Core Set of Sparse Computation Features

Deliverable No:	D1.1
Deliverable Title:	Core Set of Sparse Computation Features
Deliverable Publish Date:	31 December 2021
Project Title:	SparCity: An Optimization and Co-design Framework for Sparse Computation
Call ID:	H2020-JTI-EuroHPC-2019-1
Project No:	956213
Project Duration:	36 months
Project Start Date:	1 April 2021
Contact:	sparcity-project-group@ku.edu.tr

List of partners:

Participant no.	Participant organisation name	Short name	Country
1 (Coordinator)	Koç University	KU	Turkey
2	Sabancı University	SU	Turkey
3	Simula Research Laboratory AS	Simula	Norway
4	Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa	INESC-ID	Portugal
5	Ludwig-Maximilians-Universität München	LMU	Germany
6	Graphcore AS	Graphcore	Norway

CONTENTS

1 Introduction 1
1.1 Objectives of This Deliverable 1
1.2 Worked Performed 1
1.3 Deviations and Counter Measures 2
1.4 Resources 2

2 Feature Set for Sparse Matrices 3
2.1 Group load and reuse rates 8
2.2 Cache load and reuse rates 9

3 Feature Set for Graphs 12

4 Feature Set for Sparse Tensors 17

5 Conclusions 19

1 INTRODUCTION

The SparCity project is funded by EuroHPC JU (the European High Performance Computing Joint Undertaking) under the 2019 call of Extreme Scale Computing and Data Driven Technologies for research and innovation actions. SparCity aims to create a supercomputing framework that will provide efficient algorithms and coherent tools specifically designed for maximizing the performance and energy efficiency of sparse computations on emerging High Performance Computing (HPC) systems, while also opening up new usage areas for sparse computations in data analytics and deep learning.

Sparse computations are commonly found at the heart of many important applications, but at the same time it is extremely challenging to achieve high performance when performing the sparse computations. SparCity delivers a coherent collection of innovative algorithms and tools for enabling both high efficiency of sparse computations on emerging hardware platforms. More specifically, the objectives of the project are:

- to develop a comprehensive application and data characterization mechanism for sparse computation based on the state-of-the-art analytical and machine-learning-based performance and energy models,
- to develop advanced node-level static and dynamic code optimizations designed for massive and heterogeneous parallel architectures with complex memory hierarchy for sparse computation,
- to devise topology-aware partitioning algorithms and communication optimizations to boost the efficiency of system-level parallelism,
- to create digital SuperTwins of supercomputers to evaluate and simulate what-if hardware scenarios,
- to demonstrate the effectiveness and usability of the SparCity framework by enhancing the computing scale and energy efficiency of challenging real-life applications.
- to deliver a robust, well-supported and documented SparCity framework into the hands of computational scientists, data analysts, and deep learning end-users from industry and academia.

1.1 OBJECTIVES OF THIS DELIVERABLE

The objective of this deliverable is to provide the core set of features of sparse computation that can be used as an input to performance modeling and prediction, performance and energy optimizations of sparse computation.

The features are crucial for static, dynamic, and machine-learning based optimizations. By using these feature sets, one can select storage formats, reordering or partitioning algorithms, as well as an optimized solver for a given sparse matrix, tensor or graph. We also believe that this document and source code that computes the feature sets is valuable to other researchers who wish to develop their own performance modeling or optimizations based on these features.

1.2 WORKED PERFORMED

Sparse data is typically modeled as a graph, sparse matrix or a sparse tensor. Thus, naturally we organize the feature sets for sparse data structures into three categories: (i) feature set for sparse

matrices, (ii) feature set for graphs, and (iii) feature set for sparse tensors. Although there are a few common features for these categories, most of the features are specific to each group and thus they are discussed in separate sections.

We have performed structural analysis of the input (i.e. graph, matrix or tensors) such as their sparsity pattern, connectivity, number of non-zeros etc. by using the existing sparse matrix databases (such as SuiteSparse[1]) and graphs. For each analyzed sparse data structure, i.e., graph, matrix and tensor, the features identified are presented as a table along with their short descriptions and formulas to calculate them. In addition, the source codes to compute each of these features are being developed and will be made available to the public before the end of the project.

1.3 DEVIATIONS AND COUNTER MEASURES

There was no deviation from the work plan.

1.4 RESOURCES

It is likely that in the course of the project, new features will be added to the core list and their implementation will be improved. Because of this reason, in addition to this document, the links to the source code and feature set website are also available. The content of the links will be regularly updated.

- This document is open to public and available in the project website
  http://sparcity.eu.
- Source code that computes the feature sets is available at the project github repository
  https://github.com/sparcityeu
- Finally, the list of features as online tables is available at:
  https://sabanciparallelcomputing.github.io/sparsebase_website/

[1] Timothy A. Davis and Yifan Hu. *SuiteSparse Matrix Collection: The University of Florida Sparse Matrix Collection*. https://sparse.tamu.edu. 2011. DOI: 10.1145/2049662.2049663.

2 FEATURE SET FOR SPARSE MATRICES

A sparse matrix is a matrix in which most of the elements are zero. From mathematical point of view, there is no fundamental difference between a common (dense) and a sparse matrix in the corresponding single matrix, matrix-matrix and matrix-vector operations. However, operations using standard dense-matrix structures and algorithms are slow and inefficient when applied to large sparse matrices as processing and memory are wasted on the zeros. Sparse data is by nature easily compressed and requires significantly less storage. Thus from the numerical analysis, scientific computing and algorithmic implementation point of views, using of sparse matrices give the benefits of significant memory and storage space saving, as well as reduced computational costs due to a reduced number of zero-wasted arithmetical operations. Implementation of these benefits require using specialized algorithms and data structures that take advantage of the sparse structure of the matrix.

A number of sparse storage formats have been proposed over the years.[2] In the following, we briefly review these formats. We denote the number of rows, columns, and nonzero entries of the sparse matrix A with R, C, and NNZ, respectively. The *coordinate* (COO) format stores the matrix in three dense arrays of length NNZ called row, column, and value. The position of every nonzero value in the matrix is given explicitly. The *compressed sparse row* (CSR) format, which is the most popular format, compresses the row array to store the start positions of all rows in the corresponding column and value arrays. Formats like CSR and COO are *general*, i.e., they require $\mathcal{O}(NNZ)$ space for matrices with NNZ nonzeros. The *ELLPACK* (ELL) format stores a sparse matrix A as a dense rectangular matrix by shifting the nonzeros in each row to the left and zero-padding all rows that have fewer nonzeros than the maximum. ELL uses an index matrix to store the corresponding column index for every nonzero element. ELL thus eschews the need for a compressed row array. Furthermore, since the number of operations per row is known beforehand, the computation for each row can be unrolled completely and optimized for SIMD processing. The storage size of ELL depends on the maximum number of nonzeros in a row of matrix A, which is problematic for matrices with a large deviation in the number of nonzeros per row. The *hybrid* (HYB) format alleviates this problem by using ELL for storing most of the matrix A and COO to store additional entries in rows with many nonzeros. This reduces the required amount of padding while maintaining some advantages of ELL. Other formats, like *diagonal* (DIA), take advantage of specific sparsity patterns but can also take $\mathcal{O}(NNZ^2)$ space in the worst case.

