COMP 429/529- Parallel Programming: Assignment 1- Part 1

Part I Due: 11.00 pm on Saturday, February 29, 2020
Part II Due: 11.00 pm on Sunday, March 8, 2020

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 -20 points per day. Please submit your source code through blackboard. This assignment is worth 20% of your total grade (Part-I 8%, Part-II 12%). Part II will be released within a week and you will have an extra time for it. **This is an individual assignment - no partners.**

Part I: Game of Life

In this assignment you will implement a parallel version of the Game of Life in OpenMP. The universe of the Game of Life is a cellular automaton, in which cells live on a 2-dimensional world. They are born, live and die over successive generations. The world is defined as a binary-valued array, and each generation evolves according to the following rules:

- Each cell can be one of two possible states: alive or dead.
- Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically and diagonally adjacent.
- At each step in time, the following transitions occur:
  - Any live cell with fewer than two live neighbours dies, as if caused by under-population.
  - Any live cell with two or three live neighbours lives on to the next generation.
  - Any live cell with more than three live neighbours dies, as if by overcrowding.
  - Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The first generation can be created randomly and the successive generations will follow the above rules simultaneously to every cell. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations. Here is the pseudocode for the Game of Life:

```
1  for t in 0:T-1  //time step loop
2       forall (i,j) in 1:N x 1:N  //simulation domain
3           nNeigh = number of live neighbors of World(t)[i,j]
4           World(t+1)[i,j] = DEAD
5           if World(t)[i,j] AND (nNeigh == 2 or nNeigh ==3) then
6             World(t+1)[i,j] = LIVE
7           else
8             World(t+1)[i,j] = (nNeigh == 3)
9           end if
10      end forall
```

```
    end for
```

Serial Code

We are providing you with a working serial program that implements the Game of Life. The provided code creates a random world, distributing cells uniformly according to a specified probability. It then runs the game of life for a given number of generations, sending each generation to the gnuplot plotting program.

```
The program may be run from the command prompt as follows:
./life [-n <int>] [-i <int>] [-s <int>] [-p <float >] [-t <int>] [-d] [-step] [-g <int>]

With the arguments interpreted as follows:
-n <integer> Number of mesh points in one dimension
-i <integer> Number of generations (or iterations)
-s  <integer>  Random seed
-p  <float>  Distribution probability
-t  <integer>  Number of threads
-d disables display
-g <integer> selects a game type
-step pauses every iteration (for debugging)
```

Requirements

- To run the program, you will need the **gnuplot plotting program** to be installed on your computer. You can download the software from http://www.gnuplot.info .
- The Makefile that we have provided includes an "arch" file that defines appropriate command line flags for the compiler.

```
To compile, type
      make
To clean the objects and executables
      make clean
Example run:
       ./life -n 500 -i 200 -p 0.2
```

- Since this program requires an interactive interface, you won't be able to use the campus clusters for plotting. Please develop and test your implementations on your local machines or on the computer labs on campus.

Task Parallelism

In this part of the assignment, you will employ multithreading to handle graphical display. You will modify the life simulator to run with two threads. One thread (the plotter thread)

should handle the display (via gnu plot) while the other thread should perform the cell updates. The two activities may take place concurrently. To synchronise the execution of the threads, barriers can be used. Data should be shared between the threads via shared-memory and the mesh plot must appear for each iteration.

Data Parallelism

In this part of the assignment, you will introduce additional threads to share the computational workload in the cell update. To implement data parallelism, use a one-dimensional decomposition where the computational domain is to be divided (approximately) equally among all workers involved in the computation. Lastly, have the master thread initialize the game board. Although this approach isn't scalable because of the first touch policy, the impact on this assignment won't be noticed. Having the master initialise the game board ensures reproducible results and is extremely helpful for debugging.

You can introduce data or task parallelism in any order you like but the final implementation should use a single thread for plotting and the remaining threads to perform cell updates concurrently with the plotter thread. Starting with data parallel implementation might be easier.

Testing Correctness

- You'll notice that the serial program we provided allows you to input a random seed via the -s parameter. If you don't specify this parameter, you obtain a seed based on the time of day. The program outputs this seed so that you will be able to reproduce a given run. In order to establish correctness, run your program on differing numbers of threads, including a single thread.
- Another approach is to run with a problem that has a known solution. The literature is full of problems with known solutions. The simplest problems to test have the static patterns that do not change. We have provided one, which is called block still. You can find some examples on the Wikipedia page for Conway's Game of Life Others include "blinkers" and gliders. A command line option -g can be used to select a game number. You can add the glider (spaceship) game if you like.
- Another command line option -step has been added to allow you to "single step" through the game, one iteration at a time. This is handy for watching moving patterns.
- Finally, another simple way of checking your results against the single thread implementation is to output the contents of all game board locations in a systematic order (say, row major order), printing a single digit (1 or 0) for each position. You can then use the diff program to compare results. Of course, this assumes that your single processor implementation is correct!

Experiments

You are going to conduct an experimental performance study on KUACC. The aim of this experimental study is to get performance data across different numbers of threads and under different input sizes. In these experiments, **you will disable plotting feature**. You should still spare a thread for handling the display but this thread will not perform plotting.

- The first experiment will be a strong scaling study such that you will run your parallelized code multiple times under different thread counts. Please use the command line arguments below for this study.

```
bash$ ./life -n 2000 -i 500 -p 0.2 -d -t <num-of-threads>
```

  <num-of-threads>will be 1, 2, 4, 8, 16, and 32. Plot the speedup and execution time figures as a function of thread count and include them in your report.

- In the second experiment, you will evaluate your code's performance under different input sizes, but with fixed thread count. The command that you will run is as follow.

```
bash$ ./life -n <input-sizes> -i 500 -p 0.2 -d -t 16
```

  <input-sizes>that you will test are 2000, 4000, 6000, 8000, and 10000. Plot the execution time as a function of input size. To observe the scaling, also include a plot which shows the execution time per data point ($runningtime/n^2$), where n is the input size. Include your figures in the report.

To ensure that all of your performance data is taken from the same machine, you can run your commands for the first experiment and the second experiment in the same job script. In other words, all commands for both experiments will be submitted as a single job.

More details on how to run your code in KUACC cluster can be found in Section Environment below.

Submission

- Document your work in a well-written report which discusses your findings.
- Your report should present a clear evaluation of the design of your code, including bottlenecks of the implementation, and describe any special coding or tuned parameters.
- We have provided a timer to allow you to measure the running time of your code. Observe the running time with and without the mesh plotter is on. You will see how much time is spent doing I/O.

- Submit both the report and source code electronically through blackboard.
- You only need to submit the final implementations for OpenMP if your task+data parallelism works properly.
- Please create a parent folder named after your username(s). Your parent parent folder should include a report in pdf and a subdirectory for source code. Include all the necessary files to compile your code. Be sure to delete all object and executable files before creating a zip file.
- GOOD LUCK.

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
```

- To check the status of your currently running job, you can run the following command:

```
bash$ squeue -u <your-user-name>
```

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

Report (10 points), Task (15 pts) and Data Parallel (15 pts) implementation of Game of Life in OpenMP. You may lose points both for correctness (e.g. race condition) and performance bugs (e.g. unnecessary synch. point) in your implementations.

References

http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
Prof. Scott Baden http://cseweb.ucsd.edu/ baden/

Part II

To be announced.