A Framework for Analyzing the Performance of Sparse Matrix and Graph Operations

Khaled Abdelaal
School of Computer Science
The University of Oklahoma
Norman, OK, USA
khaled.abdelaal@ou.edu

Richard Veras
School of Computer Science
University of Oklahoma
Norman, OK, USA
richard.m.veras@ou.edu

***Abstract*—Thorough performance analysis is crucial for developing and optimizing algorithms, particularly for compute-intensive operations like those in scientific libraries such as the Basic Linear Algebra Subprograms (BLAS). While dense computations can predict performance based on dataset dimensions and strides, sparse matrix operations require more detailed analysis due to their complexity. Standard performance evaluations for sparse operations use a canonical set of matrices, but generalizing these results to new datasets is limited. This paper presents a framework to evaluate sparse matrix and graph operations by visualizing performance through parameterized graph models. It assesses different parameter sets and noise sources on performance, providing a modular and extensible approach, which includes system-wide performance considerations, allowing users to integrate various graph model generators, operation implementations, and noise types.**

***Index Terms*—Sparse Matrix, Performance Analysis, Graph Models**

I. Introduction

Large, sparse, and irregular data is central in the domains such as graph analytics, graph neural networks, fluid mechanics, and finite element analysis. Specifically, if the dynamic relationship between elements in a dataset can be captured as an edge-pair relationship between vertices, then graphs provide a natural representation of that data. Furthermore, if the analysis of complex relationships in the data can be performed through sequential linear algebra-like operations over the adjacency matrices of these datasets, then operations such as Sparse Matrix times Vector Multiplication (SpMV) are critical to the performance of computations in these domains. However, optimizing operations like SpMV is challenging because the structure of the sparse data, the implementation of the operation, and the architecture of the target all have a tremendous bearing on the execution time of these operations. If a sparse operation is tuned for one class of data, that performance may not generalize to another class. The core of this work is to provide a benchmarking framework for correlating performance with the structural features of sparse data.

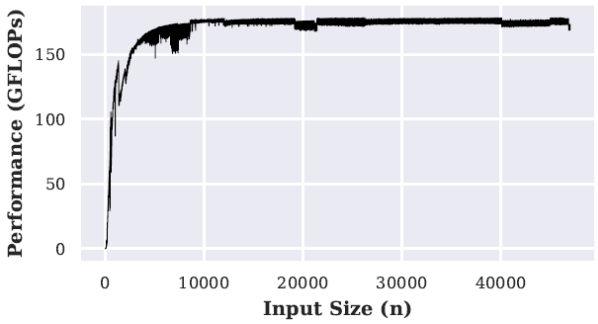


Fig. 1: Double Precision Dense General Matrix-Vector Multiplication Performance using cuBLAS on NVIDIA RTX A6000 GPU as a function of input size

For dense matrices, performance evaluation and visualization is straightforward. Figure 1 shows an example of performance evaluation of the general matrix-vector multiplication (GEMV) on a RTX A6000 GPU using cuBLAS. For simplicity, evaluated matrices are assumed to be of square dimensions: $n \times n$. The horizontal axis represents the different values of n evaluated, and the vertical axis shows the performance in GFLOPs. Moving along the horizontal axis (from left to right and from right to left) shows a clear correlation between the matrix dimensions ($n \times n$) and GEMV performance. Performance interpolation from existing data points is possible based on the dense matrix dimensions (n)

Figure 2 presents the performance evaluation of SpMV using cuSparse on matrices from the SuiteSparse collection [1]. The horizontal axis represents different matrices, and the vertical axis shows the SpMV performance in GFLOPs. However, this evaluation provides limited insights due to the distinct characteristics of each matrix, which prevents meaningful performance correlation. Using dimensions or the number of nonzeros (NNZ) on the x-axis also fails to yield useful conclusions since matrices with the same dimensions can have different sparsity ratios, and those with sim-

