NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL

DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 10

28th October 2020

Faculty: Dr. Geetha V and Mrs. Thanmayee

Bhagyashri Bhamare-181IT111

1. In a smart agriculture system in a large area like a state, sensors are deployed to collecttemperature and humidity. The sensed information are stored in a server in the cloud. A query on calculating the average temperature and average humidity of the complete state needs the processing of 10 lakh data elements. Write a parallel program using MPI in which N number of processes run in parallel to calculate the average of 10 lakh elements stored in an array, in order to improve response time. Compare the execution time with sequential code.

a) Note: You may use number of elements to be smaller than 10 lakh for testing, as you have to initialize that many elements.

b) Justify the usage of MPI routines used in the program with clear comments. [2 marks]

c) Code, results and analysis [3 marks]

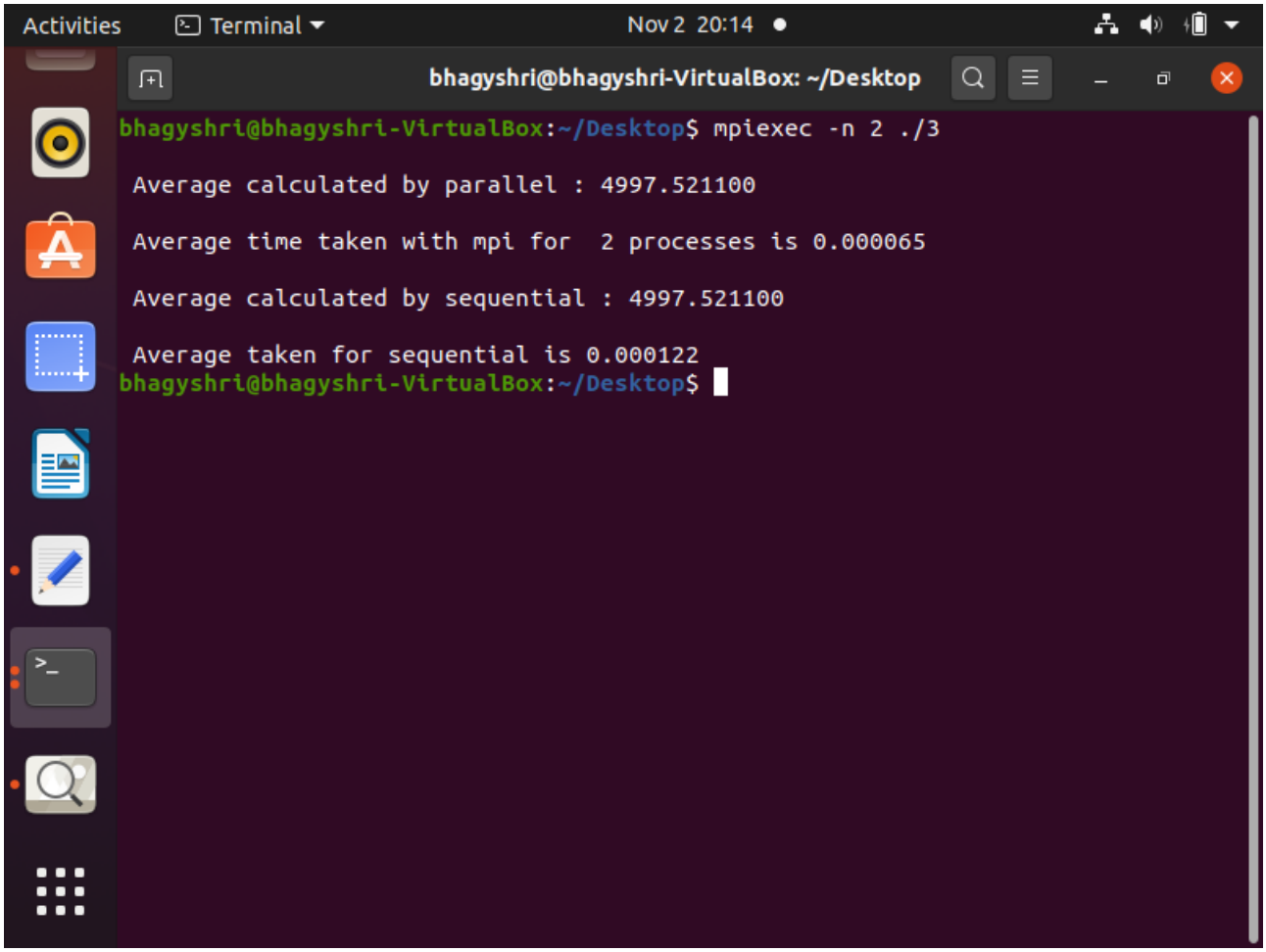
Code-

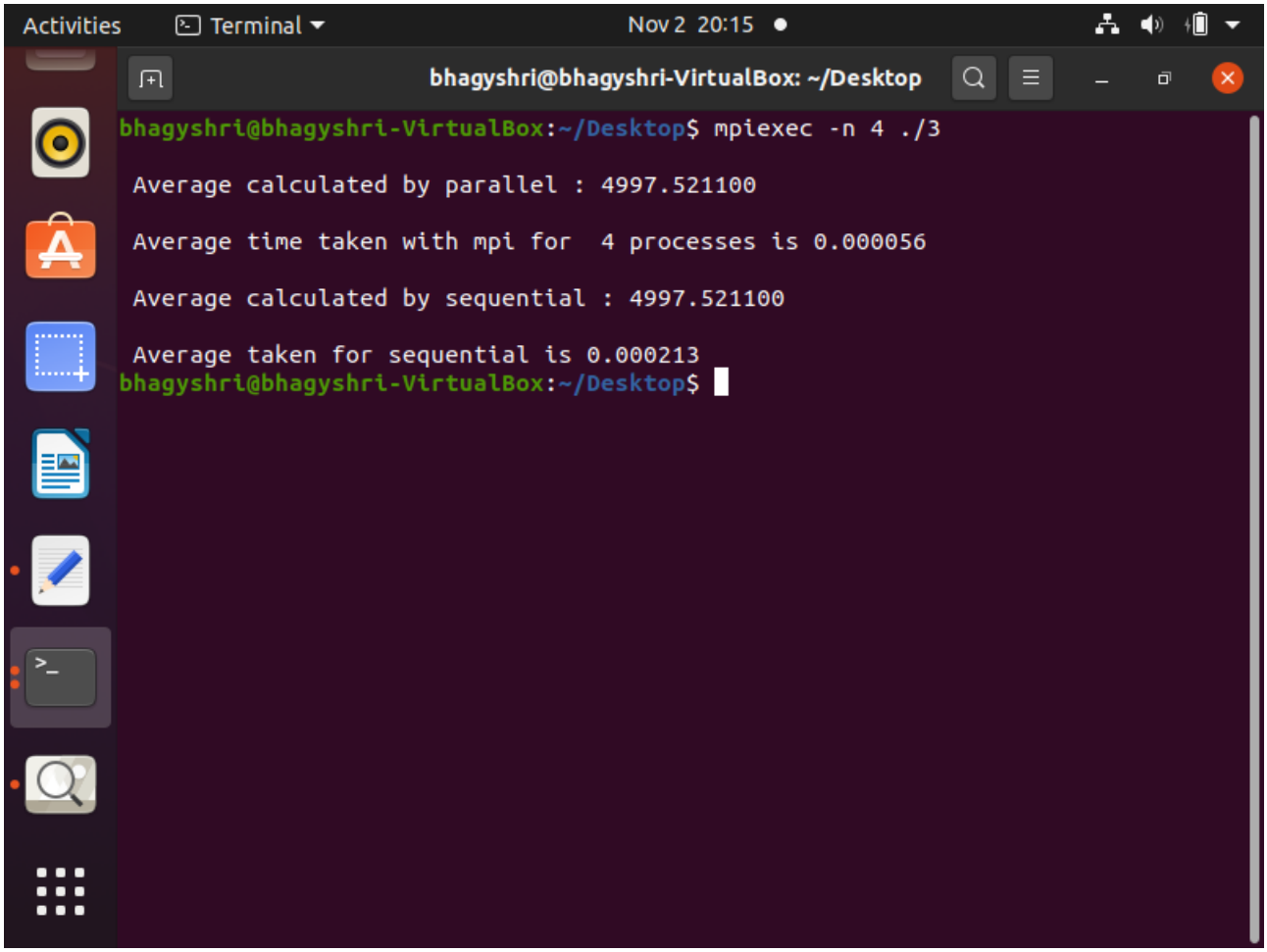
```
#include<mpi.h>
#include<stdio.h>

int main(int argc,char *argv[ ])
{
int n= 10000;
int size,myrank;

int element[n];
int buffer[n];
double sum=0;
double avg=0;
double GlobalAvg;
int indSize;
double s;
```

```
int sendcount[n],displacement[n];
double start,end,time,avgtime;
MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
indSize=n/size;
s=size;
