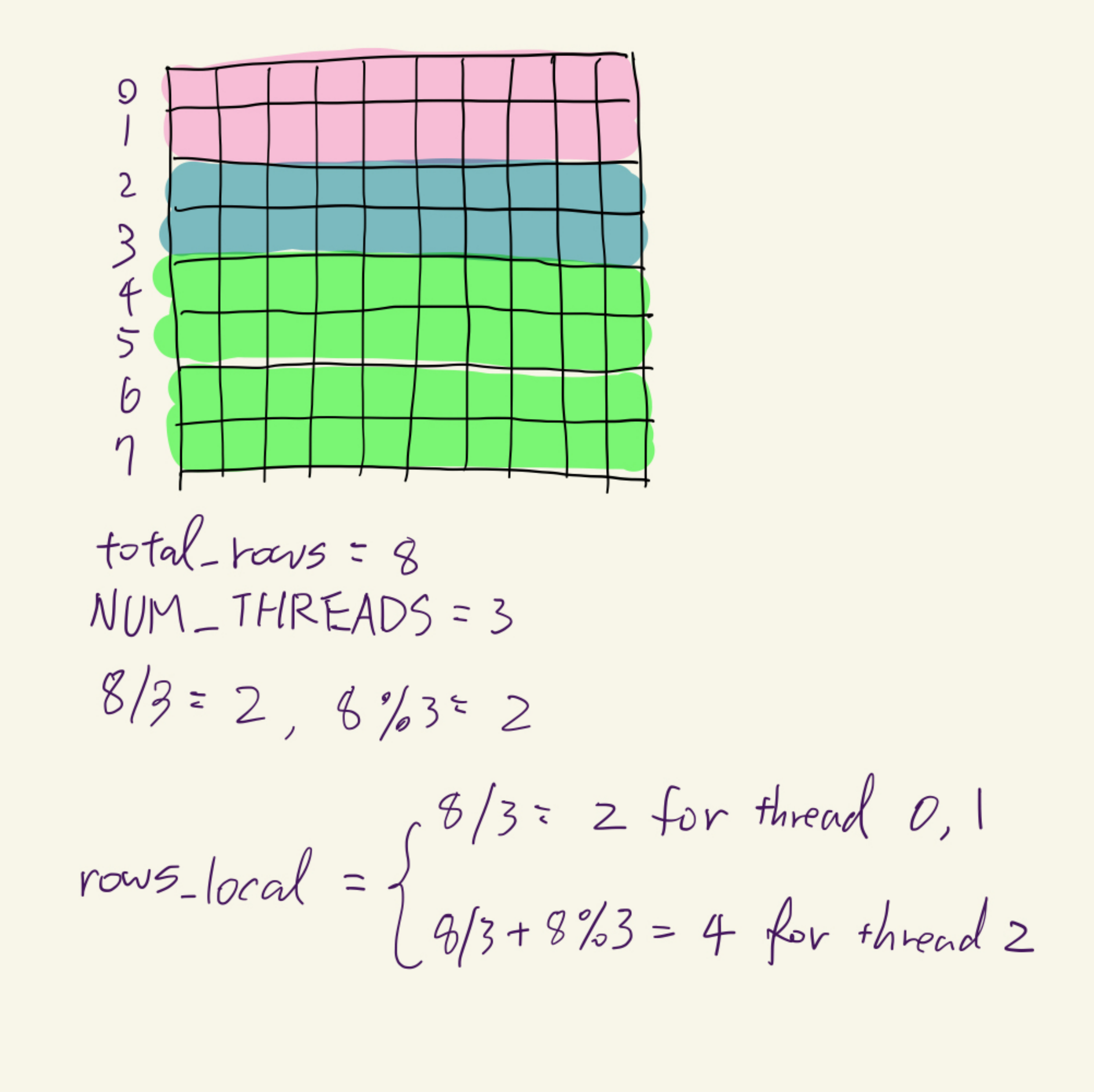
Date 2022.11.25

REPORT FOR OS HW2

1. HOW I DISPATCH THE WORK

I use 2-D block cyclic data distribution to dispatch the work, and the last threads also does the remaining rows.

Here's an example



total_rows = 8

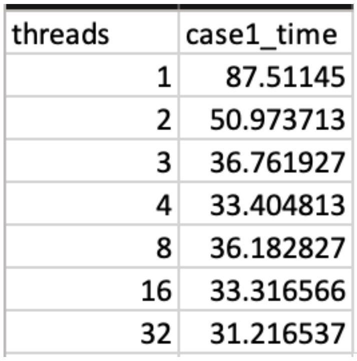
NUM_THREADS = 3

$8/3 = 2$, $8 \% 3 = 2$

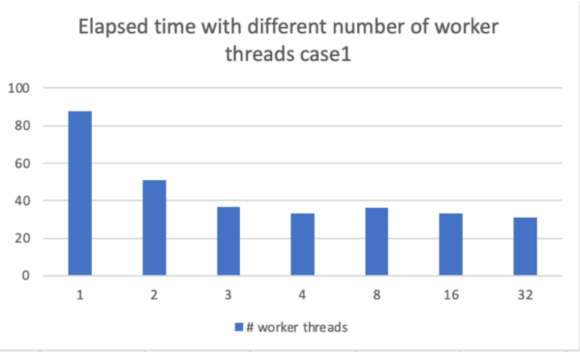
$$\text{rows\_local} = \begin{cases} 8/3 = 2 \text{ for thread } 0, 1 \\ 8/3 + 8\%3 = 4 \text{ for thread } 2 \end{cases}$$

2. TEST CASES

i.

