Trajectory-based Metaheuristics for Improving Sparse Matrix Storage

Manuel Freire*, Raúl Marichal†, Ernesto Dufrechou‡, Pablo Ezzatti§ and Martín Pedemonte¶
Instituto de Computación, INCO
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
*mfreire@fing.edu.uy, †rmarichal@fing.edu.uy, ‡edufrechou@fing.edu.uy, §pezzatti@fing.edu.uy, ¶mpedemon@fing.edu.uy

***Abstract*—Kernels in linear algebra are memory-bounded routines and their performance is dependent on the sparsity pattern of the matrix operands. Since memory is many times slower than arithmetic operations these kernels tend to exploit small fractions of the peak performance of modern architectures such as GPUs. In this sense, the improvement of the storage of the matrices to reduce the memory accesses is a main line of work.**

The problem of finding an exact solution for the best permutation of a matrix is computationally prohibitive thus it is interesting to explore metaheuristic approaches. Previous work found good results with evolutionary algorithms but with high execution time. In this context, it is compelling to explore trajectory-based metaheuristics as a way to reduce execution time since they require evaluating only one solution per iteration. In this work, we continue the efforts and present heuristics based on VNS and ILS which outperform the evolutionary algorithm in the majority of the instances evaluating half of the solutions.

***Index Terms*—sparse matrices, storage, metaheuristics, Iterated Local Search, Variable Neighborhood Search**

I. Introduction

Sparse operations arise in a myriad of scientific and engineering problems. For example, the sparse matrix-vector product (SpMV) and the sparse triangular solver (SpTRSV) are fundamental building blocks in methods for solving partial differential equations that model physical phenomena. More recently, applications to data science and machine learning brought attention to other sparse operations. Some examples are the sparse general matrix multiplication (SpGEMM), which arises in problems such as triangle counting [1], breadth-first search with multiple origins [2], or the sparse matrix-dense matrix product kernel and sampled sparse matrix-dense matrix product [3], with applications to sparse neural networks.

Sparse kernels are generally memory bound, meaning most of the computational cost lies in memory access. For example, the SpMV performs two arithmetic operations per nonzero element (a product and an addition) while it needs to read the matrix and vector coefficients involved and write the dot product of each row of the matrix by the output vector to the final result. Additionally, sparse algebra kernels have other problems, like low data locality (the elements of the input vector are accessed in the order dictated by the sparse matrix nonzero pattern) or data indirection (access to indexing data is required to retrieve the position of each nonzero in the matrix). This leads to an extremely low computational intensity compared to operations such as dense matrix multiplication. As memory access is many times slower and more power-consuming than arithmetic operations, sparse kernels tend to exploit only small fractions of the peak performance of modern hardware architectures [4].

A sustained line of research to speed up sparse operations consists of proposing storage formats of the sparse matrix that improve memory performance. Some formats aim to be compact and general-purpose, such as the Coordinate format (COO), the Compressed Sparse Row (CSR) [5], or their blocked variants BCOO and BCSR. Others are designed to accelerate a specific matrix operation in parallel hardware, such as the ELLPACK [6], or SELL-P [7] formats, which sacrifice compactness to achieve more aligned memory access or reduce the indexing overhead. Another approach is to leverage known properties of the nonzero pattern, such as the Diagonal format (DIA), intended for matrices with their nonzeros close to the main diagonal. There are also hybrid formats that combine the previous approaches.

The advances in sparse neural network applications [8] and the emergence of new hardware architectures and paradigms to accelerate machine learning computations [9] impose the need to revisit the problem of efficiently storing sparse matrices [10]–[13]. Particularly, although sparsity in neural networks reduces the overall computations and memory accesses, the irregularity of sparse representations prevents the full exploitation of new hardware accelerators.

