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:

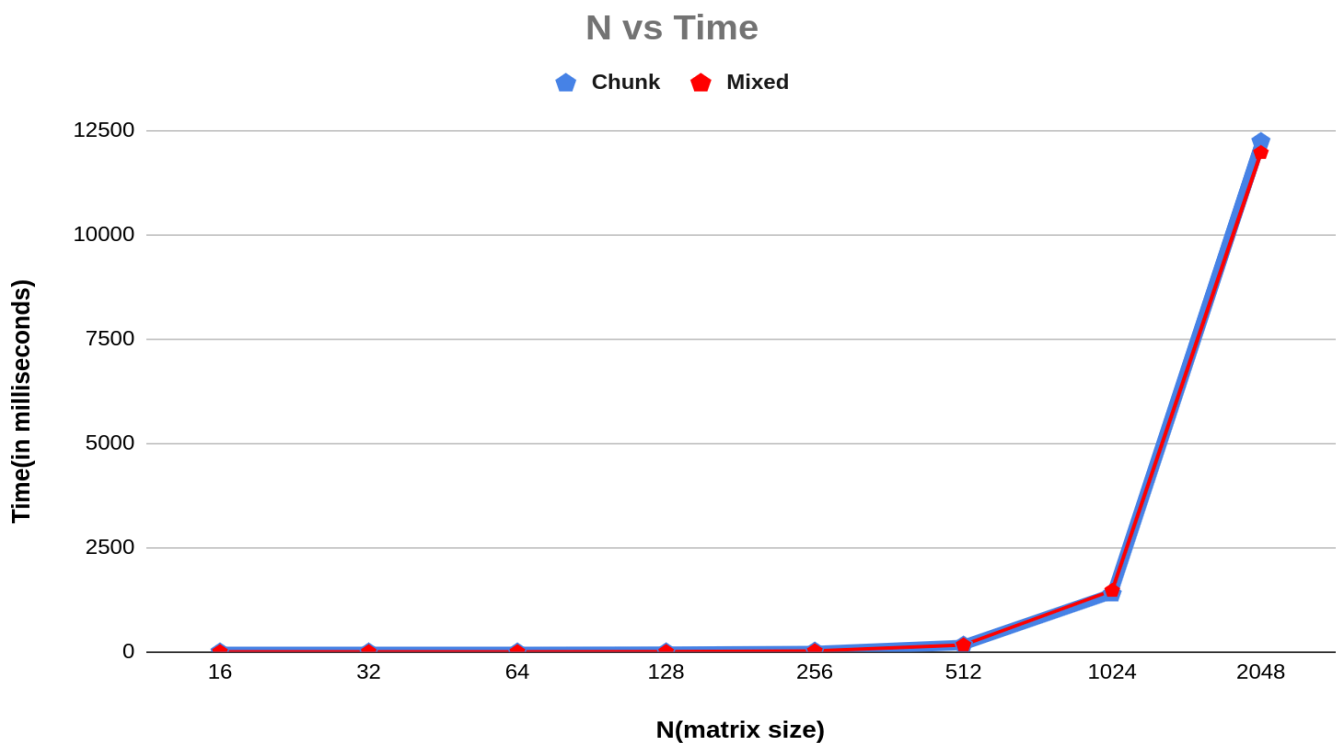
- A function that calculates a dot product of a row and column when given their numbers is written (*cellInResMatrix*) which writes the value calculated to the resultant matrix at the correct position.
- In the program I have a function (*tFunction*) which is the function implemented by threads and also 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).

Graphs and their analysis:

At K = 8 , N vs Time is as follows:



