CPSC 474
Project 2: Compressing a Sparse Matrix

Brian Montgomery

Thanh Vuong

Pseudocode

Plain English PseudoCode

```
if (“-mFile” is found)
        load the filename that follows
else
        Throw no filename error

Init MPI

if rank is root
        load the file
        while there are lines
                get line and load it into a vector
        save number of lines

for each line
        while there are letters in the line
                parse out the numbers and add them to a master vector

calculate portion that each process will have
calculate # of columns

Broadcast the portion and root values to all processes
resize arrays on all processes to the size of portion
Scatter data to all the processes’ resized arrays
Make sure all processes reach the barrier before proceeding

for each local number given to each process
        if number is not 0
                push number to a solution vector
                push number's row and col to solution vector
```

Output individual process solutions

For each non-root process
    send the size of the solution vector to the root rank
    Then send all solution data to the root rank

For the root process
    Gather the sizes of the other processes' solution vectors into a gather sizes array
    Then use the gather sizes array to calculate the offsets of the incoming data in the new array and resize the array to fit all of it.
    Ask the other processes for their solutions and add your own data
    Output the final solutions

Finalize MPI

```
//Detailed PseudoCode
rows = 0
columns = 0
numbers = []
size = 0
rank = 0
rootRank = 0

MPI_Init(argc, argv)
MPI_Comm_rank(MPI_COMM_WORLD, rank)
MPI_Comm_size(MPI_COMM_WORLD, size)

IF rank == rootRank DO:
  // read lines from file to rowVec[]
  rowVec = [][]
  WHILE line to read in matrixFile DO:
    rowVec[rows] = line
    n = count elements in line separated by comma
    IF n > columns DO:
      columns = n
    ENDIF
    rows += 1
  ENDWHILE
  close matrixFile

  // read numbers from rowVec to numbers[], fill empty cells with 0
  FOR i = 0 to rows - 1 DO:
    n = count elements in rowVec[i] separated by comma
    FOR j = 0 to columns - 1 DO:
      IF j < n DO:
        numbers[i].append(line[j])
      ELSE DO:
        numbers[i].append(0)
      ENDIF
    ENDFOR
  ENDFOR

portion = 0

IF size > 1 DO:
  portion = numbers.size / size
ENDIF

IF rows != 0 DO:
  cols = numbers.size / rows
```

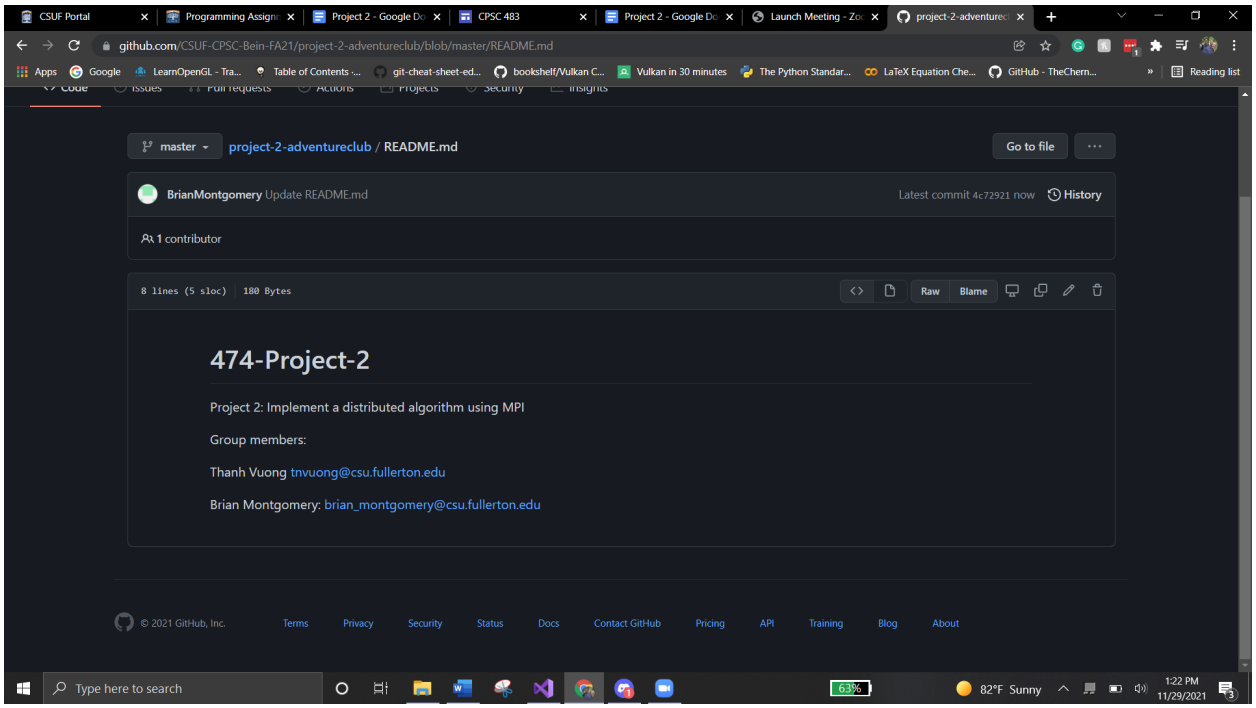
```
localNumbers = []
localNumbers.size = portion

MPI_Bcast(portion, 1, MPI_INT, rootRank, MPI_COMM_WORLD)

MPI_Scatter(numbers, portion, MPI_INT, localNumbers, portion, MPI_INT, rootRank,
MPI_COMM_WORLD)

MPI_Barrier(MPI_COMM_WORLD)

// copy non-zeros from localNumbers to solution[]
solution = []
IF size > 1 DO:
  FOR i = 0 to portion - 1 DO:
    IF localNumbers[i] != 0 DO:
      solution.append(localNumbers[i])
      solution.append((rank * portion + i) % cols))
      solution.append((rank * portion + i) / cols))
    ENDIF
  ENDFOR
ENDIF

IF rank == rootRank DO:
  IF size > 1 DO:
    IF numbers.size % size != 0 DO:
      FOR i = 0 to numbers.size % size - 1 DO:
        IF numbers[portion * size] != 0 DO:
          solution.append(numbers[portion * size])
          solution.append((rank * portion + i) % cols))
          solution.append((rank * portion + i) / cols))
        ENDIF
      ENDFOR
    ENDIF
  ELSE DO:
    IF numbers.size != 0 DO:
      FOR i = 0 to numbers.size - 1 DO:
        IF numbers[i] != 0 DO:
          solution.append(numbers[i])
          solution.append((rank * portion + i) % cols))
          solution.append((rank * portion + i) / cols))
        ENDIF
      ENDFOR
    ENDIF
  ENDIF
ENDIF

gatherSizes = []
```

