Program 1 Report

Yangxiao Wang

Contents

1 Documentation

In this project, we tried to solve Traveling salesman problem (TSP) with genetic algorithm. We are required to implement the core part of the algorithm: evaluation, crossover and mutation. Tsp problem often takes large amount of data and time. And to improve the efficiency, our approach is using OpenMP. My initial attempt is implement a greedy parallelization: parallelize the whole program from *evaluate()* to *populate()* with:

```
#pragma omp parallel for
```

however, after trying different things with the program, I realized that populate and mutate does not need to be parallelized. The time spent did increase not have much difference. And *select()* with parallelization could spend more time than when running multi-thread. Therefore, the only functions I need to parallelize are *crossover()* and *evaluate()* which contain much larger loops. I simply added #pragma omp parallel for before the outer for loop inside *crossover()* and *evaluate()*.

2 Source code

2.1 EvalXOverMutate.cpp

```
//
// Created by yangxiao on 10/8/2018.
//
#include <iostream>  // cout
#include <fstream>   // ifstream
#include <string.h>  // strncpy
#include <stdlib.h>  // rand
#include <math.h>    // sqrt, pow
#include <time.h>
#include <omp.h>
#include <bits/stdc++.h>
#include "Trip.h"

using namespace std;


float getDis(char a, char b, int coordinates[CITIES][2]);

float getDisFromIt(char str[CITIES], float disMax[CITIES][CITIES + 1]);

char *getChildB(char childA[CITIES]);

int getIndex(char a, char source[CITIES]);

int getAscii(char source);

const char cit[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

/**
* Compare tow trip based their fitness, uses as comparator for sorting
* @param t1 trip 1
* @param t2 trip 2
* @return if t1 < t2
```

```
34    */
35    bool compareTrip(Trip t1, Trip t2) {
36    return (t1.fitness < t2.fitness);
37    }
38
39    /**
40    * evaluates the distance of each trip and sorts out all the trips
41    * in the shortest-first order
42    * @param trip trips to evaluates
43    * @param coordinates coordinates of all cities
44    */
45    void evaluate(Trip trip[CHROMOSOMES], int coordinates[CITIES][2]) {
46
47    //Parallelization
48    #pragma omp parallel for
49    for (int i = 0; i < CHROMOSOMES; ++i) {
50    char temp[CITIES];
51    strncpy(temp, trip[i].itinerary, CITIES);
52    int indexIn = (temp[0] >= 'A') ? temp[0] - 'A' : temp[0] - '0' + 26;
53    double dis = hypot(coordinates[indexIn][0], coordinates[indexIn][1]);
54    for (int j = 1; j < CITIES; ++j) {
55    int index = (temp[j] >= 'A') ? temp[j] - 'A' : temp[j] - '0' + 26;
56    dis += hypot((coordinates[indexIn][0] - coordinates[index][0]),
57    (coordinates[indexIn][1] - coordinates[index][1]));
58    indexIn = index;
59    }
60    trip[i].fitness = (float) dis;
61    }
62    sort(trip, trip + CHROMOSOMES, compareTrip);
63    }
64
65    /**
66    * generates 25,000 off-springs from the parents, calculate distance
67    * based on their coordinates
68    * @param parents
69    * @param offsprings
70    * @param coordinates
71    */
72    void crossover(Trip parents[TOP_X], Trip offsprings[TOP_X],
73    int coordinates[CITIES][2]) {
74    #pragma omp parallel for
75    for (int i = 0; i < TOP_X; i += 2) {
76    int selected[100] = {0};
77    char child1[CITIES + 1];
78    char child2[CITIES + 1];
79    char *parent1 = parents[i].itinerary;
80    char *parent2 = parents[i + 1].itinerary;
81    child1[0] = parent1[0];
82    selected[child1[0]] = 1;
83
84    for (int j = 0; j < (CITIES - 1); j++) {
85    int indexInA = getIndex(child1[j], parent1) + 1;
86    int indexInB = getIndex(child1[j], parent2) + 1;
87
88    if (indexInA == -1 || indexInB == -1) {
89    printf("WHat?\n");
90    }
91
92    if (indexInA >= CITIES) {
93    indexInA = 0;
94    }
95    if (indexInB >= CITIES) {
```

```
96      indexInB = 0;
97      }
98
99      float disA = INT_MAX;
100     float disB = INT_MAX;
101
102     if (selected[parent1[indexInA]] == 0) {
