurav Muralidharan, Amit Roy, Mary Hall, Michael Garland, and Piyush Rai. 2016. Architecture-adaptive code variant tuning. *ACM SIGPLAN Notices* 51, 4 (2016), 325–338.
- [41] Saurav Muralidharan, Manu Shantharam, Mary Hall, Michael Garland, and Bryan Catanzaro. 2014. Nitro: A framework for adaptive code variant tuning. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 501–512.
- [42] Yusuke Nagasaka, Akira Nukada, and Satoshi Matsuoka. 2014. Cache-aware sparse matrix formats for Kepler GPU. In *Proceedings of the 20th IEEE International Conference on Parallel and Distributed*. IEEE.
- [43] NVIDIA. NVIDIA Management Library. Retrieved September 2017 from <https://developer.nvidia.com/nvidia-management-library-nvml>.
- [44] NVIDIA. 2017. CuBLAS: Dense Linear Algebra on GPUs. Retrieved September 2017 from <https://developer.nvidia.com/cublas>.
- [45] NVIDIA. 2017. cuSPARSE : Sparse Linear Algebra on GPUs. Retrieved September 2017 from <http://docs.nvidia.com/cuda/cusparse>.
- [46] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems*. SIAM.
- [47] Naser Sedaghati, Te Mu, Louis-Noël Pouchet, Srinivasan Parthasarathy, and P. Sadayappan. 2015. Automatic selection of sparse matrix representation on GPUs. In *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 99–108.
- [48] Jonathan Richard Shewchuk and others. 1994. An introduction to the conjugate gradient method without the agonizing pain.

- [49] Omer Spillinger, David Eliahu, Armando Fox, and James Demmel. 2015. *Matrix Multiplication Algorithm Selection with Support Vector Machines*. Technical Report. Retrieved October 2015 from <http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-29.html>.
- [50] Bor-Yiing Su and Kurt Keutzer. 2012. clSpMV: A cross-platform OpenCL SpMV framework on GPUs. In *Proceedings of the 26th ACM International Conference on Supercomputing*. ACM, 353–364.
- [51] Balaji Subramaniam, Winston Saunders, Tom Scogland, and Wu-chun Feng. 2013. Trends in energy-efficient computing: A perspective from the Green500. In *Proceedings of the 2013 International Green Computing Conference (IGCC'13)*. IEEE, 1–8.
- [52] Mícheál Verschoor and Andrei C. Jalba. 2012. Analysis and performance estimation of the conjugate gradient method on multiple GPUs. *Parallel Computing* 38, 10–11 (2012), 552–575.
- [53] Rich Vuduc, James W. Demmel, Katherine A. Yelick, Shoaib Kamil, Rajesh Nishtala, and Benjamin Lee. 2002. Performance optimizations and bounds for sparse matrix-vector multiply. In *Proceedings of the ACM/IEEE 2002 Conference on Supercomputing*. IEEE, 26–26.
- [54] Richard Wilson Vuduc and James W. Demmel. 2003. *Automatic Performance Tuning of Sparse Matrix Kernels*. Vol. 1. University of California, Berkeley.
- [55] Dong Hyuk Woo and Hsien-Hsin S. Lee. 2008. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer* 41, 12 (2008).
- [56] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32 (2008), 565–606.
- [57] Lin Xu, Frank Hutter, Jonathan Shen, Holger H. Hoos, and Kevin Leyton-Brown. 2012. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge, 2012*, 57–58.
- [58] Weizhi Xu, Hao Zhang, Shuai Jiao, Da Wang, Fenglong Song, and Zhiyong Liu. 2012. Optimizing sparse matrix vector multiplication using cache blocking method on Fermi GPU. In *Proceedings of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel and Distributed Computing (SNPD'12)*. IEEE, 231–235.
- [59] Shengen Yan, Chao Li, Yunquan Zhang, and Huiyang Zhou. 2014. yaSpMV: Yet another SpMV framework on GPUs. *ACM SIGPLAN Notices* 49 (2014), 107–118.
- [60] Xulei Yang, Qing Song, and Yue Wang. 2007. A weighted support vector machine for data classification. *International Journal of Pattern Recognition and Artificial Intelligence* 21, 5 (2007), 961–976.

Received November 2017; revised April 2018; accepted May 2018