COMP 429/529: Assignment 1-Part 2

Due: 11.00 pm on Sunday, March 8th, 2020

Notes: You must do this assignment individually. No partners are allowed. Also see notes from Part I. Post your project related questions to the Blackboard forum.

Corresponding TA: Palwisha Akhtar (pakhtar19@ku.edu.tr).
Her office hours: Thursdays 16:00-18:00 pm or by appointment, Location 110
Corresponding TA: Aditya Sagonko (msasongko17@ku.edu.tr).
His office hours: Tuesday 16:00-18:00 pm or by appointment, Location 110

Description

In this assignment you will parallelize an n-queens solver using OpenMP. This application will help you exercise your knowledge in task parallelism. You are provided with the serial version of this application and you must develop your parallel implementation on top of it. Be aware that each part requires you to collect performance data with several runs so allow yourself enough time to conduct those experiments before the deadline.

PART II: Parallel N-Queens Solver

In the second part of this assignment, you are asked to parallelize a serial n-queens solver with OpenMP. Similar to Part I, you are provided with a serial n-queens solver code, which takes an n-queens problem as an input and finds **all possible solutions** from it. We are using a brute force search for searching for all possible solutions to the problem.

For a given `nxn` chessboard, the idea is to generate every possible move by trying each cell within a column `i`, such that `0` $\leq$ `i` $<$ `n`, and then, testing to see whether queen placement in the cell satisfies the n-queens' constraint (no chess queen in the board/matrix can kill each other). After finding a cell in column `i` that satisfies the constraints, the program moves to column `i+1` to find another queen placement in that column. This process is repeated recursively until the last column `n - 1`.
A solution is printed after there is a queen in every column of the matrix with the n-queens' constraint being satisfied. After finding a solution by placing the last queen in column `n - 1`, the program backtracks to the previous column `n-2` to try other cells in the column and moves forward again to find another placement of the last queen in column `n - 1`. This backtracking process is repeated so that when the program backtracks to a column `i`, all possible placements in columns `j`, s.t. `i` $<$ `j` $<$ `n`, have been found. By the time the backtracking process completes, all possible solutions for n-queens problem have been found.

Part-II-A Task Parallelism

In the first part, you are required to parallelize the n-queens solver with task parallelism. Similar to the serial version, the parallel version is expected to find all possible solutions to a given n-queens problem.

Part-II-B Task Parallelism with Cutoff

The first implementation results in too many tasks in the system which can easily degrade the performance. As a result, you may not experience any speedup or very disappointing speedup. In this part, you will implement an optimization to improve the performance of the parallel code by using a cutoff parameter to limit the number of parallel tasks in your code. To limit task generations, beyond certain depth in the call-path tree of the recursive function, switch to the serial execution and do not generate tasks. This depth is determined by the cutoff parameter you choose.

Part-II-C Task Parallelism with Early Termination

In this part, you will start working from the parallel code that you implemented in Part-II-A. You are going to modify the code so that it stops after finding one solution instead of all possible solutions. To make a fair performance comparison, you should also modify the serial version so that it also stops after finding one solution to n-queens.

Experimental Study

You must collect the performance data on the KUACC cluster. Plot your data as figures and include them in your report along with some description for your observations and explanations. Do the experiments with the input size of 14 and with thread count of 1, 2, 4, 8, 16, and 32 threads unless stated otherwise..

a) Scalability Test

Perform the same strong scaling experiment on your implementation.

- For the parallel code from Part-II-A and B, you must compute the speedup with respect to the given serial n-queens solver.
- For the parallel code from Part-II-C, you must compute the speedup with respect to a modified version of the given serial n-queens solver that also terminates after finding one valid solution.

b) Thread Binding Test

Perform the strong scaling experiment under 2 different thread mapping schemes; compact mapping and scatter mapping. If you run the previous test with compact mapping, then you just need to repeat the same test with scatter mapping. See lecture 8 for more explanations.

c) Tests on n-queens Problems of Different Sizes

For this test, you will report the running times of the parallel code from Part-II-A on n-queens problems of different sizes by using 32 threads. The sizes that you will test are 13, 14, and 15.

