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:

```
1  #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 Wave2D.cpp

```
1  #include <iostream>
2  #include "Timer.h"
3  #include <stdlib.h>   // atoi
4  #include <math.h>
5  #include <stdio.h>
6
7  int default_size = 100;  // the default system size
8  int defaultCellWidth = 8;
9  double c = 1.0;      // wave speed
10 double dt = 0.1;      // time quantum
11 double dd = 2.0;      // change in system
12
13 using namespace std;
14
15 int main(int argc, char *argv[]) {
16    // verify arguments
17    if (argc != 4) {
18       cerr << "usage: Wave2D size max_time interval" << endl;
19       return -1;
20    }
21    int size = atoi(argv[1]);
22    int max_time = atoi(argv[2]);
23    int interval = atoi(argv[3]);
24
25    if (size < 100 || max_time < 3 || interval < 0) {
26       cerr << "usage: Wave2D size max_time interval" << endl;
27       cerr << "       where size >= 100 && time >= 3 && interval >= 0" << endl;
28       return -1;
29    }
30
31    // create a simulation space
32    double z[3][size][size];
33    for (int p = 0; p < 3; p++)
34       for (int i = 0; i < size; i++)
35          for (int j = 0; j < size; j++)
```

```
36                z[p][i][j] = 0.0; // no wave
37
38        // start a timer
39        Timer time;
40        time.start();
41
42        // time = 0;
43        // initialize the simulation space: calculate z[0][][]
44        int weight = size / default_size;
45        for (int i = 0; i < size; i++) {
46           for (int j = 0; j < size; j++) {
47              if (i > 40 * weight && i < 60 * weight &&
48                 j > 40 * weight && j < 60 * weight) {
49                 z[0][i][j] = 20.0;
50              } else {
51                 z[0][i][j] = 0.0;
52              }
53           }
54        }
55
56        // time = 1
57        for (int i = 1; i < size - 1; i++) {
58           for (int j = 1; j < size - 1; j++) {
59              z[1][i][j] = z[0][i][j] + (pow(c, 2) / 2) * pow(dt / dd, 2) * (z[0][i +↩
                    1][j] + z[0][i - 1][j] + z[0][i][j + 1] + z[0][i][j - 1] - (4.0 * ↩
                   z[0][i][j]));
60           }
61        }
62
63        // simulate wave diffusion from time = 2
64        for (int t = 2; t < max_time; t++) {
65           int time = t % 3;
66           for (int i = 1; i < size - 1; i++) {
67              for (int j = 1; j < size - 1; j++) {
68                 int time_1, time_2;
69                 //rotate z
70                 if (time == 0) {
71                    time_1 = 2;
72                    time_2 = 1;
73                 } else if (time == 1) {
74                    time_1 = 0;
75                    time_2 = 2;
76                 } else {
77                    time_1 = 1;
78                    time_2 = 0;
79                 }
80                 //calculation
81                 z[time][i][j] =
82                 2.0 * z[time_1][i][j] - z[time_2][i][j] + (pow(c, 2) * pow(dt / dd, ↩
                       2)  * (z[time_1][i + 1][j] + z[time_1][i - 1][j] + z[time_1][i][↩
                       j + 1] + z[time_1][i][j - 1] - (4.0 * z[time_1][i][j])));
83              }
84           }
85           //print out
86           if (interval != 0 && t % interval == 0) {
87              cout << t << endl;
88              for (int j = 0; j < size; j++) {
89                 for (int i = 0; i < size; i++) {
90                    cout << z[time][i][j] << " ";
91                 }
92                 cout << endl;
```

```
93              }
94              cout << endl;
95          }
96      } // end of simulation
97
98      // finish the timer
99      cerr << "Elapsed time = " << time.lap() << endl;
100     return 0;
101 }
```

2.2 Wave2D mpi.cpp

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 Execution output

Check if output is correct

```
[wyxiao_css534@cssmpi1 prog2]$ ./Wave2D 576 500 50 > reS.txt
[wyxiao_css534@cssmpi1 prog2]$ mpirun -n 4 ./Wave2D_mpi 576 500 50 1 > reF.txt
[wyxiao_css534@cssmpi1 prog2]$ diff reF.txt reS.txt
[wyxiao_css534@cssmpi1 prog2]$
```

Check output the performance improvement with four machines: 5721050 / 1575955 = 3.63 times

```
[wyxiao_css534@cssmpi1 prog2]$ ./Wave2D 576 500 0
Elapsed time = 5721050
[wyxiao_css534@cssmpi1 prog2]$ mpirun -n 4 ./Wave2D_mpi 576 500 0 1
Elapsed time = 1575955
```

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 2

Lab 2 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 Source Code