103     disA = getDis(child1[j], parent1[indexInA], coordinates);
104     }
105     if (selected[parent2[indexInB]] == 0) {
106     disB = getDis(child1[j], parent2[indexInB], coordinates);
107     }
108
109     if (disA == INT_MAX && disB == INT_MAX) {
110     int r = rand() % CITIES;
111     int found = 1;
112     while (found) {
113     if (selected[parent1[r]] != 1) {
114     found = 0;
115     child1[j + 1] = parent1[r];
116     }
117     r = ((r + 1) >= CITIES) ? 0 : (r + 1);
118     }
119     } else {
120     child1[j + 1] = (disA <= disB) ?
121     parent1[indexInA] : parent2[indexInB];
122     }
123
124     selected[child1[j + 1]] = 1;
125     }
126
127
128     strcpy(child2, getChildB(child1));
129     strncpy(offsprings[i].itinerary, child1, CITIES);
130     strncpy(offsprings[i + 1].itinerary, child2, CITIES);
131     }
132     }
133
134
135     /**
136     * Improved version, when mutate each city,
137     * perform mutation if the new one has shorter distance,
138     * otherwise, do nothing
139     *
140     * randomly chooses two distinct cities (or genes)
141     * in each trip (or chromosome) with a
142     * given probability, and swaps them
143     * @param offsprings
144     * @param disMax
145     */
146     void mutateB(Trip offsprings[TOP_X], float disMax[CITIES][CITIES + 1]) {
147     for (int i = 0; i < TOP_X; ++i) {
148     int rate = rand() % 100;
149     if (rate < MUTATE_RATE) {
150     int r1 = rand() % CITIES;
151     int r2 = rand() % CITIES;
152     char temp[CITIES + 1];
153     strncpy(temp, offsprings[i].itinerary, CITIES);
154     temp[r1] = offsprings[i].itinerary[r2];
155     temp[r2] = offsprings[i].itinerary[r1];
156     float d1 = getDisFromIt(offsprings[i].itinerary, disMax);
157     float d2 = getDisFromIt(temp, disMax);
```

```
158     //check if the new one has shorter distacne
159     if (d1 > d2) {
160     strncpy(offsprings[i].itinerary, temp, CITIES);
161     }
162     }
163     }
164     }
165
166     /**
167     * Helper function that calculate distance between two cities
168     */
169     float getDis(char a, char b, int coordinates[CITIES][2]) {
170     int indexA = (a >= 'A') ? a - 'A' : a - '0' + 26;
171     int indexB = (b >= 'A') ? b - 'A' : b - '0' + 26;
172     float dis = (float) hypot((coordinates[indexA][0] - coordinates[indexB][0]),
173     (coordinates[indexA][1] - coordinates[indexB][1]));
174     return dis;
175     }
176
177     /**
178     * Get distacne of the provided route from distance matrix
179     */
180     float getDisFromIt(char str[CITIES], float disMax[CITIES][CITIES + 1]) {
181     int indexIn = (str[0] >= 'A') ? str[0] - 'A' : str[0] - '0' + 26;
182     double dis = disMax[indexIn][CITIES];
183     for (int j = 1; j < CITIES; ++j) {
184     int index = (str[j] >= 'A') ? str[j] - 'A' : str[j] - '0' + 26;
185     dis += disMax[indexIn][index];
186
187     indexIn = index;
188     }
189     return dis;
190     }
191
192     /**
193     * Get the position of the target character
194     * @param a target character
195     * @param source source string
196     * @return index of target char, -1 if not found
197     */
198     int getIndex(char a, char source[CITIES]) {
199     for (int i = 0; i < CITIES; ++i) {
200     if (a == source[i]) return i;
201     }
202     return -1;
203     }
204
205     /**
206     * get ascii representation of target char, and map it to its index
207     * based on ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
208     */
209     int getAscii(char source) {
210     return (source >= 'A') ? source - 'A' : source - '0' + 26;
211     }
212
213     /**
214     * generate childA's complement
215     */
216     char *getChildB(char childA[CITIES]) {
217     char *childB = (char *) malloc(sizeof(char) * CITIES);
218     for (int i = 0; i < CITIES; ++i) {
219     childB[i] = cit[(CITIES - getAscii(childA[i]) - 1)];
```

```
220     }
221     return childB;
222     }
223
224
225     /**
226     * Improved version with distance matrix, avoid calculating distance
227     * generates 25,000 off-springs from the parents,
