

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

«ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ»

ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

1η ΕΡΓΑΣΙΑ

ΚΟΖΟΜΠΟΛΗΣ ΙΩΑΝΝΗΣ ΜΠΣΠ15036

ΕΜΜΑΝΟΥΗΛ ΓΕΩΡΓΙΟΣ ΜΠΣΠ15025

ΛΙΟΛΙΟΥ ΧΡΥΣΟΥΛΑ ΜΠΣΠ15046

ΠΕΙΡΑΙΕΥΣ, ΦΕΒΡΟΥΑΡΙΟΣ 2016

Εργασία 1

**Ψευδοκώδικας 1ης άσκησης**

1) Δήλωση μεταβλητών

2) Αρχικοποίηση του MPI

3) Ζητάτε από τον χρήστη να δώσει το όνομα του αρχείου με τα tuples για ταξινόμηση. Ανοίγει το αρχείο μετράει τα tuples. Αποθηκεύει τα tuples στην μνήμη της διεργασίας 0. Ζητάει από το χρήστη να δώσει το Pmin, Pmax και s στην κονσόλα.

4)Αρχικοποιείτε το p\_i ίσο με το Pmin.

5) Μοίρασμα των tuples από την διεργασία 1 στις υπόλοιπες διεργασίες και στον εαυτό της (MPI\_Scatterv)

6)Αντιγραφή μόνο των Keys σε νέο πίνακα από κάθε διεργασία για τα tubles που έχει λάβει

7)Υπολογισμός των pivots από τις διεργασίες με μέγεθος όσο το p\_i .

8) Μάζεμα από την διεργασία 0 όλων των pivots από όλες τις διεργασίες και δημιουργία ενός πίνακα από τον κάθε πίνακα pivot από κάθε διεργασία.(MPI\_Gather). Κοινοποίηση αυτού του πίνακα σε όλες τις διεργασίες. (MPI\_Bcast).

9) Υπολογισμός ιστογράμματος σε κάθε διεργασία.

10) Άθροισμα ιστογραμμάτων σε ένα τελικό από την διεργασία 0 (MPI\_Reduce). Κοινοποίηση του αθροισμένου ιστογράμματος και στις υπόλοιπες (MPI\_Bcast).

11) Υπολογισμός από την διεργασία 0 ποιά tuples θα ταξινομήσει κάθε διεργασία και πόσα θα είναι αυτά βάση του ιστογράμματος. Ενημέρωση κάθε διεργασίας με MPI\_Send, MPI\_Recv.

12)Υπολογισμός του s. Αν το s είναι μικρότερο από αυτό που δώσαμε στην κονσόλα τότε ο αλγόριθμος συνεχίζει με το επόμενο βήμα. Αν είναι μεγαλύτερο τότε διπλασιάζει το p\_i. Αν το p\_i είναι μικρότερο από το pmax επαναλαμβάνονται τα βήματα από το βήμα 7. Αν είναι μεγαλύτερο το πρόγραμμα τερματίζεται και ζητείτε να ξανατρέξουμε το πρόγραμμα και να δώσουμε νέα pmin,pmax και s.

13) H κάθε διεργασία ξεχωρίζει ποια tubles είναι τα δικά της και ποιά θα στείλει στις υπόλοιπες σε κάδους.

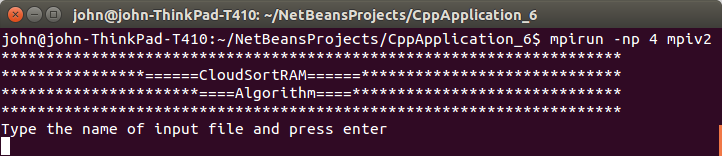
14) Η κάθε διεργασία μαζεύει τα tubles που έχει ξεχωρίσει κάθε διεργασία και την αφορούν και τα αποθηκεύει σε μία λίστα.(MPI\_Gatherv)

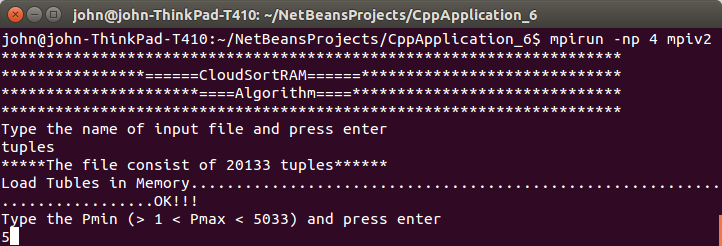
15) Η κάθε διεργασία ταξινομεί την δική της λίστα.

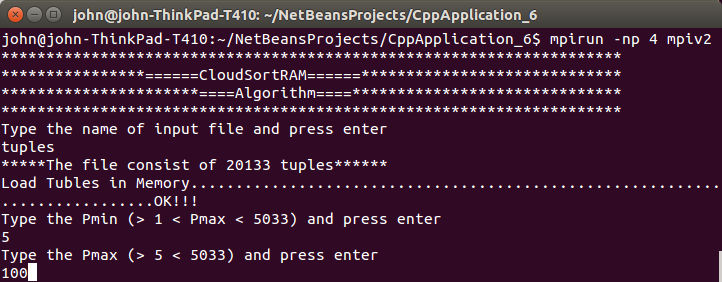
16) Η διεργασία 0 μαζεύει την ταξινομημένη λίστα κάθε διεργασίας και εκτυπώνει την τελική ταξινομημένη λίστα στην κονσόλα.

