Efficient Integer Sparse Matrix Storage and Operations

Tim Nonet
Supervised by: Dr. Rahul Mazumder

October 8, 2019

https://www.mathcha.io/editor

Abstract

The main idea of this project is to create user friendly implementations of sparse matrices and their basic operations to allow for efficient storage and computation of SNP data.

1 Introduction

Storing many specific data sets, such as SNP, as dense matrices is quite inefficient due to the discrete and skewed distribution of entries in the data set.

1.1 Compressed Row Format (CSR)

Depending on the specific structure of a sparse matrix different formats for efficiently storage exist. In addition, the anticipated use of the matrix can make certain formats more beneficial than others [1]. Our need for fast matrix vector multiplication suggest the use of CSR format. However, the need for the transpose operation adds some disadvantages to CSR.

Advantages

1. Extremely quick matrix-vector multiplication
2. Straight forward multi-threading
3. Easy extension to parallel GPU/CPU calculations
4. Linear storage in number of non-zero elements

Disadvantages

1. Slow and memory expensive transpose operation
2. Non-negligible Initialization Cost
3. Expensive to alter or update matrix

1.1.1 CSR Definition

The standard CSR format is defined by four parameters.

1. Row Pointer: A list of integers that define how many non-zero values are in each row
2. Column Index: A list of integers that define which items in each row are non-zero
3. Data List: A list of the non-zero values seen in the matrix
4. Shape Tuple: The shape of the matrix

Row Pointer A list of integers of length equal to the number of rows in the matrix plus 1. If R_p is a row point for a matrix then $R_p[i+1]$ will give the index at which the i-th row starts in the Column Index list. This allows for quick access to a specific row of the matrix.

$$\begin{aligned} R_p[0] &= 0 \\ R_p[i+1] &= ||a_i||_0 + R_p[i] \end{aligned} \tag{1}$$

Column Index A list of integers of length equal to the number of non-zero elements in the matrix. If C_i is a column index for a matrix then $C_i[R_p[j] : R_p[j+1]]$ will be the column indexes for the non-zero elements in row j.

$$C_i[R_p[i+1] : R_p[i]] = \{j | a_{ij} \neq 0\} \tag{2}$$

Data List A list of the specified non-zero elements stored in matrix. If D_l is a data list for a matrix then $D_l[R_p[j] : R_p[j+1]]$ will be the list of non-zero elements in row j in the same order as $C_i[R_p[j] : R_p[j+1]]$.

$$D_l[R_p[i+1] : R_p[i]] = \{a_{ij} | a_{ij} \neq 0\} \tag{3}$$

Shape Tuple A 2-tuple of the shape of the matrix. (Number of Rows, Number of Columns)

1.1.2 Example

Given a densely stored matrix:

$$A = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 3 & 3 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

The CSR format for **Row Pointer**, **Column Index**, **Data List** and **Shape Tuple** are constructed as follows:

Row Pointer

$$\begin{bmatrix} 2 & 2 & 0 & 1 \\ 3 & 3 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix} \xrightarrow{\text{Rows}} \begin{pmatrix} 2 & 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 3 & 4 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 & 2 \end{pmatrix} \xrightarrow[\text{Counts}]{\text{Non Zero}} \begin{matrix} 3 \\ 4 \\ 0 \\ 2 \end{matrix} \xrightarrow[\text{Sum}]{\text{Cumulative}} \begin{matrix} R_p[0] = 0 & \to 0 \\ R_p[1] = R_p[0] + ||a_1||_0 & \to 3 \\ R_p[2] = R_p[1] + ||a_2||_0 & \to 7 \\ R_p[3] = R_p[2] + ||a_3||_0 & \to 7 \\ R_p[4] = R_p[3] + |a_4||_0 & \to 9 \end{matrix}$$

$$R_p = \begin{bmatrix} 0 \\ 3 \\ 7 \\ 7 \\ 9 \end{bmatrix}$$

Column Index

$$\begin{bmatrix} 2 & 2 & 0 & 1 \\ 3 & 3 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix} \xrightarrow{\text{Rows}} \begin{pmatrix} 2 & 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 3 & 4 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 & 2 \end{pmatrix} \xrightarrow[\text{Indices}]{\text{Non Zero}} \begin{matrix} 0 & 1 & 3 \\ 0 & 1 & 2 & 3 \\ 3 & 4 \end{matrix} \xrightarrow{\text{Flatten}} \begin{matrix} 0 \\ 1 \\ 3 \\ 0 \\ 1 \\ 2 \\ 3 \\ 3 \\ 4 \end{matrix}$$

$$C_i = \begin{bmatrix} 0 \\ 1 \\ 3 \\ 0 \\ 1 \\ 2 \\ 3 \\ 3 \\ 4 \end{bmatrix}$$