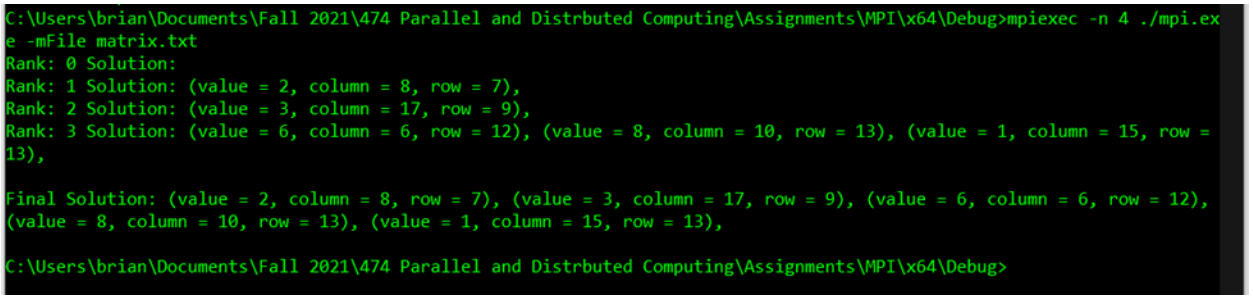
```
displacements = []
solSize = solution.size

IF rank == rootRank DO:
  FOR i = 0 to size - 1 DO:
    IF i != rootRank DO:
      gatherSizes.append(0)
      MPI_Recv(gatherSizes[i], 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
    ELSE DO:
      gatherSizes.append(solSize)
    ENDIF
  ENDFOR

  sum = 0
  FOR i = 0 to gatherSizes.size - 1 DO:
    displacements.append(sum)
    sum += gatherSizes[i]
  ENDFOR

  finalAnswer = []
  finalAnswer.size = sum

  MPI_Gatherv(solution, solution.size, MPI_INT, finalAnswer, gatherSizes,
displacements, MPI_INT, rootRank, MPI_COMM_WORLD)

  print(finalAnswer)
ELSE DO:
  // send the sizes of the solution buffer from each process
  MPI_Send(solSize, 1, MPI_INT, rootRank, 0, MPI_COMM_WORLD)
  // then gather the results into one buffer
  MPI_Gatherv(solution, solution.size, MPI_INT, NULL, NULL, NULL, MPI_INT,
rootRank, MPI_COMM_WORLD)

ENDIF

MPI_Finalize()
```

How to Run the Code

1. Have MSMPI MS-MPI SDK and Redistributions installed on a Windows machine.
2. Download the GitHub repository from https://github.com/CSUF-CPSC-Bein-FA21/project-2-adventureclub/tree/master
3. Open command prompt and navigate to the repository's /x64/Debug folder.
4. Run the command mpiexec -n 4 ./mpi.exe -mFile matrix.txt
    - This command runs the program executable with 4 processes and uses the input of matrix.txt which is also found in the /x64/Debug folder.

Screenshots





Input used in this screenshot:

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,8,0,0,0,0,1,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0