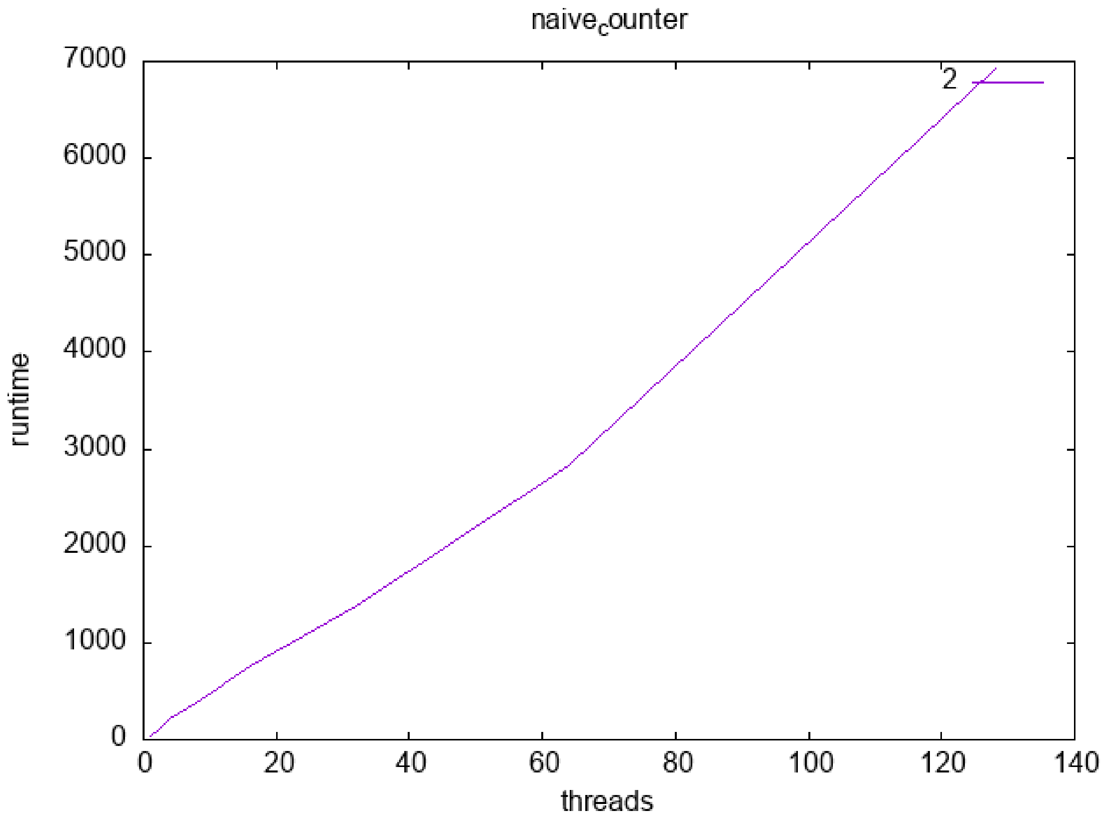
OS Project 3

Harsha Somisetty, Shivam Agrawal

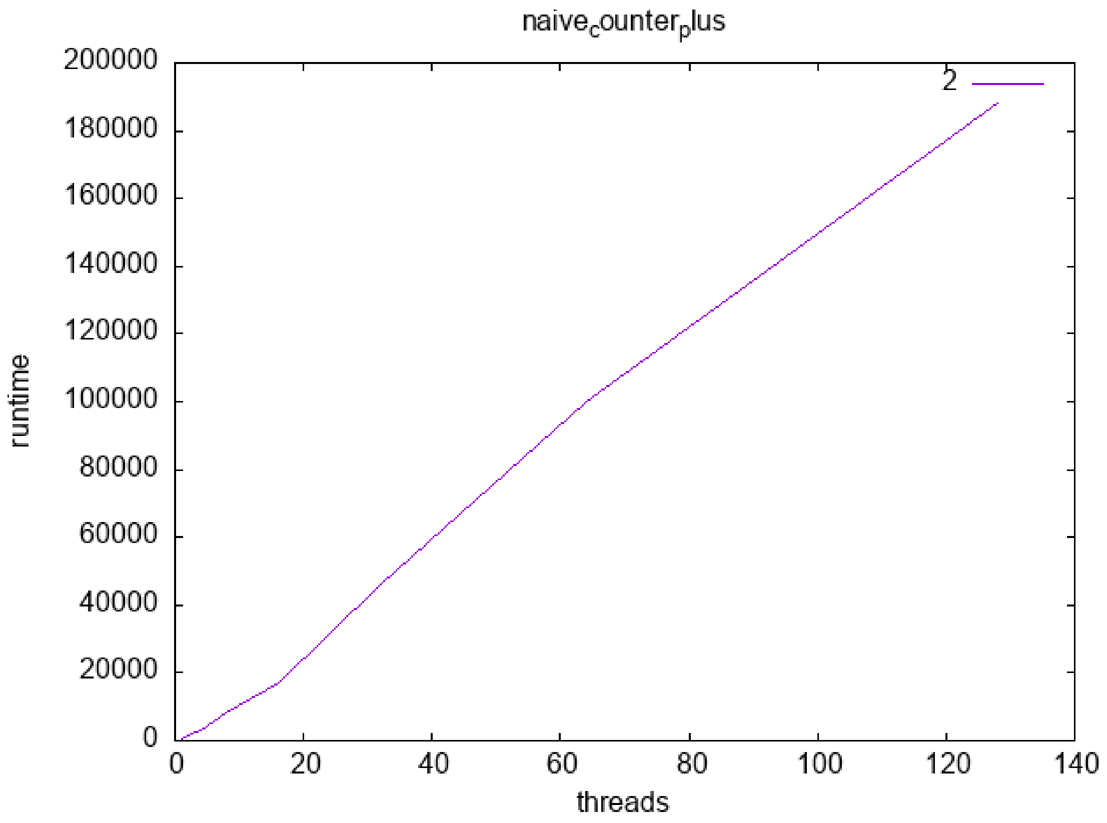