if(myrank==0)
{

//printf("\n The array elements are:");
for(int i=0;i<n;i++)
{
element[i] =1+ rand()%n;
//printf("%d ",element[i]);
}
}

MPI_Barrier(MPI_COMM_WORLD);
start = MPI_Wtime();
displacement[0]=0;
for(int i=0;i<size-1;i++)
{
    sendcount[i]=indSize;
    displacement[i+1]=displacement[i]+indSize;
}
sendcount[size-1]=n-(size-1)*indSize;
MPI_Scatterv(&element,sendcount,displacement,MPI_INT,&buffer,sendcount[myrank],MPI_INT,0,MPI
_COMM_WORLD);

for(int i=0;i<sendcount[myrank];i++)
{  //printf("Process %d Scanning %d\n",myrank,buffer[i]);
    sum+=buffer[i];
}
avg=sum/sendcount[myrank];
```

```
//printf("\nAvg element in process %d is %d\n",myrank,avg);
MPI_Reduce(&avg,&GlobalAvg,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
if(myrank ==0)
{   GlobalAvg/=size;
      printf("\n Average calculated by parallel : %f\n",GlobalAvg );
}

end=MPI_Wtime();
time=end-start;

MPI_Reduce(&time,&avgtime,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
avgtime= avgtime/s;
if(myrank==0)
printf("\n Average time taken with mpi for  %d processes is %f\n",size,avgtime);
MPI_Barrier(MPI_COMM_WORLD);
start=MPI_Wtime();
if(myrank==0)
{sum=0;
for(int i=0;i<n;i++)
{
sum+=element[i];
}
GlobalAvg=sum/n;
printf("\n Average calculated by parallel : %f\n",GlobalAvg );
}
MPI_Barrier(MPI_COMM_WORLD);
end=MPI_Wtime();
avgtime=end-start;

if(myrank==0)
{printf("\n Average taken for sequential is %f\n",avgtime);}
MPI_Finalize();
      return 0;
}
```

