Northeastern University
Department of Electrical and Computer Engineering
2021 PhD Qualifying Examination
in Computer Engineering

Student's name: Emily Costa
Problem number: 3
Submission date: May 10, 2021

1 Introduction

In this paper, I explore how high-performance computing performance improves by exploiting spatial locality in data by comparing the runtime efficiency of two different sparse matrix representations. The generated data sets are similar to those used by several HPC applications that involve problems that work with sparse input data sets such as graphs.

2 Methodology

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 3 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

Matrix 1: An $N \times N$, where $N = 5$, sparse matrix with a sparsity of 75%.

Generating Sparse Matrices. A sparse matrix is a matrix in which most of the elements are zero. An example of a sparse matrix is given in Matrix 1. In order to explore sparse matrix representations, I generate random sparse matrices to be converted and used for each benchmark. I apply two parameters that determine how the matrices are generated: (1) the matrix size and (2) the sparsity of nonzero elements in the matrix. The matrices generated are $N \times N$ such that the number of rows is equal to the number of columns.

Benchmarking. In order to benchmark the performance of algorithms ran on the sparse matrix implementations, I simply record the runtime of each function. Within each function, I convert the matrix into the respective representation then run SpMV on it. I decided to include the time to convert to the respective representations as that would be an additional step taken in production high-performance systems. The conversion would most likely be done when the matrix is compressed and stored prior to the matrix being inputted as a parameter. It should be noted that my conversion and SpMV implementations are not necessarily the most efficient ones nor are they run in parallel on a high-performance computing system.

Platform. I ran all my code on my local machine, a Dell XPS 9370 with an Intel(R) Core(TM) i7-8550U CPU quad-core processor and 16 GiB RAM. My operating system is Ubuntu v20.04.2 LTS running with a Linux kernel, which is similar to the Discovery at Northeastern University. I tested all my code on Discovery.

3 Sparse Matrix Representation Implementation

Naive. In this paper, I refer to two-dimensional array representations of matrices as naive. They serve as the baseline case for my results and analysis. A sparse matrix is a matrix in which the

$$\Big[[0\ 0\ 0\ 1\ 0], [0\ 0\ 2\ 0\ 0], [0\ 0\ 0\ 0\ 0], [0\ 4\ 0\ 3\ 5], [0\ 0\ 0\ 0\ 0]\Big] \tag{2}$$

Matrix 2: An $N \times N$, where $N = 5$, two-dimensional array representation of a sparse matrix.

Row	1	2	4	4	4
Column	4	3	2	4	5
Value	1	2	4	3	5

Table 1: A COO representation of a sparse matrix where only the nonzero values are given.

majority of values are zeroes. Matrix 2 is an example of a naive representation of based on Matrix 1, a sparse matrix. In order for applications to utilize the naive implementation, the algorithms need to traverse every element, including the unnecessary zeroes. This requires the compute node to store significantly more in its memory and, as will be shown in the following sections, demands exponentially more compute power. Hence, researchers have developed alternative methods to representation matrices that decrease compute and memory demand, achieving a higher performance. We will explore this further in Sec. 4 by delving into a commonly done operation, matrix-vector multiplication.

Coordinate-wise. Coordinate-wise (COO) matrix representation is the simplest alternative to two-dimensional, or naive, matrices. As show in Table 1, COO provides the columns and row indices of all nonzero values in a sparse matrix. In this example, the matrix in Matrix 2 is converted to COO representation. To store the matrix elements, three arrays are used to hold the value, row, and column respectively. Hence, the memory used scales linearly by a factor of three times with the number of nonzero elements in the matrix.

Compressed Sparse Row. Compressed Sparse Row (CSR) matrix representation is a commonly used alternative to COO. Similarly to COO, it uses three arrays to indicate the nonzero elements of a matrix. The three arrays store the element value, column, and row-index range. However, the row-index range array does not scale linearly to the number of nonzero elements but rather it scales linearly to the number of rows in the matrix. Therefore, though the memory still scales linearly with the number of nonzero elements in the matrix, it is by a factor of two times rather than three

Row-Index Range	1	2	3	-	-
Column	4	3	2	4	5
Value	1	2	4	3	5

Table 2: A CSR representation of a sparse matrix where only the nonzero values are given.

