# CS 550 Assignment 4 Writeup

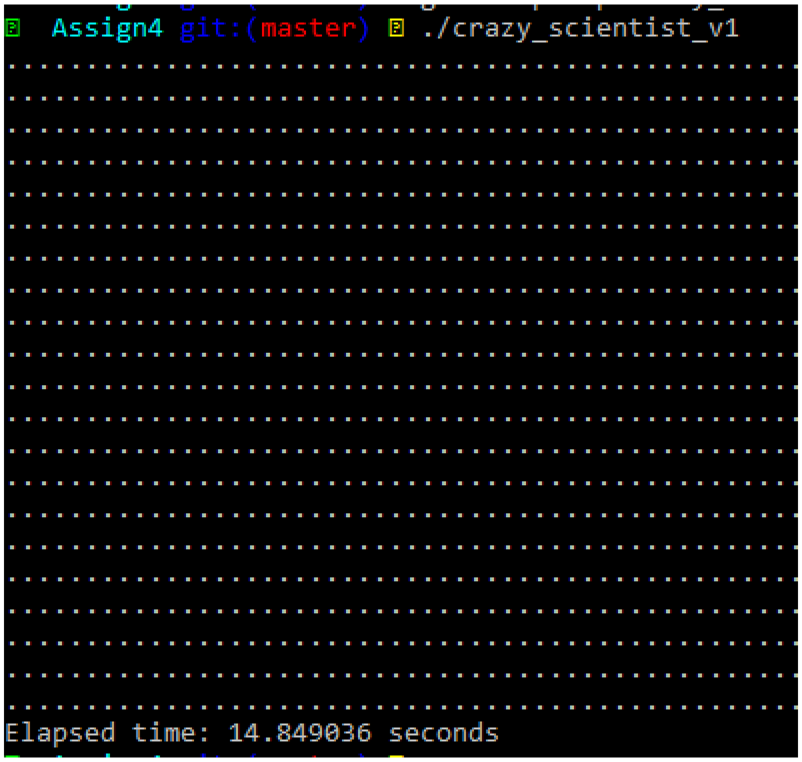
Joseph Vargovich

**Master table for timing data (values computed with a Python script I wrote):**

| Question | Total Time | Time t1 | Time t2 | Load imbalance |
|---|---|---|---|---|
| 1 | 15.090038s | NA | NA | NA |
| 2 | 14.917569s | 2.356229s | 14.915023s | 12.558793s |
| 3 | 8.850931s | 8.569474s | 8.848272s | 0.278798s |
| 4 | 8.744317s | 8.743394s | 8.460398s | 0.282996s |

**<u>Comments on Question 1:</u>**
This was completed using static scheduling, which is the default for pragma omp parallel for. I have private i and j variables since each thread needs individual indexes to compute their section of the array matrix in parallel. All threads share the matrix. The static scheduling algorithm is NOT efficient for do_crazy_computation(int x, int y) as this computation takes longer for large i and j values inputted to the function as x and y respectively.

**Sample Run for Q1:**

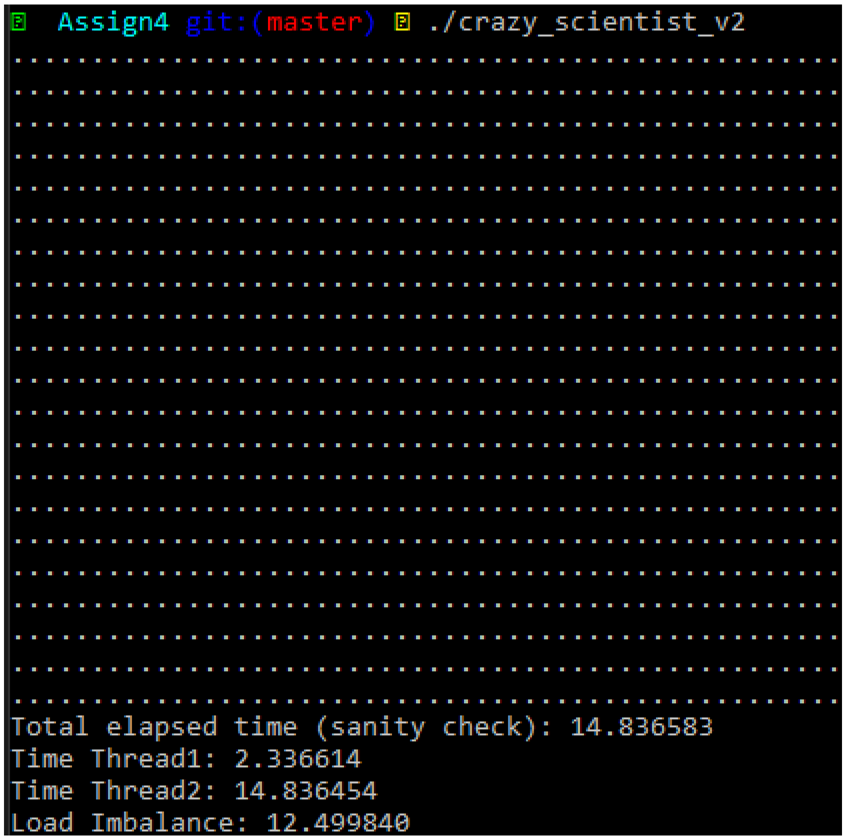


```
▯  Assign4 git:(master) ▯ ./crazy_scientist_v1
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
..................................................
Elapsed time: 14.849036 seconds
```