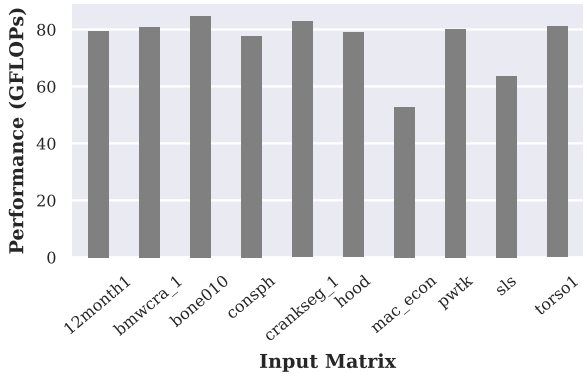


Fig. 2: Double Precision Sparse Matrix-Vector Multiplication Performance using cuSparse on NVIDIA RTX A6000 GPU for a selected set of matrices from SuiteSparse, using the COOrdinate data representation.

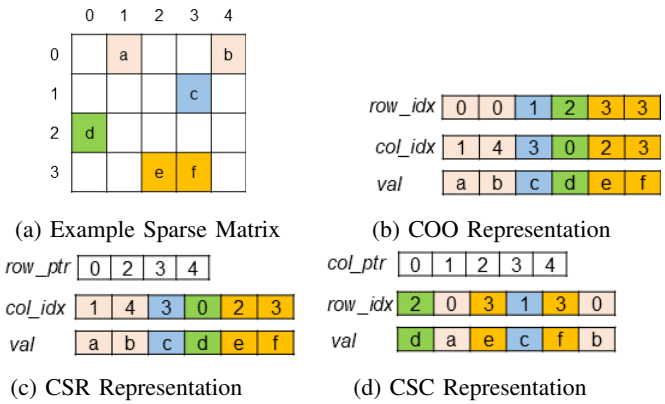
ilar NNZ can vary greatly in dimensions and sparsity ratios. Existing performance models and metrics face challenges adapting to sparse data workloads [2], [3], and relying on discrete sets of matrices for benchmarking limits performance generalization [1], [4], [5]. Moreover, many optimization techniques focus only on the sparse operation, overlooking system-level execution time and potential performance bottlenecks such as I/O.

To address the limitations in performance analysis of sparse matrices and graph operations, we propose a novel end-to-end framework that uses parameterized graph models to generate synthetic graphs and evaluate performance sensitivity to various sources of noise in model parameters. This framework systematically analyzes the relationships between input parameters of sparse matrix/graph generators and the performance of operations like SpMV. By building a predictive understanding between the generator and performance, it enables informed decisions for data approximated by these generators. Our framework focuses on identifying features and parameters that relate to performance and providing efficient performance visualization methods.

The main contributions of this work are as follows:

1) Propose an extensible framework for performance analysis and evaluation for sparse data operations and driving design choice for performance optimizations.
2) Evaluate the usage of different graph model parameters and how it relates to performance interpolation and extrapolation.
3) Provide an alternative to using discrete graph sets for benchmarking sparse data workloads.
4) Estimate the effect of different noise sources in performance and integrating it into potential performance interpolations.

II. Background and Related Work



(a) Example Sparse Matrix (b) COO Representation

(c) CSR Representation (d) CSC Representation

Fig. 3: Sparse Matrix Example (a) and its representation using (b) COO, (c) CSR, and (d) CSC

A. Sparse Matrix Storage Formats

Different storage formats have been proposed in literature mainly to reduce the memory requirements for sparse matrices. Figure 3 shows an example sparse matrix and its representation using different sparse storage formats: COOrdinate (COO), Compressed Sparse Row (CSR), and Compressed Sparse Column (CSC).

B. Graph Models for Sparse Data

Many performance evaluation techniques for sparse data have emerged in response to the generation of such data from engineering and physics problems. However, many real data is more accurately represented by large scale-free synthetic data that follow a power-law distribution such as Kronecker graphs [6], [7] or a combination of Kronecker + Random [8]. Kronecker graphs are a class of synthetic graphs that have been widely used to model real-world networks, and are generated by recursively applying the Kronecker product of a small base graph with itself. Let A and B be two matrices. Then, their Kronecker product $A \otimes B$ is given by

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix} \quad (1)$$

where a_{ij} are the entries of A. The resulting graph has a power-law degree distribution and exhibits a hierarchical structure that captures both the local and global connectivity patterns of the underlying real-world network.