228     * calculate distance based on their coordinates
229     * @param parents
230     * @param offsprings
231     * @param disMax
232     */
233     void crossoverB(Trip parents[TOP_X], Trip offsprings[TOP_X], float disMax[↩
            CITIES][CITIES + 1]) {
234     #pragma omp parallel for default(none) shared(parents, disMax, offsprings)
235     for (int i = 0; i < TOP_X; i += 2) {
236     int selected[100] = {0};
237     char child1[CITIES + 1];
238     char child2[CITIES + 1];
239     char *parent1 = parents[i].itinerary;
240     char *parent2 = parents[i + 1].itinerary;
241     child1[0] = parent1[0];
242     selected[child1[0]] = 1;
243
244     for (int j = 0; j < (CITIES - 1); j++) {
245     int indexInA = getIndex(child1[j], parent1) + 1;
246     int indexInB = getIndex(child1[j], parent2) + 1;
247
248     if (indexInA == -1 || indexInB == -1) {
249     printf("WHat?\n");
250     }
251
252     if (indexInA >= CITIES) {
253     indexInA = 0;
254     }
255     if (indexInB >= CITIES) {
256     indexInB = 0;
257     }
258
259     float disA = INT_MAX;
260     float disB = INT_MAX;
261
262     if (selected[parent1[indexInA]] == 0) {
263     int indA = getAscii(child1[j]);
264     int indB = getAscii(parent1[indexInA]);
265     disA = disMax[indA][indB];
266     }
267     if (selected[parent2[indexInB]] == 0) {
268     int indA = getAscii(child1[j]);
269     int indB = getAscii(parent2[indexInB]);
270     disB = disMax[indA][indB];
271     }
272
273     if (disA == INT_MAX && disB == INT_MAX) {
274     int r = rand() % CITIES;
275     int found = 0;
276     while (!found) {
277     if (selected[parent1[r]] != 1) {
278     found = 1;
279     child1[j + 1] = parent1[r];
280     }
```

```
281     r = ((r + 1) >= CITIES) ? 0 : (r + 1);
282     }
283     } else {
284     child1[j + 1] = (disA <= disB) ? parent1[indexInA] : parent2[indexInB];
285     }
286
287     selected[child1[j + 1]] = 1;
288     }
289
290     strcpy(child2, getChildB(child1));
291     strncpy(offsprings[i].itinerary, child1, CITIES);
292     strncpy(offsprings[i + 1].itinerary, child2, CITIES);
293     }
294     }
295
296     /**
297     * randomly chooses two distinct cities (or genes)
298     * in each trip (or chromosome) with a
299     * given probability, and swaps them
300     * @param offsprings
301     */
302     void mutate(Trip offsprings[TOP_X]) {
303     for (int i = 0; i < TOP_X; ++i) {
304     int rate = rand() % 100;
305     if (rate < MUTATE_RATE) {
306     int r1 = rand() % CITIES;
307     int r2 = rand() % CITIES;
308     char x1 = offsprings[i].itinerary[r1];
309     char x2 = offsprings[i].itinerary[r2];
310     offsprings[i].itinerary[r1] = x2;
311     offsprings[i].itinerary[r2] = x1;
312     }
313     }
314     }
315
316
317     /**
318     * Improved verison, uses distance matrix to avoid calculating distance
319     * Evaluates the distance of each trip and sorts out all the
320     * trips in the shortest-first order
321     * @param trip trips to evaluates
322     * @param disMax maxtrix that contain all the distance
323     * @param first if it's first generation
324     */
325     void evaluateB(Trip trip[CHROMOSOMES],
326     float disMax[CITIES][CITIES + 1], bool first) {
327     int start = 0;
328     if (!first) {
329     start = TOP_X;
330     }
331
332     #pragma omp parallel for default(none) firstprivate(start) shared(trip, ↩
        disMax)
333     for (int i = start; i < CHROMOSOMES; ++i) {
334     char temp[CITIES];
335     strncpy(temp, trip[i].itinerary, CITIES);
336     int indexIn = (temp[0] >= 'A') ? temp[0] - 'A' : temp[0] - '0' + 26;
337     double dis = disMax[indexIn][CITIES];
338     for (int j = 1; j < CITIES; ++j) {
339     int index = (temp[j] >= 'A') ? temp[j] - 'A' : temp[j] - '0' + 26;
340     dis += disMax[indexIn][index];
341
```

```
342     indexIn = index;
343     }
344     trip[i].fitness = (float) dis;
345     //         printf("Thread %d executes loop iteration %d\n", omp_get_thread_num↩
            (), i);
346     }
347     sort(trip, trip + CHROMOSOMES, compareTrip);
348     }
```