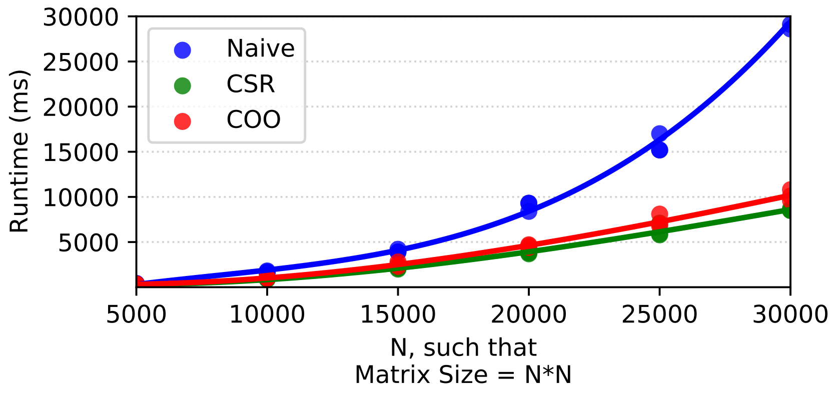

Figure 1: *Comparing the performance of matrix representations based on the size of the matrix with a set sparsity.*

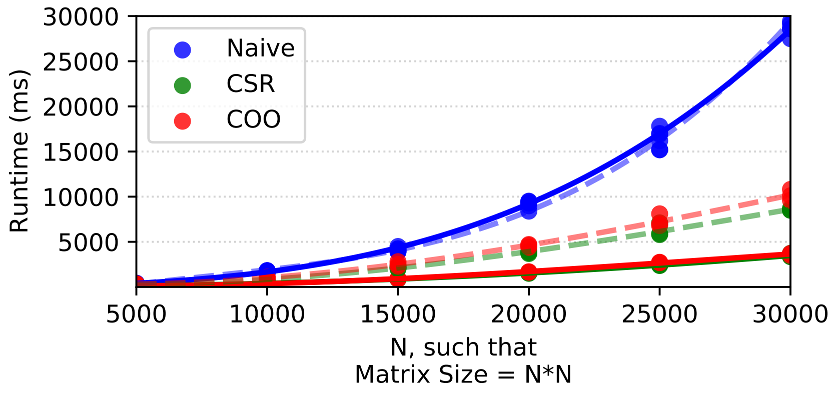

Figure 2: *Comparing the performance of matrix representations based on the size of the matrix with a set number of nonzero values.*

times. In order to achieve formatting Matrix 1 in CSR representation, entries in the same row must be consecutively located in the arrays, which is shown in Table 2.

4 Sparse Matrix-Vector Multiplication

Sparse Matrix-Vector Multiplication (SpMV) is a commonly used computation kernel in numerous high-performance computing applications. In SpMV, an M by N sparse matrix is multiplied with a N by 1 vector. However, running SpMV on a full matrix, or two-dimensional array, is known to be inefficient for memory and runtime. In this section, I explore how running SpMV with the naive, or two-dimensional array, matrix representation compares to alternative matrix representations such as Coordinate-wise (COO) and Compressed Sparse Row (CSR) matrix representations.