**Εκτέλεση αλγόριθμου με την εφαρμογή που δημιουργήθηκε**

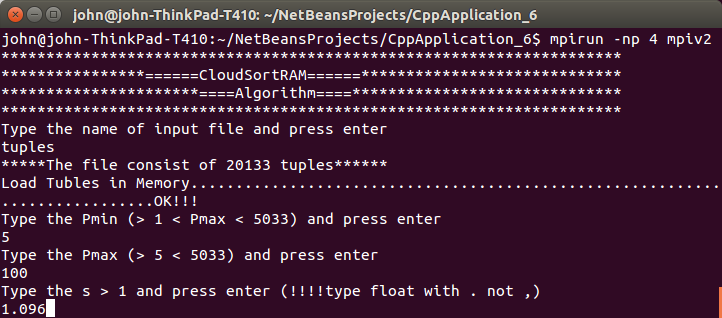
Για την εκτέλεση του αλγόριθμου χρησιμοποιήσαμε αρχέιο με 20133 tuples με τέσσερις διεργασίες. Τρέχοντας το πρόγραμμα ζητείτε να δώσουμε το αρχείο με τα tuples.

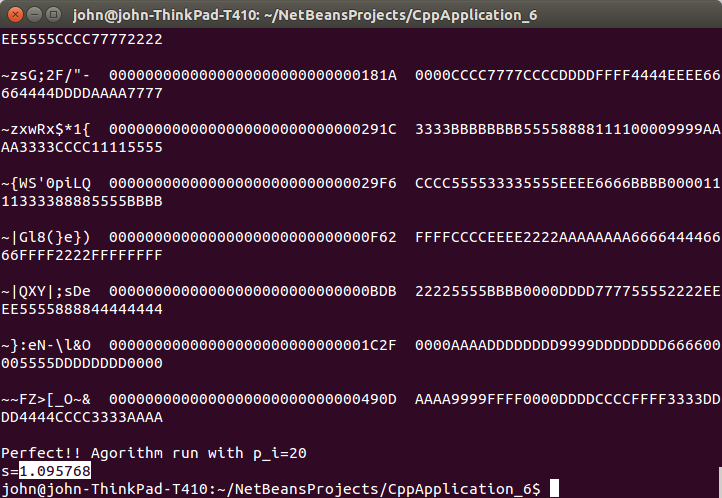
Δίνουμε το όνομα του αρχείου και πατάμε enter. Μετά μας εμφανίζει πόσα tuples διάβασε και αποθήκευσε στην μνήμη. Μας ζητάει να δώσουμε το Pmin όπου πρέπει να είναι μεγαλύτερο του 1 μικρότερο από το Pmax και μικρότερο από το πλήθος των tuples δια το πλήθος των διεργασιών που τρέχει το πρόγραμμα. To Pmin το δώσαμε 5.

Μετά μας ζητάει να δώσουμε το Pmax. To Pmax πρέπει να είναι μεγαλύτερο από το Pmin και μικρότερο από το πλήθος των tuples δια το πλήθος των διεργασιών που τρέχει το πρόγραμμα. Εμείς δώσαμε 100.



Στην συνέχεια μας ζητάει να δώσουμε το s που πρέπει να είναι μεγαλύτερο από 1 για να τρέξει το πρόγραμμα. Εμείς δώσαμε 1.096

Τρέξαμε το πρόγραμμα και μετά από λίγο μας εμφάνισε τα tuples που διάβασε από το αρχείο ταξινομημένα στην κονσόλα. Το p\_i για να ικανοποιήσει τον περιορισμό με το s που θέσαμε έφτασε το 20 και τελικά το s για αυτή την περίπτωση έφτασε στο 1.095768!!



Το πρόγραμμα έχει φτιαχτεί δυναμικά για όποιο πλήθος εγγραφών δοθεί και έχει ταχταριστεί για όποιο πλήθος διεργασιών μέχρι 6. Για μεγάλα αρχεία όπου η κάθε διεργασία παίρνει πάνω από 10.000 εγγραφές δεν τρέχει διότι δηλώνετε ένας στατικός πίνακας 3d και οι στατικοί πίνακες έχουν όριο. Σαν βελτίωση σε αυτό το κομμάτι σε επόμενη έκδοση θα προσπαθήσουμε να το κάνουμε δυναμικά. Ώστε να δουλεύει με πάνω από 40.000 εγγραφές.

Πρέπει να δίνετε το σωστό όνομα του αρχείου προς ταξινόμηση και να το s στην κονσόλα ο δεκαδικός αριθμός να είναι με τελεία “.” όχι με κόμμα “,”. Στο φάκελο εργασία 1 θα βρείτε τον κώδικα και το αρχείο με τα tuples που έγινε δοκιμή. To πρόγραμμα γράφτηκε με c.

