COMP90025 Report
A Hybrid Parallel Implementation of Ant Colony Optimisation on Travelling Salesman Problem Using Message Passing Interface (MPI) and OpenMP

Hongfei Yang <hongfeiy1>

Introduction

The Travelling Salesman Problem (TSP) is the classical problem of finding the shortest path to explore a set of n cities by only visiting each city exactly once. The problem can be represented as a set of Euclidean 2D coordinates on a plane, each refers a particular city, where an inter-city distance can be calculated using a pair of corresponding coordinates. It is known as a NP-Hard problem with a runtime complexity of O(n!) and it is impossible to compute the optimal solution given a large number of cities, even on an advanced modern computer.

To find the optimal path for a small number of cities, one may possibly brute force each possible path. Even better, one may estimate the problem size beforehand and assign every processor with a equal number of sub-problems on a PRAM model to speed up a problem solver. However, this method is still facing an exponential growth of problem size and is unable to find a solution for an even larger number of cities.

Since the applications of TSP mainly focus on logistics and modern circuitry manufacture, it is still useful to find a suboptimal solution that is good enough due to the nature of this problem. Therefore, I would like to propose solving TSP with Ant Colony Optimisation (ASO) to find a near-optimal solution.

Overview of ACO

Inspired by ants, Marco Dorigo proposed Ant Colony as a graph search method in his PhD thesis in 1992. He discovered that ants could always find the shortest path to a food source through a set of obstacles with pheromone-based clues. Similarly, ASO uses a set of artificial ants to explore a TSP problem. A colony refers to a set of cities that are fully connected to each other. It keeps a distance matrix to store the edge (line connecting two cities) distance between two cities and a pheromone matrix to store pheromone level of each edge. Each ant will traverse the entire graph by visiting each city exactly once. Upon each edge traversal, each ant leaves a substance called pheromone which is used as a communication tool to inform the colony about the utility of this edge. The pheromone level on an edge will influence an ant's choice about its next city. Therefore, after n iterations, the shortest path found would converge to an optimum.

There are several important factors regulating the behaviour of the algorithm.
The first one is the edge selection rule. Generally, The probability of the k th ant moving from city x to city y is given by this formula:

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{z \in \text{allowed}_x} (\tau_{xz}^\alpha)(\eta_{xz}^\beta)}$$

Where tau(x,y) is the amount of pheromone present on the edge of x and y. Alpha is positive decimal number to control tau(x,y). Ita(x,y) is the desirability of an edge, beta is a number not less than 1 to control ita(x,y). Tau(x,z) and ita(x,z) are attractiveness and pheromone level of all other possible edges.

The second one is the 'pheromone update rule'. This rule states that after all ants have completed a solution, the pheromone level on edges are updated by

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

,Where

$$\Delta\tau_{xy}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

, Q is a constant to control total pheromone deposited by an ant to complete a tour, L(k) is the total cost of the its solution and rho is the the evaporation constant.

Sequential implementation of ACOTSP:

```
Initialise pheromone level on all edges;

While (terminating condition is not met) do
```

```
      For (each ant) do
            For (number of cities) do
                  Choose and move to the next city according to edge
                  selection rule
            End for
      End for

      Find best

      For each edge do
            Evaporate pheromone
      End for

      For number of ants do
            For number of cities do
                  Deposit pheromone
            End for
      End for

      Reinforce the pheromone level on the best path
End while
```

One may intuitively see the parallelisable nature of ACOTSP. All artificial ants are acting independently from each other: the selection of each ant's next city is independent from other ants and the pheromone matrix update is also independent. The author uses MPI (Message Passing Interface) and OpenMP as tools to parallelise this problem. The parallel approaches used are based on the well-known master-slave approach where the master process assigns work for its slave processes and collect them once the slave processes finish their work. MPI is suitable for distributed memory system for inter-node communications while OpenMP is thread based and is suitable for shared memory system within a node. Therefore, the combination of these tools could maximise parallelisation possibility of this problem.

There are two major approaches to parallelise this problem:

- Coarsely-grained: Communication frequency is low for lower communication overhead
- Finely-grained: Communication frequency is high for better load balancing.

The parallel approaches used in this project is a combination of both coarsely-grained and finely-grained strategy. Coarsely-grained strategy is used for parallelising colonies while finely-grained strategy is used for in-colony computation like solution construction and pheromone updates.

The colony parallelisation is done using MPI master/slave approach. At the beginning of the algorithm, master read all cities and broadcast them to all slaves. Each slave computes inter-city distance and initialise their own pheromone matrix. Each slave then start to run a sequential copy of the program. Upon completion of a cycle, which means each ant in a colony has constructed a solution, each slave process finds its own best solution. The master process then gathers all best solutions from all of its slave processes and finds the best solution, broadcast it to all slave processes. Each slave process, knowing the current global best solution, reinforces this path's pheromone level. Each slave process then continues its pheromone updates. The process continues till it reaches terminating condition.

