REPORT

Distribution of tasks to threads :

When K threads are given the tasks to them are almost divided equally using the n/k factor (where n is the size of the matrix and k is the number of threads).

Details of implementation :

- A function that calculates a dot product of a row and column when given their numbers is written (*cellInResMatrix*) which writes the value calculated to the resultant matrix at the correct position.
- In the program I have a function (*tFunction*) which is the function implemented by threads and also I have created a structure (*threadAtt →*
  *char *method;*
  int i; //index of thread
  int n;
  int k;
  *int **inpMatrix;*
  *int **resMatrix;*) that contains details of threads based on which they perform the function. This can work for both mixed and chunk as they were written with if and else.
- The *threadAtt* contains input and resultant matrix as shown above.
- In this function we decide the rows and columns which need to be multiplied using the *cellInResMatrix* function based on the method which is nothing but the task division.
- If the method is **chunk**,then adjacent n/k or n/k+1 rows are given and all columns are considered for multiplication with each row.This i.e n/k or n/k+1 is decided based on thread index if the

index is <n%k it gets an extra row else only n/k. One can also perform chunk by just adding the n%k rows to the last thread.

- If the method is **mixed**,then approximately n/k rows are given based on modulo value i.e ex: (1,k+1,2k+1,..) and all columns are considered for multiplication with each row.
- The threads directly write to the resultant matrix directly. This matrix is then read again and written to the output file.
- The function chrono is used to calculate the time in microseconds (this is scaled to milliseconds for plotting graphs).
- For calculating the average time taken by each thread, I created a global vector and resized it to k in main before creation of the threads. Then based on their thread number they access the vector using it as an index and store the time taken (start and end time are calculated in function).

Distribution of threads to cores :

- In experiment 1, the threads are divided into sets of b each. Value of b is calculated using K/C which is the least nearest integral value based on which number of cores to which BT threads are restricted (**resCores**) is calculated based on the relation BT/b. When K<C all K threads are given to the same core (core 0).
- In experiment 2, *resCores* are fixed to C/2 (half of the cores).
- In both the experiments if BT=0, then this *resCores* are set to 0 as there is no restriction on the threads towards the core.

- As the system has SMP (symmetric multiprocessors - all are equivalent) the first resCores (0,1,2,....resCores-1) are alloted the bounded threads. I created an array of attributes setting affinity to each of the cores mentioned above.

