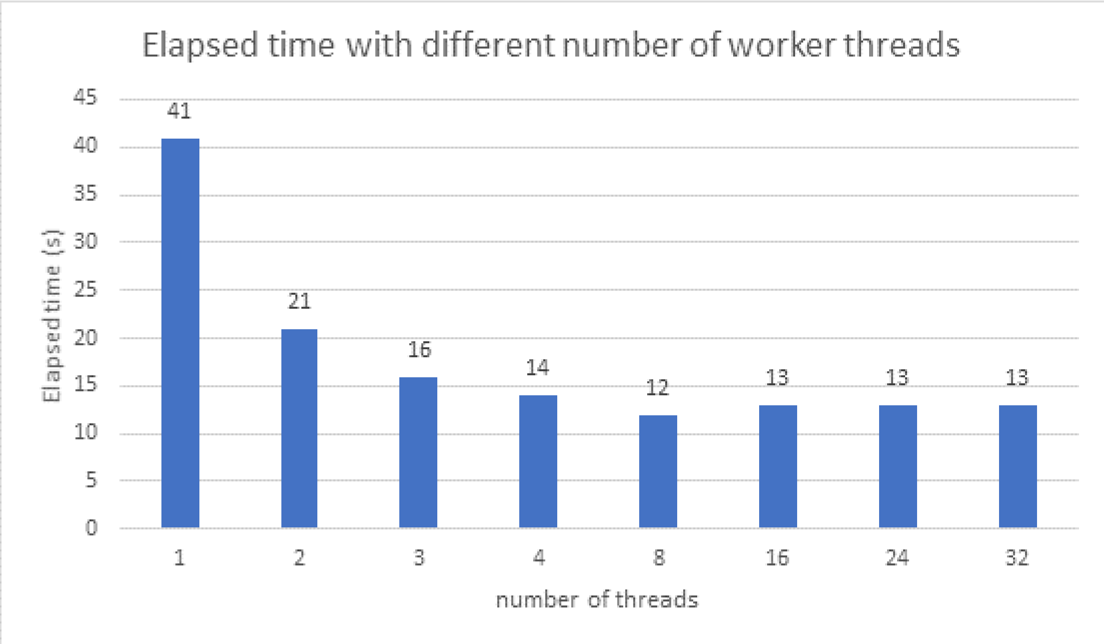
Report of Multi-threaded and kernel module programming

Work distribution:

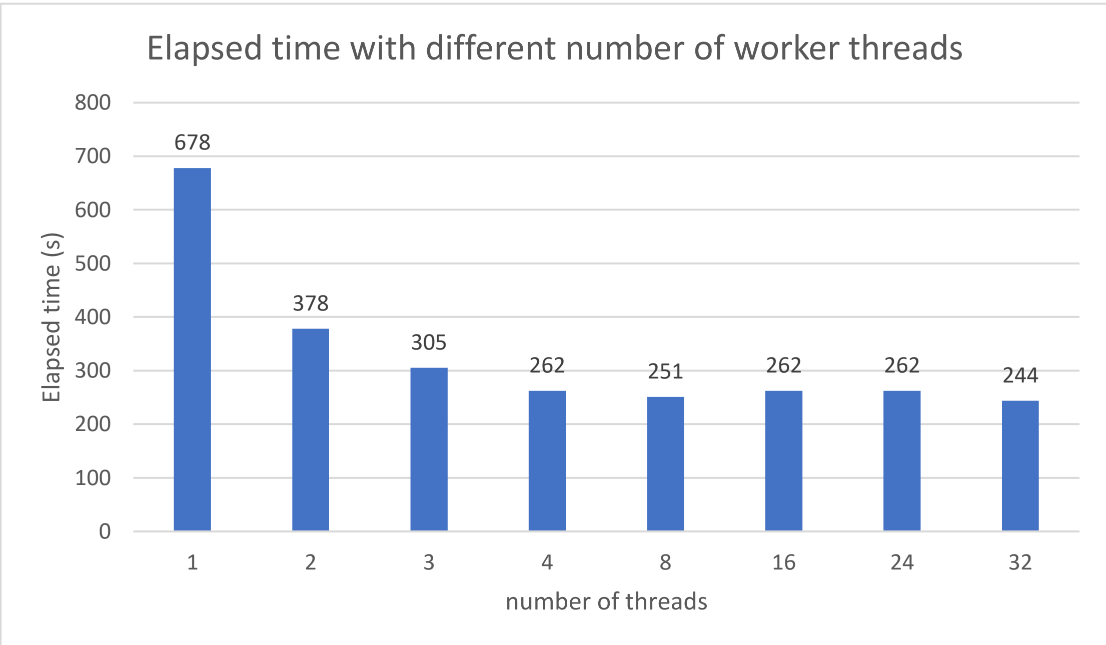
If row1 > 16 ,work distribution in matrix multiplication is by row in matrix1, that is, all threads do row_number/number_of_threads rows in matrix multiplication to matrix2, and if not divisable , the last thread will do the left rows. **Else , do it by column in matrix2.**

Execution result:

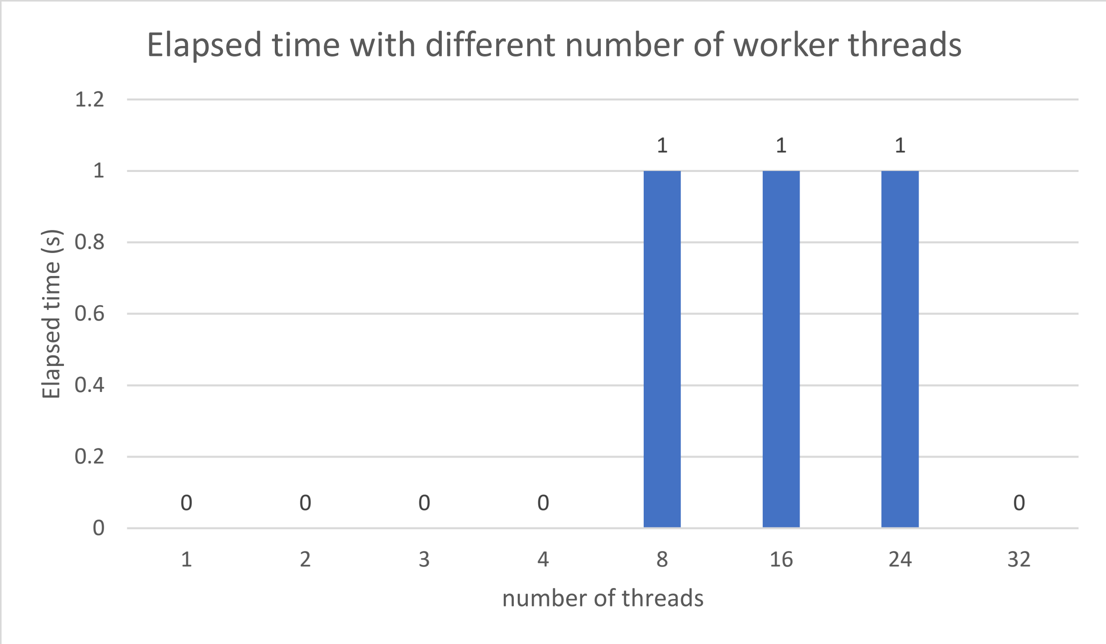
Testcase1:



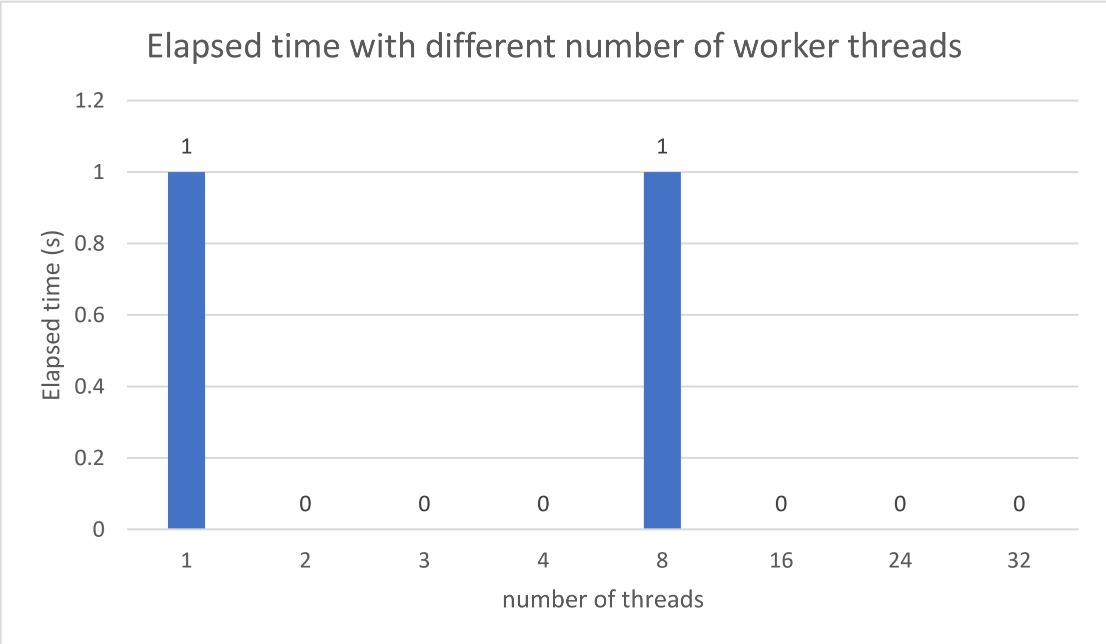
Testcase2:



Testcase3:



Testcase4:



Summerization:

From these four charts, we can get some interesting observations:

1. The more threads not always the faster.
2. If the thread number be more than core number, they may **steal** time from each other, **but only if each individual thread needs 100% CPU**. Consequently, if not all threads need 100% CPU, we create more threads to make our program be executed faster.
3. If the thread number be less than core number, we know that every thread can get

100% CPU, however, there may be some **wasted** CPUs (without being used), so , we can make the process faster with creating more threads.

4. If the execution time is too small, the result may be affected by other reasons like I/O and etc. , so it may not show the relationship between thread number : core number and execution time.

5. From my observations, I think the **fastest** way to execute program is **assign core number : thread number with about 1 : 2**, so we can make maximum use of CPU and make it faster.