Output-


Activities
Terminal
Nov 2 20:14
bhagyshri@bhagyshri-VirtualBox: ~/Desktop
bhagyshri@bhagyshri-VirtualBox:~/Desktop$ mpiexec -n 2 ./3
Average calculated by parallel : 4997.521100
Average time taken with mpi for 2 processes is 0.000065
Average calculated by sequential : 4997.521100
Average taken for sequential is 0.000122
bhagyshri@bhagyshri-VirtualBox:~/Desktop$



Analysis-sequensial excuation is taking more time than parallel as work distributed between all the process.Here I have used two important MPI routines MPI_Scatter() and MPI_Reduce(). MPI_Scatter it is used for spreading the array among all process equally MPI_Reduce is used for combining the local sum of all the processes

2. Consider random deployment of sensor nodes in field to sense the environment. Thenodes are deployed randomly and the position of each sensor node is sent to centralised server. The server would like to cluster these nodes. Use K-means algorithm to cluster the nodes. Write an MPI program to cluster the sensor nodes and compare the result with sequential and OPENMP approach. For implementation consider the following things.

a) Assume 1000 sensor nodes are deployed in 1000m x 1000m area. Generate the position of each node using random function. [Already done in Lab 7]

b) Implement the algorithm to make 2 clusters, 4 clusters and 8 clusters. Compare the result with sequential algorithm. [2 Marks]

c) Using some graphical tools, plot the clusters and positions of each node. [1 Mark]

d) Program Code with comments on MPI routines used and results [2 Marks]

