

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

**UCCD3123 Multicore Programming**

**Assignment 2**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Student Name | ID | Course |
| 1 | Quah Yu Lit | 0903245 | CS Y3T3 |
| 1 | Quah Yu Soon | 0903244 | CS Y3T3 |

**1.0 Introduction**

This program is developed to solve vehicle routing problem (VRP) by compute the minimum time of tour that visits each city exactly once by any of the given vehicle using brute force method. The additional feature of this program is it executes parts of the codes in parallel by implementing WinThread and OpenMP.

First step, we scan the data from Map.vrp and store them in a struct, called COOR.

After that, we implemented threads to calculate the distance between all the cities so that computation is running in parallel and faster. The distanced calculated will be stored in a two dimension distanceArr[][] array.

In the threads, we also include a generatecombi( ) and create file functions. Here, we create some files first for later use. Below the threads, we put a WaitForMultipleObjects( ) to make sure all the threads finish executed before proceeding to permutation function. Inside the permutation , we will use various combination to split each permutation and findPathDist () will be used to compute maximum distance traveled among the vehicles. Then, we choose the lowest value of maximum distance as minimum time of traveling all the cities. The corresponding route will be this VRP solution.

At the end, we store the cities needed to be traveled by each car into separate files which created just now in the threads,namely Solution0.vrp, Solution1.vrp, Solution2.vrp and so on.

**2.0 Perfomance**

**2.1 Perfomance Charts**

**Sequential Vs OpenMP and Windows Threading**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Time(seconds)** | | |
|  | **Sequential** | **OpenMP** | **Windows Threading** |
| **3 cars 5 cities** | 0.004 | 0.005 | 0.005 |
| **3 cars 6 cities** | 0.005 | 0.005 | 0.005 |
| **3 cars 7 cities** | 0.011 | 0.011 | 0.010 |
| **3 cars 10 cities** | 11.98 | 11.93 | 11.86 |
| **4 cars 10 cities** | 29.29 | 29.11 | 29.1 |
| **5 cars 10 cities** | 45.1 | 44.6 | 44.5 |
| **3 cars 11 cities** | 172.5 | 171.9 | 171.92 |

**2.2 Speed Up**

|  |  |  |
| --- | --- | --- |
|  | **Speed up,S(p)=ts/tp** | |
|  | **OpenMP** | **Windows Threading** |
| **3 cars 5 cities** | 0.8 | 0.8 |
| **3 cars 6 cities** | 1 | 1 |
| **3 cars 7 cities** | 1 | 1.1 |
| **3 cars 10 cities** | 1.004 | 1.01 |
| **4 cars 10 cities** | 1.006 | 1.006 |
| **5 cars 10 cities** | 1.011 | 1.013 |
| **3 cars 11 cities** | 1.003 | 1.003 |

Based on the above information, when the problem size is small, sequential version will be recommended. However, if compare both the multithreaded program, performance of windows threading is slightly better.

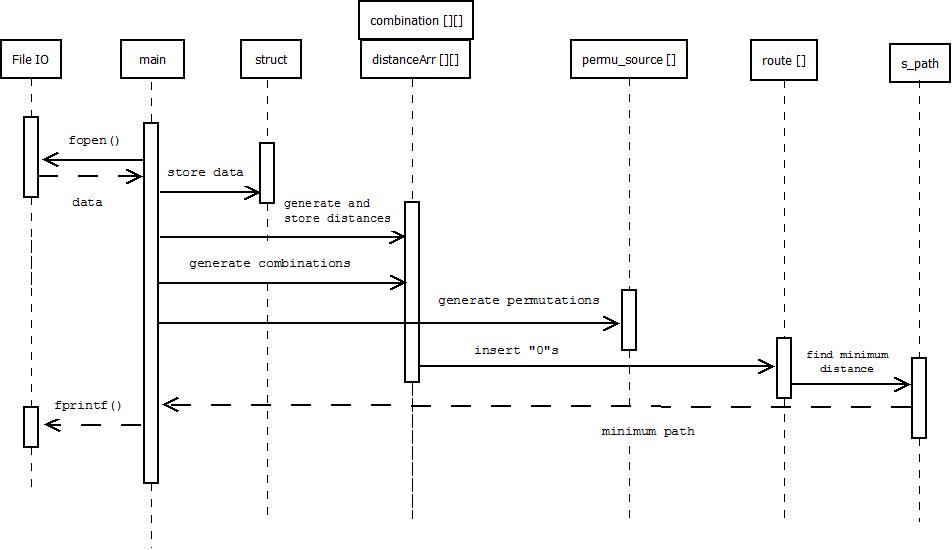
**3.0 Decomposition Techniques**

The decomposition techniques used in our multithreaded program is data decomposition. At first, we design 4 threads to generate the value into distanceArr[][]. For example, thread one generate the distance between initial point and city1, city2, city3 and so on. Thread two generate the distance between city1 and initial point, city2, city3 and so on.