Environment

Even if you develop and test your implementations on your local machine or on the computer labs on campus, you must collect performance data on the KUACC cluster.

- Accessing KUACC outside of campus requires VPN. You can install VPN through this link: https://my.ku.edu.tr/faydali-linkler/
- A detailed explanation is provided in http://login.kuacc.ku.edu.tr/ to run programs in the KUACC cluster. In this document, we briefly explain it for the Unix-based systems. For other platforms you can refer to the above link.
- In order to log in to the KUACC cluster, you can use ssh (Secure Shell) in a command line as follows: The user name and passwords are the same as your email account.

```
bash$ ssh $<$username$>$@login.kuacc.ku.edu.tr
bash$ ssh dunat@login.kuacc.ku.edu.tr //example
```

- The machine you logged into is called login node or front-end node. **You are not supposed to run jobs in the login node** but only compile them at the login node. The jobs run on the compute nodes by submitting job scripts.
- To run jobs in the cluster, you have to change your directory to your scratch folder and work from there. The path to your scratch folder is

```
bash$ cd  /scratch/users/username/
```

- To submit a job to the cluster, you can create and run a shell script with the following command:

```
bash$ sbatch <scriptname>.sh
```

  A sample of the shell script is provided in Blackboard along with the assignment folder. In the website of the KUACC cluster, a lot more details are provided.

- To copy any file from your local machine to the cluster, you can use the scp (secure copy) command on your local machine as follows:

```
scp -r <filename> <username>@login.kuacc.ku.edu.tr:/kuacc/users/<username>/
scp -r src_folder dunat@login.kuacc.ku.edu.tr:/kuacc/users/dunat/  //example
```

  -r stands for recursive, so it will copy the src_folder with its subfolders.

- Likewise, in order to copy files from the cluster into the current directory in your local machine, you can use the following command on your local machine:

```
scp -r <username>@login.kuacc.ku.edu.tr:/kuacc/users/<username>/fileToBeCopied  ./
scp -r dunat@login.kuacc.ku.edu.tr:/kuacc/users/dunat/src_code ./  //example
```

- To compile the assignment on the cluster, you can use the GNU or Intel compiler. The compilation commands and flags for the compilers are provided in a Makefile in the assignment folder. Before using the compilers, you firstly need to load their module if they are not already loaded as follows:

```
bash$ module avail //shows all available modules in KUACC
bash$ module list   //list currently loaded modules.
bash$ module load intel/ipsxe2019-u1ce  //loads Intel compiler
bash$ module load gcc/7.2.1/gcc  //loads GNU compiler
```

- If you have problems compiling or running jobs on KUACC, first check the website provided by the KU IT. If you cannot find the solution there, you can always post a question on Blackboard.
- Don't leave the experiments on KUACC to the last minutes of the deadline as the machine gets busy time to time. Note that there are many other people on campus using the cluster.

Grading

- Parallel Version of n-queens Part II-A (15 pts), Part II-B (10), Part II-C (15 pts), Performance Study for Part II (10 pts)
- A Written Report with Explanations about Implementation, Speedup Figures and Discussions (10 pts)
- Important Note: You may lose points both for parallelization bugs (e.g., race condition, inappropriate private or shared variable usage etc) and performance bugs (e.g. unnecessary synchronization point) in your implementations.

Submission

- Document your work in a well-written report which discusses your findings (10 pts).
- Explain your implementation in the report, if needed, provide a pseuedo code.
- Identify performance bottlenecks and discuss them in your report.
- Report scalability and performance study figures.
- If you achieve good performance, explain why. Similarly, if you don't achieve good performance or scaling, explain why.

- Submit both the report and source codes electronically through blackboard.
- **In the report, first paragraph should clearly mention which parts you have completed in this assignment.**
- Please create a parent folder named after your KUSIS ID. Your parent folder should include a report in pdf and a directory of n-queens_solver code. If you have intermediate implementations that are less optimized, number them with an increasing version number. Makefile should compile the most optimized version. Be sure to delete all object and executable files before creating a zip file.
- GOOD LUCK.