Code-

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
float* create_rand_nums(const int num_elements) {
float *rand_nums = (float *)malloc(sizeof(float) * num_elements);
float a = 999;
for (int i = 0; i < num_elements; i++) {
rand_nums[i] = ((float)rand()/(float)RAND_MAX) * a;
}
return rand_nums;
}
float distance2(const float *v1, const float *v2, const int d) {
float dist = 0.0;
for (int i=0; i<d; i++) {
float diff = v1[i] - v2[i];
dist += diff * diff;
}
return dist;
}
int assign_site(const float* site, float* centroids,
const int k, const int d) {
int best_cluster = 0;
float best_dist = distance2(site, centroids, d);
float* centroid = centroids + d;
for (int c = 1; c < k; c++, centroid += d) {
float dist = distance2(site, centroid, d);
if (dist < best_dist) {
best_cluster = c;
best_dist = dist;
}
}
return best_cluster;
}
void add_site(const float * site, float * sum, const int d) {
for (int i=0; i<d; i++) {
sum[i] += site[i];
}
}
void print_centroids(float * centroids, const int k, const int d) {
FILE *fp;
char *filename = "cluster_parallel.txt";
fp = fopen(filename, "w");
float *p = centroids;
printf("Centroids:\n");
for (int i = 0; i<k; i++) {
for (int j = 0; j<d; j++, p++) {
```

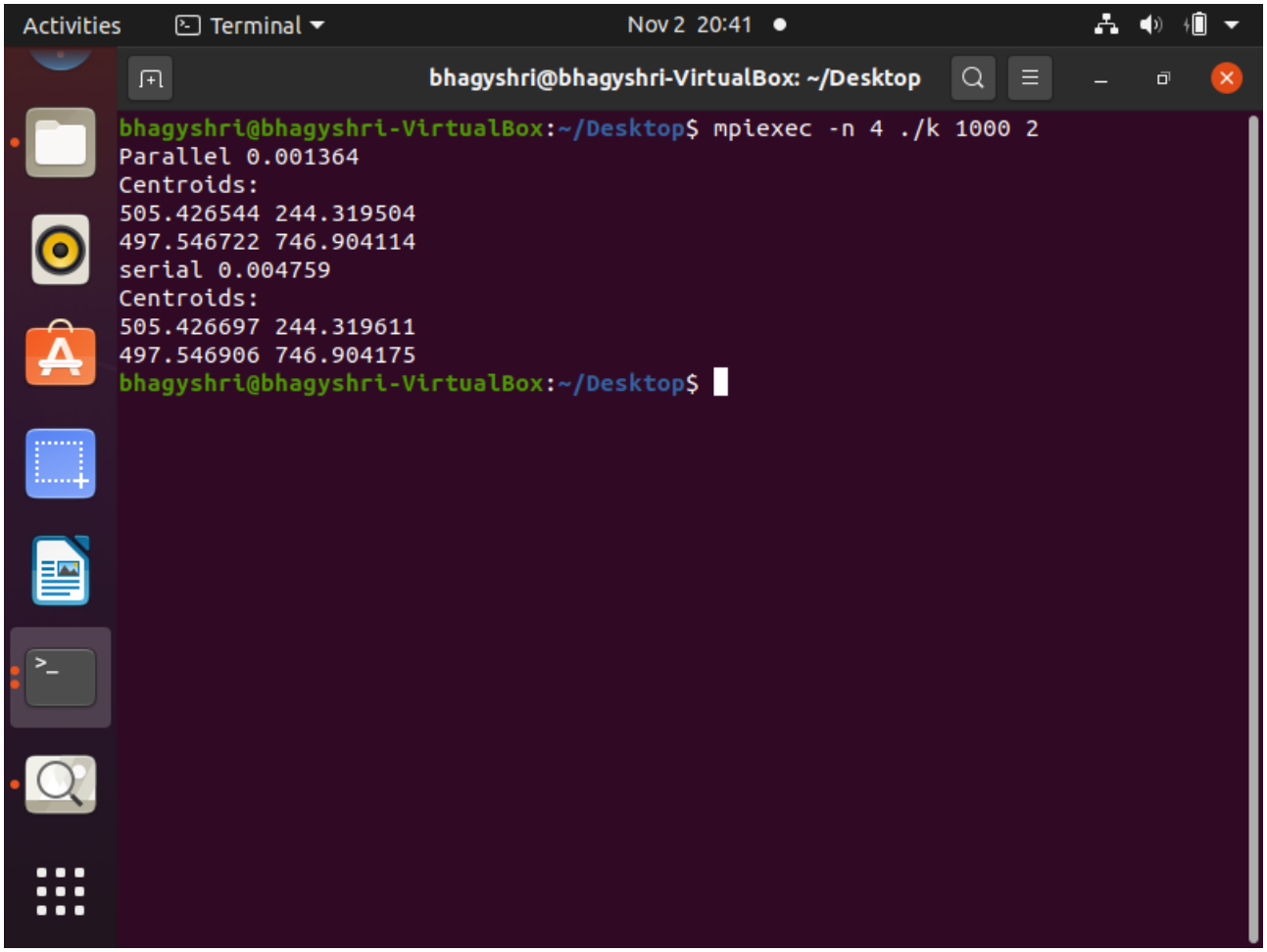
```
printf("%f ", *p);
fprintf(fp, "%f ",*p);
}
fprintf(fp, "%d\n",i);
printf("\n");
}
}
double kmeans_serial(float* all_sites,int k,int arr_size)
{
float* centroids;
centroids = malloc(k * 2 * sizeof(float));
float* grand_sums = NULL;
int* grand_counts = NULL;
int* all_labels;
for (int i = 0; i < k * 2; i++) {
centroids[i] = all_sites[i];
}
grand_sums = malloc(k * 2 * sizeof(float));
grand_counts = malloc(k * sizeof(int));
all_labels = malloc(arr_size * sizeof(int));
double start_time = MPI_Wtime();
float norm = 1.0;
while (norm > 0.00001) {
for (int i = 0; i < k*2; i++) grand_sums[i] = 0.0;
for (int i = 0; i < k; i++) grand_counts[i] = 0;
float* site = all_sites;
for (int i = 0; i < arr_size; i++, site += 2) {
int cluster = assign_site(site, centroids, k, 2);
grand_counts[cluster]++;
add_site(site, &grand_sums[cluster*2], 2);
}
for (int i = 0; i<k; i++) {
for (int j = 0; j<2; j++) {
int dij = 2*i + j;
grand_sums[dij] /= grand_counts[i];
}
}
norm = distance2(grand_sums, centroids, 2*k);
for (int i=0; i<k*2; i++) {
centroids[i] = grand_sums[i];
}
}
float* site = all_sites;
for (int i = 0; i < arr_size; i++, site += 2) {
all_labels[i] = assign_site(site, centroids, k, 2);
}
double end_time = MPI_Wtime()-start_time;
```