Each of the sparse storage formats was designed with the goal of serving storage challenges as efficiently as possible for the wide variety of sparse matrix patterns. However, the resulting performance of the sparse pipeline depends also on the corresponding algorithms that use sparse matrices for underlying computations. Different sparse computation kernels are differently sensitive to the size and the sparsity pattern of the input matrix, as well to the target hardware architecture and used sparse storage format.[3] Although CSR is general and is the most-used

[2] Y. Saad. *SPARSKIT: a basic tool kit for sparse matrix computations*. Tech. rep. RIACS-TR-90-20. Research Institute for Advanced Computer Science, 1990; Nathan Bell and Michael Garland. *Efficient Sparse Matrix-Vector Multiplication on CUDA*. tech. rep. NVR-2008-004. NVIDIA, 2008; Bian Bian et al. "CSR2: A New Format for SIMD-accelerated SpMV". 2020, pp. 350–359; Weifeng Liu and Brian Vinter. "CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication". 2015, pp. 339–350; Yishui Li et al. "VBSF: a new storage format for SIMD sparse matrix–vector multiplication on modern processors". *The Journal of Supercomputing* 76 (2019), pp. 2063–2081; B. Xie et al. "CVR: Efficient Vectorization of SpMV on x86 Processors". 2018, pp. 149–162; Nathan Bell and Michael Garland. "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors". 2009, 18:1–18:11; Jee W. Choi, Amik Singh, and Richard W. Vuduc. "Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs". 2010, pp. 115–126.

[3] Bell and Garland, "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors"; Choi,

sparse storage format, there is *no single format that suits all sparsity patterns, target architectures and sparse computation kernels*.[4] Slowdowns amounting to two orders of magnitude from suboptimal sparse formats and sparse kernels can be observed in practice. To avoid such slowdowns, a large body of work has focused on *automatically* predicting the best sparse storage format and optimal sparse kernel for a given sparse matrix.[5]

Such the automated sparse storage format and sparse computation kernel selection requires a variety of statistical features that describes the structure of sparse pattern of the particular matrix and helps differentiate classes of sparse matrices.[6] The class-differentiation along with the raw features are used for the following selection of the best storage format for the particular matrix and selection of the optimal sparse computation kernel that uses the selected storage format in the sparse computation pipeline. Here a substantial number of different feature processing algorithms are implemented[7] utilizing various sets of features in a single- and multi-stage process of predicting sparse storage formats and sparse computation kernels selection.

In this work, we aim integration of the existing sparse matrix features in one big and easily reusable set of sparse features that could be used as an input for different automated sparse format and kernel selection approaches including but not limiting to classical machine-learning (ML) classifiers, such as Random Tree (RT), Random Forest (RF), Support Vector Machine (SVM), Linear Regression (LR), Multi-Layer Perceptron, etc.; and Deep Learning (DL) classifiers, such as Convolutional Neural Networks (CNN), Deep Convolutional Neural Networks (DCNN), Generative Adversarial Networks (GAN), etc. It is important to perform a selection of features that suits and could be used as an input for methodologically different algorithms. Therefore we have divided all the sparse features into three different categories following the listed related

Singh, and Vuduc, "Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs"; N. Sedaghati et al. "Automatic Selection of Sparse Matrix Representation on GPUs". 2015, pp. 99–108; D. Merrill and M. Garland. "Merge-Based Sparse Matrix-Vector Multiplication (SpMV) Using the CSR Storage Format". 2016; S. Williams et al. "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms". Vol. 35. Elsevier Science publishers B. V., 2009, pp. 178–194.

[4]Sedaghati et al., "Automatic Selection of Sparse Matrix Representation on GPUs".

[5]Yue Zhao et al. "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection". 2018, pp. 94–108; Yue Zhao et al. "Overhead-Conscious Format Selection for SpMV-Based Applications". 2018, pp. 950–959; Jiajia Li et al. "SMAT: An Input Adaptive Auto-tuner for Sparse Matrix-vector Multiplication". 2013, pp. 117–126; Guangming Tan, Junhong Liu, and Jiajia Li. "Design and Implementation of Adaptive SpMV Library for Multicore and Many-Core Architecture". 44.4 (2018), 46:1–46:25. ISSN: 0098-3500; J. C. Pichel and B. Pateiro-López. "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques". 2018, pp. 46–54; K. Li, W. Yang, and K. Li. "Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling". 26.1 (2014), pp. 196–205; W. Zhou et al. "Enabling Runtime SpMV Format Selection through an Overhead Conscious Method". 31.1 (2020), pp. 80–93; I. Nisa et al. "Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs". *International Workshop on Automatic Performance Tuning*. 2018, pp. 1056–1065; A. Benatia et al. "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU". 2016, pp. 496–505.

[6]Benatia et al., "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU".

[7]Sedaghati et al., "Automatic Selection of Sparse Matrix Representation on GPUs"; Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection"; Zhao et al., "Overhead-Conscious Format Selection for SpMV-Based Applications"; Li et al., "SMAT: An Input Adaptive Auto-tuner for Sparse Matrix-vector Multiplication"; Tan, Liu, and Li, "Design and Implementation of Adaptive SpMV Library for Multicore and Many-Core Architecture"; Pichel and Pateiro-López, "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques"; Li, Yang, and Li, "Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling"; Zhou et al., "Enabling Runtime SpMV Format Selection through an Overhead Conscious Method"; Nisa et al., "Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs"; Benatia et al., "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU"; Bell and Garland, "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors"; Choi, Singh, and Vuduc, "Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs"; Merrill and Garland, "Merge-Based Sparse Matrix-Vector Multiplication (SpMV) Using the CSR Storage Format"; Williams et al., "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms".

work in the field of automated sparse format and kernel selection, namely single-value features, one-dimensional (1D), and two-dimensional (2D) features. The corresponding sets of the selected features are presented in Tables 1, 2 and 3.

Table 1 *Sparse matrix single-value features*

Feature	Description	Formula
Rows	Number of matrix rows	R
Columns	Number of matrix columns	C
Size	Number of all matrix elements	S
Nonzeros	Number of nonzero elements	NNZ
Fraction of nonzeros	Fraction of nonzero elements	$frac_{NNZ} = NNZ/S$
Row average	Average number of nonzeros per row	row_{avg}
Row median	Median number of nonzeros per row	row_{med}
Row minimum	Minimum number of nonzeros per row	row_{min}
Row maximum	Maximum number of nonzeros per row	row_{max}
Row STD	Standard deviation of number of nonzeros per row	row_{std}
Row delta max	Difference between maximum and average number of nonzeros per row	$row_{\Delta max} = row_{max} - row_{avg}$
Row delta min	Difference between average and maximum number of nonzeros per row	$row_{\Delta min} = row_{avg} - row_{max}$
Row variation	The coefficient of variation of nonzero elements per row	$row_{var} = row_{std}/row_{avg}$