Freire et al. worked on identifying the benefit of using compression strategies and reordering techniques to improve the storage of the matrix indices. Later, the authors extended their work using a local search in [14], [15]. Another line of work is to find permutations of the matrices rows and columns that allow their storage in compact formats such as DIA or ELL. Marichal et al. proposed in [16] an Evolutionary Algorithm and studied diverse fitness functions depending on the formats they targeted. They report some gains regarding the storage of some matrices, but the high execution times of their solution can turn the approach impractical. It is important to note that neither of these approaches produces information

979-8-3503-4807-1/23/$31.00 ©2023 IEEE

loss since the permutation is an invertible function.

In this work, we continue the study of permutations of the sparse matrices that allow storing them in efficient formats. In particular, we aim to transform the nonzero pattern of the matrix so that most nonzeros concentrate on a few central diagonals and can be stored using a dense matrix, while the rest of the matrix is stored in a general-purpose format such as COO. We propose two new trajectory-based metaheuristic approaches and compare the results against RCM [17] (the most known heuristic for bandwidth reduction) and the evolutionary algorithm of [16]. We focus on the number of non-zeros in the two percent of central diagonals as the target since it showed the most promising results in [16].

II. Background

This section briefly introduces basic concepts such as sparse matrix storage formats and the Reverse Cuthill-Mckee algorithm for reordering.

A. Sparse matrices storage formats

Sparse matrix storage formats are strategies to avoid storing the zero elements of the matrix explicitly. In general, these formats tend to explicitly store the non-zero values plus secondary structures that hold information to recover the column and row indices of those values.

The most straightforward idea is to store only the non-zeros with their respective row and column indices. This format is known as Coordinate (COO) [5], and its main problem is that each floating-point value requires eight extra bytes for the indices. If the elements (and the vectors of indices) are ordered by row or column, one index can be compressed by storing where the row (or column) starts, thus reducing the extra bytes needed to 4 per element plus an additional 4 bytes per row. These formats are called Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC), respectively [18].

While the general idea is not to store the zero values, some formats allow the storage of zeros to benefit memory access. One example of this idea is ELLEPACK (ELL) [19], which stores the sparse matrix as two dense matrices, one for the values and one for the column indices, of size $n * k$ where n is the number of rows and k is the maximum number of non-zeros in one row. The rows with less than k non-zeros are padded with zeros.

Finally, some formats exploit special characteristics of the matrices. For example, if the matrix has the non-zero elements clustered in small submatrices, it is possible to store one pair of indices for each cluster and calculate the inner indices, which leads to blocked formats [20]–[25]. If the matrix has a few dense diagonals, it can be stored as a rectangular dense matrix [26]. Moreover, if a matrix has a large group of nonzeros that follow a regular pattern, it may pay off to store this group in a regular format, such as ELL format, while the rest of the non-zeros are stored separately in COO. An example of this is the Hybrid (HYB) format.

B. Reverse Cuthill-Mckee (RCM)

The RCM algorithm is the most widely used method for reducing the bandwidth [27]. This heuristic is an improvement from the original Cuthill-Mckee (CM) algorithm [28]. The RCM does the same as the CM algorithm but reverses the resulting order. The empirical results showed that while it does not improve the bandwidth obtained by CM, it gives a lower profile.

The CM method first performs a Bread First Search (BFS) search in the graph produced by interpreting the matrix as an adjacency matrix. Once BFS has generated a level structure, it labels the nodes from level 0 upwards and traverses inside the level from low to high degree. Ties are broken arbitrarily. The node selected for starting the BFS impacts the result, and choosing it is an important part of the algorithm. The generally good idea is to start from pseudo-peripheral nodes.

III. Trajectory-based algorithms for improving the sparse matrix storage

Metaheuristics are approximate and stochastic algorithms that can be abstracted as high-level frameworks or general schemes of heuristics designed to address a wide range of optimization problems [29]–[31]. In general, they adapt to the particularities of each problem using problem-specific knowledge or heuristic information, which helps them to find high-quality solutions efficiently. Metaheuristics balance a proper exploration of the search space (diversification) with the exploitation of information from the best solutions found (intensification) [29]. Several metaheuristic algorithms have been proposed in the literature, like Evolutionary Algorithms (EAs), Ant Colony Optimization (ACO), Tabu Search (TS), Variable Neighborhood Search (VNS), and Iterated Local Search (ILS).

