Report

Assignment 1 Operating system 1

Reading

The file My input file name is “inp.txt” and output file name is “out.txt” . For OutFile I have made a file pointer that is called at the end of the program and getting written using answer matrix.For input file using ifstream from fstream library and taking input in the main () function. “In.txt” contain a line. Extracting the N and K from this by simple reading and called “read matrix” functionFrom reading determining the N and K value and making “matrix” 2d vector of N x N size and storing the value of K and N using pass by reference.

To measure Time

:Used “auto start_time = std::chrono::high_resolution_clock::now();” found in time.h library just counted the two timing and difference will be the time taken to execute on threads.

Threading Methodology

I have used Posix thread and called pthread_create function with its arguments and for parameter passing I have made struct data type. That is containing N and K and also 2d vector matrix to be passed as this is the data used by all. For passing pthread_create I have made array of pthread_t of size k where the function instance will be stored. Also I am passing ans matrix that is keeping track of the result. And last have called pthread_join function as no return value is there so passed its argument as null and thread id th[i].

for load balancing

For Chunk: - same strategy applying as guided in the question like finding ceil(N/K) and calling as Dist and Distribution each thread this amount of row in consecutive manner. Ofcourse there is chances that Last thread will lesser number of rows but it will not affect much.

For Mixed: - As the question suggests, I have made a 2d vector for this that will keep the record of row assigned to each thread. This is done using the main thread. I have done its implementation using modulo function. Like just travelling in each row index and assign to threa p if p = row%K (i.e. row modulo thread size.

why chunk is not always a good approach.

see a triangular matrix. if the things will be provided in chunk to the thread, then last threads literally have very few works to do (if it is upper triangular) as thread will get 0 value and sit idle while rest of thread will have comparatively higher value. It may induce load imbalance if chunks have significantly different computational loads. But it favors locality principal of the cache thus may improve some efficiency if matrix is not biased like triangular or other thus mixed way of matrix multiplication provide better efficiency in this case. Matrices with specific patterns often arise in structured problems. For example, in signal processing or communications, matrices with zeros coming at regular interval is

common in certain rows or columns representing inactive or irrelevant components thus in these kind of problem mixed way will be inefficient.

why mixed way of matrix multiplication also not very efficient ?

Even though it encounter the drawback of the chunk but it does not able to recover the best property of chunk that chunk provide locality .see for larger number of threads say like 8 to 10. Each row needed to do this multiplication will be k rows apart. But see in the cache perspective cash can load few rows due to locality of cache behavior (say taking 4 rows). But as in this methodology we using 8th row after second iteration thus most of the time lead to cache miss and thus may lead to higher time.

My approach

In my approach I still want to keep the things row wise because it will be good in provinding cache behavior's we can see in this example

My approach came into mind by considering Chunk methodology is good in cache management while Mixed approach provide better result when matrix is triangular or have sparse behavior. Let's say architecture is such that when a cache miss then it can load next few rows (it depends upon the architecture and associativity inside the cache) In N cross N matrix each row contain N elements and as integer is the data type so 4*N bytes is contained in each row. so, assuming using locality principal if cache miss occurs then 64 byte of 8 word will be loaded from the RAM to cache. I am assuming this to be main criteria decide to divide chink size to 8 as creating more chunk size also lead to cache miss because when cache miss happen then locality behavior load some finite data on the basis of locality like few byte depending upon block size and associativity of the cache. This motivates me to come with this idea.