Short row RMS	Root mean square of difference between row nonzero counts which are less than $\mathrm{row}_{\mathrm{avg}}$	
Long row RMS	Root mean square of difference between row nonzero counts which are greater than $\mathrm{row}_{\mathrm{avg}}$	
Row-diagonal average	Average distance of nonzeros elements from the main diagonal	$\mathrm{drow}_{\mathrm{avg}}$
Row-diagonal median	Median distance of nonzeros elements from the main diagonal	$\mathrm{drow}_{\mathrm{med}}$
Row-diagonal minimum	Minimum distance of nonzeros elements from the main diagonal	$\mathrm{drow}_{\mathrm{min}}$
Row-diagonal maximum	Maximum distance of nonzeros elements from the main diagonal	$\mathrm{drow}_{\mathrm{max}}$
Row-diagonal STD	Standard deviation of distance of nonzeros elements from the main diagonal	$\mathrm{drow}_{\mathrm{std}}$
Row-diagonal delta max	Difference between maximum and average distance of nonzeros elements from the main diagonal	$\mathrm{drow}_{\Delta\mathrm{max}} = \mathrm{drow}_{\mathrm{max}} - \mathrm{drow}_{\mathrm{avg}}$
Row-diagonal delta min	Difference between average and maximum distance of nonzeros elements from the main diagonal	$\mathrm{drow}_{\Delta\mathrm{min}} = \mathrm{drow}_{\mathrm{avg}} - \mathrm{drow}_{\mathrm{max}}$
Row-diagonal variation	The coefficient of variation of nonzero elements per row	$\mathrm{drow}_{\mathrm{var}} = \mathrm{drow}_{\mathrm{std}}/\mathrm{drow}_{\mathrm{avg}}$
Cross-row average	Average cross-row distance of nonzeros elements from the main diagonal	$\mathrm{xrow}_{\mathrm{avg}}$
Cross-row median	Median cross-row distance of nonzeros elements from the main diagonal	$\mathrm{xrow}_{\mathrm{med}}$
Cross-row minimum	Minimum cross-row distance of nonzeros elements from the main diagonal	$\mathrm{xrow}_{\mathrm{min}}$
Cross-row maximum	Maximum cross-row distance of nonzeros elements from the main diagonal	$\mathrm{xrow}_{\mathrm{max}}$

Cross-row STD	Standard cross-row deviation of distance of nonzeros elements from the main diagonal	$xrow_{std}$
Cross-row delta max	Difference between cross-row maximum and average distance of nonzeros elements from the main diagonal	$xrow_{\Delta max} = xrow_{max} - xrow_{avg}$
Cross-row delta min	Difference between cross-row average and maximum distance of nonzeros elements from the main diagonal	$xrow_{\Delta min} = xrow_{avg} - xrow_{max}$
Cross-row variation	The coefficient of cross-row variation of nonzero elements per row	$xrow_{var} = xrow_{std}/xrow_{avg}$
Group load rate	Group load rate	$group_{load}$ (see 2.1)
Group reuse rate	Group reuse rate	$group_{reuse}$ (see 2.1)
Cache load rate	Cache load rate	$cache_{load}$ (see 2.2)
Cache reuse rate	Cache reuse rate	$cache_{reuse}$ (see 2.2)

Table 2 *Sparse matrix 1D features*

Feature	**Description**	**Formula**
Row histogram	Nonzero row elements distance histogram	$hist_{row}$
Row histogram downscaled (size x)	Downscaled representation of nonzero row elements distance histogram	Separately stored sequential iterations of downscale of the $hist_{row}$ by factor of 2 until $x >= 16$
Row histogram normalized (size x)	Downscaled and size-normalized representation of nonzero row elements distance histogram	Separately stored sequential iterations of downscale of the $hist_{row}$ to the nearest smaller size $x = 2^n$ until $x >= 16$

Table 3 *Sparse matrix 2D features*

Feature	**Description**	**Formula**

Distribution downscaled (size x)	Downscaled bitmap representation of nonzero elements distribution	Separately stored sequential iterations of downscale of the bitmap representation of nonzero elements distribution by factor of 2 starting from x <= 4096 until x >= 16
Distribution normalized (size x)	Downscaled and size-normalized bitmap representation of nonzero elements distribution	Separately stored sequential iterations of downscale of the bitmap representation of nonzero elements distribution to the nearest smaller size $x = 2^n$ from x <= 4096 until x >= 16
Density downscaled (size x)	Downscaled representation of nonzero elements density	Separately stored sequential iterations of downscale of the representation of nonzero elements density by factor of 2 starting from x <= 4096 until x >= 16
Density normalized (size x)	Downscaled and size-normalized representation of nonzero elements density	Separately stored sequential iterations of downscale of the representation of nonzero elements density to the nearest smaller size $x = 2^n$ from x <= 4096 until x >= 16

2.1 GROUP LOAD AND REUSE RATES

Group load and reuse features represent a low computation cost simplified one-line cache access simulation implemented for a simple sparse matrix to vector multiplication (SpMV) algorithm. The main purpose of these features is to estimate the nonzero elements locality in the rows of the sparse matrix accessed in the sequential order and analyzed independently. We assume the single available cache line consists of N elements and all the N elements loads from the memory into the cache line at once. This single cache line load counts as one cache line load event. In this simulation the cache line is associated with the dense vector that is used in the simulated SpMV operation. Cache line load is initiated by the first nonzero element of the sparse matrix requesting the particular element of the dense vector for the SpMV operation. This first accessed dense vector element loaded as the first element in the cache line. The rest of the cache line is assumed to be loaded with the following dense vector elements. If the next sparse matrix nonzero element requesting the dense vector element that is currently already loaded to the cache line, then this access request is counted as a cache line reuse event and cache remaining unchanged. In contrast, if the next sparse matrix nonzero element requesting the dense vector element that is not presented in the current cache line, then this access request is treated as a cache miss and the described cache line load procedure is initiated again using the missing element as the first element of the newly loaded cache line. After the completion of whole SpMV simulation, the cache load and cache reuse event count are divided by the total number of nonzeros and become the resulting group load and reuse rate feature values respectively.

Group load and reuse rate features are easy to compute and do not require any complex cache line handling. The simulation algorithm uses CSR representation of the sparse matrix and is defined as the following:

```
group_load_cnt = 0
group_reuse_cnt = 0
for (row = 0; row < rows; row ++) {
  nz = -1
  nc = csr.rows_offs[row + 1] - csr.rows_offs[row]
  for (i = 0; i < nc; i ++) {
    int c = csr.cols_data[csr.rows_offs[row] + i]
```

```
    if (nz >= 0 && c - nz < cache_line_size) {
      group_reuse_cnt ++
    } else {
      group_load_cnt ++
      nz = c
    }
  }
}
group_load_rate = group_load_cnt / csr.nelem
group_reuse_rate = group_reuse_cnt / csr.nelem
```

An example of the group load and reuse rates computation is shown in Figure 1.

