INTRODUCTION

As per assignment we need to perform Chunk,Mixed and Sequential method to tell whether Sudoku is valid or not

Chunk method: In the chunk method, a thread is assigned consecutive rows, columns or subgrids to validate. For example, if the number of rows is R and the number of threads for validating rows is K1, the chunk size p is calculated as p=R/K1. Thread 1 will validate rows 1 Thread 2 will validate rows p+1 to 2p, Thread 3 will validate rows 2p+1 to 3p and so on. The same allocation method is applied for columns and subgrids.

Mixed method: Here each thread, instead of assigning consecutive rows, columns or subgrids, the workload is distributed evenly among the threads in a cyclic manner. For example, if the number of rows is R and the number of threads for validating rows is K1 Thread 1 will validate rows 1, K1+1, 2K1+1,... Thread 2 will validate rows 2, K1+2, 2K1+2,... and so on. This pattern continues for all threads.

Sequential Method :It is normal way of solving any problem without using Thread and check On each Row , Column and Subgrids .

CONTENT OF ZIP FILE

Create_matrix.cpp →Takes K and N as input and create sudoku
Assign1Readme-CO23BTECH11022.txt
Assign1Src-CO23BTECH11022-chunk.cpp
Assign1Src-CO23BTECH11022-mixed.cpp
Assign1Src-CO23BTECH11022-sequential.cpp
Assign1Src-CO23BTECH11022-earlychunk.cpp

WORKING OF CODE

0. I am defining my own data type by struct for simplicity , They are Data and Result
   Data →store following informations:
   - Start – which tell threads that from which row, column or subgrids they have to start.
   - End – tell threads that from which row, column or subgrids they have to end.
   - Innervalid – When we iterate to check validity we store validity in it.
   - Tid – stores Thread number go from 0 to N-1.

   - StartTime – Stores when is computation task starts , which act as reference By which we check which operation of thread executed first.

Result → store informations of things we have to print in output

Tid – stores Thread number vary from 0 to N-1
Index – stores indexes of rows,columns and subgrids.
Innervalid - When we iterate to check validity we store validity in it.
Totaltime – for storing total execution time (final - reference)
Whichthing – tells row,columns or subgrids.

1. In the main function , input is taken from the inp.txt file using the read_object of fstream class.
   Taking K = Number of threads , declaring it Globally.
   N = Dimension of sudoku , declaring it Globally.
   Taking Matrix of sudoku in global variables to access them anytime .

2. After Getting K and N we check whether N is perfect square (for subgrid) and K is Greater than equal to 3 to make 3 Groups of Threads .
   Defining n as the square root of N globally which will later help for checking subgrids.
   Defining results of size 3N to print because we are checking N rows , N columns and N subgrids dynamically .

3. For Chunk dividing K into K1, K2 ,K3 and their respective chunk size as P1,P2,P3.
   Taking rem1 ,rem2, rem3 which stores the remainder of N%k1,N%k2,N%k3 .
   Taking the K number of threads by pthread_t .

4. Now we are starting computation tasks So starting the clock.
   Creation of 3 variables rowsplit of size k1 , colsplit of size k2 and blocksplit of size k3.
   They all are of Data(user defined datatype) which stores information for thread.
   In case of chunk try to divide operations with help of remainder for almost uniform Distribution.

5. Creating K1 threads first by pthread_create giving (thread _ id , default attributes , rowcheckchunk function , function argument which is stored in row split)as parameters.
   Similarly for K2 threads for column check and K3 threads for block/subgrid check.