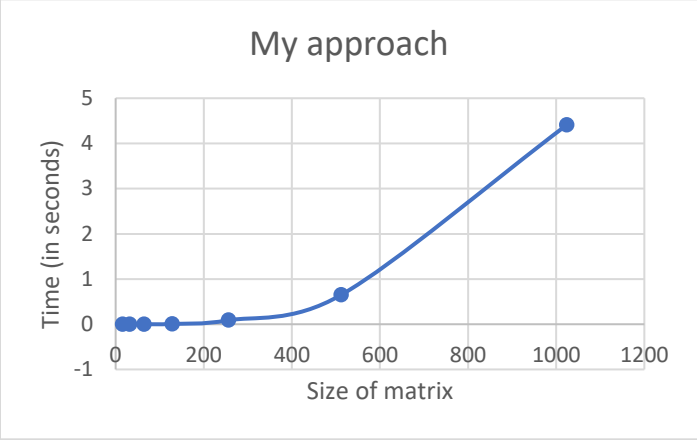
In my code First we are assigning each thread 8 number of rows and then starting second round for the similar distribution and when dist= ceil(rest of row left to assigned / K) is less than 8 then we are using

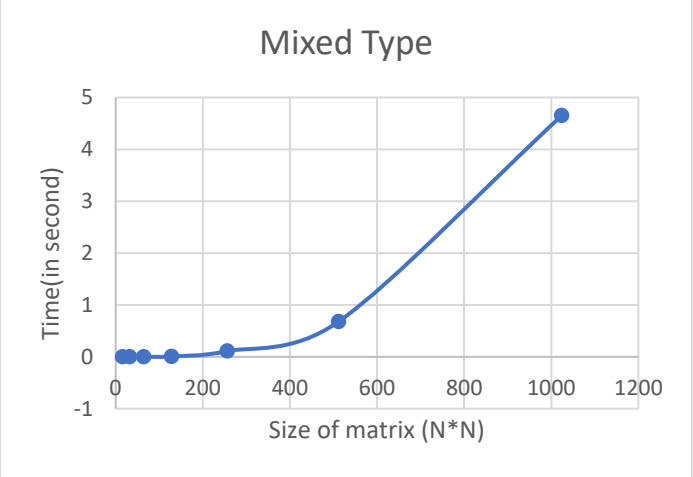
Mixed technique to rest of them. As matrix of row is assigned to thread p if p = I % K (row number modulo number of threads). Doing this will encounter the triangular matrix and some sort of symmetry up to some extent and lead to the better result. Also we are able to take the benefit of caching.

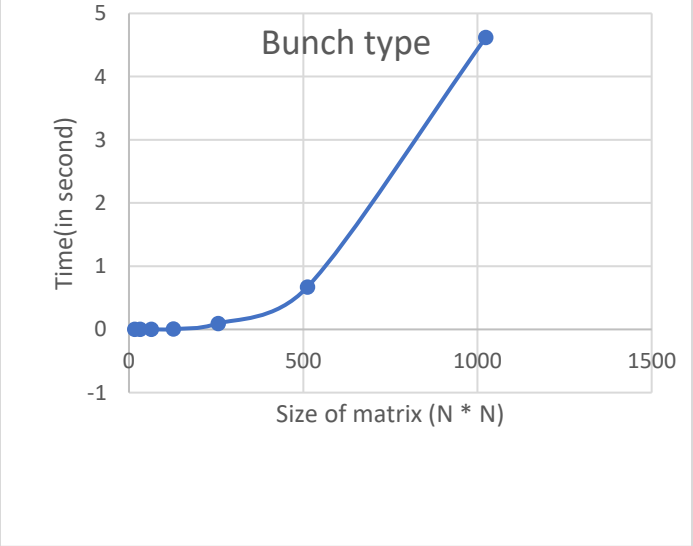
effeciency of these matrix lies in both size of the matrix and architecture upon this algo is running as cache is one of the main factor in consumption of time along with computation.

