1. Objective:
   - The programs each read a matrix from an input file (‘inp.txt’), and calculate the square of the matrix by distributing the workload among multiple threads.
   - It also sets up CPU affinity for each thread to optimize efficiency
   - The result is written to an output file , including the time taken for the computation.

2. Key Components:
   - ‘multiplyMatrices’: Function to perform matrix multiplication given certain constraints.
   - ‘setAffinity’: Sets processor affinity for each thread
   - ‘main’:
     - Reads input matrix from file.
     - Splits the work among `K` threads.
     - Measures the time taken for the computation.
     - Writes the result and time to an output file.
3. Thread Usage:

   Threads are set up exactly like the previous assignment, with BT number of them being bound to specific cores, the value of BT being decided based on the experiment

4. File Handling:

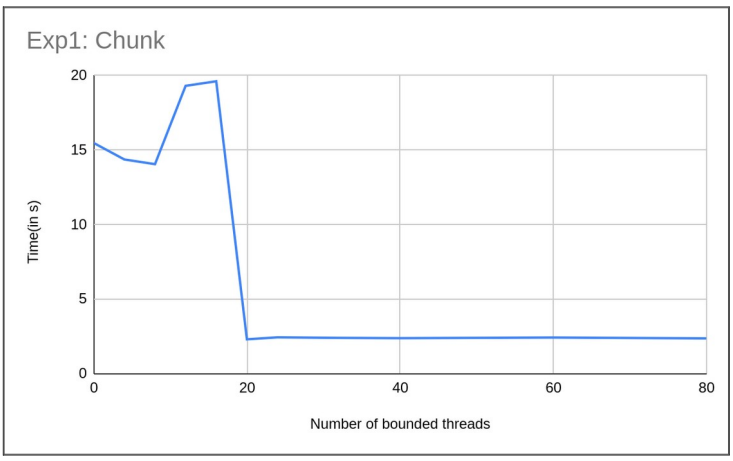
   The program uses file I/O to read the input matrix from `inp.txt` and write the result to the respective output file.

5. Data Analysis:

<u>Experiment 1</u>

{Note: Instead of a separate curve for the unbound case, it is simply plotted along with the other values with b=0. Hence, this experiment has only 2 curves instead of the suggested 4}

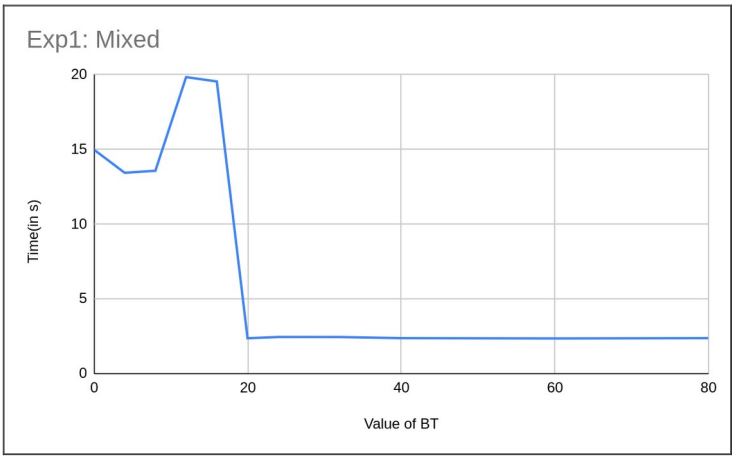
1) Chunk



K=64	Exp1	Exp2	Exp3	Exp4	Exp5	Average
0	18492	14247	14472	15243	14879	15466.6
4	14292	14059	14540	14283	14625	14359.8
8	14881	13787	13940	13883	13726	14043.4
12	19370	19239	19187	19761	18927	19296.8
16	19801	19520	19350	20120	19230	19604.2
20	2330	2288	2279	2301	2310	2301.6
24	2382	2434	2270	2898	2173	2431.4
32	2344	2734	2273	2174	2457	2396.4
40	2430	2361	2261	2319	2518	2377.8
60	2370	2172	2261	2935	2362	2420
80	2412	2367	2819	2111	2103	2362.4

(Time in ms)