Fixed sparsity, varying matrix size. First, I compare how the matrix representations perfor-

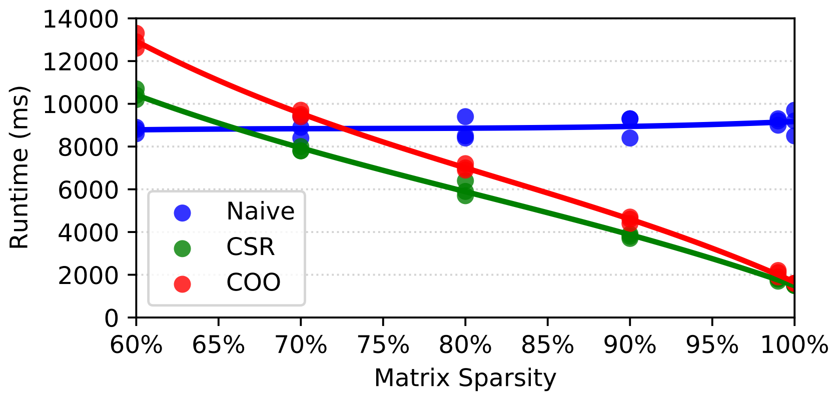

Figure 3: *Comparing the performance of matrix representations based on the sparsity of the matrix.*

mance when Sparse Matrix-Vector Multiplication (SpMV) is ran as the size of the matrix is varied. I generated several random sparse matrices as explained in Sec. 2, with varying sizes, and convert them each into the representations, naive, COO, and CSR. I set the sparsity of the matrices to be 90%, which mean that only 10% of the values are nonzero. I ran SpMV on each of the matrices for each size three times. Fig. 1 is a scatterplot to show how much time it took for the algorithm to run on each representation, the runtime of SpMV. I added a regression line to indicate the general trend of the SpMV performance. We observe that both COO and CSR linearly increase in runtime as matrix size increase whereas the naive implementation increases exponentially by a factor a N^2, N being the length of one-dimension of the matrix such that the original matrix is size $N * N$.

Fixed number of nonzero values, varying matrix size. Naturally, the question following this result would be: why does matrix size affect the performance of SpMV? To answer this, I redo the analysis but with a fixed number of nonzero values instead of a fixed sparsity. This means that each matrix, regardless of size, will have X nonzero values instead of $Y\%$ of nonzero values such that the amount of nonzero values does not increase with the matrix size. Fig. 2 shows the original fixed sparsity runtimes in dashed lines and the additional fixed nonzeroes in solid lines. We observe the COO and CSR representation significantly diverges in the original behavior and no longer increases in runtime at the same rate. Though they do increase at a slight rate, that may be due to the increased time taken to convert from a matrix into their respective representations. The original naive matrix representation does not change significantly in behavior, which suggests that the runtime is dependent on matrix size for the naive method whereas runtime is dependent on the number of nonzero values in the matrix for both COO and CSR.

Varying sparsity, fixed matrix size. To delve deeper into the finding that CSR and COO are dependent on the number of nonzeroes, I run the SpMV with a fixed matrix size of $20,000 \times 20,000$ and varying sparsity. Fig. 3 shows the runtimes of the SpMV given a certain sparsity in a large matrix. We observe that the runtime of the naive implementation is not significantly affected by the sparsity of the matrix whereas the COO and CSR are significantly affected by the sparsity. Additionally, we observe two interesting trends:
(1) CSR and COO converge in runtime as they approach a sparsity of 100%. This is because CSR improves on COO when there are more nonzero values per row, which decreases are the sparsity

increases. The implication of this is that the benefits of using the more complicated representation scheme, CSR, are insignificant after a certain threshold of sparsity and, hence, not worth the hassle of complicating the algorithms ran on the matrix representation.
(2) CSR and COO only outperform the naive representation after a certain sparsity. The runtime needed to convert matrix representation and run SpMV is consistent for the naive implementation and, hence, lower before a certain sparsity of the matrix. The benefits of alternative matrix representations are after a certain sparsity, $70 - 75\%$ in these results. The implication is that CSR and COO should be implemented only when a matrix is a certain level of sparsity.

5 Sparse Matrix Representation in Previous Works

This section explores two commonly used sparse matrix representations by summarizing two papers that utilize two sparse matrix implementations that I will evaluate in a later section of this paper.