While generative graph models (e.g. Kronecker graphs) provide a parameterized way of generating synthetic graphs similar to real graphs, tools that try to fit real data to such model (e.g. Kronfit [7]) are limited in estimation accuracy of the model parameters. Hence,

our framework uses different generation models, but accounts for different sources of noise, including noise in graph generation parameters.

C. Performance Evaluation for Sparse Data Operations

To correctly understand how modern algorithms contribute to improving performance, several frameworks have been proposed. The Graph Algorithm Iron Law (GAIL) [2] targets graph processing algorithms, and proposes the usage of more adequate metrics, other than just execution time, to quantify performance contributions in regard to graphs. The proposed metrics include algorithmic work, communication volume, and bandwidth utilization. While these metrics can provide a better understanding of the performance improvements of different algorithms, the main focus of this work is performance metrics (vertical axis of performance plots), and not the graph model features/parameters which can affect these performance metrics (horizontal axis of performance plots).

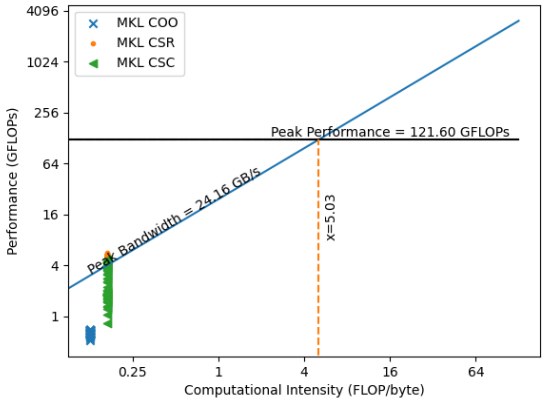


Fig. 4: Performance Evaluation of SpMV in MKL for different sparse formats (COO, CSR, and CSC) on a set of sparse matrices using the Roofline model. Since the arithmetic intensity is imposed by the sparse data format, little insights are provided on how to optimize performance. Arithmetic intensity was estimated based on memory footprint for different storage formats and SpMV FLOPs.

The roofline model [3] has been the standard model used in performance evaluation, where theoretical machine peak performance and bandwidth bounds are calculated, and application performance is recorded as a point under the theoretical bounds curve. The x value for each point is the operational intensity (FLOPs/byte), and the y value is performance (FLOPs). Additionally, many derived variants of the roofline model have been developed to accommodate for different memory hierarchy assumptions [9], capture the hardware changes in modern architectures such as GPUs [10]–[12], and work on finer granularity than a FLOP/byte such as instruction/transaction [13]. However, the application of the roofline model for applications on sparse data is less than adequate. Assuming we try to optimize SpMV operation, and we evaluate the performance of each algorithm using operational (arithmetic) intensity, and FLOPs. Since SpMV operations are directly coupled to a specific sparse data format (COO, CSR, CSC), the operational intensity for any implementation is fixed for a specific sparse data format, since the format imposes the size (bytes) of the data pieces involved in the SpMV operation. Figure 4 illustrates this issue, where the SpMV performance was evaluated using COO, CSR, and CSC for a set of input graphs. Most of the data points lie on the same vertical line in the figure, since they have similar arithmetic intensity imposed by the sparse data format. This kind of plots provides little insights on how to optimize such operations on sparse data.

In response to the above limitations, we developed our framework to evaluate using different features on the horizontal axis of performance plots, which have the potential of being exploited for performance optimization. These features can be parameters used to generate graphs/sparse matrices using a specific graph model. In addition, our framework is flexible to incorporate any performance metric (vertical axis) similar to the ones proposed in existing work (e.g. GRAIL), while providing a better representation of datasets to enable performance interpolation.