Metaheuristics are usually classified according to the number of solutions used at the same time by the algorithm [29]. Algorithms that work with a set of solutions, like EAs and ACO, are known as population-based metaheuristics. For this reason, these algorithms provide an intrinsic mechanism for exploring the search space. On the other hand, algorithms that consider a single solution (i.e., a single point of the search space) in each iteration of the algorithm are called trajectory-based metaheuristics. These algorithms start from an initial point of the search space and update the position of the candidate solution by exploring the solution's neighborhood (usually through local search operators). Thus, the candidate solution describes a trajectory in the search space during the search process. TS, VNS, and ILS are trajectory-based algorithms.

Considering the limitations of the approach proposed with Evolutionary Algorithms [16], we tackle the problem of efficiently storing sparse matrices with single solution-based methods in this work. On the one hand, our goal is to propose approaches that can explore the neighborhood of the solution (which showed to be promising in the previous work) and, on the other hand, to achieve reasonable execution times.

A. Iterated Local Search (ILS)

Iterated Local Search (ILS) is one of the simplest yet most powerful trajectory-based metaheuristics [29], [32]. ILS alternates stages of intensification and diversification. In the intensification stage, ILS iteratively applies a local search method until it finds a local optimum. Once the algorithm gets trapped in the local optima, it uses a perturbation as a diversification mechanism to escape from it and restart the local search. Algorithm 1 shows the pseudocode of the ILS algorithm.

```
1  s = generateInitialSolution()
2  while not stopCondition() do
3  |   repeat
4  |   |   s = localSearch(s)
5  |   until no improvement is possible;
6  |   s = Perturbation(s)
7  end
```

Algorithm 1: Pseudocode of Iterated Local Search.

B. Variable Neighborhood Search (VNS)

Variable Neighborhood Search (VNS) is based on the exploration of a set of neighborhoods $N_1(s), N_2(s), \ldots, N_{max}(s)$ [31], [33]. The algorithm starts from an initial solution s and computes a local search on the first neighborhood. If a better solution cannot be found, the neighborhood structure is changed, the search continues in the next neighborhood, and so on. Otherwise, if a better solution s' is found in the current neighborhood, the current solution s is replaced by s', and the local search is restarted in the first neighborhood. The neighborhoods used in VNS are usually nested: $N_1(s) \subset N_2(s) \subset \ldots \subset N_{max}(s)$. Algorithm 2 shows the pseudocode of the VNS algorithm.

```
1  s = generateInitialSolution()
2  while not stopCondition() do
3  |   k = 1
4  |   while k ≤ max do
5  |   |   s^neigh = s
   |   |   // Local search with p tries
6  |   |   for i = 1 to p do
7  |   |   |   Generate neighborhood solution s' ∈ N_k(s)
8  |   |   |   if f(s') > f(s) then
9  |   |   |   |   s^neigh = s'
10 |   |   |   end
11 |   |   end
12 |   |   if f(s^neigh) > f(s) then
13 |   |   |   s = s^neigh
14 |   |   |   k = 1
15 |   |   else
16 |   |   |   k = k + 1
17 |   |   end
18 |   end
19 end
```

Algorithm 2: Pseudocode of VNS.

C. Our proposals

Our goal is to concentrate most nonzeros on a few central diagonals of the sparse matrices. This allows storing these few diagonals in a dense matrix and the rest of the nonzeros elements using a general-purpose sparse format. For this reason, we consider maximizing the number of nonzeros in the two percent of central diagonals as our objective function. Solutions with a larger number of nonzeros in the two percent of the diagonals are preferred since they minimize the number of nonzeros elements that must be specially stored in the sparse format. The solutions are permutations that represent how to reorder the rows of the original matrix to generate the new one.