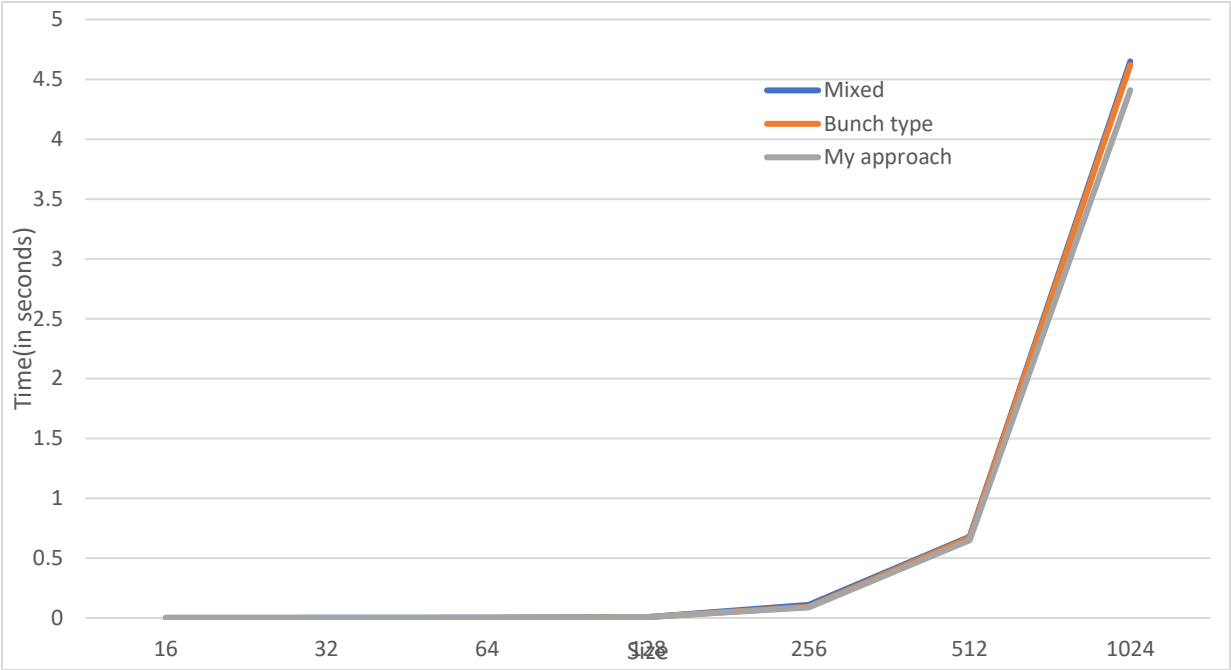
The mix way can be more efficient for smaller matrices or situations where load balancing is critical.

Keeping the Number of thread constant 8 .Note each time values are averaged 5 times then plot to remove the outlier.

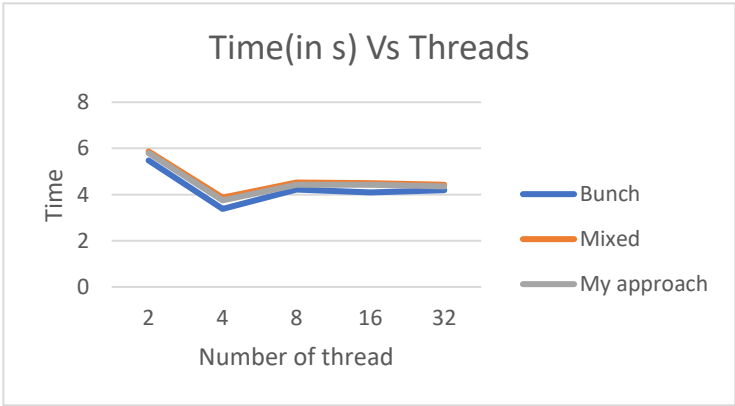








Observation: --If we compare between Bunch and Mixed type. Initially Bunch is providing better performance in comparison to Mixed but at the last mixed is achieving approx. same time> This can be stated like too large chunk size may not always provide better efficiency. While my approach is initially taking more time than Bunch one but at last Beating all others.



Observation: --This tells that increasing amount of thread too much not always increase the time. Here in my experiment, I found that Bunch Is providing better performance then all others.