D. Benchmarking Sparse and Graph data Workloads

In order to benchmark sparse and graph data workloads, appropriate input data needs to be fed to developing algorithms. Most of existing work in literature on different performance optimization for sparse matrices and graphs uses SNAP dataset [4], SuiteSparse Matrix Collection [1], and GAP [5]. These benchmarks provide a set of synthetic and real graphs/matrices from different applications and structures. However, they are limited in the sense that they are a discrete collection of graphs. Interpolating the performance of unseen graphs from a set of discrete graphs with no common continuity feature is challenging. For example, the GAP benchmark graph dataset consists of only five graphs. Tuning new algorithms on a discrete set of graphs, it is difficult to expect the performance interpolations to generalize across other sparse matrices and graphs. A recent work [14] proposed the use of artificial sparse matrix generators to tackle the issue of potential biased performance decisions based on discrete sets of matrices. However, the proposed generator parameters are limited to the observed SpMV bottlenecks features (average nnz per row, standard deviation of nonzeros per row,

bandwidth of matrix, etc.). Hence, additional work is needed to provide a more comprehensive performance analysis framework that takes into consideration any input graph/sparse matrix generation models.

III. Methods

Our framework aims at providing a modern infrastructure for describing the performance of applications where sparse data and graphs are involved. As demonstrated in Section II, existing techniques fall short in this category of irregular memory access applications. Our framework provides a means of interpolating and extrapolating the performance of different sparse matrices/graphs, based on models they closely fit.

The main goal of our framework is to find a relationship between the graph generation mechanism and the resulting performance of operations in which the graph is involved as an operand. Such analysis allows for the identification of promising graph generation parameters/features that show direct influence on performance. Algorithms can be developed to exploit these parameters to optimize the performance of operations where graphs of this model are used as operands.

The advantage of using our framework over profiling a single application is that it allows for a deeper understanding of the performance of applications that would operate over any matrix/graph that fits a specific graph model. It uses more representative features beyond dimensions or arithmetic intensity that might not be suitable in many cases.

Algorithm 1 General Framework Description

```
1:  for each param in model_gen_params do
2:      for each val in param_legal_values do
3:          new_params ← val∪(params−param_old_value)
4:          G ← gen(new_params)
5:          append G to Gs
6:          for n0 = 0 to nA_thresh step nA_step do
7:              G_NA ← gen_noiseA(G, n0)
8:              append G_NA to Gs_NA
9:          end for
10:         for n1 = 0 to nB_thresh step nB_step do
11:             G_NB ← gen_noiseB(G, n1)
12:             append G_NB to Gs_NB
13:         end for
14:         for each op in operations do
15:             for each impl in implementations[op] do
16:                 for each fmt in sparse_formats do
17:                     for each graph in Gs ∪ Gs_NA ∪ Gs_NB do
18:                         r ← record(eval(impl(fmt(graph))))
19:                         append r to results
20:                     end for
21:                 end for
22:             end for
23:         end for
24:     end for
25: end for
26: for each feature in features do
27:     visualize(results, feature)
28: end for
```

A. High-Level Overview

A general overview of the framework is shown in Algorithm 1. All Graphs discussed are directed weighted graphs, where the vertices are row/column indices, and the weights on edges are non-zero values. Initial Graphs are generated using a parameterized graph model (generator). Each model takes as input a set of parameters. In addition to the initial set of graphs, the framework generates additional sets G_s by varying the input parameters within the legal range of values for each, while fixing the rest of the values.

Then, the framework induces two forms of noise indicated in Algorithm 1 as `noiseA` and `noiseB`. `noiseA` tries to capture noisy prediction of real data to the model parameter. Graph models are expected to produce synthetic graphs with features similar to real-world graphs, but `noiseA` tests the effect of errors in these graph model parameters. `noiseB` on the other hand assesses the cases in which the model alone does not entirely describe the real data. Real graphs do not appear as pure representation of a model, noisy data might be added in the process of reading, transmitting, or pre-processing such graphs. Also, those graphs do not hold any node ordering guarantees.

The framework injects `noiseA` into G_s through the arithmetic addition of G_s with a set of random sparse matrices generated using a parameterized density that varies from (`n0`) to `nA_thresh` from a uniform distribution, and a user-defined step of `nA_step` producing a new set of Graphs: Gs_{NA}. Injecting this noise is an arithmetic matrix addition between the adjacency matrices of the random sparse graph and the original graph. In this process, new edges may be added, and/or existing edges weight may change. Additionally, `noiseB` is added to G_s as a random sparse matrix with density `n1` in steps of `nB_step` up to a maximum of `nB_thresh`, generating the Gs_{NB} graph set.

