Data Sciences
Project 3
Shrinidhi Sondur

Task 1:

How to compile:

- Both input file and .cpp file should be put in the same directory.
- `tar –zxvf Project3_Sondur_Shrinidhi.tar ./`
- `cd Project3_Sondur_Shrinidhi`
- `cd Task1/src/`
- `g++ -fopenmp PageRank.cpp –o PageRank.o`
- `./PageRank.o`

Implementation:

- Read File and Initialize matrix: Read the file and put the values in a vector of vectors.
- Rank Vector: The rank vector is initialized with the value 1/N, where N is the number of nodes in the graph.
- Rank Calculation: Iterations are performed till the rank of each node converges to the threshold which is calculated as a difference of the newly calculated Matrix and the the existing one.
  Threshold **(**new rank vector – previous rank vector) <= 10^{-7}
- Calculation: Parallel constructs in OpenMP are used. Each column of the matrix is multiplied with the rank vector parallel-y, which generates a new rank vector. Each element of the rank vector (row) is multiplied with the corresponding matrix entry (column entries) and added using reduction construct in openMP. The values calculated are then written into the new rank vector using critical construct to ensure write safety.
- Output: The values are sorted by PageRank. It is written to a file named Output_Task1.txt.
- Result: Sorted PageRank values for the egoFacebook graph.

- Dampening Factor = 0.85;

OPENMP Constructs used:

Observations Table:

Number of Threads	Convergence Threshold	Number of Iterations	Convergence Time
5	10^{-5}	12	user 2.52 sys 3.37 CPU usg 161
10	10^{-5}	12	user 2.76 sys 4.64 CPU usg 156%
15	10^{-5}	12	user 2.42 sys 0.38 CPU usg 154%
20	10^{-5}	12	user 2.54 sys 0.33 CPU usg 156%
5	10^{-7}	12	user 2.39 sys 0.36 CPU usg 156%
10	10^{-7}	12	user 2.38 sys 0.35 CPU usg 153 %
15	10^{-7}	12	user 2.50 sys 0.39 CPU usg 155%
20	10^{-7}	12	user 2.52 sys 0.90 CPU usg 156%

When Run Serially:
- for 10^{-5}
real 24.15
user 23.85
sys 0.26

-for 10^{-7}
real 64.00
user 63.41
sys 0.39

Performance Measurement:

The observations suggest that the parallel code works faster than the serial code for reasonably optimal values of number of threads, as the number of threads increase, the task of distributing and gathering back information from threads adds extra overhead.

Task 2

How to compile:

- Both input file and .cpp file should be put in the same directory.
- `tar –zxvf Project3_Sondur_Shrinidhi.tar ./`
- `cd Project3_Sondur_Shrinidhi`
- `cd Task2/src/`
- `mpiCC Reducer.cpp -o Reducer`
- `mpirun -n 5 Reducer`

Implementation:

1. Read the file: Master process distributes chunks to each processor
2. Each Processor performs local reduction of the keys meant for it depending on the mod function and sends the rest of the keys to the appropriate processor who perform loca reduction of the keys..
3. These reduced keys are sent back to the master who puts aggregates them and puts them in the output file.

I use blocking send and receive for my communications with the Master thread. However for inter process communication this was not possible and hence I use blocking communication.