**MY NAME : NIKHIL RAJPOOT**
**CO23BTECH11015**

# REPORT

## ALGORITHM OF CODE

1. Reading a file from input.txt in which user will give K and N in first line and Sudoku from the second line. Users must ensure that N is a perfect square .

2. Use malloc storing os sudoku in the global variable sudoku.

3. Starting of clock after closing input file because above then closing statement for all 3 Methods reading time is the same so no need to calculate it.
I use the gettimeofday built function to calculate time which is more accurate then clock.

.4. Splitting of threads in almost 3 equal parts and according to distribution of k into K1,K2,K3 , assign them that each group thread has to execute from where to where for which we Make our own data type by struct which stores starting and ending points and thread id.

5. Creation of thread using pthread_create taking parameter thread_id , default arguments Runner functions for row , column ang subgrid check and parameter of runner functions.

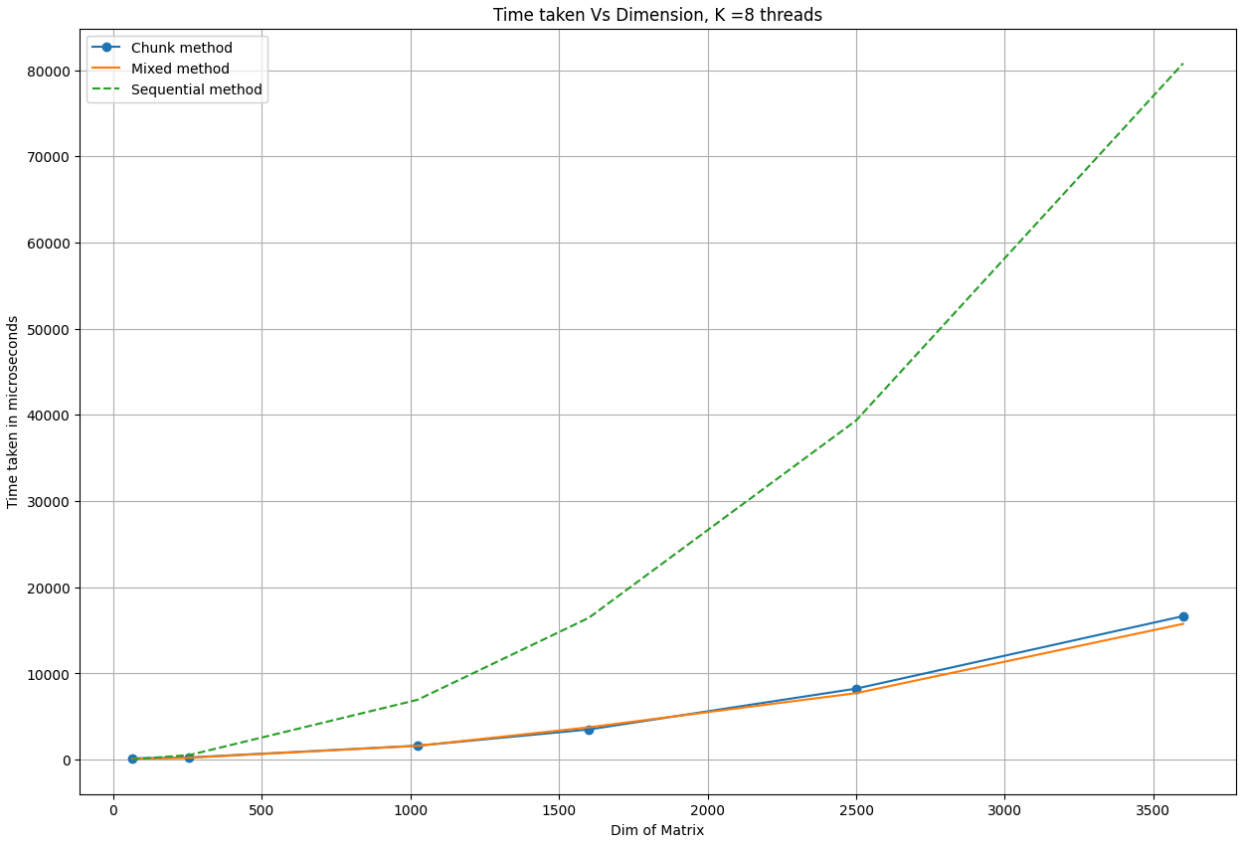
6. Waiting for execution of all threads using pthread_join.