```
printf("serial %f \n",end_time);
print_centroids(centroids,k,2);
return 0;
}
int main(int argc, char** argv) {
if (argc != 3) {
fprintf(stderr,
"Usage: kmeans num_sites_per_proc num_means num_dimensions\n");
exit(1);
}
int sites_per_proc = atoi(argv[1]);
int k = atoi(argv[2]); // number of clusters.
srand(31359);
MPI_Init(NULL, NULL);
int rank, nprocs;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
float* sites;
sites = malloc(sites_per_proc * 2 * sizeof(float));
float* sums;
sums = malloc(k * 2 * sizeof(float));
int* counts;
counts = malloc(k * sizeof(int));
float* centroids;
centroids = malloc(k * 2 * sizeof(float));
int* labels;
labels = malloc(sites_per_proc * sizeof(int));
float* all_sites = NULL;
float* grand_sums = NULL;
int* grand_counts = NULL;
int* all_labels;
if (rank == 0) {
all_sites = create_rand_nums(2 * sites_per_proc * nprocs);
for (int i = 0; i < k * 2; i++) {
centroids[i] = all_sites[i];
}
grand_sums = malloc(k * 2 * sizeof(float));
grand_counts = malloc(k * sizeof(int));
all_labels = malloc(nprocs * sites_per_proc * sizeof(int));
}
MPI_Scatter(all_sites,2*sites_per_proc, MPI_FLOAT, sites,
2*sites_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
double start_time = MPI_Wtime();
float norm = 1.0;
while (norm > 0.00001) {
MPI_Bcast(centroids, k*2, MPI_FLOAT,0, MPI_COMM_WORLD);
```

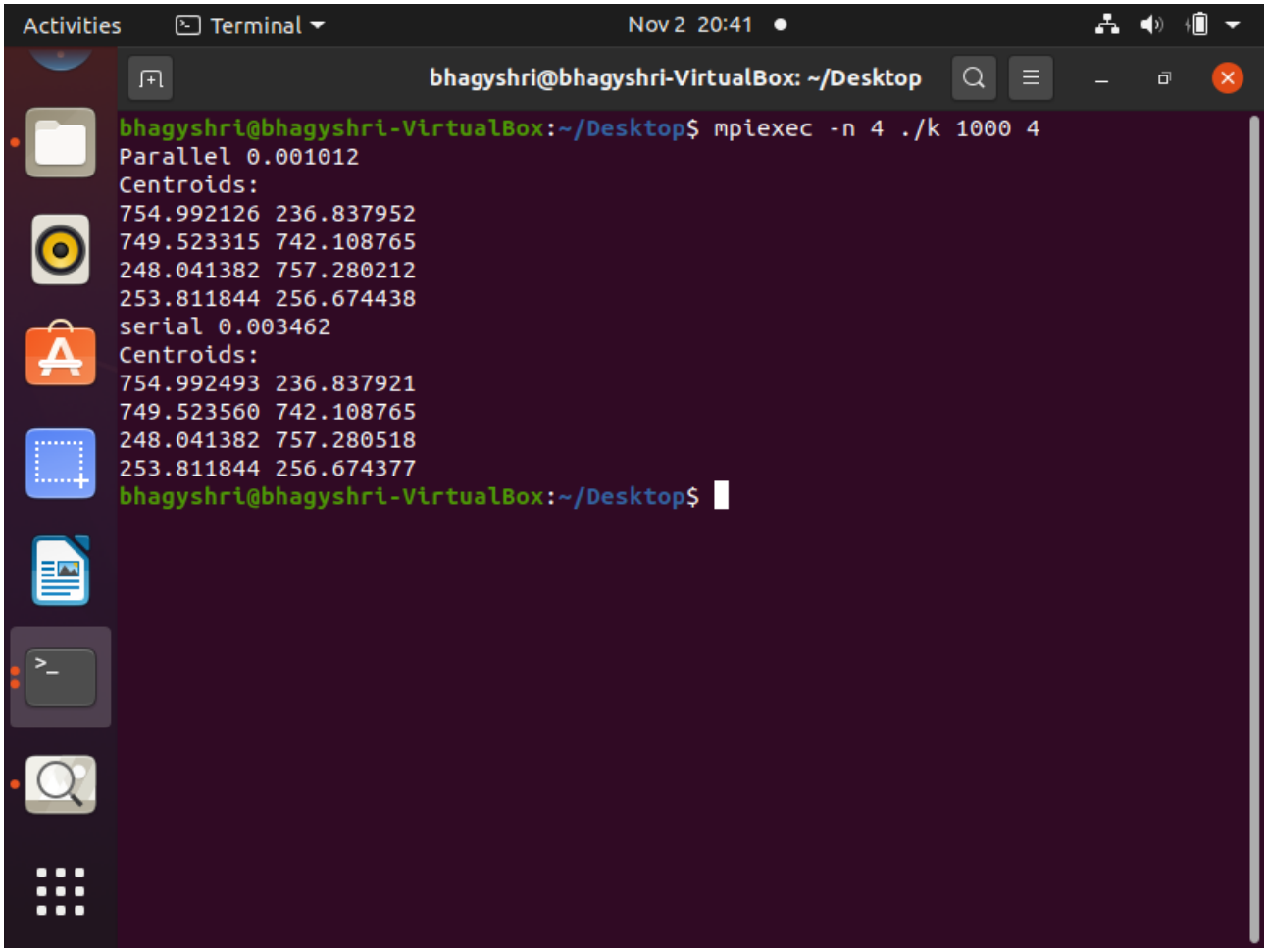
```
for (int i = 0; i < k*2; i++) sums[i] = 0.0;
for (int i = 0; i < k; i++) counts[i] = 0;
float* site = sites;
for (int i = 0; i < sites_per_proc; i++, site += 2) {
int cluster = assign_site(site, centroids, k, 2);
counts[cluster]++;
add_site(site, &sums[cluster*2], 2);
}
MPI_Reduce(sums, grand_sums, k * 2, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce(counts, grand_counts, k, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if (rank == 0) {
for (int i = 0; i<k; i++) {
for (int j = 0; j<2; j++) {
int dij = 2*i + j;
grand_sums[dij] /= grand_counts[i];
}
}
norm = distance2(grand_sums, centroids, 2*k);
for (int i=0; i<k*2; i++) {
centroids[i] = grand_sums[i];
}
}
MPI_Bcast(&norm, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
}
float* site = sites;
for (int i = 0; i < sites_per_proc; i++, site += 2) {
labels[i] = assign_site(site, centroids, k, 2);
}
MPI_Gather(labels, sites_per_proc, MPI_INT,all_labels, sites_per_proc, MPI_INT, 0,
MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
double end_time = MPI_Wtime()-start_time;
if(rank == 0)
{
printf("Parallel %f \n",end_time);
print_centroids(centroids,k,2);
}
if ((rank == 0) && 1) {
FILE *fp;
char *filename = "point_parallel.txt";
fp = fopen(filename, "w");
float* site = all_sites;
for (int i = 0;i < nprocs * sites_per_proc;i++, site += 2)
{
fprintf(fp, "%f %f %d \n", site[0],site[1],all_labels[i]);
}
fclose(fp);
```