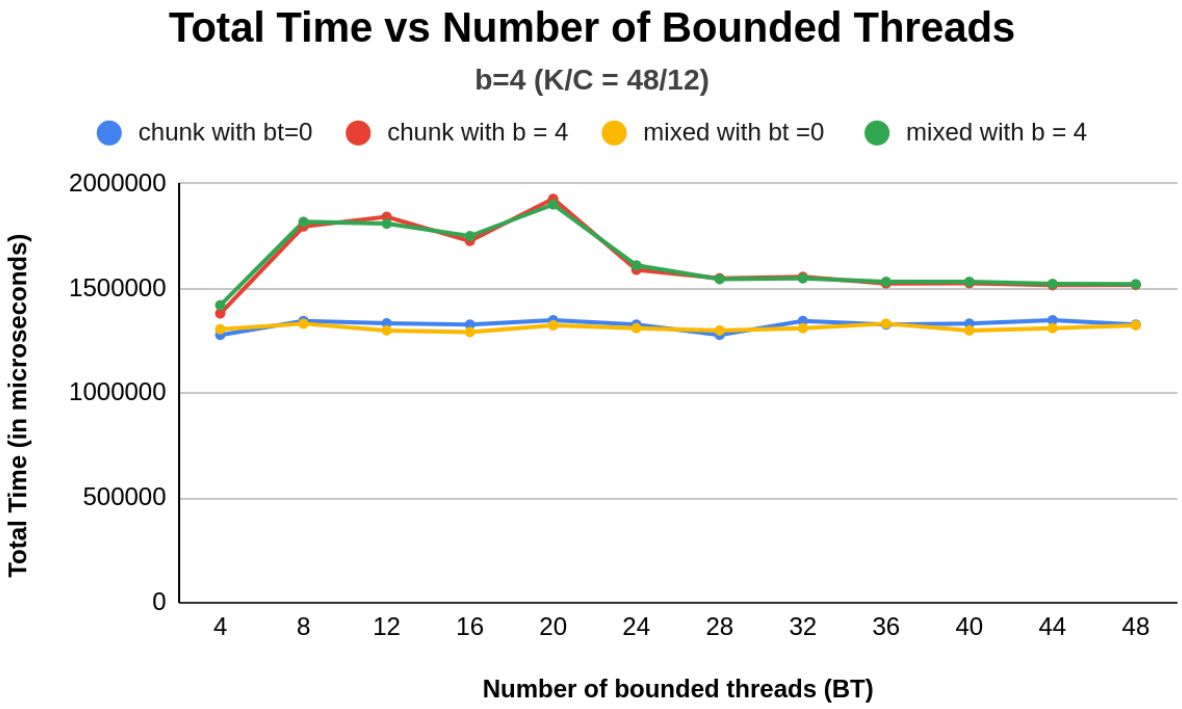
- The CPUs are decided based on modulus (thread number modulus resCores) so that the threads are uniformly distributed among the cores.
- To set affinity **pthread_attr_setaffinity_np**(&attr,sizeof(cpu_set_t), &cpus) function is used where the parameters of function are attr,sizeof(cpu_set_t),&cpus where attr is the attribute given to the thread during pthread_create (as the2nd parameter),cpu set contains the CPU under restriction. This helps to set affinity to the threads before creation.
- It is first initialised using **pthread_attr_init**(&attr) and then affinity is added using the above function.

Modifications done while considering the value of K value :

- For experiment 1, as the cores in my computer are 12, k was taken 48 (a multiple of 12).to get an integral b value (4 here).
- Even for experiment 2 the values that were taken as 12*2^i ,i from -1 to 3 taking into account that the number of cores are 12 in my computer.

Experiment 1

BT	chunk with bt=0	chunk with b = 4	mixed with bt =0	mixed with b = 4
4	1277965	1380143	1304778	1419454
8	1344519	1795360	1331500	1817399
12	1333592	1841371	1299284	1808856
16	1327239	1725914	1291751	1749829
20	1348144	1927256	1323883	1900355
24	1327239	1588581	1310239	1609810
28	1277965	1548397	1299284	1543978
32	1344519	1554972	1310239	1547878
36	1327239	1523439	1331500	1531632
40	1331975	1524460	1299284	1531273
44	1348144	1515100	1310239	1521724
48	1327239	1516466	1323883	1520044



