# Task1

**Compilation Step** -

i) g++ Pagerank.cpp -fopenmp
ii) ./a.out <filename>

**Output File -**
Output_Task1.txt

**Convergence Criteria -**
Difference of Newrank and oldrank is less than $10^{-9}$ for each node in the Graph

**Algorithm-**

1) Calculate size of graph by inserting each node in a STL set and then calculate set size
Function- GraphSizeCalculation(filename)

2) - Read the input file and create a sparse matrix .
- Sparse matrix is vector of vector where each cell store element of "pairs" type.
pairs store "node" and "rank"
struct pairs { int node;double rank;}

3) Create a "rank" array which store rank of the Node after each iteration. Initialize rank to 1/Total_Number_of_Node

4) Repeatedly calculate newrank for each Node until it converge.
- To compute new rank of a Page , multiply each outlink with corresponding rank and sum it.
- Add a damping value (1-damp)/Total_no_of_node and multiply sum to damping factor.
- Parallelize inner for loop by using reduction. Reduction will make own local copy of sum and then add it to a Final value.
- Code Snippet

```
#pragma omp parallel for reduction(+:sum) schedule(static,size/4)
for(int j=0;j<vec[i].size();j++)
      sum=sum+(vec[i][j].rank)*(rank[vec[i][j].node]);
newrank[i]=(1-damp)/size+damp*sum;  }
```

- Check the Convergence , if all node satisfy convergence criteria exit from the loop

5) Write the Final rank into the "Output_task1" file
Function: WriteFile(rank,size)

**Performance Analysis**

I ran it for "facebook_combined.txt" dataset for different number of thread on my dual core Machine with .

| **Number Of Thread** | **Average Time** |
|---|---|
| **1** | 0.422892 |
| **2** | 0.285998 |
| **3** | 0.345126 |
| **4** | 0.279126 |
| **5** | 0.31556 |
| **6** | 0.303608 |

1) Parallelization decrease run time for the job and we get best performance when Number of thread is 4.

2) As we ran it on dual core machine ,increasing number of thread more than 4 does not increase runtime of the job and give slightly bad performance than 4 thread.

3) If we set Number of thread 2 , it gives better result when Number of thread is set to 3. It may be due to lesser overhead in parallelization if number of thread is even.

# Task2

**Compilation Step** -

i) mpic++ Reducer.cpp
ii) mpirun -np <Num of Processor> ./a.out

**Output File -**
Output_Task2.txt

**Algorithm Steps**

1) - In processor 1 Calculate total Number of distinct key and total size ( Number of key Value pair) by traversing file.

**Handling if key are not sequential -**
Insert each key into a map which map different key sequentially .
e.g If we have 3 key 2,5,7 so map will do following mapping

2----0
5----1
7----2

It will perform all operation using this mapping and give back to original key in final output.

2) Broadcast "total Number of Key" and "total size" to each processor.

3) Create a 2-D matrix of Number of row= total size and 2 column.

4) In processor 1 traverse the file again and store key and value( mapped value) in the matrix.

**Handling Size if it is not Multiple of Number of Processor**-
If total size is not divisible by Number of Processor then extra value are padded into the matrix .It take an existing key and add <Key,0> in the matrix and size is set to
size=size+(numofProcessor - size% numofProcessor)

5) Do **MPI_Scatter** to the matrix , so that each processor will receive size/Numberofprocessor element

6) Do local reduction at each processor and add value corresponding to each key.

```
for(int i=0;i<(size/nprocs);i++)
   recmatrix2[recmatrix[i][0]]=recmatrix2[recmatrix[i][0]]+recmatrix[i][1];
```

Here recmatrix is each processor local array which they get after scatter. and recmatrix2 array they get after local reduction whose size is equal to keysize.