7. Print the output according to least time of execution and output will be print in output.txt And the last two lines tell whether sudoku is valid or not and time of execution in microseconds.

**EXPERIMENT 1 Varying sudoku size keeping number of thread constant k=8**

| | N=64 | N=256 | N=1024 | N=1600 | N=2500 | N=3600 |
|---|---|---|---|---|---|---|
| | | | | | | |
| sequential | 48 | 495 | 6954 | 16972 | 39629 | 84825 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 44 | 509 | 6837 | 16301 | 41555 | 79184 |
| | 44 | 523 | 7018 | 16676 | 38705 | 81580 |
| | 44 | 604 | 6769 | 15819 | 38064 | 79531 |
| | 45 | 524 | 7127 | 16477 | 38836 | 78820 |
| Average | 45 | 531 | 6941 | 16449 | 39357.8 | 80788 |
| | | | | | | |
| chunk | 218 | 321 | 1868 | 3405 | 7808 | 17635 |
| | 147 | 259 | 1569 | 3816 | 7794 | 16772 |
| | 145 | 245 | 1551 | 3374 | 8949 | 15923 |
| | 115 | 225 | 1567 | 3419 | 7933 | 16736 |
| | 107 | 202 | 1525 | 3484 | 8709 | 16177 |
| average | 146.4 | 250.4 | 1616 | 3499.6 | 8238.6 | 16648.6 |
| | | | | | | |
| mixed | 107 | 226 | 1888 | 5058 | 7466 | 15573 |
| | 104 | 220 | 1517 | 3467 | 7778 | 15802 |
| | 103 | 203 | 1537 | 3455 | 7762 | 16796 |
| | 102 | 215 | 1519 | 3430 | 7821 | 16137 |
| | 105 | 207 | 1531 | 3333 | 7736 | 14455 |
| average | 104.2 | 214.2 | 1598.4 | 3748.6 | 7712.6 | 15752.6 |
| | | | | | | |




**Observation:**

For small dimensions sequential is performing well because multithreaded program like mixed And Chunk have Overhead time and context switching time which makes them slow but As we increase the size sequential is taking much more time because of performing single Task of calculation at a time means one by one executing all things , while for large N mixed and Chunks are fast due to splitting of work among threads , using multi core functionality making them faster.

