MSc in CSTE
High Performance Technical Computing

Assignment

December 8, 2022

Hand in date: 06/02/2023 (FT), 20/02/2023 (PT), 09:30am

1 Introduction

In this assignment you are asked to examine the application of distributed-memory parallel programming techniques for the implementation of a combinatorial optimisation problem, the wandering salesman problem. The objective of the wandering salesman problem (WSP) is to find the shortest route for a traveling salesman so that the salesman visits every one of a set of cities exactly once. Unlike the traveling salesman problem, in the WSP the salesman does not return home. The WSP is a hard combinatorial optimization problem. For N cities, there are at most $N!$ possible routes, assuming cities are fully connected.

You are asked to consider branch-and-bound solutions to combinatorial optimisation problems. Initially, you may consider a simple exhaustive evaluation as a way of solving the WSP. One way you might think about evaluating every possible route is to construct a tree that describes all the possible routes from the first city. Figure 1 shows an eample for a four city problem.

For this WSP, $d[1][2] = 5$, $d[1][3] = 40$, $d[1][4] = 11$, $d[2][3] = 9$, $d[2][4] = 6$, $d[3][4] = 8$. All possible routes can be found by traveling from the root to a different leaf node three times, once for each unique path and then back to the root. For example, by taking the middle branch from the root node and the right branch after that, the route $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$ is produced, and has a distance of $40 + 8 + 6 = 54$. There are six unique root to leaf paths in this tree. Each possible route is represented twice, once in each direction.

A simple exhaustive evaluation of WSP for this example would then be to follow the six root to leaf paths and determine the total distance for each path by adding up the distances indicated by each edge in the tree and then adding the distance back to the root. The route with the smallest distance is then chosen.

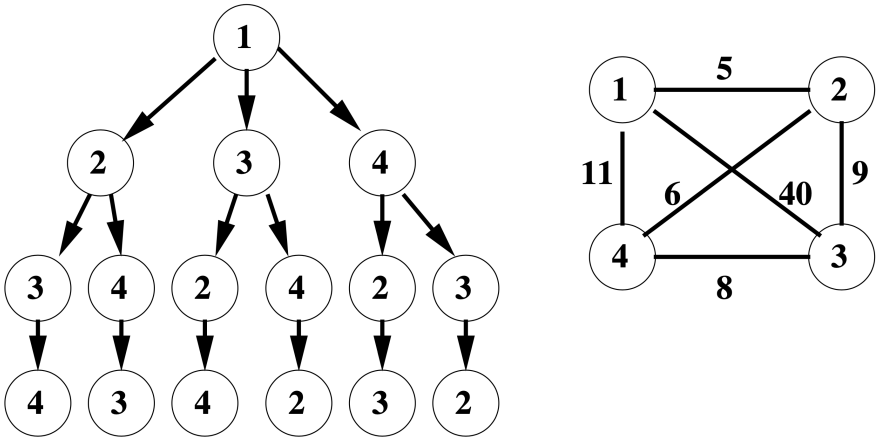

Figure 1: A four-city wandering salesman problem. (L) A tree representing all unique paths through all four cities (R) A graph providing distances between cities.

A better way to traverse each tree is to do it recursively. Here the summation of the earlier parts of each route is not repeated every time that route portion is reused in several routes. The Branch-and-Bound approach uses this type of problem formulation, but with some added intelligence. It uses more knowledge of the problem to prune the tree as much as possible so that less evaluation is necessary. The basic approach works as follows:

1. Evaluate one route of the tree in its entirety, (for example, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$) and determine the distance of that path. Call this distance the current "bound" of the problem. The bound for this path in the above tree is $5 + 9 + 8 = 22$.

2. Next, suppose that a second path is partially evaluated, for example, path $1 \rightarrow 3$, and the partial distance, 40, is already greater than the bound. If that is the case, then there is no need to complete the traversal of any part of the tree from there on, because all of those possible routes (in this case there are two) must have a distance greater than the bound. In this way the tree is pruned and therefore does not have to be entirely traversed.

3. Whenever any route is discovered that has a better distance than the current bound, then the bound is updated to this new value.

The Branch-and-Bound approach always remembers the best path it has found so far, and uses that to prevent search down parts of the tree that couldn't possibly produce better routes. For larger trees, this could result in the removal of many possible evaluations.

For the purposes of this assignment, the input to the problem is given in the form of a matrix. An element of the matrix, $d[i][j]$ gives the distance between city i and city j. The input to your program is a file organized as follows:

$$
\begin{array}{llll}
N & & & \\
d[1][2] & & & \\
d[1][3] & d[2][3] & & \\
d[1][4] & d[2][4] & d[3][4] & \\
. & & & \\
. & & & \\
. & & & \\
d[1][N] & d[2][N] & d[3][N] & \ldots \quad d[N-1][N]
\end{array}
$$

where N is the number of cities, and $d[i][j]$ is an integer giving the distance between cities i and j. The output from the program should be an ordered list of cities (numbers between 1 and N). Clearly, there are 2 equivalent permutations - either one is acceptable.

2 HPC Tasks

1. First devise a serial algorithm to solve the Branch-and-Bound algorithm for the wandering salesman problem and then devise a way to parallelise it.

2. Analyse your parallel algorithm; present and describe the flow of the algorithm, significant data structures, problem decomposition, load balancing, and potential synchronisation points where applicable.

3. Implement your algorithm in a program written in a programming language of your choice, between C, C++ and FORTRAN, and coupled with MPI. Your program should accept the command-line argument `-i` to specify the input file name, which should be in the format described above.

4. Provide your solution to the problem given in `input/distances`. This file contains a 17-city problem. For debugging purposes, you may want to use some of the smaller input files included in the same directory. The city locations, corresponding to the distance files, are provided in the corresponding `city` files and you may wish to use them for visualisation. Compare the solutions obtained by your parallel algorithm with that of the serial program.

5. Measure the cost of communications and the cost of computations for your parallel solution. How is the communication cost affected by the use of different communication patterns?

6. Measure the performance of your serial and parallel codes and discuss. How does the parallel program performance compare to the theoretical one? Is the performance of your parallel approach the expected/theoretical one?

7. Based on your results, reason on the problem size deemed necessary in order for MPI parallelisation to become efficient for the above algorithm.

3 Source Code and Report Requirements

The source program will need to compile on Crescent using the Intel compilers and Intel MPI. Your simulations for the above tasks should be performed on Crescent using the queueing system and working nodes.

Write a report to present and discuss your findings. The report should be no less than 1,500 words and must not exceed 3,000 words. The report can contain any number of figures/tables, however all figures/tables should be numbered and discussed. The report should include a description of the design of your solution explaining your choices. The source code, a sample scheduler script and the solution to the `input/distances` problem should be included as Appendices to the report.

4 Assignment Submission

The source code and the sample scheduler script must be submitted electronically via the **Canvas Coding Files submission point** by 9:30am on the 6^{th} February (full-time students) or the 20^{th} February (part-time students).

The report should be submitted electronically via the **TurnItInUK Report submission point** by the prescribed deadline, for the assignment submission to be considered complete.

5 Marking

The assignment will be assessed based on the following marking scheme:

- 20% Introduction, methodology, conclusions
- 40% Source code, design
- 30% Discussion and analysis of the results
- 10% Report structure, presentation, references