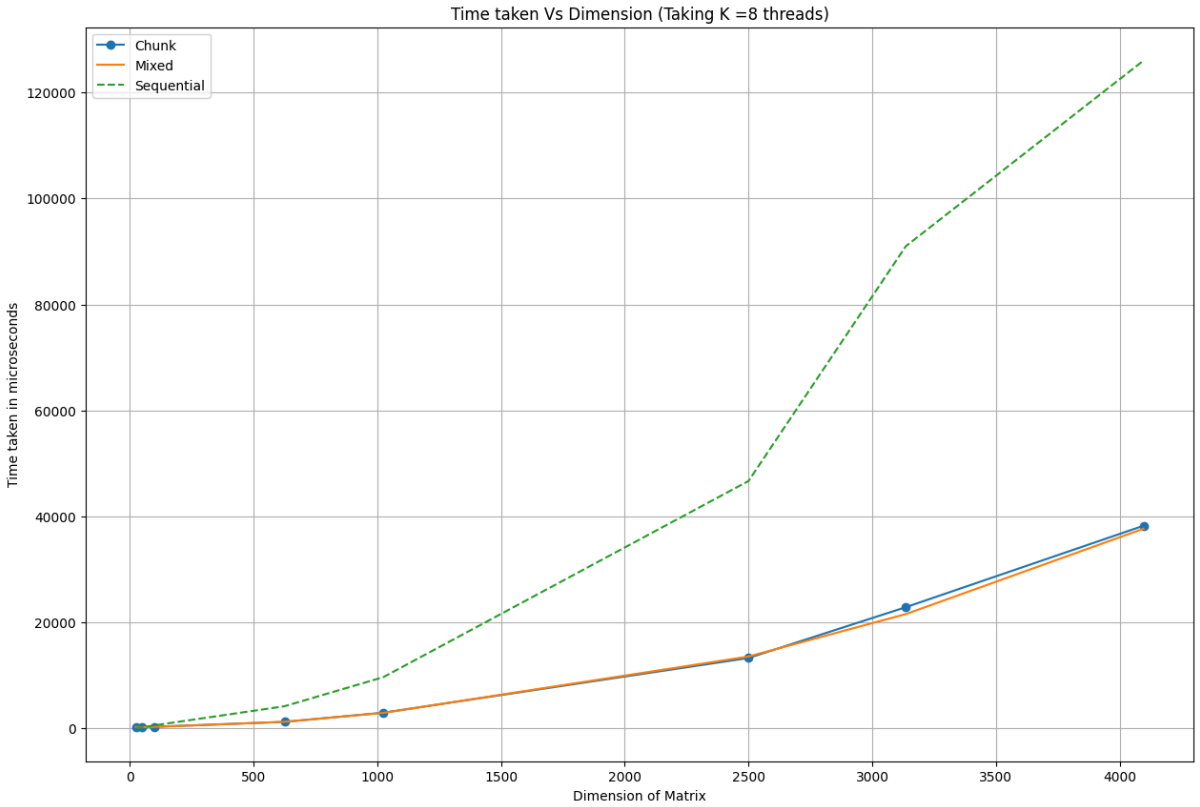
6. Wait for all threads to finish by pthread_join().

7. All computational tasks are done so calculate total duration by chrono(calculate time very precisely) and store it , we have to print it at last.

8. Now just the printing part is left , because we want to print in order of least time of execution , sort the result in ascending order of execution and print them.
   At last print Validity and time.

EXPERIMENT 1 OBSERVATIONS

		N=4096	N=3136	N=2500	N=1024	N=625	N=100
	iteration 1	39288	22786	11361	2804	1205	292
	iteration 2	34930	21657	14226	3013	1240	344
	iteration 3	36725	23722	13627	2847	1310	273
	iteration 4	39678	21671	13012	3127	1174	368
	iteration 5	40600	22454	14258	3085	1300	311
Average chunk		38244.2	22458	13296.8	2975.2	1245.8	317.6
	iteration 1	36806	20421	11426	2669	1319	300
	iteration 2	36340	21967	16721	2770	1268	246
	iteration 3	40241	24370	15680	2972	1303	326
	iteration 4	38337	23026	11156	2933	1135	300
	iteration 5	36699	25339	12906	3336	1255	282
Average mixed		37684.6	23024.6	13577.8	2936	1256	290.8
	iteration 1	138858	76516	48673	9607	4304	688
	iteration 2	124131	76154	46047	9725	4135	600
	iteration 3	123434	76877	45650	9682	4237	485
	iteration 4	122390	76599	45900	9687	4069	548
	iteration 5	121227	148858	47167	9674	4197	652
Average seque		126008	91000.8	46687.4	9675	4188.4	594.6



We observe that while keeping the number of threads as constant when we increase The dimension of sudoku sequential is taking maximum time with respect to chunk And mixed Because sequential does not use any threading and executes only one task at a time . Multithreading allows multiple tasks to run simultaneously to multi core CPU
Thus increasing the performance which we see in graphs both mixed and chunk taking less time .

As dimension increases we see mixed is taking less time make it most efficient for Higher dimensions because in mixed due to cyclic order of execution ,work loaded Is distributed among all threads equally or number of operations are almost same But in chunk due to allotment according to chunk size force us to assign all leftover Operation to the last thread causing extra load for the last thread resulting in extra time.

