CS426
Spring 2019
Project 4
Due 2/6/2019

In this project, you will implement sparse matrix-vector multiplication. You will use CUDA to implement this problem.

Problem Statement

This problem concerns multiplication of a sparse matrix with a vector. You have to implement the operation $x_{i+1} = Ax_i$, where A is your sparse matrix and x is the vector (Store result of the product of A and x into x at the end of i^{th} iteration and use it in the $i+1^{th}$ iteration).

In this project you have the following data structures:

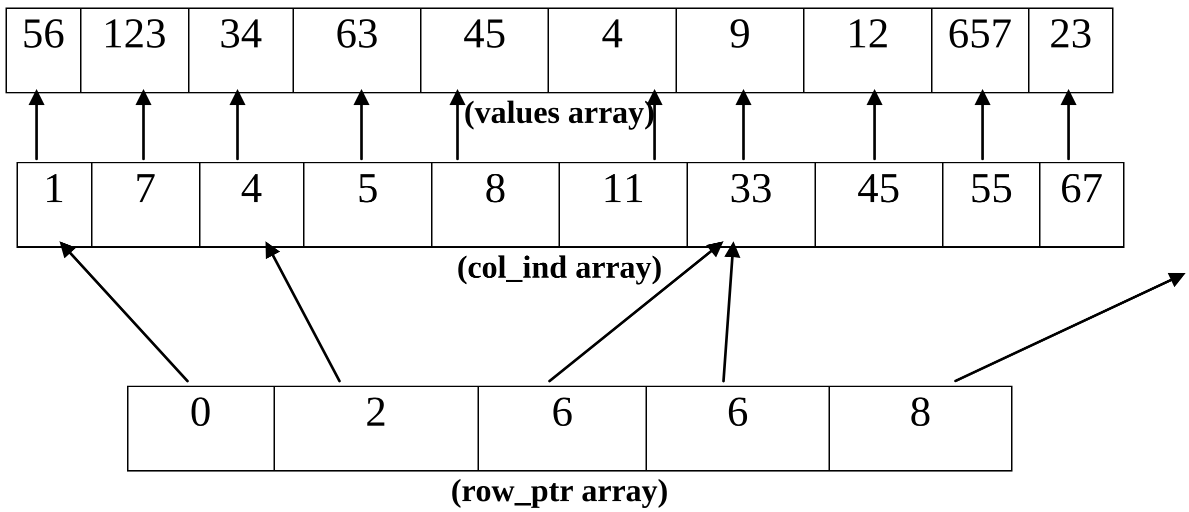
x array: Used to store the vector. **Initialize x to all 1s** before the Matrix-vector product iterations begin.

The sparse matrix is represented using the following three data structures:

row_ptr array: For each row *i* of the sparse matrix, *row_ptr[i]* contains an index number associated with the first nonzero element in this row. This index can be used in *col_ind* and *values* arrays (see below). If row *i* does not contain a non-zero element (i.e. it is all zeros), then *rowptr[i]==rowptr[i+1]*.

col_ind array: This array contains the column indices of the non-zero elements. If the matrix element at row *i* & column *j* is a non-zero element, then *col_ind[k]==j* for some *k* such that *row_ptr[i] <= k < row_ptr[i+1]*.

values array: This array contains the values of non-zero elements. This array is indexed similar to the *col_ind* array. If the matrix element at row *i* column *j* has the non-zero value *v*, then for some *k* such that *rowptr[i] <= k < rowptr[i+1]*, you have *values[k]==v*.



The arrows indicate the relations – for every k in the range row_ptr[i] <= k < row_ptr[i+1], you have col_ind[k]=j such that A[i][j] is a non zero value and it is stored in values[k].

For example in 0^{th} row, column 1 and 7 contain non-zero values 56 and 123 respectively. Other columns in row 0 contain 0s. Row 2 contains all 0s, which is reflected in the fact that *row_ptr[2]==row_ptr[3]*. Row 4 also contains all zeros, but it is the last row. So *row_ptr[4]* points somewhere outside *col_ind* array. (Handle end conditions carefully, take care that you don't get segmentation faults by accidentally trying to access a *row_ptr* location that is out of bounds.)

Write a program that uses CUDA that performs the Matrix-vector product. (No need to use CUDA for reading the input, printing etc.) You can assume that the sparse matrix is a square matrix.

The iterations of $x_{i+1} = Ax_i$ could be implemented using the following loop. The time to be noted is the total time required for all the iterations to get over:

```
for(iter=0; iter<number_of_iterations; iter++) {
           matrix-vector computation
}
```

Language

You should use C language with CUDA (You can refer to course slides for specifications). You may use the following command to compile your files:

nvcc filename.cu

Machines

You will need to use a machine with NVIDIA GPU card. If you want to use a GPU available in our servers, email kaan.akyol@bilkent.edu.tr to ask for an account. Note that, one GPU card cannot be used by two programs at the same time so your timing results will be correct.

Testing

- **Initialize x to all 1s** before the Matrix-vector product iterations begin.
- Your programs should accept as input from the first three command line arguments. –
    1. The number of threads used to compute Matrix-vector product
    2. The number of repetitions and
    3. An argument to print on stdout (See below).
    4. Test-file name
- The command line argument #3 controls whether or not the program is to print the initial matrix, vector and the resulting vector after all the $x_{i+1}=Ax_i$ iterations have completed. The programs should print on stdout when this parameter is set to 1. This will be used to test if your matrix-vector product is correct.
- You have sample input files at
    - http://www.cs.bilkent.edu.tr/~ozturk/cs426/fidapm08.mtx
    - http://www.cs.bilkent.edu.tr/~ozturk/cs426/fidapm11.mtx
    - http://www.cs.bilkent.edu.tr/~ozturk/cs426/cavity02.mtx
- These test matrices will contain the sparse matrix in the following format –

  #rows #columns #non-zero-entries-in-A
  row column non-zero-value-at-A[row][column]
  row column non-zero-value-at-A[row][column]
  .
  .
  (Till the end of file)

  These sample files should be treated as normal text files.

  Important: The rows and column numbers range start at 1 and therefore you must subtract 1 so that they start at 0 to match C style. These matrices come from the "Matrix Market" website (http://math.nist.gov/MatrixMarket/) which is an interesting place to browse if you are (and even if you are not) into sparse computations.

Report

Write a short report containing:

1. Parallelization strategy used (You can use any parallelization strategy that scales up.)
2. Three figures for each test matrix that contains
    a. parallel running time,
    b. speedup, and
    c. efficiency of your CUDA
3. Short discussion about the results.

Note that, a part of your grading criteria may be the performance of your parallel implementation. Therefore you should try to write the fastest running parallel program.

Submission

Put all relevant code, makefile, shell script and your report into a zip file.

Name the zip file: yourname_lastname_p4.zip

Mail the zip file to: kaan.akyol@bilkent.edu.tr

Mail topic: CS426_Project4