November 14, 2021

Results

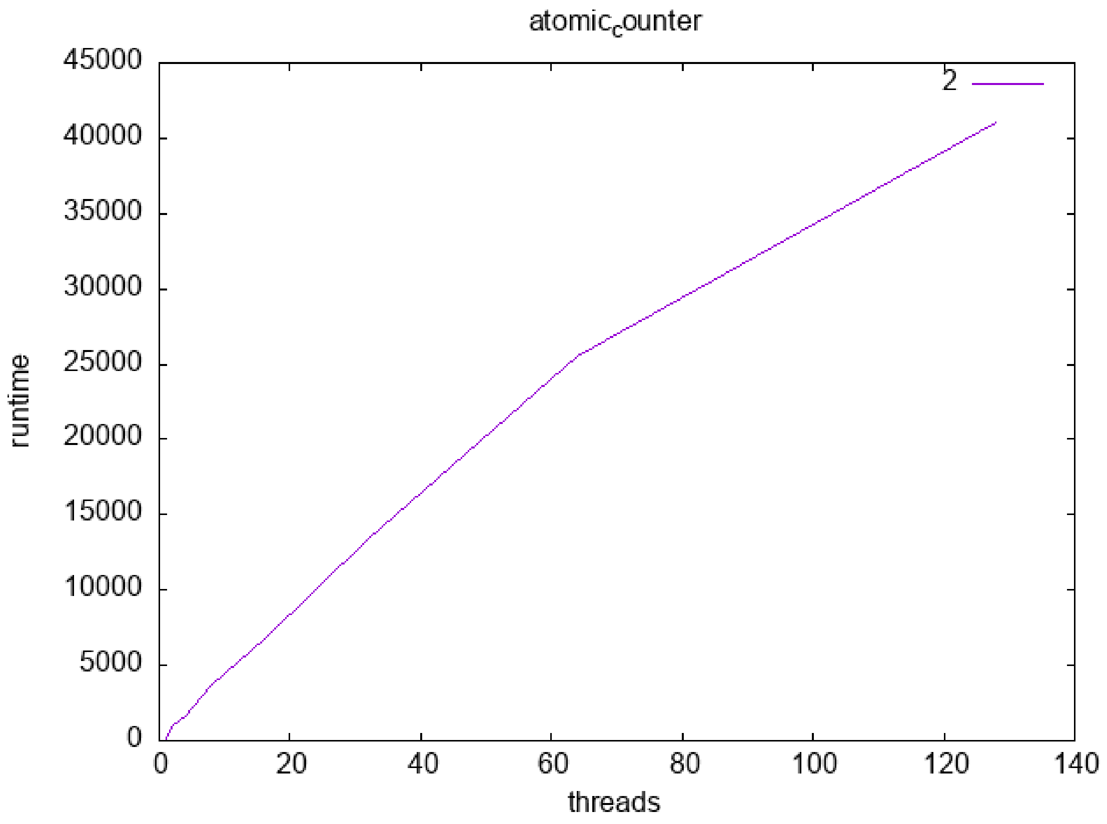
threadCount	runTime
1	29
2	93
4	224
8	378
16	753
32	1386
64	2826
128	6920