```
#include "mpi.h"
#include <stdlib.h> // atoi
#include <iostream> // cerr
#include "Timer.h"

using namespace std;

void init(double *matrix, int size, char op) {
   for (int i = 0; i < size; i++)
      for (int j = 0; j < size; j++)
         matrix[i * size + j] = (op == '+') ? i + j : ((op == '-') ? i - j : 0);
}

void print(double *matrix, int size, char id) {
   for (int i = 0; i < size; i++)
      for (int j = 0; j < size; j++)
         cout << id << "[" << i << "][" << j << "] = " << matrix[i * size + j] ↩
            << endl;
}

void multiplication(double *a, double *b, double *c, int stripe, int size) {
   for (int k = 0; k < size; k++)
      for (int i = 0; i < stripe; i++)
         for (int j = 0; j < size; j++)
            // c[i][k] += a[i][j] * b[j][k];
            c[i * size + k] += a[i * size + j] * b[j * size + k];
}

int main(int argc, char *argv[]) {
   int my_rank = 0;            // used by MPI
   int mpi_size = 1;           // used by MPI
   int size = 400;             // array size
   bool print_option = false;  // print out c[] if it is true
   Timer timer;

   // variables verification
   if (argc == 3) {
   if (argv[2][0] == 'y')
      print_option = true;
   }

   if (argc == 2 || argc == 3) {
      size = atoi(argv[1]);
   } else {
      cerr << "usage:   matrix size [y|n]" << endl;
      cerr << "example: matrix 400   y" << endl;
      return -1;
   }

   MPI_Init(&argc, &argv); // start MPI
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);

   // matrix initialization
   double *a = new double[size * size];
   double *b = new double[size * size];
   double *c = new double[size * size];

```

```
58      if (my_rank == 0) { // master initializes all matrices
59         init(a, size, '+');
60         init(b, size, '-');
61         init(c, size, '0');
62
63         // print initial values
64         if (false) {
65            print(a, size, 'a');
66            print(b, size, 'b');
67         }
68
69         // start a timer
70         timer.start();
71      } else {                // slavs zero-initializes all matrices
72         init(a, size, '0');
73         init(b, size, '0');
74         init(c, size, '0');
75      }
76
77      // broadcast the matrix size to all.
78      MPI_Bcast(&size, 1, MPI_INT, 0, MPI_COMM_WORLD);
79
80      int stripe = size / mpi_size;     // partitioned stripe
81
82      // master sends each partition of a[] to a different slave
83      // master also sends b[] to all slaves
84      if (my_rank == 0) {
85         for (int rank = 1; rank < mpi_size; ++rank) {
86            MPI_Send(a + rank * stripe * size, size * stripe, MPI_DOUBLE, rank, 0, ↩
                  MPI_COMM_WORLD);
87            MPI_Send(b, size * size, MPI_DOUBLE, rank, 0, MPI_COMM_WORLD);
88         }
89      } else {
90         MPI_Status status;
91         MPI_Recv(a, size * stripe, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
92         MPI_Recv(b, size * size, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
93      }
94
95      multiplication(a, b, c, stripe, size); // all ranks should compute ↩
          multiplication
96
97      // master receives each partition of c[] from a different slave
98      if (my_rank == 0) {
99         for (int rank = 1; rank < mpi_size; ++rank) {
100           MPI_Status status;
101           MPI_Recv(c + rank * stripe * size, size * stripe, MPI_DOUBLE, rank, 0, ↩
                  MPI_COMM_WORLD,
102           &status);
103        }
104     } else {
105        MPI_Send(c, stripe * size, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
106     }
107
108     if (my_rank == 0)
109        // stop the timer
110        cout << "elapsed time = " << timer.lap() << endl;
111
112     // results
113     if (print_option && my_rank == 0)
114        print(c, size, 'c');
115
```

```
116         MPI_Finalize(); // shut down MPI
117     }
```

5.2 Execution output

Check output is correct

```
1   [wyxiao_css534@cssmpi1 lab2]$ ./matrix 100 y > maxS.txt
2   [wyxiao_css534@cssmpi1 lab2]$ mpirun -n 4 ./matrix_mpi 100 y > maxM.txt
3   [wyxiao_css534@cssmpi1 lab2]$ diff maxS.txt maxM.txt
4   1c1
5   < elapsed time = 5185
6   ---
7   > elapsed time = 7133
8   [wyxiao_css534@cssmpi1 lab2]$
```

Check output the performance improvement: 774693 / 258639 = 2.9952 times

```
1   [wyxiao_css534@cssmpi1 lab2]$ ./matrix 500
2   elapsed time = 774693
3   [wyxiao_css534@cssmpi1 lab2]$ mpirun -n 4 ./matrix_mpi 500
4   elapsed time = 258639
5   [wyxiao_css534@cssmpi1 lab2]$
```