```
kmeans_serial(all_sites,k,nprocs * sites_per_proc);
}
MPI_Finalize();
}
```

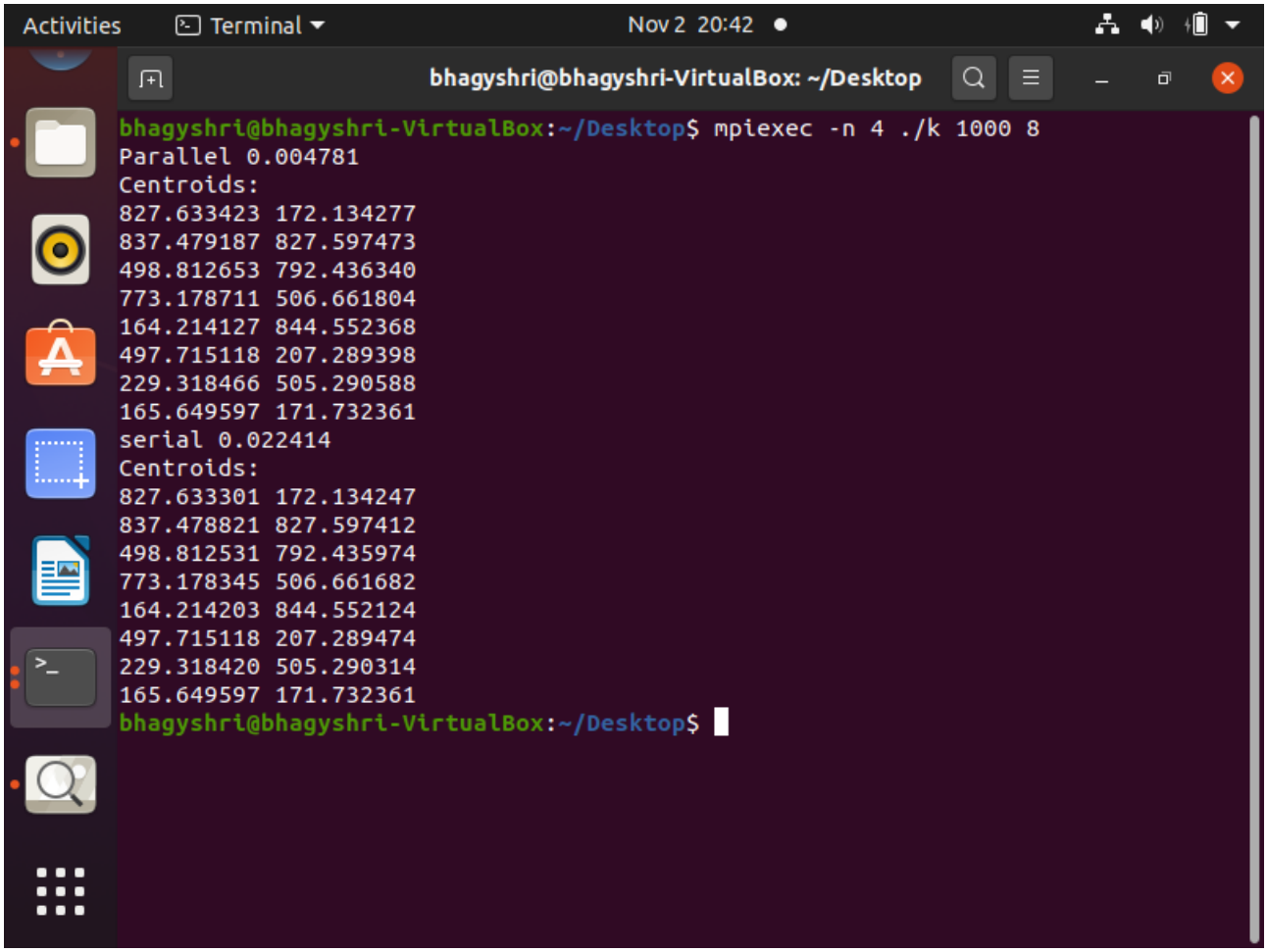
Number of cluster=2



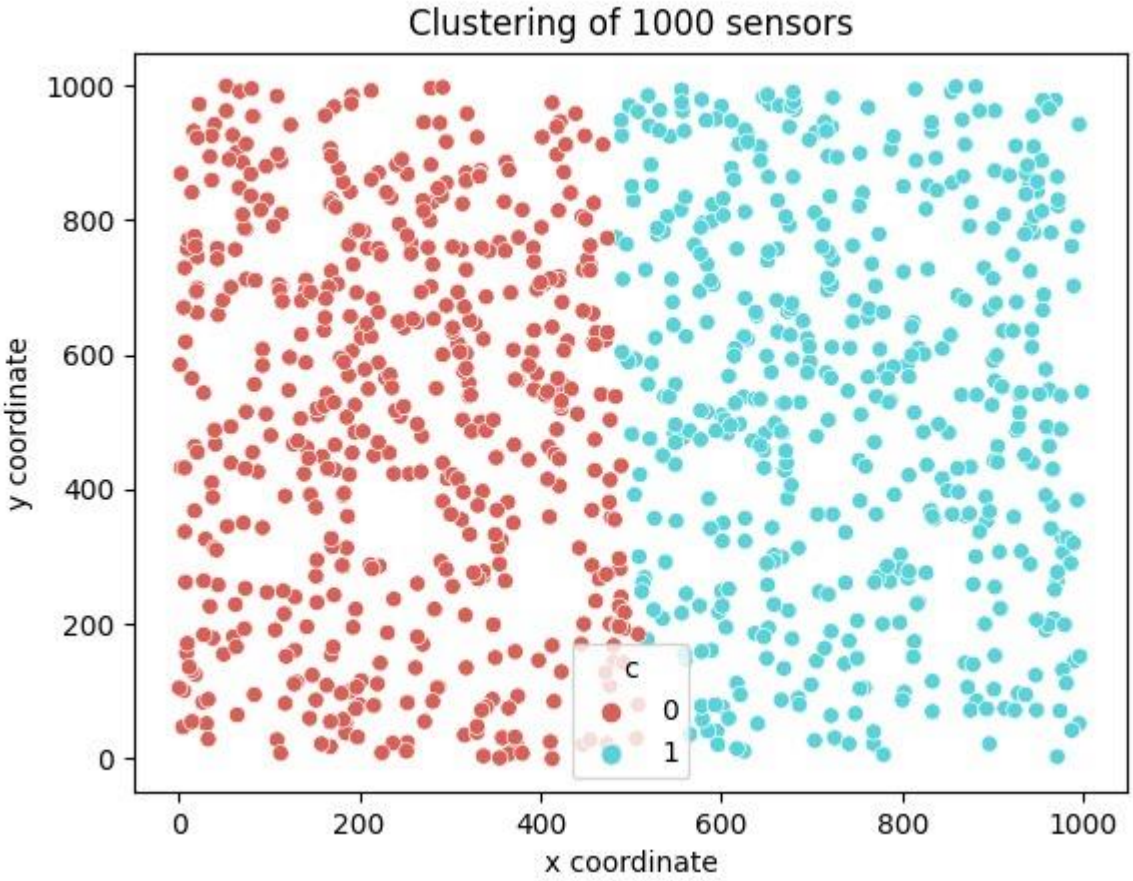
Number of cluster =4