We also used the data decomposition in generatecombi( ) function. Each thread takes one value and generates the combination starting from that value. For instance, thread one get the value 1 and he need to generate {1,1,4},{1,2,3},{1,3,2,}and so on.

Other than that, create file implemented the same techniques. So the file is created in parallel.

**4.0 Sequence**

****

**5.0 Source code**

**5.1 OpenMP**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include<Windows.h>

#include<omp.h>

int s\_path[20],combination[1000][50];

FILE \*pfile[100];

int cities=0,amountCombi=0,car;

float distanceArr[20][20],min=50000;

struct point {

int no;

int x;

int y;

};

typedef struct point COOR;

COOR city[20];

/\*===================writeFiles===================

write the shortest path into separate files which

created in threads

\*/

void writeFiles(int s\_path[],int size,int car,int cities)

{

int i;

for(int fileNum=0,arr\_position=0;fileNum<car;fileNum++)

{

fprintf(pfile[fileNum],"0\n");

if(cities>car)

{

for (i=arr\_position;i<size;i++)

{

if(s\_path[i]!=0)

fprintf(pfile[fileNum],"%d\n",s\_path[i]);

else

{

arr\_position=i+1;

break;

}

}

}

else

{

if(cities>0)

{

fprintf(pfile[fileNum],"%d\n",fileNum+1);

cities--;

}

}

fprintf(pfile[fileNum],"0\n");

fclose(pfile[fileNum]);

}

}

/\*==============generatecombi===========================

Generate all combinations based on number of cities and vehicles.

Pre number of cities and vehicles

Post generate all combinations of cities

\*/

void generatecombi(int row[],int cityNo,int carNo,int rowPosition, int a){

if (carNo==1)

{

row[rowPosition]=cityNo;

for(int i=0;i<=rowPosition;i++)

combination[amountCombi][i]=row[i];

amountCombi++;

}

else if ( rowPosition==0)

{

row[rowPosition]=a;

generatecombi(row,cityNo-a,carNo-1,rowPosition+1,a);

}

else

{

int j=1;

while(j<=cityNo-carNo+1)

{

row[rowPosition]=j;

generatecombi(row,cityNo-j,carNo-1,rowPosition+1,a);

j++;

}

}

}

/\*=====================swap=================

Swap the position of element in the array.

Pre permu\_source

Post new permutation

\*/

void swap (int permu\_src[], int i, int j) {

int temp;

temp = permu\_src[i];

permu\_src[i] = permu\_src[j];

permu\_src[j] = temp;

}

/\*================findPathDist====================

Compute the max distance traveled among vehicles

Pre route[]

Post returns max distance

\*/

float findPathDist(int route[],int num){

int i;

float totalDistance=0,max=0;

totalDistance=distanceArr[0][route[0]];

for (i=0;i<num-1;i++)

{

if (route[i+1]==0)

{

totalDistance=totalDistance+distanceArr[route[i]][0];

if(totalDistance>max)

max=totalDistance;

totalDistance=distanceArr[0][route[i+2]];

i++;

}

else

totalDistance=totalDistance+distanceArr[route[i]][route[i+1]];

}

totalDistance=totalDistance+distanceArr[route[i]][0];

if(totalDistance>max)

max=totalDistance;

return max;

}

/\*==========================perm=======================

Generate all permutations and splits them by inserting "0" in between

Pre permu\_source

Post generates all possible route

\*/

void permute(int permu\_src[ ],int i,int j,int num)

{

int a,temp[30];

int w,x,y,z,route[20],position;

float pathDistance;

if (j==num+1)

{

for(w=0;w<amountCombi;w++){

position=combination[w][0];

for (x=0,y=0,z=1;x<num+car-1,y<num;x++,y++)

{

route[x]=permu\_src[y];

if (x==position-1)