**Ο κώδικας της εφαρμογής**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <string.h>

/\*

\* Exercise 1 Implementation CloudSortRam

\*/

void getMemory(char \*\*\*data,int rows, int columns){//get memory for 2d char array

int i;

char \*p = (char \*)malloc(rows\*columns\*sizeof(char));

(\*data) = (char \*\*)malloc(rows\*sizeof(char\*));

for (i=0; i<rows; i++)

(\*data)[i] = &(p[i\*101]);

};

void keyMemory(char \*\*\*data,int rows,int columns){//get memory for 2d char array

int i;

char \*p = (char \*)malloc(rows\*columns\*sizeof(char));

(\*data) = (char \*\*)malloc(rows\*sizeof(char\*));

for (i=0; i<rows; i++)

(\*data)[i] = &(p[i\*11]);

};

int main(int argc, char\*\* argv) {

/\*Define variables and arrays\*/

int numprocs, rank, namelen, i=0, j=0,k=0, p\_i,pmin, pmax, step, g=0, lines=0, prolines=0, left=0,

\*hist, \*totalHist, \*count, \*elemCount, \*point, \*displs, \*sendcounts;

FILE \*fp;

float s, average, procaverage=0, proclines=0, ss, spec;

char \*\*dataArray, \*\*procArray, \*\*procKeys, \*\*pivots, \*\*totalPivots, \*\*calcBuff, \*\*sortedArray, \*ptr, \*ptrs, filename[20];

const int root=0;

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

/\*MPI Init\*/

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Get\_processor\_name(processor\_name, &namelen);

/\* Process 0 open the file calc the lines and store the rows to "data" array of chars\*/

if (rank==0){

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======CloudSortRAM======\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*====Algorithm====\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Type the name of input file and press enter\n");

scanf("%s",filename);//read the name of input file

char ch;

fp=fopen(filename, "rb");//open file with fp pointer

do {//Calc the number of tubles

ch = fgetc(fp);

if(ch == '\n')

lines++;

} while (ch != EOF);

printf("\*\*\*\*\*The file consist of %d tuples\*\*\*\*\*\*\n",lines);//Print the number of tuples

getMemory(&dataArray,lines,101);//Allocate memory for the number of tuples

ptr=&(dataArray[0][0]);//Set pointer on the first position of allocated memory

i=0;

rewind(fp);//Set pointer back to the start of file

while(fgetc(fp) != EOF){//Save the tuples in allocated memory

fseek(fp, -1, SEEK\_CUR);

fgets(ptr+(i\*101), 101, fp);

i++;

}

fclose(fp); //Close file

ptr=&(dataArray[0][0]);//Set pointer on the first tuble in memory

printf("Load Tubles in Memory......................."

".....................................................OK!!!\n");

/\*Set Pmin, Pmax and s from console\*/

int ver=lines/numprocs;

int var=0;

printf("Type the Pmin (> 1 < Pmax < %d) and press enter\n",ver);

do{

if (var==0){

scanf("%d",&pmin);

if (pmin<=1 || pmin > ver){

printf("False!!! Type the Pmin (> 1 < Pmax < %d) and press enter\n",ver);

}else

var=1;

}

if (var==1){

printf("Type the Pmax (> %d < %d) and press enter\n",pmin,ver);

scanf("%d",&pmax);

if (pmax>=pmin && pmax<=ver){

var=2;

}else

printf("False!!! ");

}

}while(var<2);

printf("Type the s > 1 and press enter (!!!!type float with . not ,)\n");

scanf("%f",&spec);

}

/\*Broadcast from 0 process Pmin,Pmax, s and number of saved tuples\*/

MPI\_Bcast(&pmax, 1, MPI\_INT, root, MPI\_COMM\_WORLD);//pmax

MPI\_Bcast(&pmin, 1, MPI\_INT, root, MPI\_COMM\_WORLD);//pmin

MPI\_Bcast(&spec, 1, MPI\_INT, root, MPI\_COMM\_WORLD);//s

MPI\_Bcast(&lines, 1, MPI\_INT, root, MPI\_COMM\_WORLD);//tubles

p\_i=pmin;//Define p\_i

MPI\_Barrier(MPI\_COMM\_WORLD);

displs = (int\*) malloc(numprocs\*sizeof(int));

sendcounts = (int\*) malloc(numprocs\*sizeof(int));

point=(int\*)malloc(numprocs\*sizeof(int));//Allocate array to save the number of tuples

prolines = lines/numprocs;

left = lines%numprocs;

/\*Calc how many tuples will save every process\*/

for (i=0;i<numprocs;i++){

point[i]=prolines;

if((i<left) && (left>0)){

point[i]=point[i]+1;

}

}

for (i=0;i<numprocs;i++){

if(rank==i){

prolines=point[i];

}

}

/\*Calc how many bytes will receive every process \*/

for(i = 0; i<numprocs; i++){

sendcounts[i] = sizeof(char)\*(point[i]\*101);