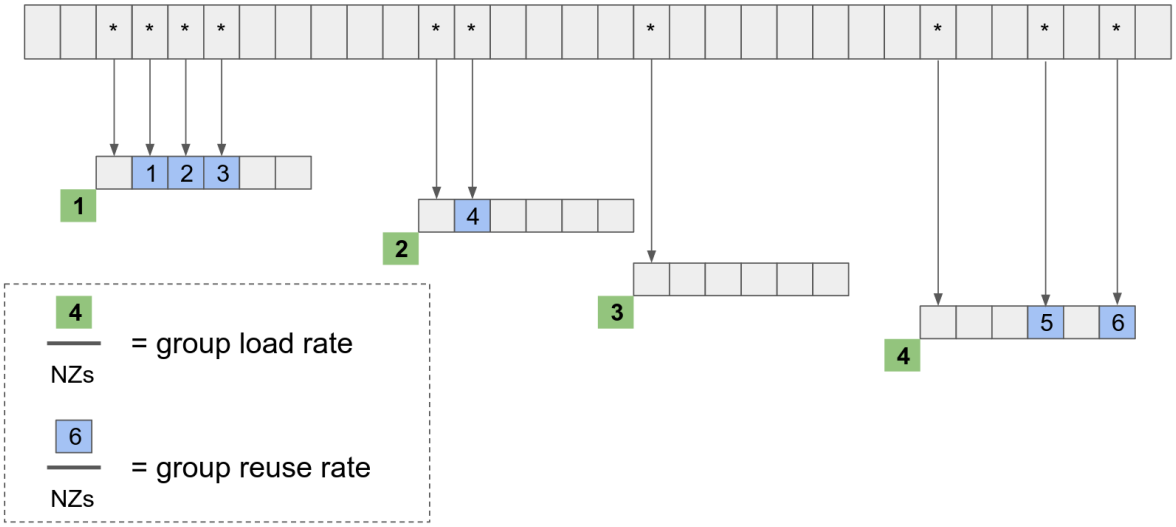

Figure 1 *An example of the group load and reuse rates computation.*

2.2 CACHE LOAD AND REUSE RATES

Cache load and reuse features represent a simplified multi-line cache access simulation implemented for a simple SpMV algorithm. The main purpose of these features is estimate the nonzero elements locality in the rows of the sparse matrix accessed in the sequential order, as well as the cross-row nonzero elements locality and spatial grouping patterns. We assume cache consist of M cache lines each consists of N elements and all the N elements loads from the memory into the cache line at once during a cache line load operation. This single cache line load counts as one cache line load event. In this simulation the cache is associated only with the dense vector that is used in the simulated SpMV operation. Cache line load is initiated by the first nonzero element of the sparse matrix requesting the particular element of the dense vector for the SpMV operation. This first accessed dense vector element loaded as the first element in the first available cache line. The rest of the cache line is assumed to be loaded with the following dense vector elements. If the next sparse matrix nonzero element requesting the dense vector element that is currently already loaded to any of the already loaded cache lines, then this access request is counted as a cache line reuse event and cache remaining unchanged. The cache line that contained the cached element is moved to the top of cache lines access list. If the next sparse matrix nonzero element

requesting the dense vector element that is not presented in any of the current cache lines, then this access request is treated as a cache miss. The simulation algorithm then lookup the cache lines access list and invalidates the lest recent accessed cache line and the described cache line load procedure is initiated again for the invalidated cache line using the missing element as the first element of the newly loaded cache line. After the completion of whole SpMV simulation, the cache load and cache reuse event count are divided by the total number of nonzeros and become the resulting cache load and reuse rate feature values respectively.

Compared the the group load and reuse rates, cache load and reuse rates gives better estimation of cache usage patterns in SpMV operations, but they are more computationally intensive. The simulation algorithm uses CSR representation of the sparse matrix and is defined as the following:

```
cache_line = [cache_lines_num](-1)
cache_time = [cache_lines_num](0)
cache_load_cnt = 0
cache_reload_cnt = 0
for (row = 0; row < rows; row ++) {
  nc = csr.rows_offs[row + 1] - csr.rows_offs[row]
  for (i = 0; i < nc; i ++) {
    c = csr.cols_data[rows_offs[row] + i]
    element_cached = false
    for (j = 0; j < cache_lines_num; j ++) {
      if (cache_line[j] >= 0 && c >= cache_line[j] && c < cache_line[j] + cache_line_size) {
      cache_reload_cnt ++
      cache_time[j] = 1
      element_cached = true
      break
    }
    if (!element_cached) {
      line_invalidate = 0
      line_invalidate_time = cache_time[0]
      for (j = 1; j < cache_lines_num; j ++) {
        if (cache_time[j] < line_invalidate_time) {
          line_invalidate = j
          line_invalidate_time = cache_time[j]
        }
      }
      cache_line[line_invalidate] = c
      cache_time[line_invalidate] = 1
      cache_load_cnt ++
    }
    for (j = 0; j < cache_lines_num; j ++) {
        cache_time[j] --
    }
  }
}
cache_load_rate = cache_load_cnt / csr.nelem;
cache_reuse_rate = cache_reuse_cnt / csr.nelem;
```

An example of the cache load and reuse rates computation is shown in Figure 2.

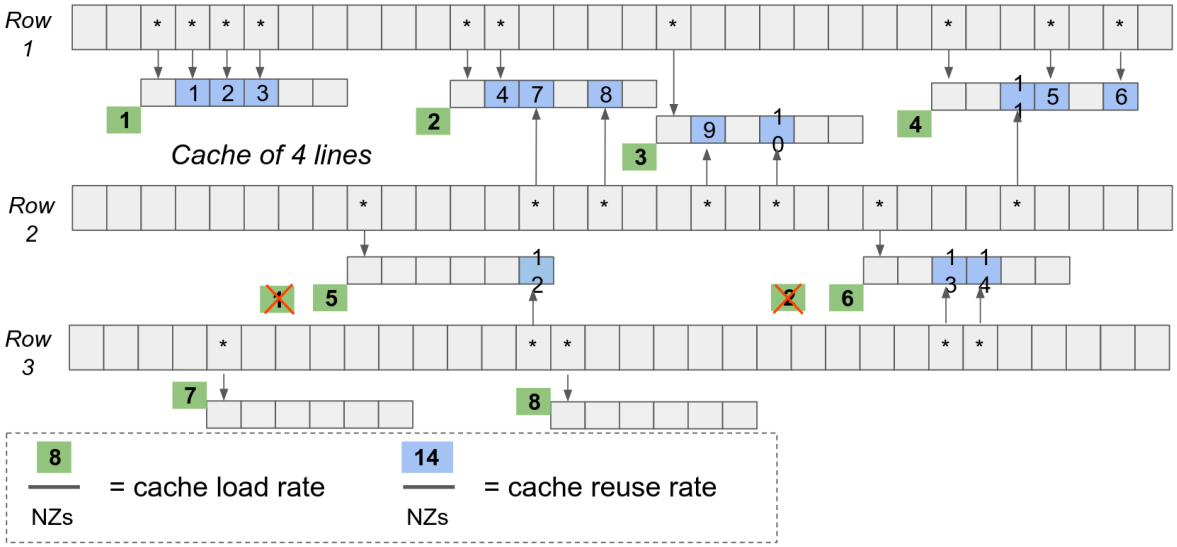

Figure 2 *An example of the cache load and reuse rates computation.*

3 FEATURE SET FOR GRAPHS

Graphs are widely used to model the interactions in real-world data such as social networks, citations in academic literature, protein-protein interactions, web data, and many more. The inherent topological information in graph models, i.e., who are connected to who, hides important and useful information for each edge, vertex, and for the graph itself. These information can be quantified via various metrics which have been defined and investigated by the literature. Some of these metrics are trivial, e.g., number of vertices in the graphs, some are easy to compute, e.g., maximum vertex degree, and some are harder, e.g., betweenness centrality for all the vertices. These metrics present valuable information and can be/have been used for different purposes in practice.