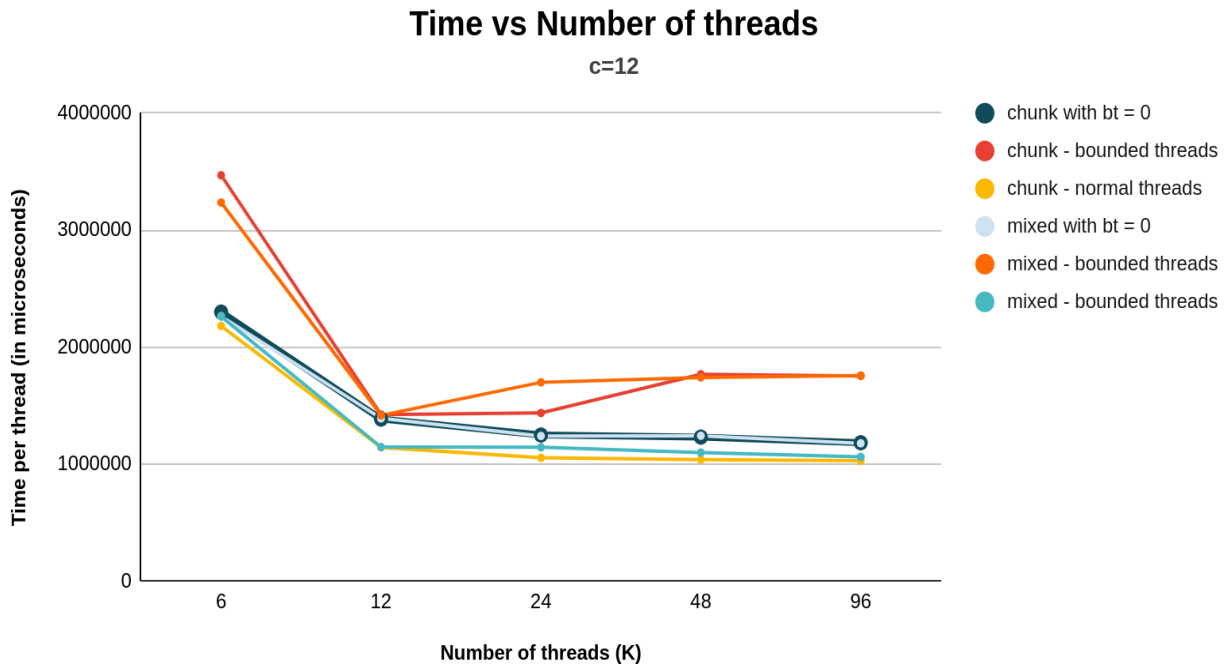
Observations:

- As introducing bounded threads adds to overheads of dividing them to respective cores. Fixing a thread to core also removes the chance of load balancing due to which the time taken to compute it could be higher than expected. Although there may be advantage of cache when affinity is set, here the cache is populated with matrix entries consistently as the size of matrix is small and also the other threads when run on the same core populates the whole matrix as they would need to work on all columns for generating a row in square matrix. Thus the benefit of cache is not much seen as it would be there without bounded threads. Hence due to the overheads the time has increased and it is reaching a constant time as the thread number increases as further optimization can’t be done . Also the effects of task division with greater concurrency and creation of threads balance out each other after a certain stage.

- Similar to the observation in the previous experiment the values of mixed and chunk do not vary much even with bounded and no bounded threads.

Experiment 2

k	chunk with bt = 0	chunk - bounded threads	chunk - normal threads	mixed with bt = 0	mixed - bounded threads	mixed - normal threads
6	2300343	3469575	2181747	2265006	3236667	2262147
12	1385724	1424765	1143801	1391970	1417702	1146928
24	1249579	1437878	1055075	1241421	1699774	1144939
48	1231810	1768993	1038840	1243240	1741750	1099483
96	1183495	1754569	1029034	1178768	1757788	1062435



Observations:

- The time taken by bounded threads is greater than the no threads under bound is greater than the normal threads and threads to which affinity is not set. This could be due to the overhead of setting affinity for bounded threads and avoiding load balance so that the normal threads are flexible in scheduling

with other processes and can be executed in any core as scheduled by OS. The time difference between normal and unbounded threads could be due to the fact that the OS now schedules k/2 of them and k of them respectively so these k/2 could have scheduled more parallelly.

- The observation that can be seen is that the average time taken by a thread where there are no threads bounded in the corresponding process is in between the average time taken by a thread in a bounded and unbounded state in the process that has both of them. This could be due to the fact that the CPU scheduler balances out the workload to get an optimal time.
- Additionally, similar to the previous graph, the mixed and chunk threads exhibit similar times across all categories (no bounded, bounded, normal), suggesting consistency in performance regardless of thread configuration.
- Time taken is high for the threads less than or equal to 12 (12 is the number of cores in my system).For more number of threads there is a small chance for fully utilising concurrency and for a small number of threads the workload for each thread is high. Therefore at 12 where maximum concurrency of threads can occur and better distribution of tasks is possible there is a local minima in each graph.