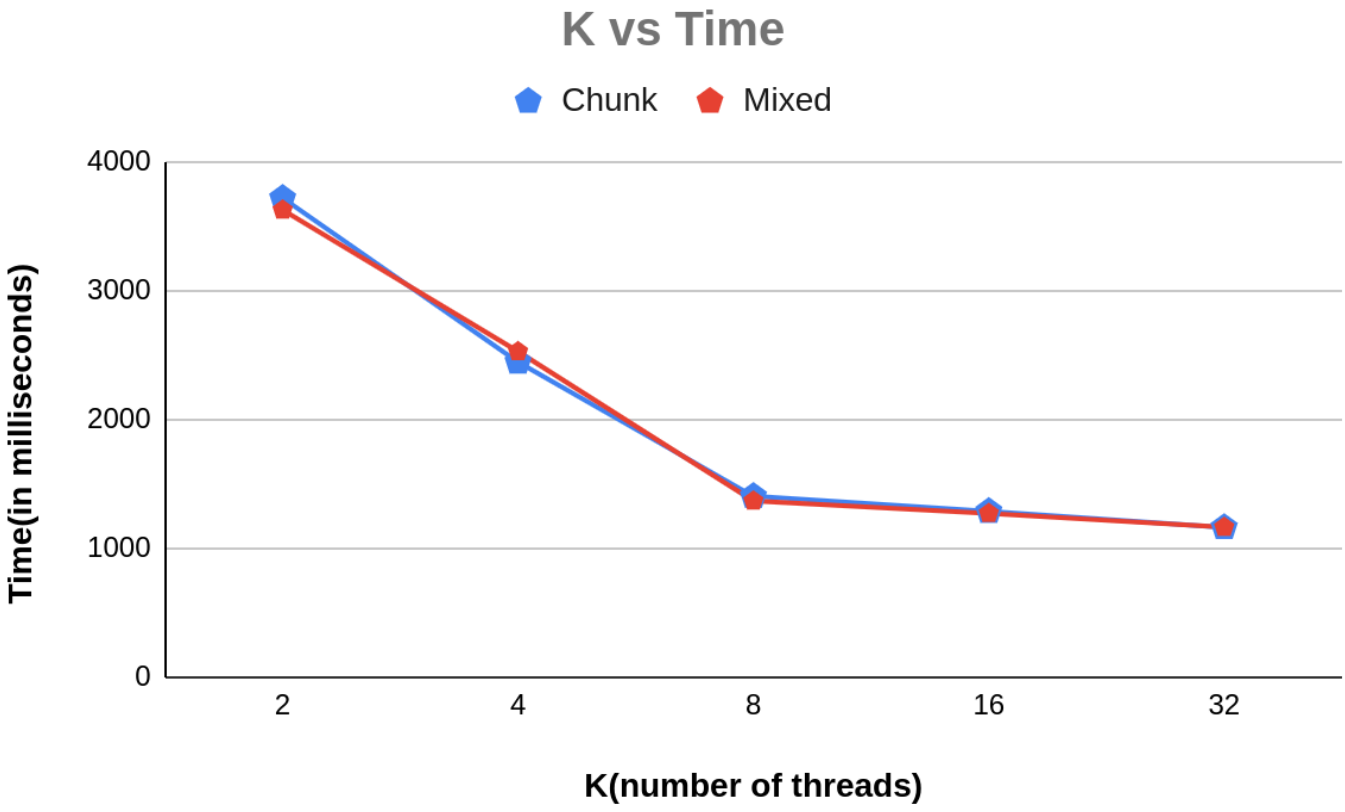
N	16	32	64	128	256	512	1024	2048
Time(in milli sec) for Chunk	0.577	0.655	0.841	2.612	21.739	164.888	1391.6	12244.06
Time(in milli sec) for Mixed	0.541	0.633	0.740	2.941	21.248	163.07	1469.92	11972.96

The table is drawn to show the slight difference in values of time in both the methods.

The graph of N vs Time appears almost the same for both mixed and chunk methods .

In both the graphs we can observe that the time increases as N increases as the task needed to be performed by each thread is high (number of threads being constant) which takes more time than before. The delay could also be because of other underlying factors such as timer constraint or scheduling delay, context switch and so on.

K vs Time at N=1024 is as follows:



The graph of K vs time appears almost the same for both mixed and chunk methods .There are very minimal differences similar to the case of N vs time.
The time taken to complete the task decreases as the number of threads increases as the task needed to be performed by each thread is smaller than before (N being constant). This ensures more parallelism (use of more cores at a time) as

thread count is high and there would be less difference as the number of threads increases much more.
Till 8 threads the decrease is more steeper from 16 it is less steeper as the cores in my PC(12) which gives a limit to have parallelism of utmost 12 threads at a time.

These when compared to normal matrix multiplication are many times faster. Approximately K times for threads less than 8, this depicts the advantage of parallelism in the computation. For smaller matrices due to overhead of declaring threads and task division the time may be smaller for them but the difference can be clearly seen for larger matrices, which are favoured by threads than multiplication function which is of time complexity O(N^3).