CS33, Spring 2020
Parallel Lab - Intro to Multi-threading

Due: Friday, June 5th at 11:59pm.

1 Getting Started

The goal of this lab is to accelerate some poorly written code we have provided. You should consider using the optimization techniques (both single threaded and multithreaded) we have covered in class to accelerate this code.

1. To start, grab the lab folder - this command will copy the lab directory to your current working directory:

   ```
   cp -r /w/class.1/cs/cs33/csbin/ParallelLab .
   ```

2. To compile the code:
   make all: This will create the Test executable.

   Before re-compiling:
   make clean: This will delete all object files.

3. Once you have successfully compiled, you can run the executable with:

   ```
   ./Test
   ```

4. Each time you modify parallel.c, you need to recompile the program using the steps detailed in 2. above.

5. Remember to add your name, UCLA ID and email details in the comment section in the parallel.c file.

6. You are allowed to modify parallel.c for this lab. You **should not edit any other files except parallel.c** as we are going to measure your parallel version against our sequential version (we may change the initial state of the input matrix so do not optimize for the specific data or functions in main.c). We will also leave utils.h and the makefile unchanged.

7. We will use **lnxsrv04.seas.ucla.edu** to evaluate your code. This machine is an 8-core system with hyperthreading (e.g. 16 virtual cores). You may develop your code elsewhere, but we will only grade based on the performance you obtain on lnxsrv04.seas.ucla.edu. The performance of your optimized code will depend on the load (what else is running) on the system. You can use programs like **top** to see the load of the system. After top starts press 1 to see the load on each core of the machine. We will run on an uncontested machine when we evaluate your code to ensure it is not impeded by other programs.
8. SEASnet servers are shared by all students in the engineering departments, including your fellow CS 33 students. When multiple students are running their programs the performance of your program may be drastically lower. To avoid testing problems due to high machine contention, START EARLY!

2 Submission

1. Submit **only parallel.c** via CCLE.
2. Remember to **add your name, UCLA ID, and email in the comment section in parallel.c**
3. The deadline for submission is **Friday, June 5th, 2020 11:59pm.**

3 Grading

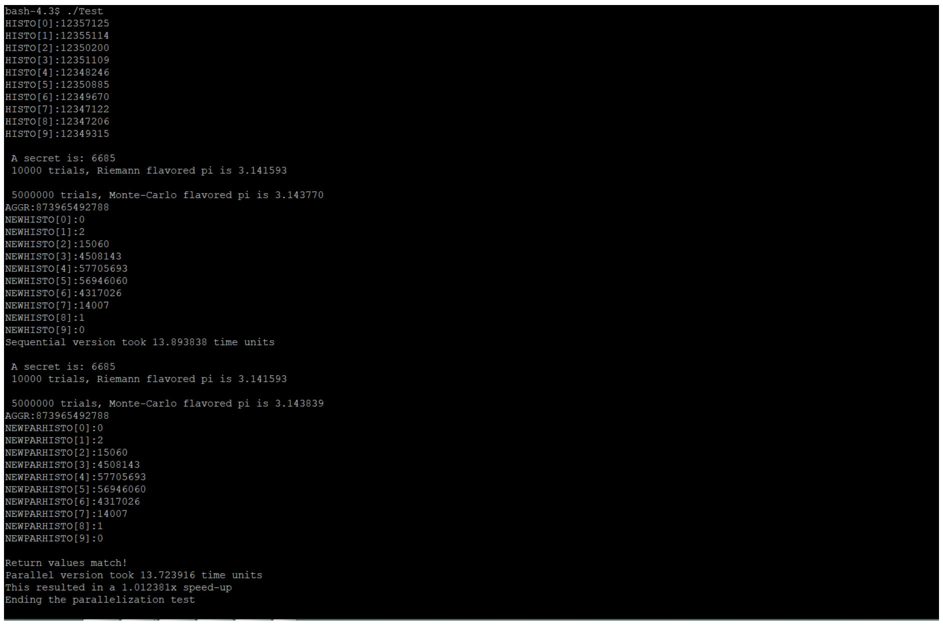
Grading will be done on lnxsrv04.seas.ucla.edu to prevent any other processes from interfering with the timing results. You will receive points proportional to the speed up. You receive 28.75 points for the first .5x speedup, then 14.25 points for each .5x speedup after that until 3.5x. Any speedup greater than 3.5x speed-up will result in extra credit. The extra credit will be awarded as half-points for any integer speed-up over 3.5x:

- $>$**3.5x** speed up: 100/100 points
- **3 - 3.49x** speed up: 85.75/100 points
- **2.5 - 2.99x** speed up: 71.5/100 points
- **2 - 2.49x** speed up: 57.25/100 points
- **1.5 - 1.99x** speed up: 43/100 points
- **1 - 1.49x** speed up: 28.75/100 points
- $< 1x$ speed up: 0/100
- The max extra credit for this lab is 15% for a speed up > 3.5. Extra credit will calculated with the following exponential decay equation: $15 * (1 - 2 * e^{-\frac{floor(your\_speedup-3.5)}{2.0}})$

- Correctness is based off the behavior of $work\_it\_par()$ matching $work\_it\_seq()$. This is determined by comparing the values printed by both functions, the values returned by both functions, and the data manipulated by both functions, e.g. the output arrays match. Float values like those printed by the Monte Carlo pi loop are acceptable within a tolerance of $\pm.005$

4 Notes

- Higher DIM values will result in longer run-time but more speedup. We will test at DIM = 500, if you run the given poorly optimized code with DIM = 500 it will take approximately 1.5 minutes to run ./Test on an empty server, and considerably longer on a crowded server. Also increasing DIM will result in a segfault as you are asking to malloc too much space on the heap.
- **DO NOT ATTEMPT TO CIRCUMVENT OPTIMIZATION BY RETURNING THE RIGHT VALUE DIRECTLY, WE WILL TEST FOR THIS, AND SCORE YOU 0.**
- **PLEASE DO NOT LET YOUR CODE ENTER AN INFINITE LOOP, THIS WILL STALL THE AUTOGRADER AND MAKE ME MANUALLY HAVE TO SCORE YOU 0.**
- I was able to get it to compile on lnxsrv03.seas.ucla.edu and lnxsrv04.seas.ucla.edu I had errors about ”multiple definitions” when I tried compiling on lnxsrv06.seas.ucla.edu and lnxsrv10.seas.ucla.edu
- **YOU WILL RECEIVE A SCORE OF 0 IF:**
  1. Your code doesn’t compile on lnxsrv04.seas.ucla.edu or lnxsrv03.seas.ucla.edu
  2. Your code does not return/output the expected values
  3. Your code enters an infinite loop
  4. Your code segfaults or otherwise produces unexpected behavior
- Correct output looks like:



```
bash-4.3$ ./Test
HISTO[0]:12357125
HISTO[1]:12355114
HISTO[2]:12350200
HISTO[3]:12351109
HISTO[4]:12348246
HISTO[5]:12350885
HISTO[6]:12349670
HISTO[7]:12347122
HISTO[8]:12347206
HISTO[9]:12349315

 A secret is: 6685
 10000 trials, Riemann flavored pi is 3.141593

 5000000 trials, Monte-Carlo flavored pi is 3.143770
AGGR:873965492788
NEWHISTO[0]:0
NEWHISTO[1]:2
NEWHISTO[2]:15060
NEWHISTO[3]:4508143
NEWHISTO[4]:57705693
NEWHISTO[5]:56946060
NEWHISTO[6]:4317026
NEWHISTO[7]:14007
NEWHISTO[8]:1
NEWHISTO[9]:0
Sequential version took 13.893838 time units

 A secret is: 6685
 10000 trials, Riemann flavored pi is 3.141593

 5000000 trials, Monte-Carlo flavored pi is 3.143839
AGGR:873965492788
NEWPARHISTO[0]:0
NEWPARHISTO[1]:2
NEWPARHISTO[2]:15060
NEWPARHISTO[3]:4508143
NEWPARHISTO[4]:57705693
NEWPARHISTO[5]:56946060
NEWPARHISTO[6]:4317026
NEWPARHISTO[7]:14007
NEWPARHISTO[8]:1
NEWPARHISTO[9]:0

Return values match!
Parallel version took 13.723916 time units
This resulted in a 1.012381x speed-up
Ending the parallelization test
```

Figure 1: ./Test Correct output reflecting correct behavior

5 Hints

- You can use the optimization techniques we have covered in class, including loop tiling, OpenMP, code motion, etc.
- It is best to test on a server without other people testing as that will affect your speedup. For example, if there is a high load during the sequential runtime of your code and a low load during the parallel runtime it will make you think you have achieved a higher speed up as your parallel code will run faster because of a smaller load. Find a server that works for you without anyone and then once you have your final results, test on lnxsrv04.seas.ucla.edu to confirm.