After the completion of the graph sets generation phase, performance of such graphs involved as operands in operations is to be evaluated. Multiple operations can be executed where these graphs are operands, for example Sparse Matrix-Vector Multiplication (SpMV), in which the graph represents the sparse matrix. The graph or the sparse matrix can be represented using different sparse formats (COO, CSR, CSC, etc.), and each of these have their own implementation. The framework evaluates the performance of SpMV using the different formats and implementations for all generated graph sets.

The final step is to relate performance to different features and parameters of the graph model. These features and parameters are then used to represent the horizontal axis of the performance plots. The goal of such representation is to find a relationship between a feature or a set of features, and the performance of the

operation on the graph. Using this information, more efficient algorithms can be tuned to optimize performance by exploiting features that exhibit strong correlation with performance.

IV. Evaluation and Results

The general framework described in Algorithm 1 generates a high-dimensional set of experiments involving different combinations of parameters and noise values. We conducted a subset of experiments to showcase the capabilities of our framework. In this section, we report planar slices of some of the experiments. Our framework was evaluated for both CPU and GPU. Table I shows the configuration for the system used in our experiments.

TABLE I: System Configuration

Component	Specification
GPU	NVIDIA RTX H100
GPU Memory	80 GB
CUDA Version	12.0
CPU	Intel Xeon Gold 6338 @ 2.00GHz
CPU Sockets	2
CPU Cores per Socket	32
CPU Threads per Core	2
MKL Version	2022.1.0
Main Memory	256 GB DDR4

A. Graphs Generated by Varying Model Parameters

1) K15 Graphs with Varying Initiator Matrix – Heatmaps: A Kronecker power of 15 was used, and the initiator matrix values were varied as follows: we start with a sample 2×2 initiator matrix of the values $[0.999, 0.437; 0.437, 0.484]$, matching the estimated initiator values by Kronfit [7] for the High Energy Physics - Phenomenology Collaboration (CA-HEP-PH) Graph [15] from the SNAP dataset. Then, we fix the first and last initiator matrix values, while varying the other two, producing different combinations of them. For each of the newly generated initiator matrices, a new Kronecker graph is generated. Finally, we evaluate the performance of each as a sparse matrix in a SpMV operation.

Figure 5 shows heatmaps generated by our framework, representing the performance of SpMV for the generated K15 graphs, using Intel MKL for different sparse data formats: COO, CSR, and CSC. The choice of heatmap for the visualization of the Kronecker graph performance enables observing relationship between two different features (parameters) of the model (two initiator matrix values), and how the performance changes with varying both of the parameters. Also, the comparison between different sparse data formats (COO, CSR, and CSC) drives the decision of choosing the ideal data format, for the given input graph model (K15), tool (MKL), and architecture (CPU). The figure clearly shows that CSR is a winner among the three evaluated formats in this specific situation. It also shows that the performance of COO is stable across different x_1 and x_2 values, so no potential benefit appears from optimizing using these two parameters for this specific format.

B. Multiple Different Models with Different Features

In this experiment, we evaluate the performance of different graph models (vertical axis) and relate that performance to different features of the models (horizontal axis). The purpose of this experiment is to show if we can directly compare the performance of different sparse matrix/graph models using common features. This shows if we can interpolate or extrapolate the performance of different model from existing performance results, by dialing different parameters/feature.

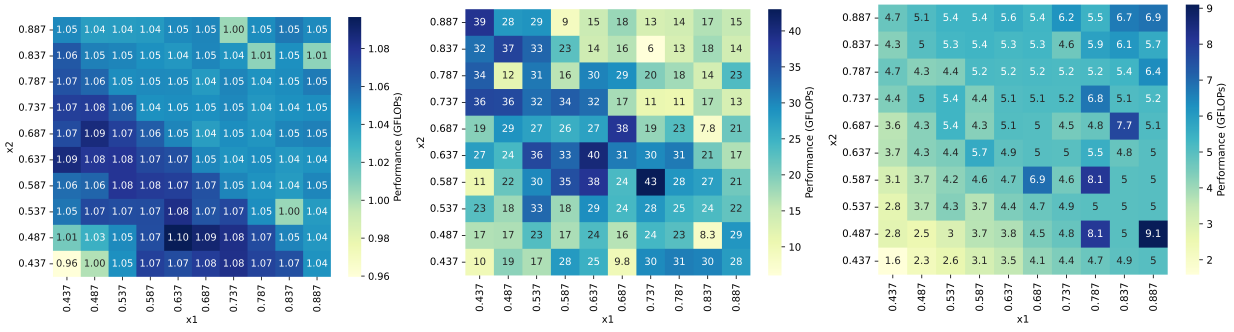
To conduct this experiment, we used three graph models: random graphs generated using the density parameter, Kronecker graphs generated using K power 15 and different initiator matrix values and select graphs from SNAP dataset collection.