7) Do **MPI_Allgather** so that each processor have result after local redicution phase in 2 D matrix

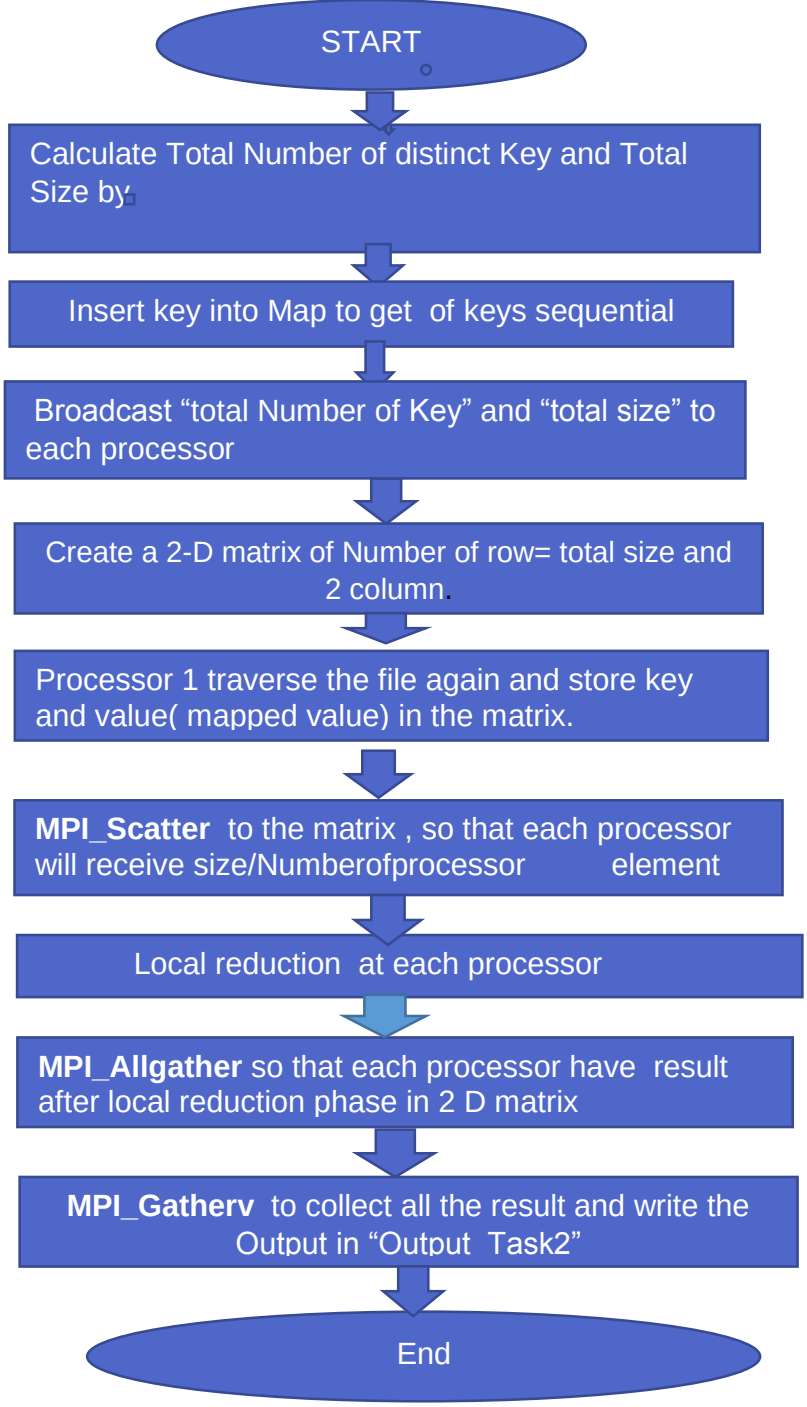
```
finmatrix[numberofprocessor][Keysize];
```

**Assignment of 2nd Reduction to each Processor**

- Divide Job based upon the keysize , Assign all process other than last -
  ciel(Number of Keys / Number of Processor )
  If Total no key are not divisible by Number of processor then last processor will receive
  ciel(Number of Keys / Number of Processor ) - (Number of Keys % Number of Processor)

9) Do a **MPI_Gatherv** to collect all the result and write the Output in “Output_Task2”

## FlowChart


START
Calculate Total Number of distinct Key and Total Size by
Insert key into Map to get of keys sequential
Broadcast “total Number of Key” and “total size” to each processor
Create a 2-D matrix of Number of row= total size and 2 column.
Processor 1 traverse the file again and store key and value( mapped value) in the matrix.
MPI_Scatter to the matrix , so that each processor will receive size/Numberofprocessor element
Local reduction at each processor
MPI_Allgather so that each processor have result after local reduction phase in 2 D matrix
MPI_Gatherv to collect all the result and write the Output in “Output Task2”
End