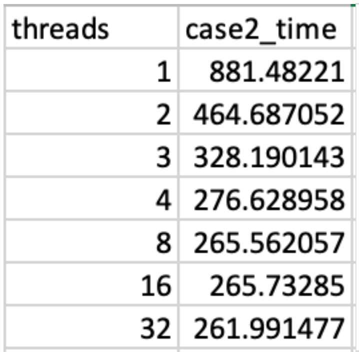


threads	case1_time
1	87.51145
2	50.973713
3	36.761927
4	33.404813
8	36.182827
16	33.316566
32	31.216537

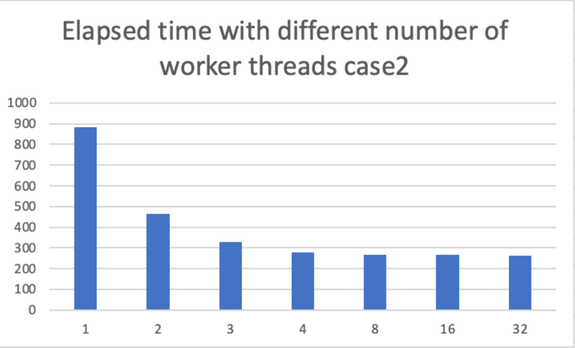


This is the expected outcome. Since the program is run on a 4-core cpu, in cases with over 4 threads, the extra threads would need to be scheduled to run on the 4 cores, which doesn't aid efficiency.

ii.

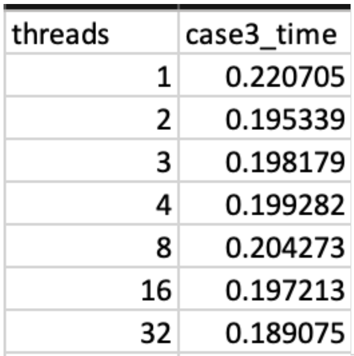


threads	case2_time
1	881.48221
2	464.687052
3	328.190143
4	276.628958
8	265.562057
16	265.73285
32	261.991477

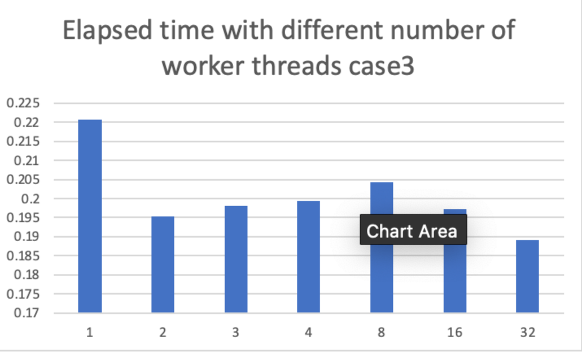


This is a similar result to case i. , just that the execution time is a lot longer due to much larger matrices.

iii.

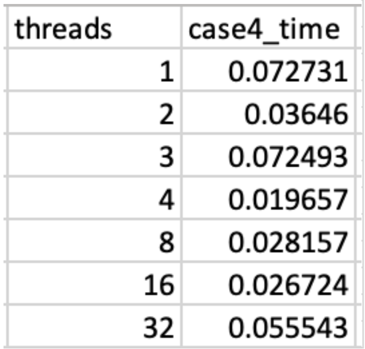


threads	case3_time
1	0.220705
2	0.195339
3	0.198179
4	0.199282
8	0.204273
16	0.197213
32	0.189075

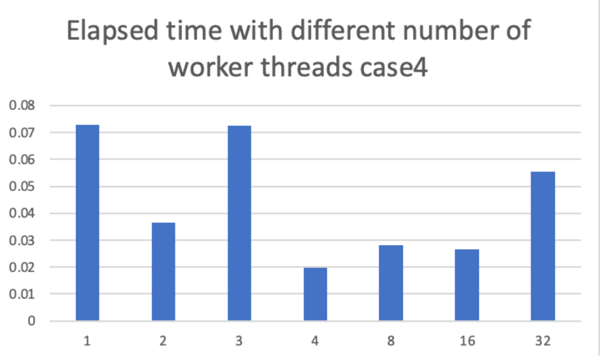


There are only one row in this case, so the time is all very close due to only the last thread doing the work.

iv.



threads	case4_time
1	0.072731
2	0.03646
3	0.072493
4	0.019657
8	0.028157
16	0.026724
32	0.055543



The times are all quite close due to m2.txt being small (4096*1).