Figure 6 shows the performance results of this experiment using cuSparse on H100 GPU, plotted against number of rows, and number of non-zeros used as features on the horizontal axis. Each subplot illustrates the performance of a specific sparse data representation out of the three we evaluated: COO, CSR, and CSC.

Looking at the relationship between Performance and number of rows (Figure 6a, 6b, 6c), one can observe that it is not a suitable feature to tune for performance, as compared to the case in dense matrices.

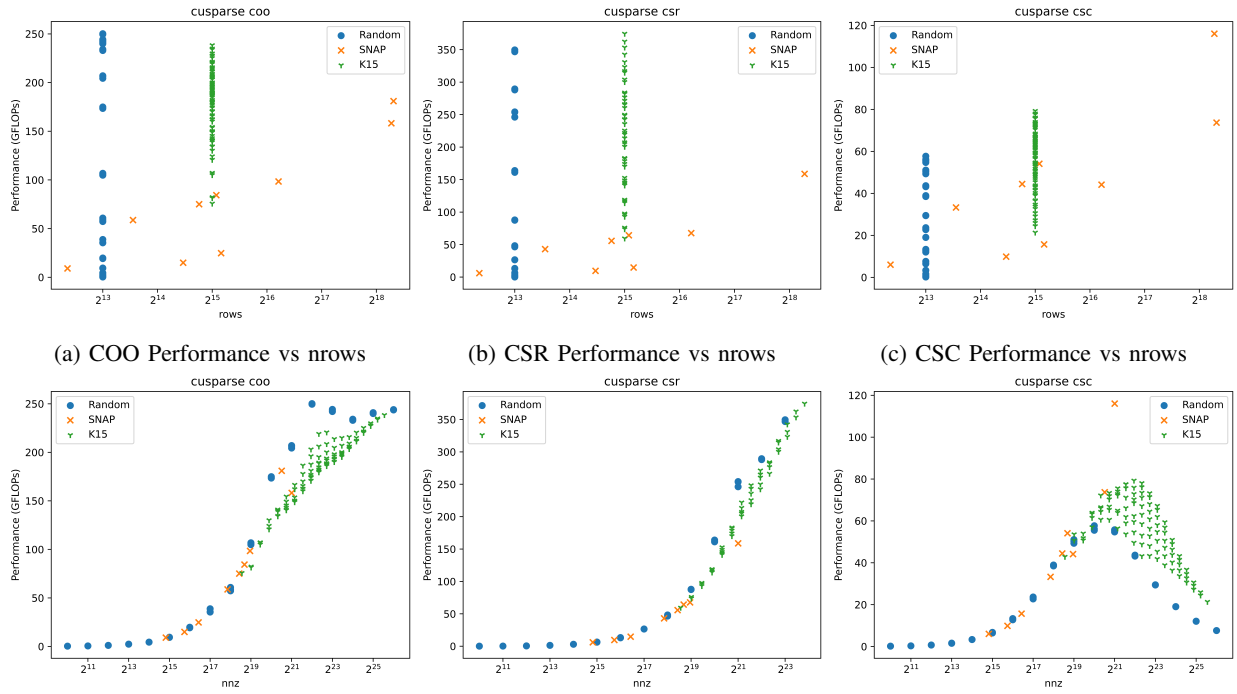
Regarding the choice of ideal format, Figure 6 shows the need of using our framework to sweep across a wide range of graph parameters and noise to generate graphs and make optimization decisions. For the SNAP subset of graphs we evaluated, the maximum attained performance was for the `web-NotreDam` in COO format, at 181 GFLOPs. If one was to tune for only this subset of SNAP graphs, a conclusion to use the COO format would have been made. However, throughout our experiment, we can see that CSR shows the global highest performance across the three graph models, using cuSparse on the H100 GPU.