Recently, leveraging graphs in Machine Learning (ML) and Deep Learning (DL) tasks like link prediction, node/graph classification, and anomaly detection became popular in applications from various domains. Graphs, as mentioned above, are powerful data models: however, when they are highly irregular and sparse, they are also harder to be used by traditional ML tools and algorithms. Note that on the contrary, a tabular representation, such as a (0/1 or weighted) adjacency matrix, is more suitable for ML usage. Unfortunately, it is infeasible to represent a large-scale graph in such a classical format due to sheer physical limitations. Hence, one needs other approaches to make graphs useful for ML tasks.

One practical solution to this problem is *graph embedding* which is the process of embedding n graph's vertices into a d-dimensional space while preserving its spatial properties, where $d \ll n$. In other words, graph embedding generates a feature vector of length d for each vertex in the graph. These feature vectors can be used for tasks like node classification, or combined with a similarity function such as the Hadamard product to be used for edge-based tasks, such as link prediction. Although *embedding* is useful in practice, it is a compute and memory intensive process. Parallelism can indeed be used to alleviate the computation requirement. Previous work, most notably[8] and many more, extensively investigated shared- and distributed-memory approaches to accelerate the embedding process. Out of these efforts, GPU-based implementations reported remarkable improvements on runtime. Unfortunately, GPUs are restricted memory devices with a limited bandwidth between the host and the device. When only the embedding matrix is used as the input, having a robust and accurate model requires a large number of features. This increases the memory requirements significantly. For a graph with 256M vertices and $d = 128$, the embedding matrix contains 32.8 billion entries. Assuming 4 bytes per entry, the memory requirement, excluding the graph itself, is 128GB, which is more than the memory of the state-of-the-art GPUs on the market.

Another approach to extract features for the graph elements is by using the node/edge metrics, i.e., the approach we are focusing in this project. One popular ML application is estimating the performance of a kernel/algorithm on a graph, or tuning the parameters of the execution by using these metrics. In addition, these metrics can be used both to shrink the embedding matrix by using less features, and to increase the accuracy with additional non-embedding information. Shrinking the embedding matrix is an important problem since this will help to use

[8] B. Perozzi, R. Al-Rfou, and S. Skiena. "DeepWalk: Online Learning of Social Representations". *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*. NY, USA: ACM, 2014, pp. 701–710; Da Zheng et al. "DGL-KE: Training Knowledge Graph Embeddings at Scale". *CoRR* abs/2004.08532 (2020); A. Taha, A. A. Amro, and K. Kamer. "GOSH: Embedding Big Graphs on Small Hardware". *49th International Conference on Parallel Processing - ICPP (ICPP '20), August 17-20, 2020, Edmonton, AB, Canada*. 2020. DOI: 10.1145/3404397.3404456; Z. Zhu et al. "GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding". *The World Wide Web Conference*. CA, USA: ACM, 2019, pp. 2494–2504; Adam Lerer et al. *PyTorch-BigGraph: A Large-scale Graph Embedding System*. 2019.

the GPUs better. In fact, recently, Mara et al.[9] benchmarked various graph embedding algorithms on link prediction and found that some simple graph metrics, i.e., *common neighbors*, *Jaccard coefficient*, *Adamic Adar*, *research allocation index*, and *preferential attachment*, outperform most of the embedding methods, which shows the effectiveness of graph metrics on downstream tasks. Calculating these metrics can be cheaper than graph embedding especially when both compute and memory requirements are considered. For instance, compared to $d = 128$ features per vertex, the embedding size requirements are reduced by a factor of 25, which also speeds up the ML pipeline of the downstream task. In this project, we will be interested in a set of graph metrics which are summarized by Tables 4 and 5. The tables were created using multiple sources.[10]

Table 4 *Notation used for the graph metrics.*

Symbol	Description
E	set of edges
V	set of vertices
A	adjacency matrix
W	weighted adjacency matrix
α	constant
β	constant
v	leading eigenvector
$d(x, y)$	shortest path distance from vertices x to y
$\sigma(x, y)$	number of shortest paths from vertices x to y
$\sigma(x, y\|z)$	number of shortest paths from vertices x to y that pass through z
$N(x)$	neighborhood of vertex x
$N_y(x)$	vertices that are at y distance from x (if $y = 1$, this is equal to $N(x)$)
$d(x)$	degree of x (number of neighbors of x)
$d_{out}(x)$	out degree of x (number of outgoing edges of x)
$d_{in}(x)$	in degree of x (number of incoming edges of x)

[9] Alexandru Mara, Jefrey Lijffijt, and Tijl De Bie. *Benchmarking Network Embedding Models for Link Prediction: Are We Making Progress?* 2020.

[10] Javier Martín Hernández and Piet Van Mieghem. *Classification of graph metrics*. 2011. URL: http://www.resumenet.eu/; Stuart Oldham et al. "Consistency and differences between centrality measures across distinct classes of networks". *PLoS ONE* 14 (7 2019). ISSN: 19326203. DOI: 10.1371/journal.pone.0220061; Zelin Wan, Virginia Tech, and Terrence J Moore. *A Survey on Centrality Metrics and Their Implications in Network Resilience*; Andry Alamsyah and Budi Rahardjo. *Social Network Analysis Taxonomy Based on Graph Representation*.

Table 5 *Graph metrics that will be used as features for nodes/edges.*

Feature	Description	Formula	Citation
Average Degree	Average degree of all vertices	$2\|E\|/\|V\|$	-
Diameter	Length of longest shortest path	$max_{v,u \epsilon V}(d(v,u))$	-
Minimum Degree	Minimum number of neighbors	$min_{i \epsilon V}\|N(i)\|$	-
Maximum Degree	Maximum number of neighbors	$max_{i \epsilon V}\|N(i)\|$	-
Degree Centrality	Number of neighbors of a vertex	$C_{deg}(i) = \|N(i)\| = d(i)$	Hanneman & Riddle, 2005[11]
In/Out Degree Centralities	Number of neighbors of a vertex	$C_{indeg}(i) = d_{in}(i), C_{outdeg}(i) = d_{out}(i)$	Wasserman & Faust, 1994[12]
Betweenness Centrality	Centrality measure based on the percentage of shortest paths passing through the node	$C_{btw}(i) = \sum_{s,t \epsilon V} \sigma(s,t\|i)/\sigma(s,t)$	Freeman, 1977[13]
K-Betweenness Centrality	Approximation of betweenness centrality using only paths of length k or less	$C_{k-btw}(i) = \sum_{s,t \epsilon V, d(s,t)<k} \sigma(s,t\|i)/\sigma(s,t)$	Ercsey-Ravasz & Toroczkai, 2010[14]
Closeness Centrality	The reciprocal of farness, or the inverse proportion of the average distance to all other nodes in the network	$C_{cls}(i) = 1/\sum_{j \epsilon V} d(i,j)$	Bavelas, 1950[15]
K-Closeness Centrality	Approximation of closeness centrality using only paths of length k or less	$C_{k-cls}(i) = 1/\sum_{j \epsilon V, d(i,j)<k} d(i,j)$	-

[11] Robert A Hanneman and Mark Riddle. *Introduction to social network methods*. 2005.