}

/\*Calc the start pointers that process 0 will use for send data to other processes\*/

displs[0]=0;

for(i = 1; i<numprocs; i++){

displs[i] = displs[i-1]+sendcounts[i-1];

}

getMemory(&procArray,prolines,101);//Allocate memory for data that will receive every process

/\*Split the data and send to other processeis\*/

MPI\_Scatterv(ptr,sendcounts, displs, MPI\_CHAR,procArray[0],prolines\*101,MPI\_CHAR,0,MPI\_COMM\_WORLD);

keyMemory(&procKeys,prolines,11);//Allocate memory to copy only the keys from payloads

MPI\_Barrier(MPI\_COMM\_WORLD);

/\*Free memory from dataArray in process 0\*/

if (rank==0){

free(&(dataArray[0][0]));

free(dataArray);

}

ptrs=&(procKeys[0][0]);//set pointer to procKeys memory

for (j=0;j<prolines;j++){//Copy the keys from payloads in every process in new memory

memcpy(procKeys[j],procArray[j],10);

ptrs=ptrs+10;

\*ptrs='\0';

ptrs=ptrs+1;

}

MPI\_Barrier(MPI\_COMM\_WORLD);

do{//do while loop to find s <= init s if fail rerun the program with other pmin pmax and s

if (g>0){//double the pivots

p\_i=p\_i\*2;

}

if (p\_i>pmax){//if p\_i > Pmax the algorith has't find <= s and print message rerun with new values

if (rank==0){

printf("Algorithm Fail Rerun the program and set new s.....\n");

}

MPI\_Finalize();

exit(0);

}

step=(int)prolines/p\_i;//calc the step that every process will choose keys for make pivots

keyMemory(&pivots,p\_i,11);//Allocate memory for pivots

j=0;

for (i=0;i<prolines;i=i+step){//calc keys for pivots

memcpy(pivots[j],procKeys[i],11);

if(((i==(step\*p\_i-2)) && (step>1 )) || (j==p\_i-1)){

i=prolines;

}

j++;

}

keyMemory(&totalPivots,p\_i\*numprocs,11);//Allocate memory for totalPivots

MPI\_Barrier(MPI\_COMM\_WORLD);

/\*Save Pivots from all processes to totalPivots array\*/

MPI\_Gather(pivots[0], p\_i\*11, MPI\_CHAR,totalPivots[0], p\_i\*11, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

/\*Process 0 sort totalPivots Array\*/

if (rank==0){

char temp[11];

for (i = 0; i < p\_i\*numprocs; i++) {

for (j = 0; j < p\_i\*numprocs - 1; j++) {

if (strcmp(totalPivots[j], totalPivots[j + 1]) > 0) {

strcpy(temp, totalPivots[j]);

strcpy(totalPivots[j], totalPivots[j + 1]);

strcpy(totalPivots[j + 1], temp);

}

}

}

}

MPI\_Barrier(MPI\_COMM\_WORLD);

free(&(pivots[0][0]));//free memory

free(pivots);//free memory

/\*Broadcast totalPivots sorted array to all processes\*/

MPI\_Bcast(totalPivots[0], p\_i\*numprocs\*11, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

hist=(int \*)calloc(p\_i\*numprocs+1, sizeof(int)); //Allocate memory for histograms

/\*Calc histogram in every process\*/

for (i=0;i<prolines;i++){

for (j=0;j<p\_i\*numprocs;j++){

if (strcmp(procKeys[i], totalPivots[j]) < 0){

hist[j]=hist[j]+1;

j=p\_i\*numprocs;

}

else if (strcmp(procKeys[i], totalPivots[j]) == 0){

hist[j+1]=hist[j+1]+1;

j=p\_i\*numprocs;

}

else if (strcmp(procKeys[i], totalPivots[p\_i\*numprocs-1]) > 0){

hist[p\_i\*numprocs]=hist[p\_i\*numprocs]+1;

j=p\_i\*numprocs;

}

}

}

totalHist=(int \*)calloc(p\_i\*numprocs+1, sizeof(int));//Allocate memory for final histogram

MPI\_Barrier(MPI\_COMM\_WORLD);

/\*Sum all histograms from from processes to final array named totalHist\*/

MPI\_Reduce(hist,totalHist,p\_i\*numprocs+1,MPI\_INT,MPI\_SUM,0,MPI\_COMM\_WORLD);

/\*Broadcast totalHist array to all processes\*/

MPI\_Bcast(totalHist, p\_i\*numprocs+1, MPI\_INT, root, MPI\_COMM\_WORLD);

i=1;

MPI\_Barrier(MPI\_COMM\_WORLD);

/\*Calc pointers from histogram to find how many tuples will calc every process and which tuples will calc every process\*/

if(rank==0){

average=lines/numprocs;

for(j=0;j<p\_i\*numprocs+1;j++){

procaverage=procaverage+totalHist[j];

if (proclines==0 && procaverage>=average){

proclines=procaverage;//save the total tuples that process 0 will calc

point[0]=j;//point which tuples will calc

procaverage=0;

