Edward Alkire - 1001722331 CSE 3320 – 003 March 24, 2023

MT Star Catalog Report

Summary:

In this report, we explore the performance characteristics of a largely CPU bound workload/application when multi-threade. The workload in question is computation of the average, min, and max angular distance of a 30,000 count star catalog. The particular subject of anomalies explored is "overthreading" or the allocation of more threads to a multithreaded process than it has access to in hardware. In the following data you will find results of testing on a "GitHub Codespaces" VM, as well as a Mac M1 personal computer for comparison.

Timing method:

To measure time, timespec with clock_gettime() was used. Timespec utilizes the OS time/clock. This as opposed to clock_t variables with clock(), which measure time via CPU time, which multiplies the time measured in multithreaded contexts — meaning that the real time passed would no longer be the subject of measurement. Alternatively, timeval could have been used, as it operates similarly to timespec.

Steps to run:

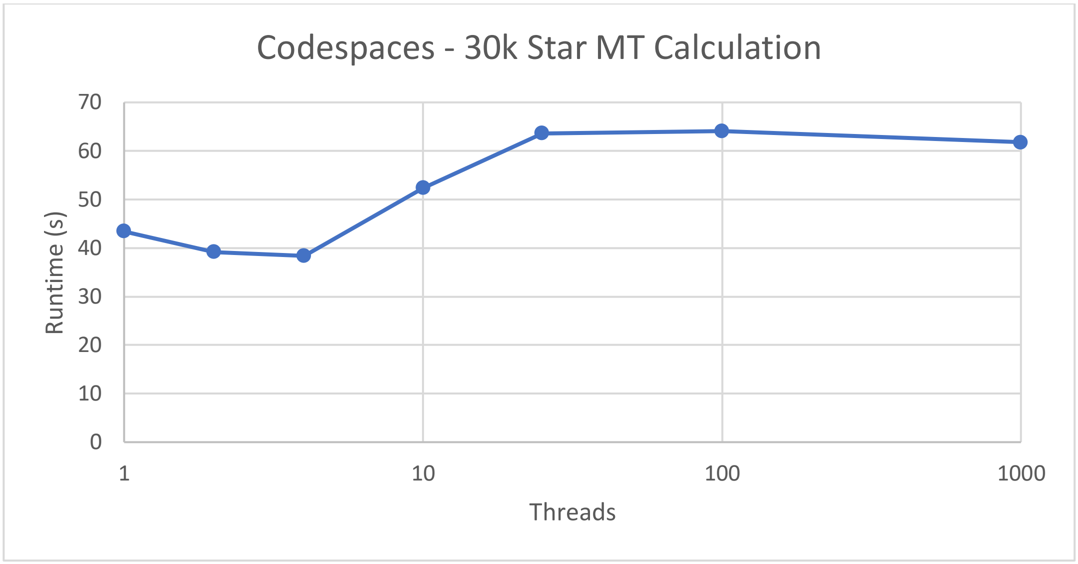
Compile with: "gcc src/main.c src/utility.c -lpthread -lm -fPIC -mcmodel=large"

then run "./a.out -t <threads number>"

Results:

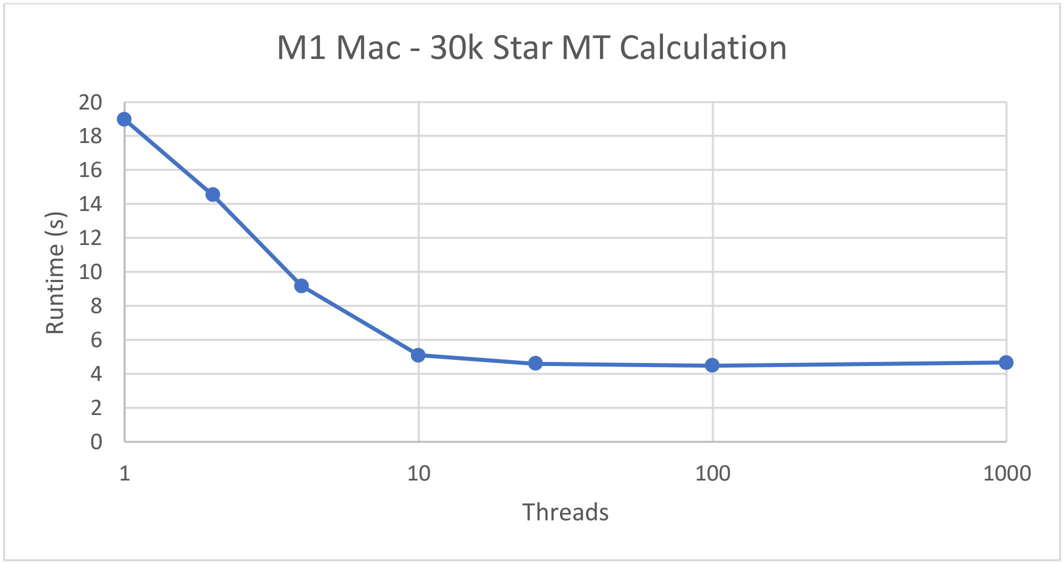
Codespaces (2c/4t) + 4GB RAM

Threads:	Time(s):
1	43.409986
2	39.197598
4	38.376815
10	52.338593
25	63.640637
100	64.090935
1000	61.742305



M1 Mac (8c/8t) + 8GB RAM

Threads:	Time (s)
1	18.976292
2	14.541522
4	9.167735
10	5.095433
25	4.612355
100	4.479866
1000	4.65788



Interpretation:

In the Codespaces VM the calculation speed was best using four threads. Perhaps uncoincidentally that is also the number of threads allocated to it. Beyond the use of four threads the time taken was worse than when single threaded. This suggests that once the thread count was sufficient to fully utilize the hardware, the overhead of increased memory usage and cycles to create threads outweighed their benefit.

Interestingly, the performance on an M1 Mac tells a different story. The best performance was found closer to 100 threads, well beyond the 8 cores/threads the program has access to in hardware.

The drastic performance difference between the machines is difficult to surmise given the many factors that could be involved. The poor performance of the VM when overthreaded probably has to do with the overhead of virtualization. Perhaps put simply it doesn't have 2c/4t to itself, only those it's allowed to use — thus blocking/switching overhead between users of the VM.

The exceptional performance of the M1 may have to do with the bottleneck at process creation and joining (lines 97-106 and 109-117 in runThreads()). The smaller the workload in each thread, the lower the potential variance between threads, where the join loop is waiting for all threads to finish to continue. This especially since process joining is done in order of thread ID, not just the order they finish. If this theory were the case, it would mean the overhead cost of threads 9 through 100 was less than the variance expected by that extra overhead itself + that from competing processes interruptions — which when put like that seems more than plausible.

In conclusion, process competition could reasonably explain why the abnormalities in multi-threading performance, where the Codespaces VM exceptionally suffers from overthreading, and where a PC benefits from overthreading. Overthreading is likely a net cost in a high competition/low efficiency VM environment, but a net positive in low competition/high efficiency environments such as a PC.