2) Mixed



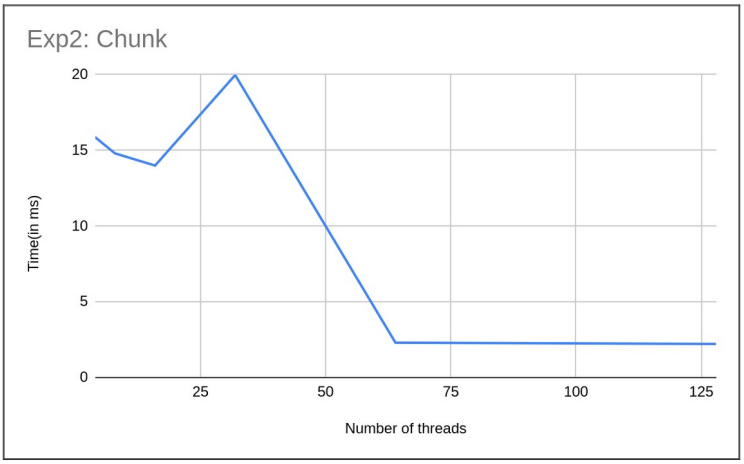
K=64	Exp1	Exp2	Exp3	Exp4	Exp5	Average
0	15803	13825	16783	14527	13897	14967
4	13341	13570	13189	13217	13768	13417
8	14300	13477	13543	13264	13189	13554.6
12	20125	19258	20256	19652	19807	19819.6
16	19776	19530	20182	19038	19126	19530.4
20	2437	2320	2272	2310	2392	2346.2
24	2359	2215	2463	2209	2361	2431.4
32	2310	2322	2456	2681	2371	2428
40	2365	2712	2173	2210	2315	2355
60	2276	2387	2452	2183	2371	2333.8
80	2422	2819	2178	2156	2200	2355

(Time in ms)

Observations:
First off, my laptop had 20 cores, which affected the way I thought the results would look like. For the values of BT ranging from 0-20, performance didn't improve, but rather became worse (With the exception of going from 0 to 4). I suspect this is because binding initially gives an improved performance due to better cache usage but till it hits 20, the os scheduled threads are not split as effectively and thus, run time increases. However, the moment we hit 20 bound threads, everything evens out and the os now splits all unbound threads amongst all 20 cores better. The fact that we hit an almost constant number indicates that the system is running in an optimum state and it is unlikely that performance would improve beyond this.

Experiment 2

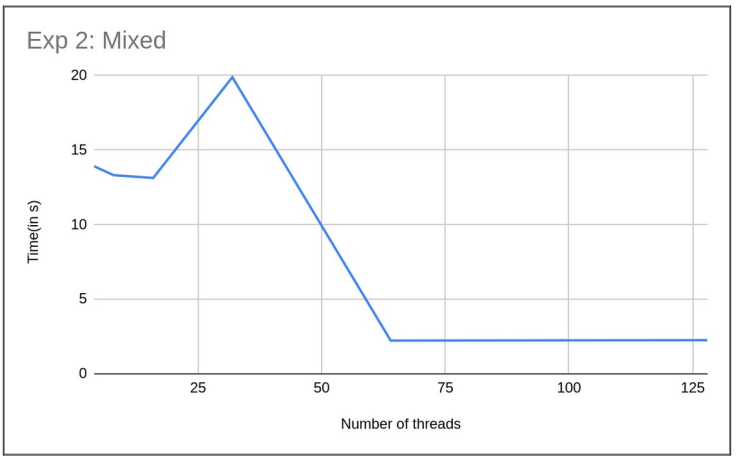
1) Chunk (Bound)



	Exp1	Exp2	Exp3	Exp4	Exp5	Average
4	16750	15467	16321	15325	15437	15860
8	14990	14576	14352	14897	15104	14783.8
16	14011	14072	13976	14076	13765	13980
32	20723	19205	19756	19943	20143	19954
64	2276	2219	2416	2317	2198	2285.2
128	2237	2145	2316	2187	2111	2199.2

(Time in ms)

2) Mixed (Bound)



	Exp1	Exp2	Exp3	Exp4	Exp5	Average
4	13778	13178	14356	14256	13987	13911
8	13259	13454	13123	13254	13442	13306.4
16	13408	13467	13245	12346	13123	13117.8
32	19671	20123	19762	20463	19356	19875
64	2336	2113	2218	2196	2200	2212.6
128	2253	2185	2233	2412	2109	2238.4

(Time in ms)

3) Chunk (Unbound) and 4) Mixed (Unbound)
Refer pages 5 and 6 of the other attached report

Observations: Very similar to experiment 1, when BT exceeds 20, ie at K=64, that is when performance for the bounded cases improves since every core has a bounded thread and the OS now effectively distributes the rest.

While the unbound case initially performs much better, once ebery core has a thread bound to it, we can see that the bound case has a better asymptotic limit of around 2.2 seconds as compared to the unbound case's 2.5 seconds. This shows that binding threads in the long run offers moderate performance improvement, which we expected.