}

else if ((proclines!=0 && procaverage>=average)||( j==p\_i\*numprocs)){

/\*Send the number of tuples that other processes will calc \*/

MPI\_Send(&procaverage, 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD);

point[i]=j;//point which tuples will calc every process

i++;

procaverage=0;

if (i==numprocs){

break;

}

}

}

}

else /\*Receive other processes the number of tuples that will calc\*/

MPI\_Recv(&proclines, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

//Broadcast average of number of tuples divided with number of processes

MPI\_Bcast(&average,1, MPI\_INT, root, MPI\_COMM\_WORLD);

free(hist);//free memory of hist array

free(totalHist);//free memory of final hist array

MPI\_Barrier(MPI\_COMM\_WORLD);

s=proclines/average;//Calc the s in every process

printf("rank %d s=%f\n",rank,s);

MPI\_Barrier(MPI\_COMM\_WORLD);

//Calc the max s from all processes

MPI\_Reduce(&s,&ss,1,MPI\_FLOAT,MPI\_MAX,0,MPI\_COMM\_WORLD);

//Broadcast the Max s

MPI\_Bcast(&ss,1, MPI\_FLOAT, root, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

g++;

}while(ss>spec);//if Max s is <= than s that we defined in console go next else go on do and double the p\_i

MPI\_Barrier(MPI\_COMM\_WORLD);

//Broadcast pointers for know where tuples have to calc every process

MPI\_Bcast(point,numprocs, MPI\_INT, root, MPI\_COMM\_WORLD);

//Define 3d char array to for put every process the tubles for other processes

char index[numprocs][prolines+1][101];

MPI\_Barrier(MPI\_COMM\_WORLD);

//Define counter to know how many tuples we have put in 3d for every process

count= (int\*) calloc(numprocs,sizeof(int));

//Put the tuples for other processes in 3d array and calc how many with counter

for (i=0;i<prolines;i++){

for (j=0;j<numprocs;j++){

if (j<numprocs-1){

if (strcmp(procKeys[i], totalPivots[point[j]]) < 0){

strcpy(index[j][count[j]],procArray[i]);

count[j]++;

j=numprocs;

}

}

else{

strcpy(index[numprocs-1][count[j]],procArray[i]);

count[j]++;

}

}

}

//Free memory from data that we no need

free(&(procKeys[0][0]));

free(procKeys);

free(&(procArray[0][0]));

free(procArray);

free(&(totalPivots[0][0]));

free(totalPivots);

//Make array to save how many tuples will receive every process for calc

elemCount=(int \*)malloc(sizeof(int)\*numprocs);

MPI\_Barrier(MPI\_COMM\_WORLD);

//calc how many tuples will receive every process from other processes

for( i = 0; i < numprocs; i++){

MPI\_Gather(&(count[i]), 1, MPI\_INT, elemCount, 1, MPI\_INT, i, MPI\_COMM\_WORLD);

}

//Sum the total tuples that every process will receive from all other processes

for(i=0;i<numprocs;i++){

k=k+elemCount[i];

}

//Allocate memory for every process to save the tuples that will receive from other processes

getMemory(&calcBuff,k,101);

//calc the size in bytes that will receive a process from every process

for(i = 0; i<numprocs; i++){

elemCount[i] = sizeof(char)\*(elemCount[i]\*101);

}

//Set the points in memory which a process must know to receive the tuples and save true in memory

displs[0]=0;

for( i = 0; i < numprocs-1; i++){

displs[i+1] = displs[i] + elemCount[i];

}

MPI\_Barrier(MPI\_COMM\_WORLD);

//Every process receive the tuples from other processes and save in memory

for( i = 0; i < numprocs; i++){

MPI\_Gatherv(index[i][0],count[i]\*101 , MPI\_CHAR,calcBuff[0], elemCount, displs, MPI\_CHAR, i, MPI\_COMM\_WORLD);

}

free(count);

MPI\_Barrier(MPI\_COMM\_WORLD);

//Every process sort the tuples

char temps[101];

for (i = 0; i < k; i++) {

for (j = 0; j < k - 1; j++) {

if (strcmp(calcBuff[j], calcBuff[j + 1]) > 0) {

strcpy(temps, calcBuff[j]);

strcpy(calcBuff[j], calcBuff[j + 1]);

strcpy(calcBuff[j + 1], temps);

}

}

}

MPI\_Barrier(MPI\_COMM\_WORLD);

//Allocate memory for save all sorted lists from all processes

getMemory(&sortedArray,lines,101);

//save in array the size of sorted list in every process

MPI\_Gather(&k, 1, MPI\_INT, elemCount, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

//calc the size in bytes of sorted lists from every process

for(i = 0; i<numprocs; i++){

sendcounts[i] = sizeof(char)\*(elemCount[i]\*101);

}

free(elemCount);

//set the pointers where the last array will receive the sorted lists from all processes

displs[0]=0;

for(i = 1; i<numprocs; i++){

displs[i]=displs[i-1]+sendcounts[i-1];