{

route[++x]=0;

position=position+combination[w][z]+1;

z++;

}

}

pathDistance=findPathDist(route,num+car-1);

if(pathDistance<min){

min=pathDistance;

for(x=0;x<num+car-1;x++)

s\_path[x]=route[x];

}

}

}

else

{

for (a=0;a<j;a++)

temp[a]=permu\_src[a];

temp[i]=j;

permute(temp,i+1,j+1,num);

a=j-1;

while (a>0 &&j>2)

{

swap (temp, a, a-1);

permute(temp,i+1,j+1,num);

a--;

}

}

}

int main()

{

omp\_set\_num\_threads(4);

int no\_of\_point,x\_value,y\_value,a;

int permu\_src[30];

int combi\_arr[10];

char fileNo[2];

char filename[20];

FILE \*inputfile=NULL;

inputfile=fopen("Map.vrp","r");

if(inputfile==NULL)

{

printf("Unable to read the file.\n");

exit(1);

}

fscanf(inputfile,"%d",&car);

while(fscanf(inputfile,"%d%d%d",&no\_of\_point,&x\_value,&y\_value)!=EOF)

{

city[cities].no=no\_of\_point;

city[cities].x=x\_value;

city[cities].y=y\_value;

cities++;

}

fclose(inputfile);

if (cities-1>car)

{

for ( a=0;a<cities;a++)

{

#pragma omp parallel

{

#pragma omp for schedule(dynamic,1)

for(int b=1;b<cities;b++)

{

if(b>a)

{

distanceArr[a][b]=sqrt(((float)(city[b].x-city[a].x)\*(city[b].x-city[a].x))+((city[b].y-city[a].y)\*(city[b].y-city[a].y)));

distanceArr[b][a]=distanceArr[a][b];

}

}

distanceArr[a][a]=0;

}

}

#pragma omp parallel

{

#pragma omp for schedule(dynamic,1)

for( a=0;a<cities-car;a++)

{

generatecombi(combi\_arr,cities-1,car,0,a+1);

}

}

#pragma omp parallel

{

#pragma omp for schedule(dynamic,1)

for(int fileNum=0;fileNum<car;fileNum++)

{

itoa(fileNum,fileNo,10);

strcpy(filename,"Solution");

strncat(filename,fileNo,10);

strcat(filename,".vrp");

pfile[fileNum]=fopen(filename,"w");

}

}

permute(permu\_src,0,1,cities-1);

}

writeFiles(s\_path,cities+car-2,car,cities-1);

return 0;

}

**5.2 Windows Threading**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <Windows.h>

#define threadsize 4

HANDLE t[threadsize];

DWORD WINAPI Distance\_Combi(LPVOID lpParam);

FILE \*pfile[100];

int s\_path[20],combination[1000][50];

int cities=0,amountCombi=0;

int car;

float distanceArr[20][20],min=50000;

struct point {

int no;

int x;

int y;

};

typedef struct point COOR;

COOR city[20];

/\*===================writeFiles===================

Write shortest path into separate file which created in threads

\*/

void writeFiles(int s\_path[],int size,int car,int cities)

{

int i;

for(int fileNum=0,arr\_position=0;fileNum<car;fileNum++)

{

fprintf(pfile[fileNum],"0\n");

if(cities>car)

{

for (i=arr\_position;i<size;i++)

{

if(s\_path[i]!=0)

fprintf(pfile[fileNum],"%d\n",s\_path[i]);

else

{

arr\_position=i+1;

break;

}

}

}

else

{

if(cities>0)

{

fprintf(pfile[fileNum],"%d\n",fileNum+1);

cities--;

}

}

fprintf(pfile[fileNum],"0\n");

fclose(pfile[fileNum]);

}

}

/\*==============generatecombi===========================

Generate all combinations based on number of cities and vehicles.

Pre number of cities and vehicles

Post generate all combinations of cities

\*/

void generatecombi(int row[],int cityNo,int carNo,int rowPosition, int a){

if (carNo==1)

{

row[rowPosition]=cityNo;

for(int i=0;i<=rowPosition;i++)

combination[amountCombi][i]=row[i];

amountCombi++;

}

else if ( rowPosition==0)

{

row[rowPosition]=a;

generatecombi(row,cityNo-a,carNo-1,rowPosition+1,a);

}

else

{

int j=1;

while(j<=cityNo-carNo+1)

{

row[rowPosition]=j;

generatecombi(row,cityNo-j,carNo-1,rowPosition+1,a);

j++;

}

}

}

/\*=====================swap=================

Swap the position of element in the array.