2.2 Tsp.cpp

```
1  #include <iostream>  // cout
2  #include <fstream>   // ifstream
3  #include <string.h>  // strncpy
4  #include <stdlib.h>  // rand
5  #include <math.h>    // sqrt, pow
6  #include <omp.h>     // OpenMP
7  #include "Timer.h"
8  #include "Trip.h"
9
10 using namespace std;
11
12 void initialize(Trip trip[CHROMOSOMES], int coordinates[CITIES][2]);
13
14 void select(Trip trip[CHROMOSOMES], Trip parents[TOP_X]);
15
16 void populate(Trip trip[CHROMOSOMES], Trip offsprings[TOP_X]);
17
18 void formDisMax(int coordinates[CITIES][2], float disMax[CITIES][CITIES + 1])↩
       ;
19
20 extern void evaluate(Trip trip[CHROMOSOMES], int coordinates[CITIES][2]);
21
22 extern void evaluateB(Trip trip[CHROMOSOMES], float disMax[CITIES][CITIES + ↩
       1], bool first);
23
24 extern void crossover(Trip parents[TOP_X], Trip offsprings[TOP_X], int ↩
       coordinates[CITIES][2]);
25
26 extern void crossoverB(Trip parents[TOP_X], Trip offsprings[TOP_X], float ↩
       disMax[CITIES][CITIES +
27 1]);
28 extern void mutate(Trip offsprings[TOP_X]);
29
30 extern void mutateB(Trip offsprings[TOP_X], float disMax[CITIES][CITIES + 1])↩
       ;
31
32 /*
33 * MAIN: usage: Tsp #threads
34 */
35 int main(int argc, char *argv[]) {
36 //Start random seed based on current time value
37 srand(time(NULL));
38
39 //static is required on windows machine
40 static Trip trip[CHROMOSOMES];       // all 50000 different trips (or ↩
       chromosomes)
41 Trip shortest;                 // the shortest path so far
42 int coordinates[CITIES][2];    // (x, y) coordinates of all 36 cities:
43 int nThreads = 1;
44 float disMax[CITIES][CITIES + 1];
45
46 // verify the arguments
47 if (argc == 2)
48 nThreads = atoi(argv[1]);
49 else {
50 cout << "usage: Tsp #threads" << endl;
51 if (argc != 1)
```

```
52      return -1; // wrong arguments
53      }
54      cout << "# threads = " << nThreads << endl;
55
56      // shortest path not yet initialized
57      shortest.itinerary[CITIES] = 0;  // null path
58      shortest.fitness = -1.0;         // invalid distance
59
60      // initialize 5000 trips and 36 cities' coordinates
61      initialize(trip, coordinates);
62
63      //from a distance matrix to improve efficiency
64      formDisMax(coordinates, disMax);
65
66      // start a timer
67      Timer timer;
68      timer.start();
69
70      // change # of threads
71      omp_set_num_threads(nThreads);
72
73      // find the shortest path in each generation
74      for (int generation = 0; generation < MAX_GENERATION; generation++) {
75      // evaluate the distance of all 50000 trips
76      evaluate(trip, coordinates, generation == 0);
77
78      // just print out the progress
79      if (generation % 20 == 0)
80      cout << "generation: " << generation << endl;
81
82      // whenever a shorter path was found, update the shortest path
83      if (shortest.fitness < 0 || shortest.fitness > trip[0].fitness) {
84      strncpy(shortest.itinerary, trip[0].itinerary, CITIES);
85      shortest.fitness = trip[0].fitness;
86
87      cout << "generation: " << generation
88      << " shortest distance = " << shortest.fitness
89      << "\t itinerary = " << shortest.itinerary << endl;
90      }
91
92      // define TOP_X parents and offsprings.
93      // static is required on windows machine