}

//Save all sorted lists in the final sorted list

MPI\_Gatherv(calcBuff[0],k\*101 , MPI\_CHAR,sortedArray[0], sendcounts, displs, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

free(&(calcBuff[0][0]));

free(calcBuff);

free(displs);

free(sendcounts);

//Process 0 print in console the sorted tuples that we read from file

if (rank==0){

for(i=0;i<lines;i++)

printf("%s\n",sortedArray[i]);

}

MPI\_Barrier(MPI\_COMM\_WORLD);

//Process 0 print the s and p\_i that algorithm ran

if (rank==0){

printf("Perfect!! Agorithm run with p\_i=%d\n",p\_i);

printf("s=%f\n",ss);

}

MPI\_Finalize();

}

Εργασία 2

**Ψευδοκώδικας 2ης άσκησης**

1: Δήλωση Μεταβλητών

2: Αρχικοποίηση του MPI

3: Άνοιγμα του αρχείου, σε περίπτωση που δεν βρεθεί, τερματισμός του προγράμματος και εμφάνιση μηνύματος, διαφορετικά διάβασμα του αρχείου και τοποθέτηση των δεδομένων σε πίνακες

4: Ταξινόμηση του πίνακα unique\_sorted\_vec και αφαίρεση των διπλών εγγραφών

5: Για κάθε τιμή του παραπάνω πίνακα, έλεγχος για την θέση τους στο αρχείο και επιλογή του γειτονικού κόμβου

6: Αναδιανομή του πίνακα network που περιλαμβάνει το σύνολο των κόμβων στις διεργασίες

7: Για κάθε τιμή του partition επιλογή του υποπίνακα Nv, ταξινόμησή του και για κάθε τιμή του έλεγχος αν είναι μέρος του partition, αν είναι τότε γίνεται εύρεση των γειτονικών του κόμβων Nu και υπολογισμός των τομών. Αποθήκευση στην μεταβλητή Τ.

8: Αν ο γείτονας Nu βρίσκεται σε διαφορικό partition, εύρεση του και αποστολή στην διεργασία που το χειρίζεται για τον υπολογισμό των τριγώνων.

9: Διάβασμα του buffer για τυχόν μηνύματα

10: Αν το μήνυμα είναι τύπου data τότε διαβάζουμε τον πίνακα των γειτόνων και υπολογίζουμε το Τ. Διαφορετικά αν είναι μήνυμα notifier προσθέτουμε +1 στο σύνολο των διεργασιών που ολοκλήρωσαν τον υπολογισμό των τριγώνων.

11: Ενημέρωση όλων των διεργασιών για την ολοκλήρωση των υπολογισμών με μήνυμα notifier και όσο ο αριθμός του των διεργασιών που ολοκληρώθηκαν δεν είναι το σύνολο αναμονή και διάβασμα τυχόν μηνυμάτων. Αν το μήνυμα είναι τύπου data τότε διαβάζουμε τον πίνακα των γειτόνων και υπολογίζουμε το Τ. Διαφορετικά αν είναι μήνυμα notifier προσθέτουμε +1 στο σύνολο των διεργασιών που ολοκλήρωσαν τον υπολογισμό των τριγώνων.

12: Συγχρονισμός των διεργασιών

13: Άθροισμα των τριγώνων στην διεργασία 0

14: Εμφάνιση του αριθμού των τριγώνων και τερματισμός του προγράμματος.

Στην εργασία 2 φτιάξαμε τον αλγόριθμο και δουλεύει σειριακά κανονικά μέσα στο φάκελο εργασία δύο θα βρείτε το σειριακό κώδικα καθώς και τον παράλληλο μέχρι εκεί που φτάσαμε. Η δεύτερη εργασία έγινε με C++.

**O παράλληλος κώδικας μέχρι το σημείο που φτάσαμε**

#include <iostream>

#include <fstream>

#include <list>

#include <algorithm>

#include <vector>

#include <sys/stat.h>

#include <set>

#include "mpi/mpi.h"

using namespace std;

inline bool fileExists (const std::string& name) { // Έλεγχος αν υπάρχει το αρχείο που πρόκειται να διαβαστεί

struct stat buffer;

return (stat (name.c\_str(), &buffer) == 0);

}

int main(int argc, char \*argv[]) {

// Δήλωση μεταβλητών

int rank, count, partitionNumber, mod;

const int root = 0;

int v = 0, u = 0, T = 0, globalT = 0, j =0, completion\_counter;

vector<int> v\_vec, u\_vec, unique\_sorted\_vec, neighbours\_vec, Nv, Nu, res\_set, partition, nodes;

vector<vector<int> > network, Vi;

// Αρχικοποίηση του MPI

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &count);

int \*point = (int\*)malloc(sizeof(int)\*count);

int \*displs = (int\*)malloc(sizeof(int)\*count);

const char \*fileName = "undirected\_test.txt"; // Το αρχείο που θα διαβαστεί

// Διάβασμα του αρχείου

if (fileExists(fileName)) {

ifstream infile(fileName);

while (infile >> v >> u) {

v\_vec.insert(v\_vec.end(), v);

u\_vec.insert(u\_vec.end(), u);