Pre permu\_source

Post new permutation

\*/

void swap ( int permu\_src[],int i, int j) {

int temp;

temp = permu\_src[i];

permu\_src[i] = permu\_src[j];

permu\_src[j] = temp;

}

/\*================findPathDist====================

Compute the max distance traveled among vehicles

Pre route[]

Post returns max distance

\*/

float findPathDist(int route[],int num){

int i;

float totalDistance=0,max=0;

totalDistance=distanceArr[0][route[0]];

for (i=0;i<num-1;i++)

{

if (route[i+1]==0)

{

totalDistance=totalDistance+distanceArr[route[i]][0];

if(totalDistance>max)

max=totalDistance;

totalDistance=distanceArr[0][route[i+2]];

i++;

}

else

totalDistance=totalDistance+distanceArr[route[i]][route[i+1]];

}

totalDistance=totalDistance+distanceArr[route[i]][0];

if(totalDistance>max)

max=totalDistance;

return max;

}

/\*==========================perm=======================

Generate all permutations and splits them by inserting "0" in between

Pre permu\_source

Post generates all possible route

\*/

void permute(int permu\_src[ ],int i,int j,int num)

{

int a,temp[30];

int w,x,y,z,route[20],position;

float pathDistance;

if (j==num+1)

{

for(w=0;w<amountCombi;w++){

position=combination[w][0];

for (x=0,y=0,z=1;x<num+car-1,y<num;x++,y++)

{

route[x]=permu\_src[y];

if (x==position-1)

{

route[++x]=0;

position=position+combination[w][z]+1;

z++;

}

}

pathDistance=findPathDist(route,num+car-1);

if(pathDistance<min){

min=pathDistance;

for(x=0;x<num+car-1;x++)

s\_path[x]=route[x];

}

}

}

else

{

for (a=0;a<j;a++)

temp[a]=permu\_src[a];

temp[i]=j;

permute(temp,i+1,j+1,num);

a=j-1;

while (a>0 &&j>2)

{

swap (temp, a, a-1);

permute(temp,i+1,j+1,num);

a--;

}

}

}

int main()

{

int no\_of\_point,x\_value,y\_value;

int permu\_src[30];

FILE \*inputfile=NULL;

inputfile=fopen("Map.vrp","r");

if(inputfile==NULL)

{

printf("Unable to read the file.\n");

exit(1);

}

fscanf(inputfile,"%d",&car);

while(fscanf(inputfile,"%d%d%d",&no\_of\_point,&x\_value,&y\_value)!=EOF)

{

city[cities].no=no\_of\_point;

city[cities].x=x\_value;

city[cities].y=y\_value;

cities++;

}

fclose(inputfile);

if (cities-1>car)

{

for ( int thdstart = 0 ; thdstart < threadsize; thdstart++)

{

t[thdstart] = CreateThread(NULL,0,Distance\_Combi,(void\*)thdstart,0,NULL);

}

WaitForMultipleObjects(threadsize,t,true,INFINITE);

permute(permu\_src,0,1,cities-1);

}

writeFiles(s\_path,cities+car-2,car,cities-1);

return 0;

}

DWORD WINAPI Distance\_Combi(LPVOID lpParam)

{

char fileNo[2];

char filename[20];

int combi\_arr[10],a;

for( a=(int)lpParam;a<cities;a+=threadsize)

{

for (int b=1;b<cities;b++)

{

if (b> a)

{

distanceArr[a][b]=sqrt(((float)(city[b].x-city[a].x)\*(city[b].x-city[a].x))+((city[b].y-city[a].y)\*(city[b].y-city[a].y)));

distanceArr[b][a]=distanceArr[a][b];

}

}

distanceArr[a][a]=0;

}

for( a=(int)lpParam;a<cities-car;a+=threadsize)

{

generatecombi(combi\_arr,cities-1,car,0,a+1);

}

for(int a=(int)lpParam;a<car;a+=threadsize)

{

itoa(a,fileNo,10);

strcpy(filename,"Solution");

strncat(filename,fileNo,10);

strcat(filename,".vrp");

pfile[a]=fopen(filename,"w");

}

return 0;

}