We designed four different approaches based on VNS and ILS metaheuristics, which we called VNS$_f$, VNS$_b$, ILS$_f$, and ILS$_b$.

In the VNS algorithms, the neighborhoods are nested and correspond to the number of rows that can be changed in the solution. For instance, $N_1(s)$ corresponds to making one change in the permutation, i.e., switching two rows, while $N_9(s)$ indicates that ten rows of the matrix exchange their positions. The implementation of the changes in the permutation, when more than two rows are involved, includes the draw of the n rows that are considered, and then the values are exchanged in order as in a "rotation" (i.e., the value of the i^{th} row goes to the position of the i^{th+1} row and so on).

Since we have noticed that RCM algorithms do not always lead to improvements over the original permutation (as the matrix is stored in the collection), we have designed an initialization mechanism that considers this. The initial solution is randomly chosen between the original permutation and the one generated by the RCM algorithm. When VNS execution reaches the maximum neighborhood ($N_9(s)$), and it cannot improve the current solution, a restart mechanism is used. This restart uses a different initial solution than the latest one (for instance, if the latest initial solution was the originally stored permutation, the new initial solution is the one generated by RCM).

We have designed two variants of VNS. The first variant, VNS$_f$, uses a first-improvement or greedy approach for neighborhood exploration. When a new solution is generated in exploring a neighborhood, if it is better than the current solution, the current solution is updated and the algorithm restarts from the first neighborhood. If the solution is worse than the current solution, a new solution is generated in the current neighborhood. When the maximum number of tries is reached in a neighborhood, the search process continues in the next neighborhood. The second variant, VNS$_b$, uses a best-improvement approach for neighborhood exploration, but since considering the full number of possible combinations of exchanges between rows is prohibitive, VNS$_b$ implements a pseudo-best-improvement strategy. Instead of choosing the first solution generated in the neighborhood that is better than the current solution, VNS$_b$ uses the maximum number

of tries to choose the best movement within the explored neighborhood.

We have also designed two variants of ILS following a similar approach. ILS_f uses a first improvement or greedy approach, while ILS_b uses a pseudo-best-improvement strategy. The `Perturbation` operator is defined according to the second mutation operator used in [16] called *flipr*. When the algorithm is stuck in a local optimum, it randomly draws two indices of the permutation and flips the values of the subvector between the indices. To be fair in our comparison of the VNS variants, we have also included a restart mechanism in ILS that allows using both the originally stored permutation and the one generated by RCM as the initial solution. When the restart is executed, the algorithm uses a different initial solution than the latest one used.

IV. Experimental Results

In this section, we describe the instances used for our experiments, the parameters setting of the algorithms, and the experimental procedure followed in this study. Then, we present and analyze the results obtained.

A. Instances used in the experiments

The instances used in this work are taken from the `SuiteSparse Matrix Collection` [34] and are publicly available. We have chosen a subset of 17 sparse matrices of the library that are presented in Table I. The features of the matrices included in the table are the dimension (since the sparse matrices used are square, only the number of rows of the matrix is included), the number of nonzero elements (*nnz*), and the density of the sparse matrix, which is the ratio of nonzeros to the total number of elements of the matrix. The subset of matrices was selected for comparison purposes with the results of [16], but the approach proposed in this paper can be applied to the whole collection. The sparse matrices selected are representative of small matrices of the collection and have a dimension between 62 and 886; and a density between 0.47 and 11.83.

TABLE I
SPARSE MATRICES SELECTED FOR THE EVALUATION.

