CSCI 2500 — Computer Organization
Lab 07 (document version 1.0) — Due 19 April 2023
Parallel Realities

- This lab is will be graded in your Wednesday lab session.
- This lab is to be completed **individually**. Do not share your code with anyone else.
- Labs are available on Mondays before your lab session. Plan to start each lab early and ask questions during office hours, in the Submitty forum, and during your lab session.

This lab will be a mild foray into the wild world of parallel computing. We will be using OpenMP and basic pragmas to parallelize summation over an array as well as compute prefix sums on that array.

1. **Checkpoint 1:** The first checkpoint is to make sure your system is setup for completing the lab. Figure out how to enable openmp compilation and compile the included test function in `lab07.c`. If necessary, you can use `bella` again for this lab (copmpile via `gcc lab07.c -fopenmp`). Your password has been reset to your RIN. See the HW4 bomb assignment documentation for login and usage. The `lab07.c` template file is in the `/home/csci2500/` directory.

   IMPORTANT: If you are using `bella`, you must keep the line `omp_set_num_threads(4);` in the code file. Using larger numbers of threads impacts shared usage for everyone. Anyone negatively affecting the ability of their classmates to use the system will be penalized.

2. **Checkpoint 2:** The second checkpoint is to compile a parallel summation of an array. Use the included file input1.txt for this. First, simply compute a summation of the array in serial to validate your output. Next, make the loop parallel with a `#pragma omp parallel for` statement above the loop.

   If you run the code immediately after including this statement, you will likely notice the output sum is now wrong. This is due to race conditions in the code, where two or more threads read the same value at a location in memory, update that same value in parallel, and then write the updated values back to memory. Since each thread has read the same value before it was updated by other threads, all updates except the last are effectively "lost".

   It is straightforward to handle this race condition in the OpenMP framework. Note that summation over an array is a form of *reduction*. We can specify that we want to perform a summation reduction on a shared variable within a parallel loop. We do that by adding `reduction(+:sum)` to the above `#pragma` we used to parallelize the loop. This essentially gives each thread its own local `sum` variable copy, which are then summed into a single shared variable at the end of the loop.

3. **Checkpoint 3:** The third checkpoint will implement parallel prefix sums. See https://en.wikipedia.org/wiki/Prefix_sum and check the serial code within the file for more detail into the basic operation. Essentially, the prefix sum of index `i` in some array is the sum of values from index `0` through index `i` in that array.

   You might note that the given serial code exhibits inherent serialization via loop-carried dependencies. There are several parallel algorithms to compute this operation. For this

checkpoint, you will be implementing one of them. One of the simpler algorithms is that of Hillis and Steele. The pseudo-code for this algorithm is given as Algorithm 1 on the wikipedia page. Alternative, you can implement Algorithm 2 if you want more of a challenge. If you want an even greater challenge, you can probably come up with your own approach.

Note: You might not notice improved performance with your parallel implementation. That is expected and contains an important lesson.

4. **Compilation and Running Details:**

   You will need to figure out how to compile with openmp enabled for your system. For `gcc` and similar compilers, you should just need the `-fopenmp` flag (note that you'll also need `-lm` to link the math library). To run, as with most of our assignemnts, you'll be directing in input from a file via redirects. You can modify the I/O for your system as necessary. A large and small test file are included for testing and running.

```
bash$ gcc -fopenmp -lm lab07.c
bash$ ./a.out < input_large.txt
Sum time: 0.001892 seconds
Parallel sum time: 0.000410 seconds
Serial sum: 193738908
Parallel sum: 193738908
Prefix sum time: 0.002882 seconds
Parallel prefix sum time: 0.057900 seconds
```

5. **Submission Details:**

   Fill in the provided template code given in `lab07.c`. You'll show your completed and functioning code to the graders of your lab session for credit.