Project Report :

Task 1 Pagerank Calculation using OpenMP

Compile : g++ task1.cpp -fopenmp -o task1.o
Run : 
export OMP_NUM_THREADS=5
./task1.o -f "facebook_combined.txt"

Details :
1. Read file to calculate the total number of nodes
2. Allocate memory for array used to store the graph matrix.
3. Populate the graph matrix by reading the file again, for each (vertexi,vertexj) pair make the value [i][j] and [j][i] in the graph martix as 1.
4. Initialize the pagerank of all the vertices as 1/N where N is the total number of nodes in the graph.
5. Once the graph matrix is populated and pagerank is initialized to 1/N for each node , then iterate over the matrix and calculate until the pagerank converges to a converging criteria.

Page RAnk formula used : Pagerank of NodeA = (1-d)/N + d*SUM((PageRank of neighbour I)/(Number of outlinks of Neighbour I))
here d= Damping factor taken as 0.85 , N = total number of nodes , As the facebook graph is an undirected graph so the number of outlinks is equal to the degree of the graph.

Converging Crietria used : the grap is assumed to have converged if for a given node the difference in the current pagerank and the pagerank in previous iteration > 0.0000000001

6. Compute the final sum of all page ranks . This sum should equal to 1.
7. Write output to file

Performance improvements :
I compared running the programming with and without openmp.With openmp a slight improvement in running time of the program was observed.

Task2 : Calculation of sum using Key Value pairs

Compile : mpicc task2_new.c -o task2_new.out -std=c99
Run : mpirun -np 4 task2_new.out -f "100000_key-value_pairs.csv"

Details :
1) Read the file and get the total number of nodes and max key.
2) Allocate memory for arrays used to store the keyvalue pair tables, partial table of size (total pairs)/(total processors) , partioned tables.
3) read the file and populate the partial tables such that each processor has equal number of key value pairs, in this case 25000 key-value pairs are distributed for a run with 4 processors.
4)At each processor locally reduce the partial tables and calculate the sum of the values of each key on each processor. Store this key-Local_sum pair array in an array in each processor.
5) Then calculate the global sum for each key and send it to processor 0.
6) On processor 0 store the global sum for each key and store it in an output file.