[12] Stanley Wasserman, Katherine Faust, et al. "Social network analysis: Methods and applications" (1994).

[13] Linton C Freeman. "A set of measures of centrality based on betweenness". *Sociometry* (1977), pp. 35–41.

[14] Mária Ercsey-Ravasz and Zoltán Toroczkai. "Centrality scaling in large networks". *Physical review letters* 105.3 (2010), p. 038701.

[15] Alex Bavelas. "Communication patterns in task-oriented groups". *The journal of the acoustical society of America* 22.6 (1950), pp. 725–730.

Eigenvector Centrality	Each node has proportional value to the sum of the score its neighbors.	$C_{eig}(i) = \lambda^{-1} \sum_{j \epsilon V} A_{ij} v_j$	Bonacich & Llyod, 2001[16]
Katz Centrality	Each node has given small amount of centrality "for free" regardless of its position in the network	$C_{katz}(i) = \alpha \sum_{j \epsilon V} (A_{ij} v_j) + \beta$	Newman, 2010[17]
PageRank Centrality	Each node has given the rank based on network neighbors proportional to their centrality divided by their out-degree	$C_{pagerank}(i) = \alpha \sum_{j \epsilon V} (A_{ij} v_j / d_{out}(i)) + \beta$	Brin & Rage, 1998[18]
Sum of Degrees	Sum of the degrees of the two vertices at either end of the edge	$\lvert N(i) \rvert + \lvert N(j) \rvert$	-
Authority	Authority is a node that contain useful information on topic of interest	$M_{auth}(i) = \beta \sum_{j} A_{ij} M_{hub}(j)$	Kleinberg et.al, 1998[19]
Hubs	Hub is a node that tells us where the best authority to be found.	$M_{hub}(i) = \alpha \sum_{j} A_{ij} M_{auth}(j)$	Kleinberg et.al, 1998[20]
Volume Centrality	Centrality based on the degrees of neighbors within a certain reach (say k)	$C_{volume}(i) = \sum_{j \epsilon N(i), d(i,j) < k} d(j)$	Kim & Yoneki, 2012[21]
Contribution Centrality	A refined variant of eigenvector centrality to account similarity of the neighbors that link to a node	$C_{contrib} = 1/\lambda \sum_{u \epsilon N(v)} W_{uv} C_{contrib}(u)$	Alvarez-Socorro, et.al., 2015[22]

[16] Phillip Bonacich and Paulette Lloyd. "Eigenvector-like measures of centrality for asymmetric relations". *Social networks* 23.3 (2001), pp. 191–201.

[17] Mark Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[18] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual web search engine". *Computer networks and ISDN systems* 30.1-7 (1998), pp. 107–117.

[19] Jon M Kleinberg et al. "Authoritative sources in a hyperlinked environment." *SODA*. vol. 98. Citeseer. 1998, pp. 668–677.

[20] Ibid.

[21] Hyoungshick Kim and Eiko Yoneki. "Influential neighbours selection for information diffusion in online social networks". *2012 21st international conference on computer communications and networks (ICCCN)*. IEEE. 2012, pp. 1–7.

[22] AJ Alvarez-Socorro, GC Herrera-Almarza, and LA Gonzalez-Diaz. "Eigencentrality based on dissimilarity measures reveals central nodes in complex networks". *Scientific reports* 5.1 (2015), pp. 1–10.

Clustering Coefficient	Probability of node's neighbors being neighbors of each other	$C_{cluster}(v) = 1/(\|N(v)\|^2 - N(v)) \sum_{r,s \epsilon N(v)} A_{rs}$	Burt, 1995[23]
Hopcount	Given two nodes, the number of edges in the shortest path between the two	$hop(u,v) = d(u,v) - 1$	Van Mieghem, 2009[24]
Eccentricity	The eccentricity of a node is the longest Hopcount between the node and any other node in the graph	$ecc(u) = max_{v \epsilon V}(hop(u,v))$	Van Mieghem, 2009[25]
Persistence	Smallest number of edges whose removal causes the graph the be disconnected or increase in diameter	–	Boesch et.al., 1981[26]
Expansion	The average fraction of nodes in the graph that fall within a ball of radius h (in hops) centered at a random node in the topology	$expansion(v,h) = 1/(\|V\|^2)\|N_h(v)\|$	Van Mieghem, 2009[27]
Central Point Dominance	A measure of the maximum betweenness of any point in the graph (it equals 0 for complete graphs and 1 for star graphs)	$cpd = 1/(N-1) \sum_{v \epsilon V} (max_{u \epsilon V}(C_{btw}(u)) - C_{btw}(v))$	Freeman, 1977[28]

[23] RS Burt. *Structural Holes: The Social Structure of Competition, Harvard Univ.* 1995.

[24] Piet Van Mieghem. *Performance analysis of communications networks and systems*. Cambridge University Press, 2009.

[25] Ibid.

[26] Frank T Boesch, Frank Harary, and Jerald A Kabell. "Graphs as models of communication network vulnerability: Connectivity and persistence". *Networks* 11.1 (1981), pp. 57–63.

[27] Van Mieghem, *Performance analysis of communications networks and systems*.

[28] Freeman, "A set of measures of centrality based on betweenness".

4 FEATURE SET FOR SPARSE TENSORS

Tensors are multi-dimensional arrays which can have three or more dimensions. A tensor with M dimensions is called an Mth order or M-mode tensor. Mode m of a tensor refers to its mth dimension. Fibers are defined as the one-dimensional sections of a tensor obtained by fixing all but one index. Slices are two-dimensional sections of a tensor obtained by fixing every index but two. Figure 3 depicts sample horizontal (mode-1), lateral (mode-2) and frontal (mode-3) slices along with column (mode-1), row (mode-2) and tube (mode-3) fibers of a third-order tensor.