Data List

$$\begin{bmatrix} 2 & 2 & 0 & 1 \\ 3 & 3 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix} \xrightarrow{\text{Rows}} \begin{pmatrix} 2 & 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 3 & 4 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 & 2 \end{pmatrix} \xrightarrow[\text{Values}]{\text{Non Zero}} \begin{matrix} 2 & 2 & 1 \\ 3 & 3 & 4 & 1 \\ -1 & 2 \end{matrix} \xrightarrow{\text{Flatten}} \begin{matrix} 2 \\ 2 \\ 1 \\ 3 \\ 3 \\ 4 \\ 1 \\ -1 \\ 2 \end{matrix}$$

$$D_l = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 3 \\ 3 \\ 4 \\ 1 \\ -1 \\ 2 \end{bmatrix}$$

Sparse A

$$A = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 3 & 3 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix} \rightarrow R_p = \begin{bmatrix} 0 \\ 3 \\ 7 \\ 7 \\ 9 \end{bmatrix}, C_i = \begin{bmatrix} 0 \\ 1 \\ 3 \\ 0 \\ 1 \\ 2 \\ 3 \\ 3 \\ 4 \end{bmatrix}, D_l = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 3 \\ 3 \\ 4 \\ 1 \\ -1 \\ 2 \end{bmatrix}, S = (4, 4)$$

To get the first row.

$$\{i | a_1 \neq 0\} \; = \; C_i[R_p[0] : R_p[1]] \; = C_i[0:3] \; = \; \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

$$\{a_{1i} | a_{1i} \neq 0\} \; = D_l[R_p[0] : R_p[1]] \; = \; D_l[0:3] \; = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

$$a_1 \; = \begin{bmatrix} 2 & 2 & 0 & 1 \end{bmatrix}$$

1.1.3 Memory Cost and Binary Matrix

If all of the elements of a matrix are known to be the same. It is not necessary to store the Data List, D_l and in stead just store one value. This can decrease the memory cost of storing a matrix.

Sparse Binary B example

$$B = \begin{bmatrix} a & a & 0 & a \\ a & a & a & a \\ 0 & 0 & 0 & 0 \\ 0 & 0 & a & a \end{bmatrix} \rightarrow R_p = \begin{bmatrix} 0 \\ 3 \\ 7 \\ 7 \\ 9 \end{bmatrix}, C_i = \begin{bmatrix} 0 \\ 1 \\ 3 \\ 0 \\ 1 \\ 2 \\ 3 \\ 3 \\ 4 \end{bmatrix}, D_l = [a], S = (4, 4)$$

Integer Type	Number of Rows N_r	Number of Columns N_c	Number of Non-Zero-Elements N_{nnz}
8-bit Unsigned Int	255/K	255	255
16-bit Unsigned Int	6.5e4/K	6.5e4	6.5e4
32-bit Unsigned Int	4.3e9/K	4.3e9	4.3e9
64-bit Unsigned Int	1.8e19/K	1.8e19	1.8e19

Memory Cost The memory usage of storing a Sparse CSR matrix is the cost of storing **Row Pointer**, **Column Index**, **Data List** and **Shape Tuple**. Luckily, textbfRow Pointer, **Column Index** are integers so they are much cheaper to store than floating point numbers. We can choose to many different types of integers which will give different storage capacities and difference memory costs seen below in the following table: (Notice that all formats are unsigned as indexing is always positive and that each row has on average K entries)

2 Notation

This is the

3 References

References

[1] Youcef Saad. Sparskit: a basic tool kit for sparse matrix computations. *Research Institute for Advanced Computer Science*, 1990.