Matrix name	Dimension	*nnz*	Density
bfwb62	62	342	8.90
dwt_503	503	6027	2.38
S10PI_n1	528	1317	0.47
Spectro_10NN	531	7422	2.63
lshp_577	577	3889	1.17
LowThrust_1	584	6133	1.80
steam2s	600	5660	1.57
662_bus	662	2474	0.56
nos6	675	3255	0.71
Trefethen_700	700	12654	2.58
dendrimer	730	63024	11.83
Si2	769	17801	3.01
lshp_778	778	5272	0.87
bfwb782	782	5982	0.98
goddardRocket_1	831	8498	1.23
young4c	841	4089	0.58
orsirr_2	886	5970	0.76

B. Experimental Settings

In our experimental evaluation, we include two VNS algorithms, VNS_f and VNS_b, and two ILS algorithms, ILS_f and ILS_b, which differ on the neighborhood exploration strategy (first-improvement or pseudo-best-improvement). In order to set an actual comparison basis, we also include the numerical results of the EA from the previous work [16] in the experimental analysis. In EA results only the best found solution was reported in the previous work.

The parameter settings of the algorithms are:

- Trials in first-improve strategy: 100.
- Neighborhood solutions evaluated in pseudo-best-improvement strategy: 1000.
- Neighborhoods for VNS: $N_1(s) \subset N_2(s) \subset \ldots \subset N_9(s)$.

To provide a fair comparison between the results obtained in this work with [16], the stopping criterion used is to have 300,000 function evaluations (while in [16] the criteria used was 600,000 function evaluations).

Since the algorithms proposed are stochastic, we use statistical tests to assess the significance of the numerical results. We use the following statistical procedure [35], [36]. First, we perform thirty independent runs for each algorithm and each instance. To analyze the results across multiple instances, we use Friedman's test as an omnibus method for comparing the median of the results. Since more than two algorithms are involved in the study, in case the results are statistically different, a pairwise comparison using Holm's post-hoc procedure is performed. The statistical tests are performed with a confidence level of 95%.

C. Experimental Results

Table II shows the median and the interquartile range of the distribution of the number of nonzero elements (*nnz*) in the two percent of diagonals for the algorithms proposed in this work. The table also includes the best-found solution of the four algorithms studied in this work, the EA from previous work, the RCM method, and the original matrix. Dark gray and light gray backgrounds indicate the best and the second-best performing algorithm considering the median of the distribution of *nnz*. Bold is used to indicate the best-found solution over all the algorithms.

The results regarding the median of *nnz* in the 2% of the diagonals show that VNS_b is the best performing algorithm in 15 out of 17 instances (in one instance it is the second best). ILS_b and VNS_f are the second-best performing algorithms and present similar results. ILS_b obtains the highest value of the median in 5 instances and the second highest value in 6 instances, while VNS_f obtains the highest and second highest value in 5 each. Finally, ILS_f is the worst-performing algorithm, being able to obtain the highest value of the median only in two scenarios where none of the algorithms improve the original matrix (`nos6` and `young4c`), and in two scenarios where all the algorithms obtain the same value (`Trefethen_700` and `orsirr_2`).