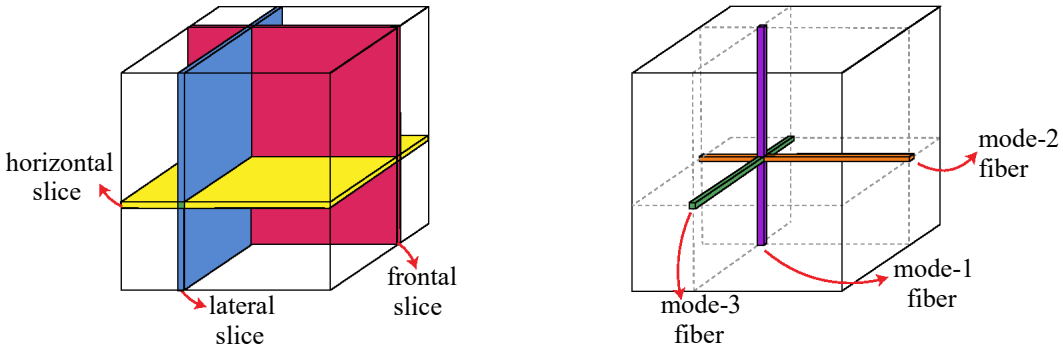

Figure 3 *Sample slice and fibers of a 3rd-order tensor.*

To reveal the relationship of data across different modes, tensor decomposition techniques are widely used. Canonical polyadic decomposition (CPD) and Tucker decomposition are the most two popular ones among them. In Tucker decomposition, a tensor is decomposed into a much smaller core tensor and a set of matrices; whereas in CPD, tensor is factorized as a set of rank-1 tensors. CPD can be considered as a generalization of matrix singular value decomposition (SVD) method for higher-order tensors. CPD is often solved by the Alternating Least Square (ALS) algorithm, in which the bottleneck operation the Matricized Tensor Times Khatri-Rao Product (MTTKRP) requiring operations on very large sizes of matrices.

The features of a sparse tensor has the capacity to reveal the most suitable storage format, partitioning or reordering method that fits best with that tensor if well-examined. For example, tensor features can be used for automatically predicting the optimal storage format of a tensor for operating MTTKRP.[29] As a first step, we plan to extract an extensive set of features for sparse tensors on both CPU and GPU.

Table 6 shows the sparse tensor features that are planned to be extracted, which can be extended as needed. A tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \cdots \times I_M$ is assumed to be given as an input. We denote the set of fibers and slices of the tensor as $\mathcal{F}$ and $\mathcal{S}$, respectively. $\mathcal{F}_m$ refers to the set of fibers in mode-m, whereas $\mathcal{S}_{k,\ell}$ refers to the set of slices obtained by fixing indices in modes k and ℓ. In the formulas, $\texttt{nnz}(\cdot)$ and $\texttt{nnzFib}(\cdot)$ are used as functions returning the number of nonzeros and number of nonzero fibers, respectively. Here, the term "nonzero fibers" refers to the fibers that include at least one nonzero value.

Most of the features given in Table 6 are adopted from the study.[30] In the original work, the fiber ratio is found as the ratio of number of fibers over the number of mode-1 fibers, yet we adopt this feature as the ratio of number of fibers over the minimum number of fibers along

[29] Qingxiao Sun et al. "Input-aware Sparse Tensor Storage Format Selection for Optimizing MTTKRP". *IEEE Transactions on Computers* (2021).

[30] Ibid.

Table 6 *Sparse tensor features. *Due to the complex formula, the formula is omitted but described in the text.*

Feature	Description	Formula
size_m	Tensor mode size (in mode m)	$\mathrm{I_m}$
fiberCnt	Number of fibers	$\lvert\mathcal{F}\rvert = \sum \lvert\mathcal{F}_\mathrm{m}\rvert$
sliceCnt	Number of slices	$\lvert\mathcal{S}\rvert = \sum \lvert\mathcal{S}_{\mathrm{k},\ell}\rvert$
fiberRatio	The ratio of fibers	$\lvert\mathcal{F}\rvert/\mathtt{min}(\lvert\mathcal{F}_\mathrm{m}\rvert)$
sliceRatio	The ratio of slices	$\lvert\mathcal{S}\rvert/\mathtt{min}(\lvert\mathcal{S}_{\mathrm{k},\ell}\rvert)$
nnz	Number of nonzeros	$\mathtt{nzz}(\mathcal{X})$
density	Density of nnz in the tensor	$\mathtt{nzz}(\mathcal{X})/\prod \mathrm{I_m}$
avgNnzPerSlice	Average nnz per slice	$\sum \mathtt{nnz}(\mathcal{S}_{\mathrm{k},\ell})/\lvert\mathcal{S}\rvert$
maxNnzPerSlice	Maximum nnz per slice	$\mathtt{max}(\mathtt{nnz}(\mathcal{S}_{\mathrm{k},\ell}))$
minNnzPerSlice	Minimum nnz per slice	$\mathtt{min}(\mathtt{nnz}(\mathcal{S}_{\mathrm{k},\ell}))$
adjNnzPerSlice	Average nnz difference of adjacent slices	*
devNnzPerSlice	The deviation of nnz per slice	*
cvNnzPerSlice	The coefficient of variation of nnz per slice	*
avgNnzPerFiber	Average nnz per fiber	$\sum \mathtt{nnz}(\mathcal{F}_\mathrm{m})/\lvert\mathcal{F}\rvert$
maxNnzPerFiber	Maximum nnz per fiber	$\mathtt{max}(\mathtt{nnz}(\mathcal{F}_\mathrm{m}))$
minNnzPerFiber	Minimum nnz per fiber	$\mathtt{min}(\mathtt{nnz}(\mathcal{F}_\mathrm{m}))$
adjNnzPerFiber	Average nnz difference of adjacent fibers	*
devNnzPerFiber	The deviation of nnz per fiber	*
cvNnzPerFiber	The coefficient of variation of nnz per fiber	*
aveFibersPerSlice	Average number of nonzero fibers per slice	$\sum \mathtt{nnzFib}(\mathcal{S}_{\mathrm{k},\ell})/\lvert\mathcal{S}\rvert$
maxFibersPerSlice	Maximum number of nonzero fibers per slice	$\mathtt{max}(\mathtt{nnzFib}(\mathcal{S}_{\mathrm{k},\ell}))$
minFibersPerSlice	Minimum number of nonzero fibers per slice	$\mathtt{min}(\mathtt{nnzFib}(\mathcal{S}_{\mathrm{k},\ell}))$
adjFibersPerSlice	Average nonzero fiber count difference of adjacent slices	*
devFibersPerSlice	The deviation of number of nonzero fibers per slice	*
cvFibersPerSlice	The coefficient of variation of nonzero fiber count per slice	*

any dimension to get a more wide and generalized view of the tensor. The slice ratio is defined similarly in terms of number of slices. The deviation of nonzero count per fiber (or slice) means the difference between the maximum and the minimum nonzero count for all fibers (or slices). We also add the coefficient of variation metric for both slices and fibers to acquire an intuition regarding the distribution of nonzeros among different slices or fibers. Coefficient of variation is defined as the ratio of standard deviation over mean.

5 CONCLUSIONS

The first technical deliverable of the SparCity project provided an extensive set of features of sparse computation by focusing on the three most common sparse data structures, i.e., sparse matrices, graphs and sparse tensors. Given the uniqueness of the features for each analyzed data structure, we organized the feature sets into three categories: *i*) feature set for sparse matrices, *ii*) feature set for graphs, and *iii*) feature set for sparse tensors.

We identified more than 100 core features for sparse matrices, graphs and tensors, for which we also provided in-depth descriptions, formulas, as well as the source codes to compute each of these features. This information will be made available to the public before the end of the project via the SparCity project website (http://sparcity.eu) and the SparCity github repository (https://github.com/sparcityeu). It is likely that in the course of the project, new features will be added to the core list and their implementation will be improved.