**__Comments on Question 2:__**
This question uses the nowait clause of omp parallel for to compute individual timings of when threads finish. Here we can see that static scheduling is poor for this algorithm because the lower indexes of i,j are much faster to compute than the higher indexes. This is seen within the high load imbalance between thread1 and thread2. Thread1 is assigned the lower indexes to compute, so it always finishes quite a bit before thread2 does. While the first thread usually took around 2 seconds to execute, the second thread takes much longer, at almost 15 seconds on average, which severely slows down the overall performance of the program, even though it is technically parallelized.

Now each thread has their own tid, tstart, tend, and telapsed, and index values. They continue to share the matrix as well.

**Sample run for Q2:**

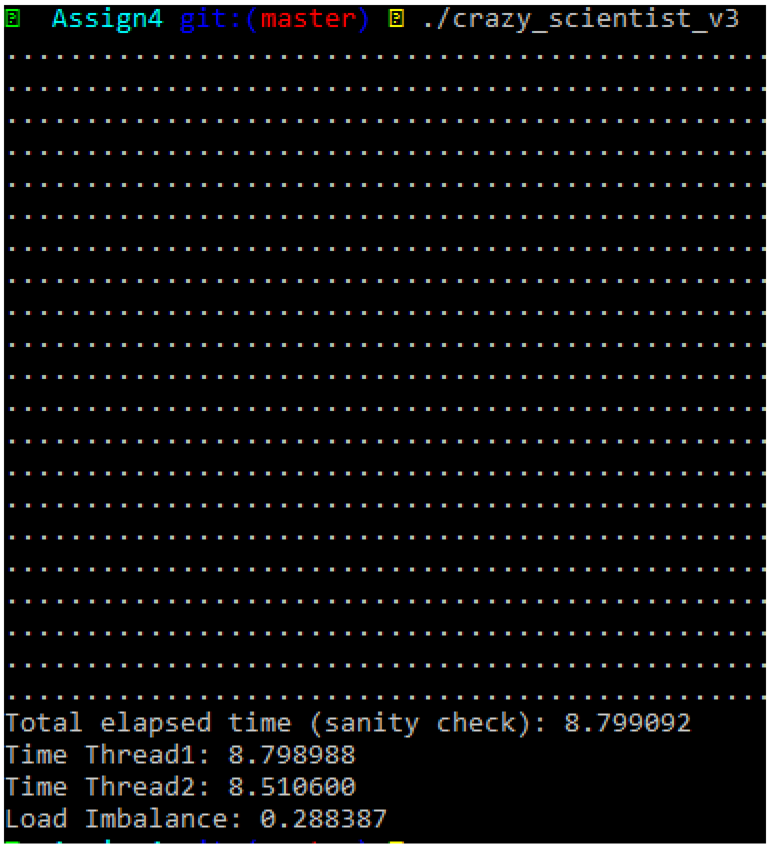


```
  Assign4 git:(master)  ./crazy_scientist_v2
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
Total elapsed time (sanity check): 14.836583
Time Thread1: 2.336614
Time Thread2: 14.836454
Load Imbalance: 12.499840
```

**<u>Comments on Question 3:</u>**
This question calls for dynamic scheduling to be added to the omp parallel for loop. This will give each thread a new row to compute on demand. Dynamic scheduling is quite applicable here as the computational complexity increases with increasing i,j values/iterations. Guided scheduling was also tested, but it performed similarly to dynamic scheduling. Both of these scheduling algorithms are far superior to static scheduling with very low load imbalances between threads. The average load imbalances and execution time for dynamic scheduling are reported in the table above.

Each thread has the same shared and private values as question 2, only the scheduling algorithm of the omp parallel for loop has changed.