TABLE II
NUMBER OF NONZERO ELEMENTS IN THE TWO PERCENT OF DIAGONALS

Instance	Median and Interquartile Range				Best						
	ILS_f	ILS_b	VNS_f	VNS_b	ILS_f	ILS_b	VNS_f	VNS_b	Original	RCM	EA
bfwb62	148_{4}	156_{4}	$152_{3.5}$	160_{4}	154	162	160	**166**	94	86	108
dwt_503	2883_{12}	3130_{91}	2918_{10}	3148_{43}	2911	**3251**	2937	3249	2190	1919	2813
S10PI_n1	1275_{0}	1275_{0}	1281_{2}	1287_{2}	1279	1275	1285	**1293**	1223	1058	1256
Spectro_10NN	$3208_{75.5}$	$3182_{68.5}$	3269_{24}	2010_{1372}	3370	3462	3340	**3826**	208	2182	3130
lshp_577	$3309_{3.5}$	3309_{0}	3323_{4}	$3371_{17.5}$	3335	3309	3333	**3397**	2487	2680	3072
LowThrust_1	3541_{2}	$3560_{15.5}$	$3559_{3.5}$	$3568_{13.5}$	3565	**3595**	3573	3589	2377	836	2554
steam2s	3231_{0}	$3234_{6.75}$	3235_{3}	$3237_{5.5}$	3232	3258	3238	**3263**	2413	2116	2518
662_bus	1525_{28}	1708_{55}	$1559_{19.5}$	$1767_{45.5}$	1582	1820	1590	**1874**	951	966	1701
nos6	3255_{0}	3255_{0}	3255_{0}	3255_{0}	**3255**	**3255**	**3255**	**3255**	**3255**	2893	**3255**
Trefethen_700	12654_{0}	12654_{0}	12654_{0}	12654_{0}	**12654**	**12654**	**12654**	**12654**	9610	2818	9610
dendrimer	$12765_{401.5}$	$14188_{391.5}$	$12633_{79.5}$	14034_{349}	12986	**14722**	12734	14468	8000	4551	9168
Si2	$8461_{1.5}$	8486_{35}	8473_{2}	$8491_{27.5}$	8471	**8559**	8485	8529	7029	4256	7200
lshp_778	4730_{2}	4730_{0}	4747_{6}	$4799_{11.5}$	4742	4730	4754	**4820**	3583	3920	4310
bfwb782	4372_{35}	4358_{0}	4411_{14}	4770_{64}	4434	4358	4444	**4882**	3733	2926	3785
goddardRocket_1	4504_{9}	4559_{9}	$4544_{11.5}$	4570_{10}	4560	**4584**	4562	4582	2937	1472	4291
young4c	4089_{0}	4089_{0}	4089_{0}	4089_{0}	**4089**	**4089**	**4089**	**4089**	**4089**	2295	**4089**
orsirr_2	5260_{0}	5260_{0}	5260_{0}	5260_{0}	5260	5260	**5262**	5260	5170	2491	5170

As mentioned, the original matrix cannot be improved in nos6 and young4c instances. This affects the statistical results, so the statistical tests were computed using only data from the remaining 15 instances.

Table III presents Friedman's ranking according to the mean values, while Table IV presents the p-values adjusted with Holm's procedure. In Table IV, '$\lhd$' indicates that the result of the first algorithm is statistically better than the second, and '$-$' states that no statistically significant differences are found. The tests show that the VNS_b algorithm is significantly better than the other algorithms proposed in this work.

TABLE III
AVERAGE RANKINGS OF FRIEDMAN TEST OF THE ALGORITHMS.

ILS_f	ILS_b	VNS_f	VNS_b
3.43	2.57	2.53	1.47

TABLE IV
STATISTICAL ASSESSMENT OF THE ALGORITHMS.

VNS_b vs. ILS_f	VNS_b vs. ILS_b	VNS_b vs. VNS_f
3.02e-05 $\lhd$	0.0196 $\lhd$	0.0237 $\lhd$

VNS_f vs. ILS_f	VNS_f vs. ILS_b	ILS_f vs. ILS_f
0.056 $-$	0.943 $-$	0.066 $-$

To continue with the analysis, let us consider the results for the best-found solution of the algorithms. The results on the best-found solution allow us to confirm the trend shown with the mean values. VNS_b systematically outperforms the rest of the algorithms studied in this work and is the best-performing algorithm in 11 out of 17 instances.

Moreover, VNS_b can obtain better solutions than the EA from previous work and the RCM method, as well as improve over the original matrix in all the instances considered. It should be noted that not only VNS_b is superior to the EA, but it also is able to obtain such results with less computational effort because it evaluates half of the solutions used by the EA. To clarify, Figure 1 shows the percentage of *nnz* in the 2% of the diagonals obtained by VNS_b and the three previous approaches. Since the matrices are sorted by their density in the figure, these results also suggest that instances with higher density are more difficult to solve and that there is still room for improvement. This is expected, since when the average non-zeros per row grows, it is more likely that any move to bring a non-zero to the 2% of diagonals also pushes outside another already there.