The above procedure are implemented using MPI, where each node runs an independent colony to communicate its own best path to each other. Within a node, it still runs an sequential version of the original algorithm. This brings up a parallelism opportunity using OpenMP. Each slave process could run a OpenMP parallelised version of the sequential algorithm. Four independent sections can be parallelised: Initialisation of pheromone matrix could be done in parallel, each ant could choose next city and construct solution independently in parallel, and

the pheromone update is independent. More specifically, the pheromone update section consists of two subsections, pheromone deposition and pheromone evaporation. Pheromone deposit is the process of a local pheromone matrix collects all pheromone deposited by all of its ants after completion of a cycle. Pheromone evaporation is to reduce every cell in the local pheromone matrix by a constant factor. Both these subsections could run in parallel.

The algorithm is described below:

On Master

```
Read input cities in cityList
Broadcast cityList to all processor

While (termination condition is not met) do
      gather all local best solutions from all processors
      Find the best global solution
      Broadcast best global solution to all processors
End while
```

On slave

```
Receive input cityList
Initialise pheromone matrix

While (termination condition is not met) do
      Find local best using OpenMP parallelised sequential algorithm
      Send local best to master
      Receive global best solution from master
      Reinforce local pheromone matrix with global best
End while
```

OpenMP parallelised sequential algorithm running on slave process:

```
Initialise pheromone level on all edges

For (number of tours) do
      For (each ant) do in parallel
            For (number of cities) do in parallel
                  Choose and move to the next city according to edge
                  selection rule
            End for
      End for

      Find best solution and store in a global variable

      For each edge do in parallel
            Evaporate pheromone
      End for

      For number of ants do in parallel
            For number of cities do in parallel
                  Deposit pheromone
            End for
      End for
```

```
End for

Send local best to master
Receive global best solution from master
Reinforce local pheromone matrix with global best
```

The number of threads is set to the number of cores in a node. This is because no threads will be blocked while waiting for a system resource by the nature of ACOTSP, therefore the number of threads is set to be the typical number of cores here. There are hyper threading processors that can do two threads in one core, but creating of excess threads would not be much useful here according to Amdahl's law which states that sequential part put a upper limit to the maximum performance gain in parallel. A `#pragma omp parallel num_threads(n)` is put before each parallel sections to maximise performance, A `#pragma omp for` is put before each for loops to parallelise for loops. Dynamic scheduling is also employed to increase the load balancing of parallelism and is done using `schedule(dynamic,1)` to keep all threads busy.

The pheromone deposit phase may involve race conditions, where an edge is traversed by more than one ant. Therefore a `#pragma omp critical` block will be employed to ensure the correct updates of the pheromone matrix.

Some may notice that the communication overhead between colonies are very high if it takes a large number of iterations to make the solution converge to an optimal. To address this issue, one could limit the number of communication calls on the master thread, by letting each colony optimises its own optimal solution with more cycles instead of one and communicate a better solution to master.

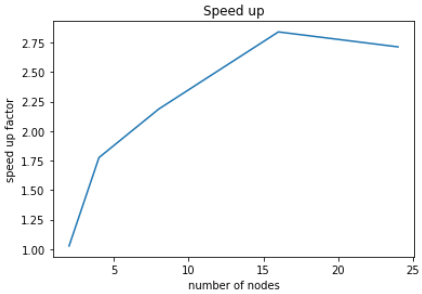
Experimentation

The test graph is selected from National TSP (http://www.math.uwaterloo.ca/tsp/vlsi/index.html) and is called XQF131.tsplib. There are 131 cities in this set and only the city coordinate parts were used. Since this algorithm depends on randomised input to increase the probability of discovering better solution, each comparison was taken of the average time of 4 runs, hopefully to reduce bias.

The parallel implementation of the algorithm is tested against a serialised one. The constants used in this algorithm are fixed, where Q = 80, rho = 0.5, alpha = 1.0, beta = 2.0. The number of ants is set to 100, number of tours is set to 30 and the number of communications limit is set to 4. For a sequential algorithm, the total number of iterations is set to be the multiplication between number of communications, number of tours and number of nodes. One may vary these constants to get better performance.

The sequential and parallel algorithms are put to test. On Spartan, the number of CPUs per node are set to 2.

The speed up graph is plotted below:



One may notice that the speedup starts to drop after 16 nodes. This may be caused by the high amount of communication cost. As the communication starts to increase, the amount of overhead eventually put an upper limit to the parallelisation.

Conclusions

The parallelisation ACOTSP is intuitive, however one must pay more attention to the high communication overhead involved in this problem, therefore only the most computational intensive part should be parallelized to achieve better performance.