94      static Trip parents[TOP_X], offsprings[TOP_X];
95
96      // choose TOP_X parents from trip
97      select(trip, parents);
98
99      // generates TOP_X offsprings from TOP_X parenets
100     crossover(parents, offsprings, coordinates);
101
102     // mutate offsprings
103     mutateB(offsprings, disMax);
104     // populate the next generation.
105     populate(trip, offsprings);
106     }
107
108     // stop a timer
109     cout << "elapsed time = " << timer.lap() << endl;
110     return 0;
111     }
112     /**
113     * Initializes trip[CHROMOSOMES] with chromosome.txt and coordiantes[CITIES↩
```

```
    ][2] with cities.txt
114 *
115 * @param trip[CHROMOSOMES]:      50000 different trips
116 * @param coordinates[CITIES][2]: (x, y) coordinates of 36 different cities: ↩
        ABCDEFGHIJKLMNOPQRSTUVWXYZ
117 */
118 void initialize(Trip trip[CHROMOSOMES], int coordinates[CITIES][2]) {
119 // open two files to read chromosomes (i.e., trips) and cities
120 ifstream chromosome_file("chromosome.txt");
121 ifstream cities_file("cities.txt");
122
123 for (int i = 0; i < CHROMOSOMES; i++) {
124 chromosome_file >> trip[i].itinerary;
125 trip[i].fitness = 0.0;
126 }
127
128 for (int i = 0; i < CITIES; i++) {
129 char city;
130 cities_file >> city;
131 int index = (city >= 'A') ? city - 'A' : city - '0' + 26;
132 cities_file >> coordinates[index][0] >> coordinates[index][1];
133 }
134
135 chromosome_file.close();
136 cities_file.close();
137
138 if (DEBUG) {
139 for (int i = 0; i < CHROMOSOMES; i++)
140 cout << trip[i].itinerary << endl;
141 for (int i = 0; i < CITIES; i++)
142 cout << coordinates[i][0] << "\t" << coordinates[i][1] << endl;
143 }
144 }
145
146 /**
147 * Select the first TOP_X parents from trip[CHROMOSOMES]
148 *
149 * @param trip[CHROMOSOMES]: all trips
150 * @param parents[TOP_X]:    the firt TOP_X parents
151 */
152 void select(Trip trip[CHROMOSOMES], Trip parents[TOP_X]) {
153 // just copy TOP_X trips to parents
154 for (int i = 0; i < TOP_X; i++)
155 strncpy(parents[i].itinerary, trip[i].itinerary, CITIES + 1);
156 }
157
158 /**
159 * Replace the bottom TOP_X trips with the TOP_X offsprings
160 */
161 void populate(Trip trip[CHROMOSOMES], Trip offsprings[TOP_X]) {
162 // just copy TOP_X offsprings to the bottom TOP_X trips.
163 for (int i = 0; i < TOP_X; i++) {
164 strncpy(trip[CHROMOSOMES - TOP_X + i].itinerary, offsprings[i].itinerary, ↩
        CITIES);
165 }
166 // for debugging
167 if (DEBUG) {
168 for (int chrom = 0; chrom < CHROMOSOMES; chrom++)
169 cout << "chrom[" << chrom << "] = " << trip[chrom].itinerary
170 << ", trip distance = " << trip[chrom].fitness << endl;
171 }
172 }
```

```
173
174     /**
175     * Generate CITIES distances for each city, total CITIES * CITIES distances
176     * @param coordinates (x, y) coordinates of CITIES different cities
177     * @param disMax Distance matrix
178     */
179     void formDisMax(int coordinates[CITIES][2], float disMax[CITIES][CITIES + 1])↩
            {
180     for (int i = 0; i < CITIES; ++i) {
181     for (int j = 0; j < CITIES; ++j) {
182     disMax[i][j] = hypot((coordinates[i][0] - coordinates[j][0]),
183     (coordinates[i][1] - coordinates[j][1]));
184     }
185     disMax[i][CITIES] = hypot(coordinates[i][0] , coordinates[i][1]);
186     }
187     }
```

