ECE454, Fall 2017
Homework5: Parallelization
Assigned: Nov 20th, Due: Dec 6th, 11:59PM

Yongle(yongle.zhang@mail.utoronto.ca) and Seyed(sa.jokar@mail.utoronto.ca) are the TAs for this assignment.

1 Introduction

OptsRus will now put all of its optimization knowledge to work in its biggest challenge to date: parallelizing and optimizing an artificial life simulator. Given original `C` code from the client, OptsRus can use pthreads to parallelize the code to exploit multicore machines, as well as perform other optimizations to improve performance.

2 Background

You will be parallelizing/optimizing Conway's "Game of Life" (GoL), a famous, simple algorithm for computing artificial life (cellular automota), that through very simple rules can generate very complex and interesting behavior. You can read more about the history and details of GoL here:

`http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`

You can also play with a GoL emulator here:

`http://www.bitstorm.org/gameoflife/`

3 Setup

Copy and extract the GoL source code from `/cad2/ece454f/hw5/hw5gol.tar.gz` into a protected directory in your UG account.

GoL should build by simply typing `make`. Input and output files are in a simple image format called .pbm. The executable `initboard` can create an arbitrary-sized input file by running:

```
initboard <num_rows> <num_cols> > <generated_file_name>
```

The executable `gol` is run as follows:

```
gol <num_iterations> <infilename> <outfilename>
```

If you want to visualize an input or output file, you can convert it to a jpg for viewing in a browser with the command:

```
convert filename.pbm filename.jpg
```

Visualizing can potentially help you think of opportunities for optimization.

4 Parallelizing and Optimizing

Your main task in this homework is to parallelize GoL using pthreads. You can use as many threads as you like, but you must use at least two (and you probably want to use four since there are four CPUs in the UG machines). Your parallelized GoL should be called or forked from `game_of_life`. You are also free to do any optimization you wish on your program, including adjusting the Makefile and/or compilation flags.

4.1 Rules

1. You may work alone or in teams of two.
2. Given some input file and some number of iterations, your program must produce the identical output file as the original unmodified GoL program.
3. Your program must be parallelized using pthreads to exploit at least two threads and improve performance by doing so.
4. The code must be your own, i.e., you cannot incorporate GoL-specific acceleration code or libraries written by others.
5. The optimizations must successfully run on the UG machines. For example, Intel Core i7 optimizations are not allowed (unfortunately).

4.2 Assumptions

- You can assume the dimensions of all input game boards to be $N \times N$, where N is a power of two.
- You can assume a maximum world size of 10000×10000. However, your program should at least exit gracefully if a larger size is attempted.
- Your code does not have to parallelize for world sizes less than 32×32.

5 Measurement

For this homework your implementation should be measured from start to finish, using `/usr/bin/time`. In other words, you will include the entire computation in your measurement, including reading and writing files, initialization, and thread creation or other overheads associated with parallelization.

We will measure your speedup only by running GoL for 10,000 iterations on the `1k.pbm` file provided. Please test your speedup with the command below which provides the "wall clock" timing of your program in seconds. If you find this execution is prohibitively long for quick optimization prototypes, feel free to experiment with smaller files, or fewer Life generations, but recall that your speedup is based on the configuration below.

```
/usr/bin/time -f "%e real" ./gol 10000 inputs/1k.pbm outputs/1k.pbm
```

6 Correctness

As stated above, for the same input, the output of your optimized/parallelized program must match the output of the original program. We will test the correctness of your code using **several input files of varying sizes and initial configurations**. We have provided one (read-only) output file `1k_verify_out.pbm`, which you can use to verify your program in the "speedup" configuration above.

Verify correctness by running

```
diff outputs/1k.pbm outputs/1k_verify_out.pbm
```

Be sure to frequently test/debug your current program state on many different inputs. Consider generating a few new input files (and the original output file) beyond those we have provided.

7 Evaluation

The total grade of this homework is **70 points**, divided into the following four components:

- *Correctness (10 points).* We may test any size/configuration under the assumptions given in Section 4.2.
- *Performance (50 points).* The speedup of your parallelized/optimized program determines the grade you are awarded. Speedup will be calculated as:

$$s = \frac{\bar{t}_{orig}}{\bar{t}_{opt}} \tag{1}$$

where $\bar{t}_{orig}$ is the average original execution time (with /usr/bin/time), and $\bar{t}_{opt}$ is the average execution time of the optimized program. Times are averaged over 5 trials.

We will determine a curve to assign performance points, where the exact curve will maximize fairness given the performance of programs submitted. However, we will award you at least 80% of performance (40/50) for a speedup of 16×. A speedup of 8× will earn at least 50% (25/50).

You will receive 0/50 if `1k.pbm` fails the correctness test or you break Rule #3 in Section 4.1.

You will receive 0 points for Correctness and Performance if you do not respect Rule #4 and #5 in Section 4.1 or your submitted source code fails to compile and run on UG machines.

- *Report (5 points).* In a brief report describe how your parallelization works and list/describe any optimizations that you did. If you added new source files, list them and briefly explain their purpose. For 2-member teams, clearly list the names of both team members.
- *Style (5 points).*
  - Your code should be decomposed into functions and *not* use global variables.
  - Your code should begin with a header comment that briefly summarizes your report, including parallelization strategy, and optimizations.
  - Each function should be preceded by a header comment that describes what the function does.
  - Each subroutine should have a header comment that describes what it does and how.

8 Submission

Note: You must add the following comment into your gol.c file, or else you and your team won't get score. (Replace XXX and 123 with your team name, your name and student ID.)

```
/*
Team: XXX
Student1: XXX
ID1: 123
Student2: XXX
ID2: 123
*/
```

Please submit your report in .pdf or text format, as well as a tarball of all source code and Makefiles needed to compile and run your GoL solution. Only one submission is required per team. Suppose your source code is in the directory `src` (you can name the directory whatever you wish). Enter `src`, remove the input/output images and directories, clean all object/binary files in the directory, and compress.

```
cd src
rm -r inputs outputs
make clean
cd ..
tar -cvvf gol.tar src/
gzip gol.tar
submitece454f 5 gol.tar.gz report.pdf
```