COMP 429/529: Project 1

Due: 11.59 pm on Tuesday, March 19th, 2019

Notes: You may discuss the problems with your peers but the submitted work must be your own work. Late assignment will be accepted with a penalty of -10 points per day up to three days. Please submit your assignment through blackboard. This assignment is worth 15% of your total grade. **You must do this assignment individually. No partners are allowed.** Post your project related questions to the Blackboard forum.

Corresponding TA: Muhammad Aditya Sasongko (msasongko17@ku.edu.tr).
His office hours: Wednesdays 16:00-18:00 pm or by appointment, Location 110

Description

In this assignment you will parallelize two different applications; an image blurring algorithm and sudoku solver using OpenMP. While the first application exercises your knowledge in data parallelism, the second application helps you gain knowledge in task parallelism. You are provided with the serial version for each application and you must develop your parallel implementation on top of those. 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 I: Image Blurring

In the first part of this assignment you will implement a parallel version of a simple image blurring algorithm with OpenMP. You are provided with a serial program which takes an input image and outputs a blurred image.
The image is represented as a 2-dimensional grid with three components. We have provided you with image reading and writing utilities already. After reading the image to be filtered, the program generates an n by n filter. The filter is then applied to blur every pixel in the image. For pixels which are located along/near the edges of the image, zero padding technique is used to add additional zero-valued pixels beyond the edges of the image.

Experimental Study

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. For details, please refer to the Environment section in this document. Before conducting performance studies, first make sure that your parallel code is correct by comparing the average the pixel values. Note that because of the floating point precision, the average values might slightly differ after the 5 or 6th decimal point, which is fine.

Input Images

Use cilek and coffee images provided in the assignment for the experiment study.

a) Scalability Test

For the scalability test, you will report the speedup of the parallel code under different numbers of threads.

- Here, your baseline is the serial code.
- In addition, report the parallel execution time with single thread and observe if there is any parallelization overhead by comparing the running times of the serial version with the parallel version with one thread.
- Test your performance under serial, 1, 2, 4, 8, 16, and 32 threads. Which thread count gives you the best performance?
- Explain the speedup curve you observe
- If you don't experience any speedup, your implementation probably has some performance bugs. Fix those first, then collect the performance numbers again.

b) Thread Binding Test

For this test, you will report the speedups of the parallel code when the number of threads is 16 and all threads are mapped to cores with two different mapping strategies; *compact and scatter*. In the compact mapping, multiple threads are mapped as close as possible to each other, while in the scatter mapping, the threads are distributed as evenly as possible across all cores. Here is an example how to set the affinities for the Intel Compiler. You can refer to this website for details: https://software.intel.com/en-us/node/522691

```
bash$ export KMP_AFFINITY=verbose,granularity=fine,scatter
bash$ export KMP_AFFINITY=verbose,granularity=fine,compact
```

Which mapping provides better performance? Why?

PART II: Parallel Sudoku Solver

In the second part of this assignment, you are asked to parallelize a serial sudoku solver with OpenMP. Similar to Part I, you are provided with a serial sudoku solver code, which takes a sudoku 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. The idea is to generate every possible move by trying each number within the game and then test to see whether it satisfies the sudoku (every number appears only once in a column, once in a row and once in a square block). We provided 9 by 9 matrix for debugging purposes, which doesn't take

much time to solve. 16 by 16 matrices are the ones you will be using for performance study. For fun, you can use 25 by 25 matrices but those may take hours, if not, minutes to solve.

Part-II-A Task Parallelism

In the first part, you are required to parallelize the sudoku solver with task parallelism. Similar to the serial version, the parallel version is expected to find all possible solutions to a given sudoku 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-A. You are going to modify the code so that it stops after finding one solution. To make a fair performance comparison, you should also modify the serial version so that it also stops after finding one solution to sudoku.

Experimental Study

Similar to Part-I, you must collect the performance data on the KUACC cluster. We have provided some sudoku problems in the assignment. These are solved fairly quickly but you can get started with those. For performance studies, you should be using 4x4_hard_3.csv.

a) Scalability Test

Perform the same scalability test as described in Part-I-(a) with the hard sudoku problem that will be provided for Part-A, Part-B and C.

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

b) Thread Binding Test

Perform the same binding test as described in Part-I-(b) with the hard sudoku problem that will be provided for Part-A, Part-B and C.

c) Tests on Sudoku Problems of Different Grids

For this test, you will report the running times of the parallel code from Part-B on sudoku problems of different sizes and difficulties by using 32 threads. Three sudoku problems are provided to you on Blackboard, those are 4x4_hard1, 2 and 3.

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 Image Blurring (20 pts), Performance Study for Part I (10 pts)
- Parallel Version of Sudoku Part A (15 pts), Part B (15), Part C (20 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. Do not upload image or sudoku input files (we already have them!).
- **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 two subdirectories: image_blurring and sudoku_solvers. The two subdirectories will contain your source 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. Do not upload image files.
- GOOD LUCK.