The outcomes of this deliverable can be used as an input to performance modeling and prediction of sparse computations, as well as for static, dynamic, and machine-learning based optimizations. For example, by relying on the identified set of features, one can select storage formats, reordering or partitioning algorithms, as well as an optimized solver for a given sparse matrix, tensor or graph. We also believe that other researchers will find the feature sets invaluable for their research and contribute to them in terms of both development and use-cases.

REFERENCES

Alamsyah, Andry and Budi Rahardjo. *Social Network Analysis Taxonomy Based on Graph Representation*.

Alvarez-Socorro, AJ, GC Herrera-Almarza, and LA Gonzalez-Diaz. "Eigencentrality based on dissimilarity measures reveals central nodes in complex networks". *Scientific reports* 5.1 (2015), pp. 1–10.

Bavelas, Alex. "Communication patterns in task-oriented groups". *The journal of the acoustical society of America* 22.6 (1950), pp. 725–730.

Bell, Nathan and Michael Garland. *Efficient Sparse Matrix-Vector Multiplication on CUDA*. Tech. rep. NVR-2008-004. NVIDIA, 2008.

— "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors". 2009, 18:1–18:11.

Benatia, A. et al. "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU". 2016, pp. 496–505.

Bian, Bian et al. "CSR2: A New Format for SIMD-accelerated SpMV". 2020, pp. 350–359.

Boesch, Frank T, Frank Harary, and Jerald A Kabell. "Graphs as models of communication network vulnerability: Connectivity and persistence". *Networks* 11.1 (1981), pp. 57–63.

Bonacich, Phillip and Paulette Lloyd. "Eigenvector-like measures of centrality for asymmetric relations". *Social networks* 23.3 (2001), pp. 191–201.

Brin, Sergey and Lawrence Page. "The anatomy of a large-scale hypertextual web search engine". *Computer networks and ISDN systems* 30.1-7 (1998), pp. 107–117.

Burt, RS. *Structural Holes: The Social Structure of Competition, Harvard Univ*. 1995.

Choi, Jee W., Amik Singh, and Richard W. Vuduc. "Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs". 2010, pp. 115–126.

Davis, Timothy A. and Yifan Hu. *SuiteSparse Matrix Collection: The University of Florida Sparse Matrix Collection*. https://sparse.tamu.edu. 2011. DOI: 10.1145/2049662.2049663.

Ercsey-Ravasz, Mária and Zoltán Toroczkai. "Centrality scaling in large networks". *Physical review letters* 105.3 (2010), p. 038701.

Freeman, Linton C. "A set of measures of centrality based on betweenness". *Sociometry* (1977), pp. 35–41.

Hanneman, Robert A and Mark Riddle. *Introduction to social network methods*. 2005.

Hernández, Javier Martín and Piet Van Mieghem. *Classification of graph metrics*. 2011. URL: http://www.resumenet.eu/.

Kim, Hyoungshick and Eiko Yoneki. "Influential neighbours selection for information diffusion in online social networks". *2012 21st international conference on computer communications and networks (ICCCN)*. IEEE. 2012, pp. 1–7.

Kleinberg, Jon M et al. "Authoritative sources in a hyperlinked environment." *SODA*. Vol. 98. Citeseer. 1998, pp. 668–677.

Lerer, Adam et al. *PyTorch-BigGraph: A Large-scale Graph Embedding System*. 2019.

Li, Jiajia et al. "SMAT: An Input Adaptive Auto-tuner for Sparse Matrix-vector Multiplication". 2013, pp. 117–126.

Li, K., W. Yang, and K. Li. "Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling". 26.1 (2014), pp. 196–205.

Li, Yishui et al. "VBSF: a new storage format for SIMD sparse matrix–vector multiplication on modern processors". *The Journal of Supercomputing* 76 (2019), pp. 2063–2081.

Liu, Weifeng and Brian Vinter. "CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication". 2015, pp. 339–350.

Mara, Alexandru, Jefrey Lijffijt, and Tijl De Bie. *Benchmarking Network Embedding Models for Link Prediction: Are We Making Progress?* 2020.

Merrill, D. and M. Garland. "Merge-Based Sparse Matrix-Vector Multiplication (SpMV) Using the CSR Storage Format". 2016.

Newman, Mark. *Networks: An Introduction*. Oxford University Press, 2010.

Nisa, I. et al. "Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs". *International Workshop on Automatic Performance Tuning*. 2018, pp. 1056–1065.

Oldham, Stuart et al. "Consistency and differences between centrality measures across distinct classes of networks". *PLoS ONE* 14 (7 2019). ISSN: 19326203. DOI: 10.1371/journal.pone.0220061.

Perozzi, B., R. Al-Rfou, and S. Skiena. "DeepWalk: Online Learning of Social Representations". *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*. NY, USA: ACM, 2014, pp. 701–710.

Pichel, J. C. and B. Pateiro-López. "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques". 2018, pp. 46–54.

Saad, Y. *SPARSKIT: a basic tool kit for sparse matrix computations*. Tech. rep. RIACS-TR-90-20. Research Institute for Advanced Computer Science, 1990.

Sedaghati, N. et al. "Automatic Selection of Sparse Matrix Representation on GPUs". 2015, pp. 99–108.

Sun, Qingxiao et al. "Input-aware Sparse Tensor Storage Format Selection for Optimizing MTTKRP". *IEEE Transactions on Computers* (2021).

Taha, A., A. A. Amro, and K. Kamer. "GOSH: Embedding Big Graphs on Small Hardware". *49th International Conference on Parallel Processing - ICPP (ICPP '20), August 17-20, 2020, Edmonton, AB, Canada*. 2020. DOI: 10.1145/3404397.3404456.

Tan, Guangming, Junhong Liu, and Jiajia Li. "Design and Implementation of Adaptive SpMV Library for Multicore and Many-Core Architecture". 44.4 (2018), 46:1–46:25. ISSN: 0098-3500.

Van Mieghem, Piet. *Performance analysis of communications networks and systems*. Cambridge University Press, 2009.

Wan, Zelin, Virginia Tech, and Terrence J Moore. *A Survey on Centrality Metrics and Their Implications in Network Resilience*.

Wasserman, Stanley, Katherine Faust, et al. "Social network analysis: Methods and applications" (1994).

Williams, S. et al. "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms". Vol. 35. Elsevier Science publishers B. V., 2009, pp. 178–194.

Xie, B. et al. "CVR: Efficient Vectorization of SpMV on x86 Processors". 2018, pp. 149–162.

Zhao, Yue et al. "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection". 2018, pp. 94–108.

Zhao, Yue et al. "Overhead-Conscious Format Selection for SpMV-Based Applications". 2018, pp. 950–959.

Zheng, Da et al. "DGL-KE: Training Knowledge Graph Embeddings at Scale". *CoRR* abs/2004.08532 (2020).

Zhou, W. et al. "Enabling Runtime SpMV Format Selection through an Overhead Conscious Method". 31.1 (2020), pp. 80–93.

Zhu, Z. et al. "GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding". *The World Wide Web Conference*. CA, USA: ACM, 2019, pp. 2494–2504.