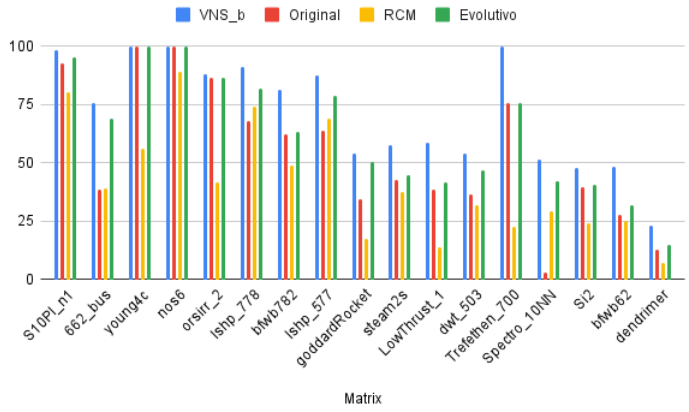

Fig. 1. Percentage of *nnz* in 2% of the diagonals.

Within the context of this experimental evaluation, it is noticeable that VNS_b has shown promising results, being the best-performing algorithm for maximizing the number of non-zeros in the two percent of diagonals of sparse matrices.

V. CONCLUSIONS AND FUTURE WORK

In this work, we continued previous efforts on the problem of finding better permutations to store sparse matrices in HYB. This format stores the main block as ELL and the other elements as COO. Thus, it is important to find permutations that maximize the elements in the block. The VNS variant VNS_b proposed in this work outperforms the previous evolutionary

algorithm while doing half of the evaluations. This leaves our approach as an interesting one to this type of problem.

There are many interesting lines of future work. First, expanding the test group to more (and larger) matrices of the collection would allow us to confirm the density hypothesis. Secondly, we want to test the impact of these changes in linear algebra kernels. Finally, our approaches used random local search operators that do not use information about the problem. It is interesting to explore more specific operators, for example, an operator that checks that the rows selected to be moved have non-zeros outside the band.

VI. Acknowledgments

This work is partially funded by the UDELAR CSIC-INI project *CompactDisp: Formatos dispersos eficientes para arquitecturas de hardware modernas* . The authors also thank PEDECIBA Informática and UDELAR, Uruguay.

References

[1] T. A. Davis, "Graph algorithms via suitesparse: Graphblas: triangle counting and k-truss," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, 2018, pp. 1–6.

[2] J. Gilbert, S. Reinhardt, and V. Shah, "High-performance graph algorithms from parallel sparse-matrices," in *Proceedings of the 8th International Conference on Applied Parallel Computing*, 2006, p. 260–269.

[3] T. Gale, M. Zaharia, C. Young, and E. Elsen, "Sparse GPU kernels for deep learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[4] H. Anzt, S. Tomov, and J. J. Dongarra, "On the performance and energy efficiency of sparse linear algebra on gpus," *International Journal of High Performance Computing Applications*, vol. 31, pp. 375 – 390, 2017.

[5] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. Siam, 1994, vol. 43.

[6] F. Vázquez, J. J. Fernández, and E. M. Garzón, "A new approach for sparse matrix vector product on nvidia gpus," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 8, pp. 815–826, 2011.

[7] A. Monakov, A. Lokhmotov, and A. Avetisyan, "Automatically tuning sparse matrix-vector multiplication for gpu architectures," in *High Performance Embedded Architectures and Compilers*, 2010, pp. 111–125.

[8] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 806–814.

[9] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 15–28.

[10] E. Trommer, B. Waschneck, and A. Kumar, "dcsr: A memory-efficient sparse matrix representation for parallel neural network inference," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*, 2021, pp. 1–9.

[11] S. Wiedemann, K.-R. Müller, and W. Samek, "Compact and computationally efficient representation of deep neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 772–785, 2020.

