Ong Teck Wu
1001539
Lab 2

Files
meanThreadCode.c
medianThreadCode.c

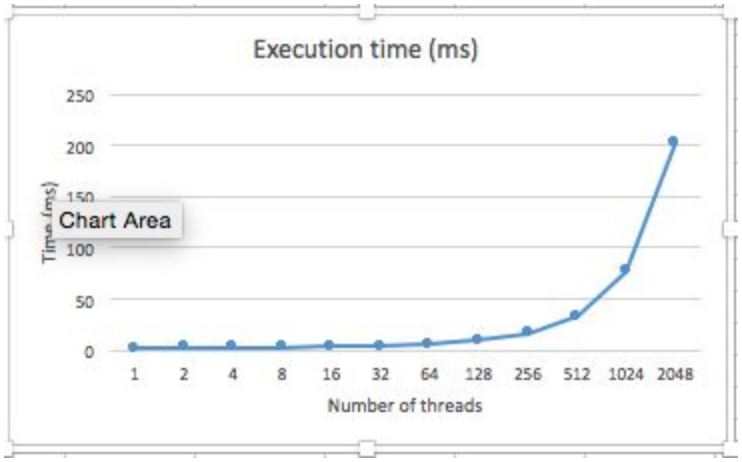
How to run code
gcc -lpthreads meanThreadCode -o meanThreadCode.c
./meanThreadCode input.txt 1 2 4 8 16 32 64 128 256 512 1024 2048

gcc -lpthreads medianThreadCode -o medianThreadCode.c
./medianThreadCode input.txt 1 2 4 8 16 32 64 128 256 512 1024 2048

Set up
I performed the analysis by running the algorithm on each number of threads 5 times and getting the average run time.

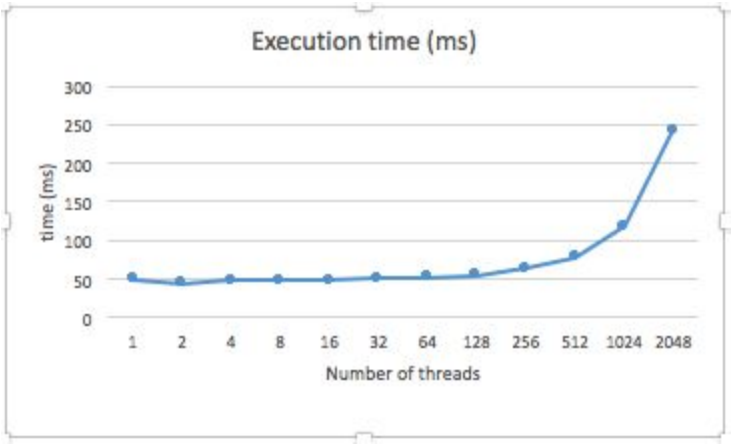
Mean

Number of threads	1	2	4	8	16	32	64	128	256	512	1024	2048
Execution time (ms)	2.13	2.18	2.72	2.85	3.25	4.05	5.49	9.28	16.75	33.17	76.77	202.37



Median

Number of threads	1	2	4	8	16	32	64	128	256	512	1024	2048
Execution time (ms)	49.39	44.27	48	47.9	48.29	50.42	52.47	54.95	64.15	78.32	118.38	242.83



Observations

Mean: The execution time increases steadily but negligibly from 1-16 threads but starts to increase exponentially from 16-2048.

Median: The execution time increases steadily but negligibly from 1-128 threads but starts to increase exponentially from 128-2048. For median the minima is at 2 and the only after 16 did the runtime exceed that of 1. However, that is also negligible.

Analysis

For both mean and median, the runtime increases exponentially as a result of
(1) the overhead from context switching,
(2) having only two processors i.e. max two threads running at the same time,
(3) and having to wait for the last worker thread to end before the next line of computation can start.

For such map-reduce computation operations, running more threads would only be ideal if there are more processors and an optimized maximum time-slice for processes to run in (to reduce redundant context switching overhead)

Problems I faced

For the “merge” function in medianThreadCode, due to the stack size having a limit of 2^16 bytes, I had to use malloc to create temporary arrays for the naive implementation of the merge algorithm.