Vectorized Sparse Matrix Multiply for Compressed Row Storage Format. This paper [D'Azevedo et al., 2005] introduces an alternative algorithm that performs sparse matrix-vector multiplication (SpMV) which utilizes compressed sparse row (CSR) matrices. The algorithm does the SpMV on compressed sparse row (CSR) matrix representation, which the authors point out to be a commonly used matrix storage format. The form of CSR used in the algorithm builds on the original; the CSR include a permutation vector such that rows with the same number of nonzeros are grouped together.

In order to improve the performance of applications that use SpMV, the authors introduce a means of computing SpMV that will max-out the resources in a high-performance computing system. At the time, vector processors were being introduce to modern high-performance computing (HPC) systems. This paper seems to foreshadow the transition to HPC systems that implemented GPUs and require applications to take advantage of the hardware as to achieve maximum performance. More recent papers that utilize CSR do so on GPUs [Benatia et al., 2020; Flegar and Quintana-Ortı, 2017; Guo and Zhang, 2018; Nisa et al., 2018].

Performance Prediction for CSR-Based SpMV on GPUs Using Machine Learning. In fact, GPU-accelerated SpMV using a CSR kernel has become so relevant to the HPC community that efforts have been made to predict the performance. This paper [Guo and Zhang, 2018] proposes using machine learning techniques to predict the performance of SpMV using CSR as storage format. Support vector machine regression (SVR) is used to build the model.

Use of hybrid recursive CSR/COO data structures in sparse matrix-vector multiplication. Another interesting application of the matrix formats is a hybrid created from recursive CSR or COO. This paper [Martone et al., 2010] utilizes both COO and CSR to build up a new recursive method for storing sparse matrices. The sparse matrix is formed as a quad-tree structure with submatrices partitioned among the leaves. Each leaf uses CSR or COO to store the respective submatrix.

6 Conclusion

In this paper, I discussed alternative sparse matrix representations to the two-dimensional array, naive representation. I reviewed and summarized related works to two sparse matrix representations, Compressed Sparse Row (CSR) and Coordinate-wise (COO). I converted sparse matrices to the alternative representations and compared how sparse matrix-vector multiplication (SpMV) performed. During our benchmarking, we observed the following findings and explanations:

- The naive matrix representation takes exponentially more memory and compute for SpMV as the matrix size increases whereas COO and CSR scales linearly.
- Matrix size is the main factor in determining the runtime of SpMV on the naive matrix representation whereas the number of nonzero values, or sparsity, of the matrix is the main determinate for COO and CSR.
- As the sparsity of the matrix increases, the runtime of COO and CSR converges.
- Practically, COO and CSR only improve the runtime of SpMV when the matrix have a sparsity greater than a certain threshold.

7 Code

Code is available at https://github.com/emilyjcosta5/ce_phd_qual

References

Benatia, A., Ji, W., Wang, Y., & Shi, F. (2020). Sparse matrix partitioning for optimizing spmv on cpu-gpu heterogeneous platforms. *The International Journal of High Performance Computing Applications*, *34*(1), 66–80.

D'Azevedo, E. F., Fahey, M. R., & Mills, R. T. (2005). Vectorized sparse matrix multiply for compressed row storage format. *International Conference on Computational Science*, 99–106.

Flegar, G., & Quintana-Ortı, E. S. (2017). Balanced csr sparse matrix-vector product on graphics processors. *European Conference on Parallel Processing*, 697–709.

Guo, P., & Zhang, C. (2018). Performance prediction for csr-based spmv on gpus using machine learning. *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 1956–1960. https://doi.org/10.1109/CompComm.2018.8780598

Martone, M., Filippone, S., Tucci, S., Gepner, P., & Paprzycki, M. (2010). Use of hybrid recursive csr/coo data structures in sparse matrix-vector multiplication. *Proceedings of the International Multiconference on Computer Science and Information Technology*, 327–335. https://doi.org/10.1109/IMCSIT.2010.5680039

Nisa, I., Siegel, C., Rajam, A. S., Vishnu, A., & Sadayappan, P. (2018). Effective machine learning based format selection and performance modeling for spmv on gpus. *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 1056–1065.