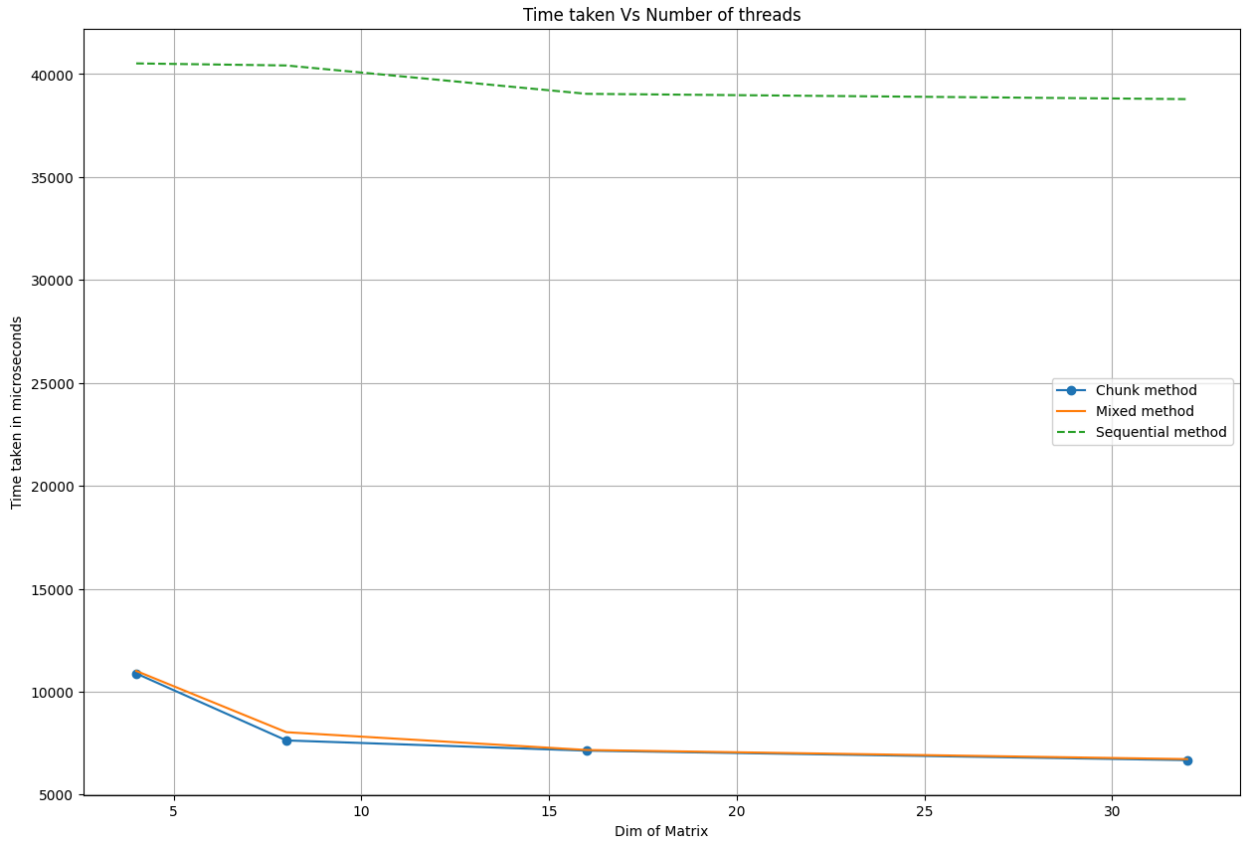
But in mixed VS chunk , initially chunk is faster but as size increases mixed performs better than Chunk because mixed has almost equal distribution of work among all threads due to which performance is fast while in chunk we are giving work according to chunk size but there Will be cases when some threads have load of executing more task than others due to which

It is slow but for small N both have almost the same overhead and switching time of context makes them take almost equal time.

**EXPERIMENT 2** : Keeping N constant and varying K

| | | | N=2500 | |
|---|---|---|---|---|
| | | | | |
| | K=4 | K=8 | K=16 | K=32 |
| | | | | |
| | | | | |
| CHUNK | 10786 | 7732 | 6993 | 6375 |
| | 10889 | 7858 | 7135 | 6810 |
| | 10892 | 7663 | 6809 | 6768 |
| | 10910 | 7605 | 7142 | 6913 |
| | 10941 | 7298 | 7592 | 6503 |
| average | 10883.6 | 7631.2 | 7134.2 | 6673.8 |
| | | | | |
| | | | | |
| MIXED | 10869 | 8079 | 7058 | 6637 |
| | 10896 | 7707 | 7414 | 6596 |
| | 10903 | 7742 | 7213 | 6388 |
| | 11139 | 7777 | 6831 | 6563 |
| | 11227 | 8849 | 7320 | 7436 |
| average | 11006.8 | 8030.8 | 7167.2 | 6724 |
| | | | | |

| | | | | |
|---|---|---|---|---|
| SEQUENTIAL | 42152 | 39821 | 40108 | 40109 |
| | 38374 | 39834 | 39275 | 38462 |
| | 39293 | 39472 | 38237 | 38370 |
| | 43377 | 42958 | 38752 | 38625 |
| | 39386 | 39999 | 38822 | 38343 |
| average | 40516.4 | 40416.8 | 39038.8 | 38781.8 |
| | | | | |




**Observation:**

Because i took N as 2500 , large number we see Sequential is taking maximum time and almost equal time while changing number of threads , therefore it means sequential is independent of

Number of threads.

We see for K =8 there is significant drops starts for multithreaded methods because as k Increases , there are more threads for performing tasks which reduced the workload on each Thread causing gradual decrease in time .

Distribution of Workload into a large number of threads all running in multicore decreases execution time.

For K=4 it’s taking more time because the distribution is of threads to each group in less like 1, 2,1 making it less efficient because only one thread is checking for all row or column or grid That's why they are taking a little more time.