Name: Yifeng Zhang

Date: Oct 15, 2019

Note: please read readme.txt to know the files detain in my submission folder

CSS534 Program1

Parallelizing Traveling Salesman Problem with OpenMP

The Traveling Salesman Problem is a well know NP-Hard problem with a computational time complexity of O(n!). A more scalable solution of TSP is much more favorable even at the cost optimality. This program is an implementation of a traveling salesman problem (TSP) based on the concept of genetic algorithms (GA) and parallelize it with OpenMP.

Algorithms:

. The key to GA-based TSP is to design a suitable crossover algorithm.

Use a greedy crossover algorithm to generate 2 child from 2 parents: parent[i], parent[i+1]. I start from the first city of parent[i][0], compares the cities int parent[i] and parent[i+1]. For example: the current city in the parent[i][j1], parent[i+1][j2]. Compare the distances from current city to parent[i][j1-1] and parent[i][j1+1] to get next city next1, and compare the distances from current city to parent[i + 1][j2-1] and parent[i+1][j2+1] to get next city next2. If both are not selected yet, choose the closer one. If one city already selected, choose another one. If both are selected, get random next city.

. Set Distance Matrix: distance[CITIES][CITIES]

Since assume that a travel will start from (0,0), we also need calculate the distance between each city to (0,0). I use distance[i][i](suppose to be 0) to store the distance value between each city to (0,0).

OpenMP

OpenMP(Open multi-processing) support shared memory multiprocessing in C/C++. Using the omp pragma, each thread executes a copy of the code within the loop body. I use the omp pragma in each of loop body in my program to improve the performance with multi-threads.

In lab1, pi\_intergral\_omp is a good example that the performance improvement with OpenMp is much better than without using openmp(single thread). But for pi\_monte\_omp, we didn’t see the same result as pi\_intergral\_omp. Because in pi\_monte\_omp, use rand() function in the loop body and rand() function will create the same sequence number. In order to avoid the same rand() function behavior, should initialize random numbers generator in each parallel thread with various values.

For meeting the requirement of performance’s improvement with more than 2.4 time (3 threads to 1 thread), I must use as less as possible rand() function. I improved my performance by changing MUTATE RATE from 50(50%) to 10(10%) in Trip.h for my mutate() function. In addition, I tried to use a changing global variable plus the current city index to generate a next random city index without using rand() functions(I used both solution. Using rand() method has worse performance improvement).

The result: I got multiple results and pick one of result showing the best performance.

Shortest trip:

I got the correct shortest trip is always equal or less than 449.658.

The shortest trip I even got is:

shortest distance = 431.669

itinerary = V1YZH5CWS20EUJ6OI84TNXGK9FAL7R3MPBDQ

Performance: Since cssmpi machines have only 3 CPU cores. I test my code with 1 - 3 threads

I run my code with 1 ,2 ,3 threads

The one of my best performance improvement with 3 threads is ***15284102/6228208 = 2.45*** times better than with only 1 threads.

**Note**: mostly the performance improvement with 3 threads is 2.2 – 2.5 time better than with only 1 threads.

**[15:12:37] yifengzh\_css534@cssmpi7: ~/css534/hw1/try7 $** ./tsp1 3

# threads = 3

generation: 0 shortest distance = 1265.9 itinerary = V1SPMBQAN26G4J37DX8OTF95ZUH0EYRLCWKI

generation: 54 shortest distance = 433.582 itinerary = V1YZH5CWS20EUJ6OI84NTXGK9FAL7R3MPBDQ

generation: 60

generation: 75 shortest distance = 431.669 itinerary = V1YZH5CWS20EUJ6OI84TNXGK9FAL7R3MPBDQ

generation: 80



Discussions: about the parallelization, the limitation, and possible performance improvement of your program.

Parallelization is the act of designing a computer program or system to process data in parallel. Large problems can often be divided into smaller ones, which can then be solved at the same time.

prog1 use OpenMP to improve the performance with TSP. OpenMP is portable, shared-memory threading API. By default, each thread executes the parallelized section of code independently. Work-sharing constructs can be used to divide a task among the threads so that each thread execute its allocated part of the code.

The limitation:

1. It does not support parallelization over multiple computer nodes(cores). Cssmpi machines have only three CPU cores. Therefore, using beyond three threads have no further performance improvement.

2.The algorithms must be managed in such a way that they can be handled in the parallel mechanism

3. The algorithms or program must have low coupling and high cohesion. But it’s difficult to create such programs.

possible performance improvement:

1. In my program, found the shortest trip mostly before the 100th generation. At least in the 1/3 generation, my algorithm didn’t use it to find efficient shortest trip. I will add if syntax in the GA master loop to check if need break the loop or continue the next generation.
2. better loop contracture to utilize parallel computing
3. may try sections construct + loop construct

Lab1 - Calculate PI using OpenMP with 2 different methods.

Method1: Using trapezoid rule;

When using OpenMP for this solution, all threads parallel execute the loop:

Thread 1: 0 - N/nThreads;

Thread 2: N/nThreads - 2 \* N/nThreads;

...

Thread N: (N - 1)/nThreads - N;

It is obvious the performance using OpenMP is much better than without it. Ideally, the performance improvement with threads using OpenMP should be about 3 times larger than without using OpenMP.

From my code, the performance improvement = ***17105/6164 = 2.78*** times.

Methods2: Using Monte Carlo method

This method using rand() function in the loop and rand() function will create the same sequence number. In order to avoid the same rand() function behavior, should initialize random numbers generator in each parallel thread with various values. So, the performance with OpenMP is not improve much than without using OpenMP.

From my code, the performance improvement = ***108056/887347 = 1.22*** times.

***Note****: all source code and output files in the lab1 folder.*