CS60027: PARALLEL ALGORITHMS
Assignment1: Multiplication of Sparse Matrices

Description of Sparse Matrix Data-Structure

An **NxM** matrix has **N** rows and **M** columns. For a square matrix, **N == M**.
All matrices are square in this assignment.
Unless otherwise mentioned, a matrix is generally considered dense, i.e., all **n^2** entries in the matrix are assumed to have some valid numbers.
In contrast, most of the **n^2** entries are **0 (zero)** in the case of a sparse matrix. Therefore, it makes sense to store sparse matrices by using special data-structures which economize on the total amount of storage memory required.
Following is a description of **Row-compressed sparse matrices**.
You can easily design a **Column-compressed matrix** using the same basic principles.

A row-compressed **NxN** sparse matrix consists of 3 arrays :
1. **row_Ptr:** Row Pointer is an integer array of size **N+1**
   Let numNonZeros is the total number of non-zeros in the matrix
2. **col_Ind:** Column Index is an **integer array of size numNonZeros**,
3. **nNZ:** Non-Zeros is an integer/float/double array of size numNonZeros

The Row Pointer and Columns Index arrays define the structure of the sparse matrix, while the Non-Zeros array contains the numerical entries.

The non-zeros in the i^th row of the matrix can be accessed as follows :
1. Find starting column index, colStart = rowPtr[i] and last column index, colEnd = row_Ptr[i+1]-1
2. Let col_j = colInd[j] for j in [colStart, colEnd]
2. For every j in [colStart, colEnd], nnz[j] is a non-zero entry at the **i**^th row and **col_j**^th column

Matrix-Vector Multiplication

Multiplication of an **NxN** matrix **A** with an **Nx1** vector **x** to get the vector **y** (i.e., **y = A*x**) can be described in terms of **N dot-products of vectors** :
1. Let **r_i = A(i , :)**, the i^th row of A
2. Then **y[i] = dot-product(r_i , x)**

Normalized Product Vector

A **1xN** normalized vector **y_norm** is defined as follows:
y_norm[i] = y[i]/ (y[1] + y[2] + ... + y[n]),
where **y** is a given **1xN vector** generated from matrix-vector .

In this Assignment

1. Implement the serial version of normalized sparse matrix-vector multiply in the multiply(...) function. Generate the product vector first, and then normalize it.

2. Implement a parallel version of multiply(...) using OpenMP.

3. Run your program on the 4 matrices provided in the matrices directory of the assignment folder.

4. Try different scheduling methods and find the one which produces the best multi-threaded scaling. You need to run your program with 1,2,3,...,24 threads and observe the performance scaling with the number of threads.

5. Write a short report in latex with scaling charts (or graphs/bar plots) for the different scheduling methods tried by you.

Explain the result with emphasis on why a particular scheduling method works better than the others.

How is it related to the type of sparse matrices provided in this assignment?

Would the scaling performance turn out to be different for some other type of sparse matrices?