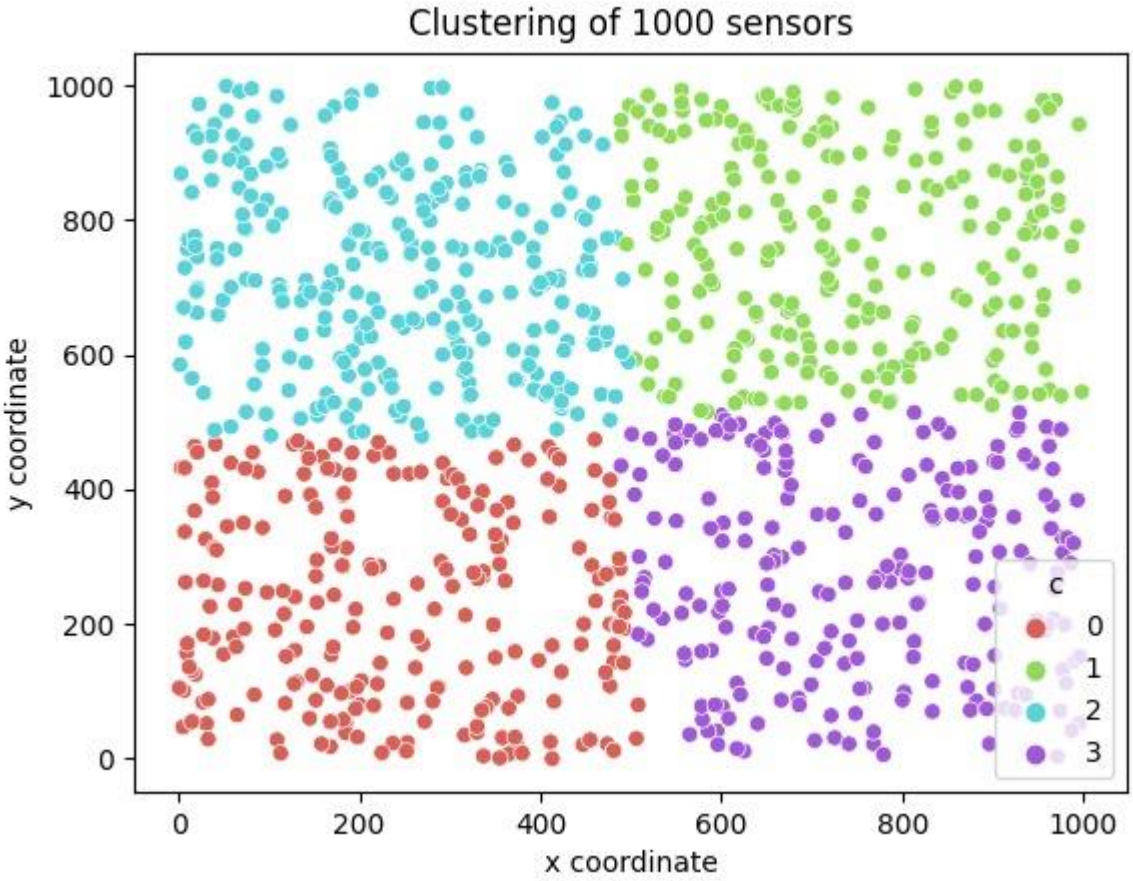
Number of cluster=8



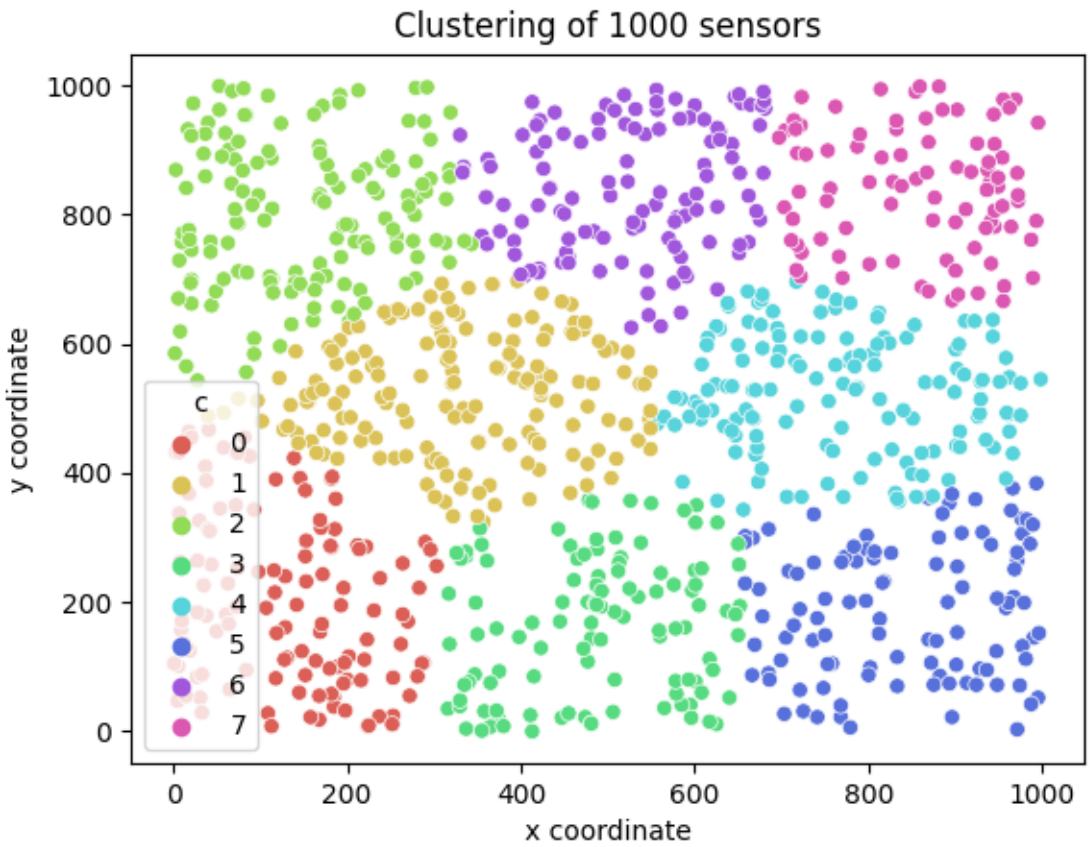
For number of cluster=2



For number of cluster=4



For number of cluster=8



Analysis-Here we can see that the execution time of parallel is less than the time of serial execution. In the program I have used MPI routines like MPI_Bcast(), MPI_Scatter(), MPI_Gather, MPI_Reduce(), MPI_Barrier().