Or large chunks can lead to thread imbalance , where some threads finish much Earlier while others are still processing their chunks.
While mixed reduces the chance of one thread becoming bottleneck , leading Better resource utilization.

For small k or for k which are multiple of 2 we see chunk and mixed takes almost equal time.

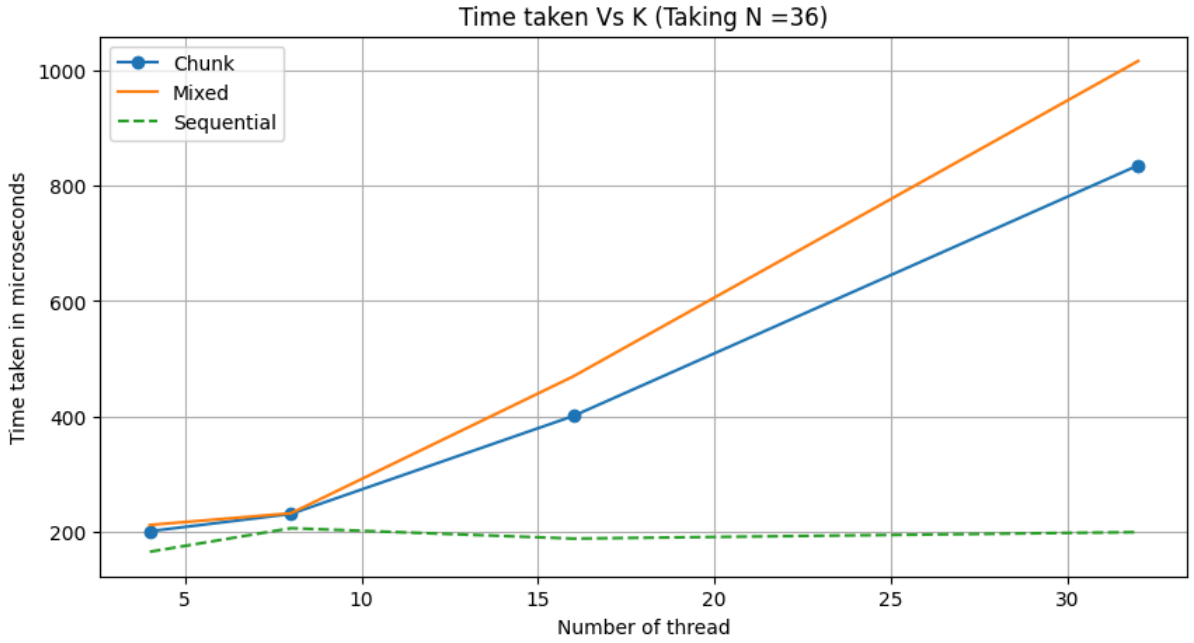
To optimise chunk what we can do is find out remainder for every group , remainder

Tell extra operations , as number of threads in each group is always greater than Remainder distributes remainder to some threads by adding 1 to those threads.

Ex - N=1024 K=342 k1=k2=k3=114 P1=P2=P3=8 which means each total perform 8*114 =912 so one way is to assign 8 operations for 113 threads and rest operation for last which increases the workload of that thread instead of that we can distribute it to all.
Remainder is 112 give this to 112 threads , so 112 threads execute 9 operations
And 2 threads perform 8 operations.

