ECE 5750 – Advanced Computer Architecture
Programming Assignment #2

Objective

The objective of this assignment is to implement a parallel shared-memory version of the well-known ***N-Queens*** problem from scratch. You will again use POSIX threads. You will have the opportunity to run and evaluate your shared-memory program on a 20-core processor.

This assignment is due on Oct 30 at 5pm EST. You will again work in group of twos like PA1, but **you must work with a different partner**.

Problem description: The N-Queens Problem

The well-known *N-Queens* problem asks you to find out all the possible ways of putting N queens on an $N \times N$ chessboard such that no two of them attack each other. A queen can attack another if it is on the same row, the same column or the same diagonal.

In this assignment the problem that you are going to solve is a slightly modified version of the *N-Queens* problem. We assign a profit to each of the squares of the chessboard. The (i, j) entry has a profit of $|i-j|$ for $0 \leq i \leq N-1$ and $0 \leq j \leq N-1$ (use ***abs*** function of C to calculate the absolute value). Thus if you place a queen on the (i,j) square you gain $|i-j|$ units. In this assignment, you will parallelize this problem and write a shared-memory version of it. Your program should take two parameters from the command line, namely, N and P. The program on termination should print out the total number of solutions found, the solution that maximizes the profit (find a good way to print this solution) and the profit corresponding to this best solution. Also, it should print out the timing breakdown i.e. initialization time, computation time, and finish time. Use the same methods for measuring timing statistics as done in PA1.

Production runs

Once you have tested your program to run correctly, you should execute the following steps to submit your program for execution on the production machine. Please remember that the head node (where you log in) is a virtual machine, whose purpose is solely to give you access to your files and to the production machine. It is not meant to run extensive tests, much less production runs. On the other hand, please don't clog up the production machine with tests—it's much better if you run these on your laptop, for example.

Submission

You must submit a **correct** parallel implementation of the algorithm described. You are allowed to use the web or any other consulting resource to find the best possible approach; however, **you must properly cite all your references**. Your solution should be shared-memory and use POSIX threads where appropriate.

You must also submit a four-page report (PDF), in which you explain:

- ***The Algorithm:*** what algorithm you used to parallelize the problem and the rationale behind it. Discuss how you partitioned the task and the trade-offs involved.

- ***Experimental results:*** prepare a table showing the number of solutions and the maximum profit achievable for *N* increasing from 8 to 15. Also list the solutions that achieve the maximum profit for these cases. Next, report (in a meaningful format) the total time and computation time respectively for *N*=8, 9, … 15 and *P* up to 20, and the corresponding speedup.

- ***Discussion:*** Comment on the results regarding the speedup and discuss whether they agree with the theoretically expected results. Also, include a discussion on the shortcomings and the possible improvements of your implementation.

In summary, each group should submit two files: *nqueens.c*, and *report.pdf*. To avoid violating double-blindness in peer review, use the exact file names, and don't include any identify information in any of your files. You could submit multiple c files with multiple implementations of nqueens, if they show different tradeoffs and it's hard to choose the best (note that it's not required and not encouraged to do so). Name them as *nqueens1.c*, *nqueens2.c*, etc. If you work for the extra credit below, submit another file called *latin.c*.

Extra Credit

The following should only be attempted after finishing your *N-Queens* code above. Your efforts can add ***up to 20*** points. The assignment is to write a parallel shared-memory version of ***Latin Squares***. In the *Latin Squares* problem there is an $N \times N$ matrix and N colors. The goal of the program is to color the matrix so that a color appears only once in each row and each column. Clearly, if you started with an empty (un-colored) matrix, the problem is trivial. Similarly, given a pre-colored matrix with S set (colored) cells and some "holes," when starting with only a small number of holes (S is close to $N \times N$) it is easy (quick) to determine whether the matrix is colorable or not. The difficult cases (and long running!) are when $S = N^2/2$ but can vary substantially.

Your first task is to come up with a scheme to generate random solvable instances of the problem with a parametrizable S (you may want these random instances to be repeatable for testing purposes). Then for S=0.3×, 0.4×, and $0.5 \times N^2$ solve as large a problem (N) as possible in under 2 minutes.

In your report, state how large a problem you were able to run, how long it took, and outline the algorithm and any design decisions you care to share. There are many choices for matrix representation, color data structures, etc. It has been observed that randomness helps the convergence of this problem, so you may want to keep that in mind when making some color choices.