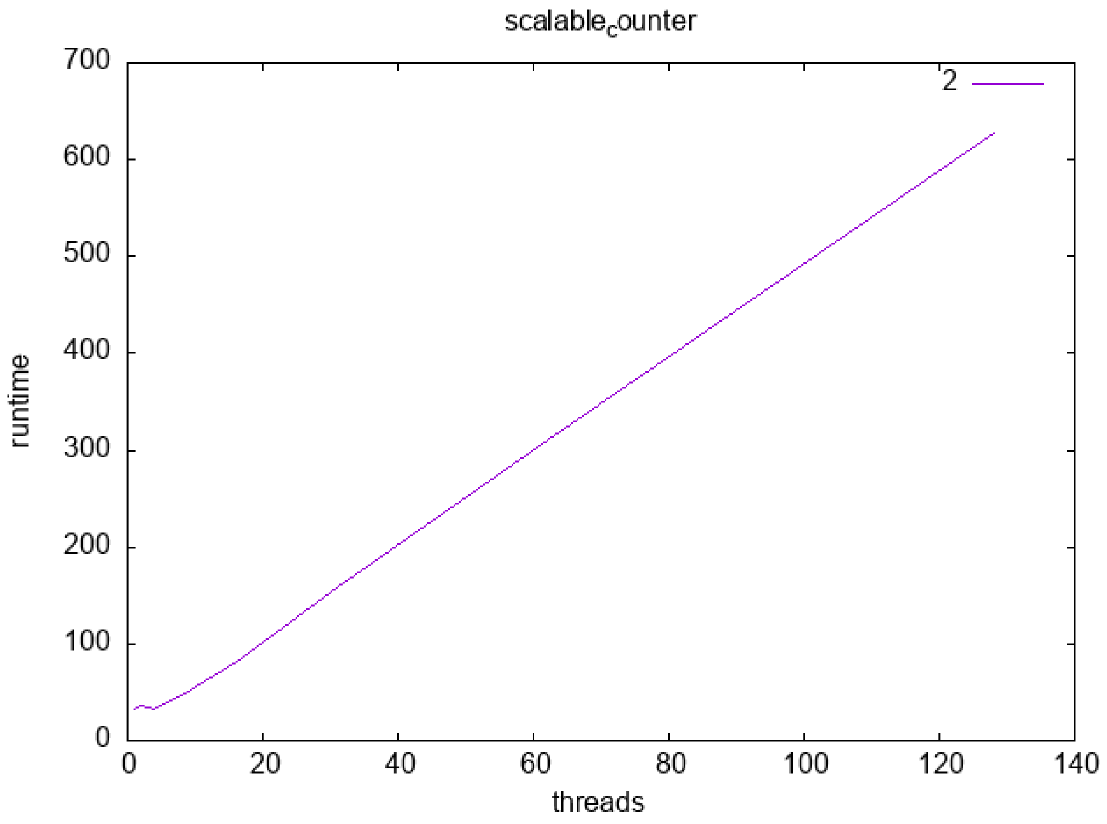
threadCount	runTime
1	310
2	1490
4	3121
8	8077
16	17213
32	46182
64	99748
128	187970



threadCount	runTime
1	121
2	882
4	1665
8	3683
16	6717
32	13371
64	25573
128	41070



threadCount	runTime
1	33
2	36
4	32
8	47
16	81
32	164
64	319
128	627



Questions

Naive Counter error

Naive counter is very off from the true value, since the addition operations are not atomic. Specifically, the naive counter requires fetching the value of the pointer, and then move the pointer value to a higher value, and then set the pointer equal to this new value. In this multistep action, multiple threads can all get the value of the pointer simultaneously, and rewrite the value to the same value multiple times, therefore, not saving progress.

For example, if 10 threads all query the pointer value at the same time, then increment the value to 11, and then save that value, the real answer should be 20, but instead only 11 is saved. This is why the Naive counter has a huge difference from the correct value

Atomic Counter vs Naive plus

The Atomic counter increments the global counter atomically and saves time, (and avoids race conditions where multiple threads act on the same data in a critical section at once). It does this by directly acting on the counter value stored in a register, during which the register is blocked off to the other threads.

On the other hand, the Naive Plus Algo wastes a lot of time waiting to make sure that only one thread is in the critical section, retrieve the data, then release a lock. This locking and unlocking to get into the critical section wastes a lot of time, compared to the hardware solution of Atomic

Atomic Counter vs Naive Counter

With the atomic implementation, there is still a limit on the speed of incrementing the register. Specifically, the register can still only be accessed one at a time, albiet with very little overhead. The benefit is of course an accurate count, but still slow in time.

Naive on the otherhand has many threads running at once, and since a lock isnt considered, that limit of *atomically* accessing the critical section data is non-existant, allowing for Naive's threads to continue together, and finish the entire counter process multiple times faster.