3 Execution output

3.1 Output analysis

1. The shortest trip in my program is equal to 447.638
2. The performance improvement with four threads in my program is equal to 50548672 / 23005048 = 2.2 times

3.2 Output

```
# threads = 1
generation: 0
generation: 0 shortest distance = 1265.72 itinerary = V1SPMBQAN26G4J37DX8OTF95ZUH0EYRLCWKI
generation: 1 shortest distance = 1031.81 itinerary = I61YHO9F48KGATL7UJR3BQ20ENXVSCZWP5MD
generation: 2 shortest distance = 979.47 itinerary = KFL94NT86OJAGX7RD3IU2W5PBS10EZHYVCMQ
generation: 3 shortest distance = 882.109 itinerary = V1O6XG84K9AFLPDBM3Q7NTIUZJE02HYCSW5R
generation: 4 shortest distance = 724.804 itinerary = V1YHEJUZ02CSW5MBQDR794X6IONTK8FALG3P
generation: 5 shortest distance = 627.852 itinerary = V1YHEZ02CSW5MPBQDR3JU7ALGF4NT89KX6IO
generation: 7 shortest distance = 626.585 itinerary = V1IO6JE02WCS5HZYFAL7R3DBQMPUKX4NT8G9
generation: 8 shortest distance = 603.274 itinerary = V1YHEUZJWSC205MPQ3RDB7LAF9KGX48TN6OI
generation: 10 shortest distance = 556.14 itinerary = V1YZHUE0MSWC5DBQR37LA9KFGXNT48IO6J2P
generation: 11 shortest distance = 542.062 itinerary = V1YZHUE0MPBQR37LA9KFGXNT48IO6J2SWC5D
generation: 13 shortest distance = 477.948 itinerary = V1YZHUE02WSC5MPBQDR37LA9KFGXNT48IO6J
generation: 16 shortest distance = 474.401 itinerary = V1YZHUE02WSC5MPBQDR37LAFK9GXNT48IO6J
generation: 18 shortest distance = 467.935 itinerary = V1YZHUE0J6OI84NTGXKF9AL7R3DBQPMSWC52
generation: 19 shortest distance = 467.25 itinerary = V1YZH5CWSMPQBDR37LAF9KGXNT48IO6J0E2U
generation: 20
generation: 20 shortest distance = 464.471 itinerary = V1YZHUE20J6OI84NTGX9FKAL7R3DBQPMWSC5
generation: 24 shortest distance = 461.723 itinerary = V1YZHUE025CWSMPQBDR37LA9KFGXNT48IO6J
generation: 30 shortest distance = 459.941 itinerary = V1YZHUE20J6OI84NTGX9FKAL7R3DBQPMSWC5
generation: 33 shortest distance = 459.436 itinerary = V1YZH5CWSMPQBDR37LAF9KGXNT48IO6JUE20
generation: 35 shortest distance = 458.176 itinerary = V1YZHUE025CWSMPQBDR37LAFK9GXNT48IO6J
generation: 37 shortest distance = 457.536 itinerary = V1YZH5CWSMPQBDR37LAF9KGXNT48IO6J02EU
generation: 40
generation: 43 shortest distance = 456.303 itinerary = V1YZHUE20J6OI84NTGX9KFAL7R3DBQPMSWC5
generation: 44 shortest distance = 452.975 itinerary = V1YZHUE025CWSMPQBDR37LAF9KGXNT48IO6J
generation: 54 shortest distance = 449.658 itinerary = V1YZHUE025CWSMPQBD3R7LAF9KGXNT48IO6J
generation: 60
generation: 61 shortest distance = 449.552 itinerary = V1YZHUE20J6OI84NTXGK9FAL7R3DBQPMSWC5
```

generation: 71 shortest distance = 447.638 itinerary = V1YZHUE20J6OI84TNXGK9FAL7R3DBQPMSWC5
generation: 80
generation: 100
generation: 120
generation: 140
elapsed time = 50548672