}

} else {

cout << "File Not Found!" << endl; // Σε περίπτωση όπου το αρχείο δεν βρεθεί

MPI\_Finalize(); // το πρόγραμμα τερματίζει εμφανίζοντας μήνυμα

return EXIT\_FAILURE;

}

unique\_sorted\_vec = v\_vec;

sort(unique\_sorted\_vec.begin(), unique\_sorted\_vec.end()); // Σορτάρισμα του πίνακα

unique\_sorted\_vec.erase(unique(unique\_sorted\_vec.begin(), unique\_sorted\_vec.end()), unique\_sorted\_vec.end()); // Αφαίρεση διπλών κόμβων

// Ο βρόγχος for προσθέτει σε έναν δυναμικό πίνακα πινάκων network όλους τους γείτονες για τους κόμβους που βρίσκονται

for (int i = 0; i < unique\_sorted\_vec.size(); i++) { // στον πίνακα unique\_sorted\_vec

for (int y = 0; y < v\_vec.size(); y++) {

if (unique\_sorted\_vec[i] == v\_vec[y]) {

neighbours\_vec.insert(neighbours\_vec.end(), u\_vec[y]);

}

}

network.insert(network.end(), neighbours\_vec);

neighbours\_vec.clear();

}

// Αναδιανομή του πίνακα network με βάση τον αριθμό των διεργασιών

int num = network.size();

partitionNumber = num / count;

mod = num % count;

for (int i=0; i<mod; i++) {

if(rank==i) {

partitionNumber++;

}

}

MPI\_Gather(&partitionNumber,1, MPI\_INT, point, 1,MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(point, count, MPI\_INT,root, MPI\_COMM\_WORLD);

displs[0] = 0;

for (int i = 1; i<count; i++) {

displs[i] = displs[i-1] + point[i-1];

}

for (int i = 0; i < count; i++) {

if (rank==i && i< count-1) {

for (int y = displs[i]; y< displs[i+1]; y++) {

Vi.insert(Vi.end(), network[y]);

}

} else if (rank==i) {

int k = 1;

for (int y = displs[i]; y<=displs[i]; y++) {

Vi.insert(Vi.end(), network[y]);

}

}

}

for (int i = 0; i < count; i++) {

if (rank==i && i< count-1) {

for (int y = displs[i]; y< displs[i+1]; y++) {

nodes.insert(nodes.end(), unique\_sorted\_vec[y]);

}

} else if (rank==i) {

int k = 1;

for (int y = displs[i]; y<=displs[i]; y++) {

nodes.insert(nodes.end(), unique\_sorted\_vec[y]);

}

}

}

// Υπολογισμός τριγώνων που βρίσκονται στον πίνακα πινάκων Vi

for (int i = 0; i < Vi.size(); i++) {

for (int j = 0; j < Vi[i].size(); j++) {

Nv.insert(Nv.end(), Vi[i][j]); // Για κάθε υποπίνακα του Vi τον περνάμε στον πίνακα Nv

}

sort(Nv.begin(), Nv.end());

for (int y = 0; y < Nv.size(); y++) {

vector<int>::iterator itB = find(nodes.begin(), nodes.end(), Nv[y]); // Βρίσκουμε τον γείτονα και το index του

if (itB != nodes.end()) { // στον πίνακα, εάν δεν υπάρχει εκεί

int pos = itB - nodes.begin(); // τον στέλνουμε στην διεργασία που

for (int j = 0; j < Vi[pos].size(); j++) { // έχει αναλάβει τον κόμβο

Nu.insert(Nu.end(), Vi[pos][j]);

}

sort(Nu.begin(), Nu.end());

set\_intersection(Nv.begin(), Nv.end(), Nu.begin(), Nu.end(), std::inserter(res\_set, res\_set.end()));

T += res\_set.size(); //Υπολογίζουμε τις τομές και το μέγεθος του πίνακα res\_set δίνει τον αριθμό των τριγώνων για την τομή

res\_set.clear();

Nu.clear();

} else {

vector<int>::iterator itA = find(nodes.begin(), nodes.end(), Nv[y]);

int pos = itA - unique\_sorted\_vec.begin();

int f = 0, sendRank;

for (int i = 0; i < count; i++) { // Βρίσκουμε σε ποιο partition υπάρχει ο γείτονας

f = f + point[i];

if (f >= pos) {

sendRank = i;

break;

}

}

MPI\_Send(); // Στέλνουμε τους εναπομείναντες γείτονες στην διεργασίας που περιλαμβάνει τον πρώτο γειτονικό κόμβο

}

}

MPI\_Recv(); // Ελέγχουμε για τυχόν μηνύματα που υπάρχουν στον buffer

if (tag == data) { // Αν το μήνυμα που λάβαμε είναι μήνυμα data διαβάζουμε τον πίνακα με τους γείτονες

//Υπολογίζουμε τα τρίγωνα

} else {

completion\_counter++; // Διαφορετικά αν είναι notifier θεωρούμε ότι μια διεργασία τελείωσε τον υπολογισμό των τριγώνων