EXPERIMENT 2 OBSERVATION

CASE1 : FOR SMALL N

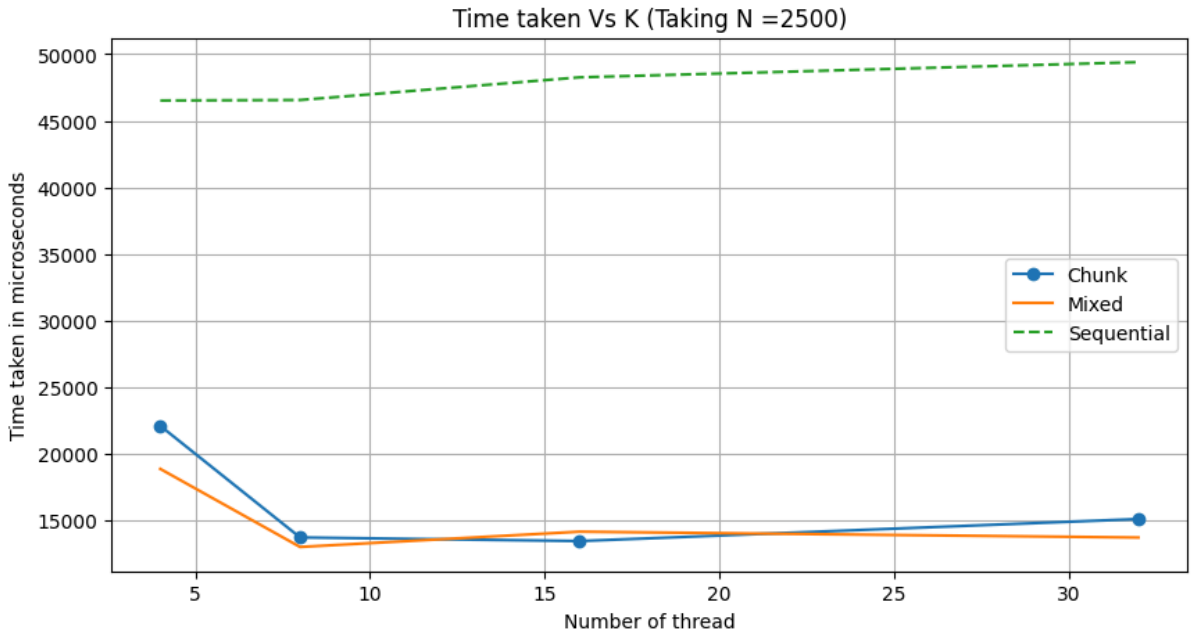


For small dimension like N=32 as above we see Sequential is performing better the Reason is because chunk and mixed have some overhead time like context Switching overhead or added thread management overhead which makes both Method slow.

Mixed is taking little more time , the reason we cannot say surely because of less size but it might be due to overhead of threads and for small dimensions continuous Execution is fast with respect to cyclic Execution.

CASE 2 : FOR LARGE N VALUES

		k=4	k=8	k=16	k=32
	iteration 1	18302	13928	12176	15642
	iteration 2	18619	14763	12351	15006
	iteration 3	18452	13284	14108	15307
	iteration 4	29163	12858	13697	15119
	iteration 5	25775	13551	14687	14229
Average chunk		22062.2	13676.8	13403.8	15060.6
	iteration 1	18776	11663	15670	11910
	iteration 2	19536	13785	14501	14418
	iteration 3	19161	14592	12731	16465
	iteration 4	18386	13705	14132	12639
	iteration 5	18276	11057	13514	12914
Average mixed		18827	12960.4	14109.6	13669.2
	iteration 1	47387	48745	47716	52164
	iteration 2	46124	46218	49187	48379
	iteration 3	46473	46295	48323	45904
	iteration 4	45997	45618	49800	52978
	iteration 5	46626	45922	46288	47591
Average seque		46521.4	46559.6	48262.8	49403.2



We see that sequential is taking almost same time because sequential working is Independent of k (Number of threads).

As k increases we are seeing sometimes mixed is faster and sometimes chunk Maybe the computation of splitting threads vary there execution time and overhead Also changes every time which results in fluctuations .

Near K =8 we see a dip which tell having lesser threads is taking more time with Respect to right side of dip because for lower thread computation and overhead is More and multi processing is less.

EARLY TERMINATION PART FOR CHUNK

1. I used one global variable isChunkvalid which Shared between threads and help to tell invalid Sudoku and stop further checks.

2. I used pthread_kill() which sends a signal to a specific thread , it takes 2 parameters to which the signal should be sent and a signal handler .

3. For signal handling we use pthread_exit(NULL) which exits the calling thread safely and cleans up resources associated with it and does not return anything.

4. I used signal(SIGUSR1, signal_handler) where SIGUR1 is a user defined signal and signal_handler is a function that will be called when SIGUSR1 is received.

OBSERVATIONS OF EARLY TERMINATION

1. Time of execution is highly dependent on how much valid sudoku means suppose there is only one or few cells which have duplicate values then it will take about as long as a Normal chunk (For small N , i tried till N =9,16,25,36) .

2. But suppose we are generating Sudoku by random Numbers then there is very high probability of mostly duplicate values causing invalidity of rows or coloums or subgrids. So in those cases early termination takes very less time,
with respect to the Normal chunk method, almost constant because randomly generated sudoku is terminated within 1,2 or 3 operations .

3. Therefore it's highly dependent on how many subparts of sudoku are valid , more proportion of valid parts cause increase in time .

GRAPH OF TIME THAT NORMAL CHUNK METHOD AND EARLY TERMINATION