[12] G. Du, Z. Yang, Z. Li, D. Zhang, Y. Yin, and Z. Lu, "Nr-mpa: Non-recovery compression based multi-path packet-connected-circuit architecture of convolution neural networks accelerator," in *2019 IEEE 37th International Conference on Computer Design*, 2019, pp. 173–176.

[13] D. T. Vooturi and K. Kothapalli, "Efficient sparse neural networks using regularized multi block sparsity pattern on a gpu," in *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2019, pp. 215–224.

[14] M. Freire, R. Marichal, E. Dufrechou, and P. Ezzatti, "Towards reducing communications in sparse matrix kernels," in *JCC-BD&ET 2023 Proceedings*, 2023.

[15] ——, "Enhancing the sparse matrix storage using reordering techniques," in *Under review-CARLA*, 2023.

[16] R. Marichal, E. Dufrechou, and P. Ezzatti, "Optimizing sparse matrix storage for the big data era," in *Cloud Computing, Big Data & Emerging Topics - 9th Conference, JCC-BD&ET, La Plata, Argentina, June 22-25, 2021, Proceedings*, ser. Communications in Computer and Information Science, vol. 1444, 2021, pp. 121–135.

[17] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th National Conference*, ser. ACM '69. Association for Computing Machinery, 1969, p. 157–172.

[18] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Society for Industrial and Applied Mathematics (SIAM), 2003.

[19] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. Association for Computing Machinery, 2009.

[20] Jee W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (15th PPOPP'10)*. Bangalore, India: ACM SIGPLAN, January 2010, pp. 115–125.

[21] Shengen Yan, C. Li, Y. Zhang, and H. Zhou, "yaSpMV: yet another SpMV framework on GPUs," *ACM SIGPLAN Notices*, vol. 49, no. 8, pp. 107–118, August 2014.

[22] V. Karakasis, G. I. Goumas, and N. Koziris, "A comparative study of blocking storage methods for sparse matrices on multicore architectures," in *CSE*. IEEE Computer Society, 2009, pp. 247–256.

[23] J. Zhang and L. Gruenwald, "Regularizing irregularity: Bitmap-based and portable sparse matrix multiplication for graph data on gpus," ser. GRADES-NDA '18. New York, NY, USA: Association for Computing Machinery, 2018.

[24] G. Berger, M. Freire, R. Marini, E. Dufrechou, and P. Ezzatti, "Unleashing the performance of bmsparse for the sparse matrix multiplication in GPUs," in *Proceedings of the2021 12th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, 2021, pp. 19–26.

[25] ——, "Advancing on an efficient sparse matrix multiplication kernel for modern gpus," *Concurrency and Computation: Practice and Experience*, 2022.

[26] J. Godwin, J. Holewinski, and P. Sadayappan, "High-performance sparse matrix-vector multiplication on gpus for structured grid computations," in *The 5th Annual Workshop on General Purpose Processing with Graphics Processing Units, GPGPU-5, London, United Kingdom, March 3, 2012*. ACM, 2012, pp. 47–56.

[27] A. George, "Computer implementation of the finite element method," Ph.D. dissertation, Computer Science Department, School of Humanities and Sciences, Stanford Univesity, CA, USA, 1971.

[28] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*. ACM Press, 1969, pp. 157–172.

[29] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[30] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[31] E.-G. Talbi, *Metaheuristics: from design to implementation*, 2009.

[32] H. H. Hoos and T. Stützle, *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[33] I. Decia, R. Leira, M. Pedemonte, E. Fernández, and P. Ezzatti, "A vns with parallel evaluation of solutions for the inverse lighting problem," in *Applications of Evolutionary Computation: 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part I 20*. Springer, 2017, pp. 741–756.

[34] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[35] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[36] M. Pedemonte, F. Luna, and E. Alba, "A theoretical and empirical study of the trajectories of solutions on the grid of systolic genetic search," *Information Sciences*, vol. 445, pp. 97–117, 2018.