threads = 4
generation: 0
generation: 0 shortest distance = 1265.72 itinerary = V1SPMBQAN26G4J37DX8OTF95ZUH0EYRLCWKI
generation: 1 shortest distance = 1043.94 itinerary = I61YHO9F48KGATL7UJR3BQ2P5SCZW0ENXVMD
generation: 2 shortest distance = 871.976 itinerary = VU25BSCWMPQDR37JZHT8IONX9AF46K0EY1GL
generation: 3 shortest distance = 825.757 itinerary = V8TNKA9FXGIO1HZYUE5C6RL73DMBQPSW2J04
generation: 4 shortest distance = 805.385 itinerary = 1VKF9X7AGTJ6OI4N83RL0SWC25BPMQDEUHYZ
generation: 5 shortest distance = 716.93 itinerary = 8TN4X9KGAF6OIJ02EHYZU1VSCWM5BDL7R3QP
generation: 7 shortest distance = 652.966 itinerary = 1V6OI9KFAL7J0WC5SMPBQ3RDHZYUE284GXNT
generation: 8 shortest distance = 635.406 itinerary = V1YIO6NTXK948GR3DBPMS5CW02ZHUEJ7LFAQ
generation: 9 shortest distance = 583.042 itinerary = V1YZUH5CSWMPBQD3R7AF9KTNX4GOI68L2E0J
generation: 10 shortest distance = 564.252 itinerary = V1YHUZ5CWSMPQBDR37LJOI4NT89FKXGA6E02
generation: 11 shortest distance = 558.654 itinerary = UHZY1VO6I8KGXNT49FAL7R3QBPMSWC520EJD
generation: 12 shortest distance = 545.421 itinerary = V1YZHUE20J6OI9FK84NTGXAL73RDBQSWCMP5
generation: 13 shortest distance = 536.668 itinerary = V1YZHUE20J6OI4N8T9XGKFAL73R5CWSMPQBD
generation: 14 shortest distance = 520.507 itinerary = V1YZHU5CWSE0J6OI84NTXG9KFAL7R3DBPM2Q
generation: 15 shortest distance = 496.876 itinerary = V1YZHUE0J6OI84NTXGK9FLA7R325CWSMPBDQ
generation: 16 shortest distance = 496.376 itinerary = V1YZHUE02WSC5MPQBD3R7LAGXKF984NTJ6OI
generation: 17 shortest distance = 484.545 itinerary = V1YZHUE02J6OI84NT9FKXGAL7R3QBDPMSWC5
generation: 18 shortest distance = 469.679 itinerary = V1YZHUE025CWSMPQBDR37LAFKGXNT849IO6J
generation: 20
generation: 22 shortest distance = 467.855 itinerary = V1YZHUE02J6OI84NT9FKXGAL7R3DBQPMSWC5
generation: 24 shortest distance = 460.906 itinerary = V1YZHUE20J6OI84NT9FKXGAL7R3DBQPMSWC5
generation: 29 shortest distance = 460.657 itinerary = V1YZHUE02J6OI84NTXGKF9AL7R3DBQPMSWC5
generation: 36 shortest distance = 453.709 itinerary = V1YZHUE20J6OI84NTXGKF9AL7R3DBQPMSWC5
generation: 40
generation: 43 shortest distance = 449.552 itinerary = V1YZHUE20J6OI84NTXGK9FAL7R3DBQPMSWC5
generation: 60
generation: 60 shortest distance = 447.638 itinerary = V1YZHUE20J6OI84TNXGK9FAL7R3DBQPMSWC5
generation: 80
generation: 100
generation: 120
generation: 140
elapsed time = 23005048

4 Discussions

In addition, I tried to improve the whole efficiency by replacing calculating distance with a cached matrix that contains all the 36 * 36 distance. This implementation decrease significant amount of time spent, although this made the program program with single thread runs faster than that with multi-thread. This also explained why select() with multi-thread could be slower than that with single thread. Starting a multi-thread could require some time to analysis the for-loop and distribute the tasks. Therefore, it is best to use multi-thread when the loop is large and require large computational power.

To improve the performance of current program, just use the implementation I mentioned above to shorten the time required to calculate the distance. However, this only works with the current data set, it could be less efficient with larger data sets.

5 Lab Sessions 1

Lab 1 we parallelize two programs that compute Pi using Monte Carlo methods and Integration. As the result shown, multi-thread decrease significant amount of time that required to finish the program. However, if the number of iterations is too small, the performance would decrease.

5.1 Execution output

pi_integral_omp

```
Enter the number of iterations used to estimate pi: 1000000000
Enter the number of threads: 1
elapsed time for pi = 10850791
# of trials = 1000000000, estimate of pi is 3.1415926535899708, Error is 0.0000000000001776
```

```
Enter the number of iterations used to estimate pi: 1000000000
Enter the number of threads: 4
elapsed time for pi = 2831500
# of trials = 1000000000, estimate of pi is 3.1415926535898211, Error is 0.0000000000000280
```

pi_monte_omp

```
Enter the number of iterations used to estimate pi: 1000000000
Enter the number of threads: 1
elapsed time for pi = 44118944
# of trials = 1000000000, estimate of pi is 3.1416131730000001, Error is 0.0000205194102070
```

```
Enter the number of iterations used to estimate pi: 1000000000
Enter the number of threads: 4
elapsed time for pi = 14253004
# of trials = 1000000000, estimate of pi is 3.1415967750000000, Error is 0.0000041214102069
```