Random graph generators use a main parameter: density. All of the generated random graphs were of the same dimensions (square). We can see that nnz (Figure 6d, 6e, 6f) as a feature on the x-axis does not work as well for Kronecker graphs; multiple graphs with the same number of nonzeros exhibit different performance characteristics. Also, for SNAP, looking at the scattered performance points, one cannot interpolate or extrapolate the performance (using existing performance data) at different non-zero values that have not been evaluated.



(a) COO (b) CSR (c) CSC

Fig. 5: MKL SpMV performance of K15 Kronecker Graphs with varying initiator matrix values x_1 (x-axis), and x_2 (y-axis). The graph is represented in (a) COO, (b) CSR, and (c) CSC formats.



(a) COO Performance vs nrows (b) CSR Performance vs nrows (c) CSC Performance vs nrows

(d) COO Performance vs nnz (e) CSR Performance vs nnz (f) CSC Performance vs nnz

Fig. 6: cuSparse SpMV performance for: random, SNAP, and K15 graphs plotted against number of rows, and number of non-zeros (nnz) on the horizontal axis. COO, CSR, and CSC formats are evaluated.

V. Conclusion

In this paper, we propose a highly modular framework for evaluating and analyzing the performance of sparse matrix and graph operations. Our proposed framework makes use of parameterized graph models to generate graphs by varying these parameters and observing performance. It also evaluates the effect of inducing different types of noise to the performance of sparse data operations: noise due to error in model fitting tools, and noise rising from using the wrong model for the data. Our framework focuses on evaluating performance (using different metrics) against representative parameters/features (horizontal axis of performance plots), from which performance interpolations and extrapolation can be performed. It also aims at overcoming the existing limitation of using discrete graph sets to tune the performance of sparse matrix and graph kernel. We show results from sets of experiments, conducted through our framework to show the potential it provides to draw insightful performance optimization decisions.

REFERENCES

[1] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, dec 2011. [Online]. Available: https://doi.org/10.1145/2049662.2049663

[2] S. Beamer, K. Asanović, and D. Patterson, "Gail: The graph algorithm iron law," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, ser. IA^3 '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2833179.2833187

[3] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[4] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[5] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," 2017.

[6] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in *Knowledge Discovery in Databases: PKDD 2005*, A. M. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 133–145.

[7] J. Leskovec and C. Faloutsos, "Scalable modeling of real graphs using kronecker multiplication," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 497–504. [Online]. Available: https://doi.org/10.1145/1273496.1273559

[8] C. Seshadhri, A. Pinar, and T. G. Kolda, "An in-depth study of stochastic kronecker graphs," in *2011 IEEE 11th International Conference on Data Mining*, 2011, pp. 587–596.

[9] A. Lopes, F. Pratas, L. Sousa, and A. Ilic, "Exploring gpu performance, power and energy-efficiency bounds with cache-aware roofline modeling," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 259–268.

[10] E. Konstantinidis and Y. Cotronis, "A quantitative roofline model for gpu kernel performance estimation using micro-benchmarks and hardware metric profiling," *Journal of Parallel and Distributed Computing*, vol. 107, pp. 37–56, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731517301247

[11] C. Yang, T. Kurth, and S. Williams, "Hierarchical roofline analysis for gpus: Accelerating performance optimization for the nersc-9 perlmutter system," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 20, p. e5547, 2020.

[12] K. Z. Ibrahim, S. Williams, and L. Oliker, "Performance analysis of gpu programming models using the roofline scaling trajectories," in *International Symposium on Benchmarking, Measuring and Optimization*. Springer, 2019, pp. 3–19.

[13] N. Ding and S. Williams, "An instruction roofline model for gpus," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2019, pp. 7–18.

[14] D. Galanopoulos, P. Mpakos, P. Anastasiadis, N. Koziris, and G. Goumas, "Invited paper: An artificial matrix generator for multi-platform spmv performance analysis," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2023, pp. 574–577.

[15] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 1, p. 2–es, mar 2007. [Online]. Available: https://doi.org/10.1145/1217299.1217301