**Sample run for Q3:**



```
➜  Assign4 git:(master) ✗ ./crazy_scientist_v3
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
Total elapsed time (sanity check): 8.799092
Time Thread1: 8.798988
Time Thread2: 8.510600
Load Imbalance: 0.288387
```

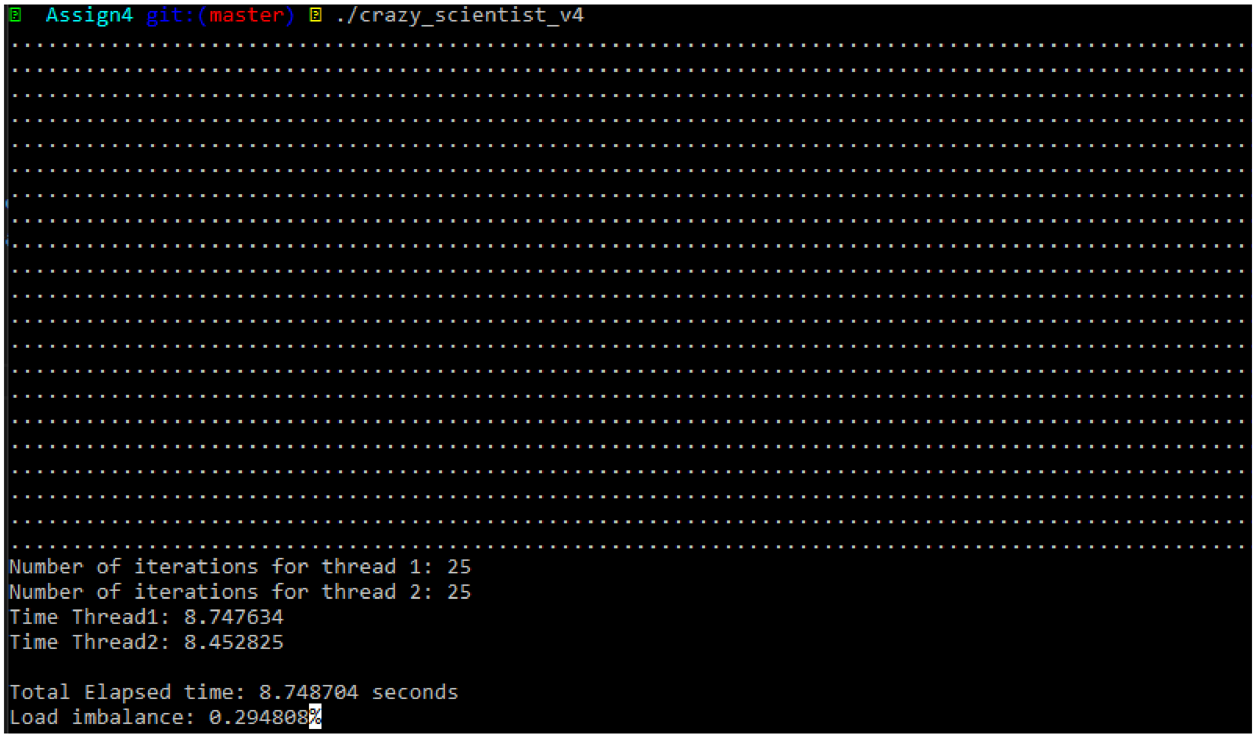
**<u>Comments on Question 4:</u>**
Here I implemented Q3 using pthreads and mocking a dynamic scheduling algorithm to give each row on demand to awaiting threads. I accomplished this with one compute_rows() function, where tid is passed in as the only parameter. Global variables are used to communicate state between threads and to send values back for use in main. These values are then used to compute the load imbalance for example.

Compute_rows() uses an atomicUpdateFunction to make sure there is still another row ready to compute. We use a lock to check the currentRow again to make sure we still have a valid row to compute once we enter the while loop. Additionally, we assign the thread its own local version of the current row to use as an array row index for its for loop. We then increment the currentRow's shared value for the other thread to read and use. We then unlock the shared mutex and compute the assigned row using the localRow index.

This implementation follows dynamic scheduling, so the load imbalance is low and each timing for it is very similar to Q3, which also follows dynamic scheduling through OMP.

However, this dynamic scheduling implementation is much faster than the static scheduling implemented within Q2 since the load imbalance is far lower with dynamic scheduling.

**Sample Run for Q4:**



```
  Assign4 git:(master)  ./crazy_scientist_v4
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
Number of iterations for thread 1: 25
Number of iterations for thread 2: 25
Time Thread1: 8.747634
Time Thread2: 8.452825

Total Elapsed time: 8.748704 seconds
Load imbalance: 0.294808%
```