}

}

MPI\_Bcast(); // Έχουμε υπολογίσει τα τρίγωνα για το partition και ενημερώνουμε τις υπόλειπες διεργασίες με μήνυμα notifier

while (completion\_counter <= count) // Αν κάποια διεργασία δεν έχει ολοκληρώσει με τον υπολογισμό, της αναμένουμε

// για την πιθανή λήψη γειτόνων

MPI\_Recv(); // Ελέγχουμε για τυχόν μηνύματα που υπάρχουν στον buffer

if (tag == data) { // Αν το μήνυμα που λάβαμε είναι μήνυμα data διαβάζουμε τον πίνακα με τους γείτονες

//Υπολογίζουμε τα τρίγωνα

} else {

completion\_counter++; // Διαφορετικά αν είναι notifier θεωρούμε ότι μια διεργασία τελείωσε τον υπολογισμό των τριγώνων

}

MPI\_Barrier(MPI\_COMM\_WORLD); // Συγχρονισμός διεργασιών

MPI\_Reduce(&T, &globalT, 1, MPI\_INT, MPI\_SUM, root, MPI\_COMM\_WORLD); // Άθροισμα των τριγώνων στην διεργασία 0

if (rank == root) {

cout << "Ο αριθμός των τριγώνων είναι: " << globalT << endl;

}

MPI\_Finalize(); // Ολοκλήρωση και τερματισμός του προγράμματος

return 0;

}

**Ο σειριακός κώδικας που φτιάξαμε αρχικά**

#include <iostream>

#include <fstream>

#include <list>

#include <algorithm>

#include <vector>

#include <sys/stat.h>

#include <set>

using namespace std;

inline bool fileExists (const std::string& name) { // Έλεγχος αν υπάρχει το αρχείο που πρόκειται να διαβαστεί

struct stat buffer;

return (stat (name.c\_str(), &buffer) == 0);

}

int main() {

// Το αρχείο που θα διαβαστεί

const char \*fileName = "facebook\_combined.txt"; //facebook\_combined.txt

int v = 0, u = 0, t = 0;

vector<int> v\_vec, u\_vec, unique\_sorted\_vec, neighbours\_vec, Nv, Nu, res\_set;

vector<vector<int> > network;

// Διάβασμα του αρχείου

if (fileExists(fileName)) {

ifstream infile(fileName);

while (infile >> v >> u) {

v\_vec.insert(v\_vec.end(), v);

u\_vec.insert(u\_vec.end(), u);

}

} else {

cout << "File Not Found!" << endl; // Σε περίπτωση όπου το αρχείο δεν βρεθεί το πρόγραμμα τερματίζει εμφανίζοντας μήνυμα

return EXIT\_FAILURE;

}

unique\_sorted\_vec = v\_vec;

sort(unique\_sorted\_vec.begin(), unique\_sorted\_vec.end()); // Σορτάρισμα του πίνακα

unique\_sorted\_vec.erase(unique(unique\_sorted\_vec.begin(), unique\_sorted\_vec.end()), unique\_sorted\_vec.end()); // Αφαίρεση διπλών κόμβων

// Ο βρόγχος for προσθέτει σε έναν δυναμικό πίνακα πινάκων network όλους τους γείτονες για τους κόμβους που βρίσκονται

for (int i = 0; i < unique\_sorted\_vec.size(); i++) { // στον πίνακα unique\_sorted\_vec

for (int y = 0; y < v\_vec.size(); y++) {

if (unique\_sorted\_vec[i] == v\_vec[y]) {

neighbours\_vec.insert(neighbours\_vec.end(), u\_vec[y]);

}

}

network.insert(network.end(), neighbours\_vec);

neighbours\_vec.clear();

}

// Υπολογισμός τριγώνων που βρίσκονται στον πίνακα πινάκων network

for (int i = 0; i < network.size(); i++) {

for (int j = 0; j < network[i].size(); j++) {

Nv.insert(Nv.end(), network[i][j]);

}

sort(Nv.begin(), Nv.end());

for (int it=0; it < Nv.size(); it++) {

vector<int>::iterator itB = find(unique\_sorted\_vec.begin(), unique\_sorted\_vec.end(), Nv[it]);

int pos = itB - unique\_sorted\_vec.begin();

if (itB != unique\_sorted\_vec.end()) {

for (int j = 0; j < network[pos].size(); j++) {

Nu.insert(Nu.end(), network[pos][j]);

}

sort(Nu.begin(), Nu.end());

//Υπολογίζουμε τις τομές και το μέγεθος του πίνακα res\_set δίνει τον αριθμό των τριγώνων για την τομή

set\_intersection(Nv.begin(), Nv.end(), Nu.begin(), Nu.end(), std::inserter(res\_set, res\_set.end()));

t += res\_set.size();

res\_set.clear();

Nu.clear();

}

}

Nv.clear();

}

cout << "Ο αριθμός των τριγώνων για το αρχείο " << fileName <<" είναι: " << t << endl;

return 0;

}