CSCI-GA.3033-016

Multicore Processors: Architecture & Programming

Lab 1

In this lab you will implement a method for solving a modified version of the traveling salesman problem in OpenMP.

What is the problem definition?

You have a group of n cities (from 0 to n-1). The traveling salesman must start from city number 0, visit each city *once*, and does **not** have to come back to the initial city. You will be given the distance between each city (cities are fully connected to each other). You must pick the path with shortest distance. Note that there may be several paths that satisfy the above conditions, you just need to find one of them.

What is the input?

The input to your program is a text file that contains an NxN matrix, one row per line, representing the distance between any two cities. Your program will be called ptsm (for parallel tsm). The command line then will be:

./ptsm x t filename.txt

Where :
x is the number of cities
t is the number of threads
filename.txt is the file that contains the distance matrix

What is output?

You program must output, to screen, the best path you found and the total distance.
Here is an example output:

Best path: 0 1 3 4 5 2
Distance: 36

This means the best path your program found is 0->1->3->4->5->2 with total distance of 36.

How will you solve this?

This problem is a combinatorial optimization problem (n! possible solutions for n cities). This means an exact optimal solution will take very long time. However, in this lab we will test your submission with a relatively small number of cities. So, you will implement an exact solution.

We will not test your submission with more than 12 cities and we will not test with more threads than cities.
For example if we have 10 cities and we have two threads. We always start with city 0. Thread 0 will be responsible for exploring paths that start from 0 and go to either 1, 2, 3, 4, or 5; while thread 1 will explore threads going from 0 to 6, 7, 8, or 9. Each thread can use recursion to check the paths. At the end, each thread will pick the shortest path from the paths it has explored. The final answer is the shortest path found among all threads.

How will we grade this?

The total grade is out of 40

- 5 points if your program outputs a legal solution (i.e. each city visited once) but not necessarily one of the shortest paths.
- 20 points for the report
- 15 points if your solution is the optimal solution.

To help you:

We are providing you with two executables:

- ./tsm : This program generates test files (usage: ./tsm numcities). The output of this program is cities*x*,txt that contains the matrix of the x city distances. It will also print a (sub) optimal solution on the screen. The file citiesx.txt is a text file that you can open and take a look.
- ./tsmoptimal : This programs prints the optimal solution on the screen (usage: ./tsmoptimal x cities*x*.txt) for the x cities problem. This is a sequential program.

The report

You need to write a report that includes the following:

- A graph that shows speedup over single-thread version for a problem of size 4 cities, another graph of problem of size 8 cities, and a third graph for 12 cities. For each graph, the x-axis the number of threads (1, 2, 3, and 4) and y-axis is the time generated by *time* command (real, not user or system).
- Explain your analysis of the graphs. Your analysis must not be generic but must explain why, for example, a solution with larger number of threads takes longer time than one with lower number of threads. You may need to measure the time taken by different parts of your program (hint: check the usage of clock() inside your code).

Regarding compilation

- Name your source file: netID.c (where netID is your netID number) and add as many comments as you can. This will help us give partial credit in case your program does not compile.

- Do the compilation and execution on crunchyx (x = 1, 3, 5, or 6) machines.
- Use the latest version of gcc on crunchy by typing: module load gcc-9.2

What to submit?

Add the source code netID.c as well as the pdf file that contains your graphs and analysis to a zip file named: lastname.firstname.zip
Where lastname is your last name, and firstname is your first name.

How to submit? NYU classes → Assignments → lab1

Have Fun!