

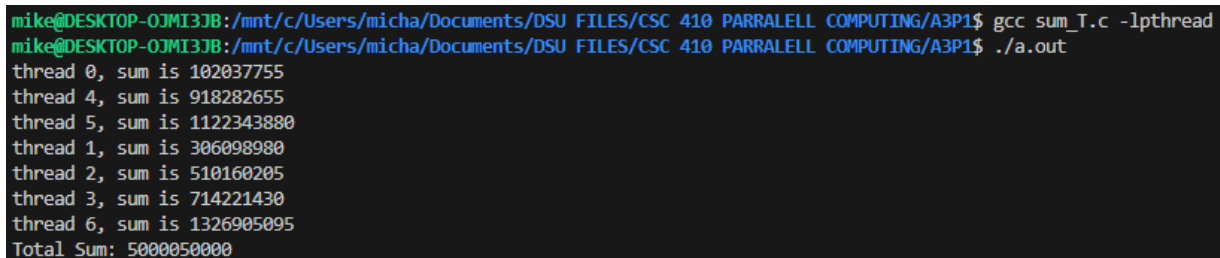
Writeup:

I did not use a synchronization technique for either program here is the justification:

For **partialSums**:

I noticed that the partial sums function was working up until around results $1e10$ in size. I added some debugging that would print each thread's values before putting it in the partial sums array and noticed it was only thread 3 that was negative, and threads zero through two were correct (I used $n=100000$ and 4 threads for my test case, starting at $n=10$ and multiplying by 10 each time). I also noticed that whenever I ran it, I got the same results, which was also odd, if it was a synchronization issue there should be a certain amount of randomness. I also had it print out the rolling sum in the array during thread 3, which were all positive, but would suddenly turn negative around the billions mark. This was my answer, I was dealing with an integer overflow issue. So I ported the rolling sum to each thread and changed it to a long and then assigned that value to the partial sums after the summation was complete. The individual thread sums were correct, but the total was still off, I then realized that I needed to change everything to longs, and now it works. So in summary, it wasn't a synchronization issue, it was an integer overflow.

Working code, 7 threads $N = 100000$



```
mike@DESKTOP-OJMI3JB:/mnt/c/Users/micha/Documents/DSU FILES/CSC 410 PARRALELL COMPUTING/A3P1$ gcc sum_T.c -lpthread
mike@DESKTOP-OJMI3JB:/mnt/c/Users/micha/Documents/DSU FILES/CSC 410 PARRALELL COMPUTING/A3P1$ ./a.out
thread 0, sum is 102037755
thread 4, sum is 918282655
thread 5, sum is 1122343880
thread 1, sum is 306098980
thread 2, sum is 510160205
thread 3, sum is 714221430
thread 6, sum is 1326905095
Total Sum: 5000050000
```

For **matrixMultiply**:

The code in here seemed to be working fine, I tested up to $N=100000$ and the results were as expected. I think we aren't running into synchronization issues because the code is writing to its own part of a shared memory so locking would not actually impact anything and only reading from the same shared memory location which is its thread number.

Screenshot of working code at $N=15$, 7 threads:

```
Matrix multiplication complete!
```

[illegible]

```
mike@DESKTOP-OJMI3JB:/mnt/c/Users/micha/Documents/DSU FILES/CSC 410 PARRALELL COMPUTING/A3P1$
```