ASSIGN2: SET AFFINITY

Gaurav Choudekar - CS22BTECH11015

February 23, 2024

1 Introduction

This report is regarding programming assignment-2 conducted on efficient matrix squaring with set affinity through threads.

2 About Code

The code is written in C++.Square matrix is given as input with number of threads for calculation and resulting square matrix is obtained in output file.

3 Input format

A file is given as input,name of file is modified accordingly in code.N,K,cores,bt are given first in input files with spaces and then matrix numbersone by one,covering first all numbers from left to right from one row then going to lower one,bt is bounded threads,cores are number of cores to be used for bounded threads.

4 Code Flow

4.1 Main Function

In the main function we are giving a file as input for which we want to obtain square matrix.we will then tokenise one by one elements of matrix from file and store them in global array. we will one by one call functions with threads for this different techniques individually.

4.2 Other Functions

We have different functions for different techniques for namely multiplychunk,multiplimixed. createnode function is used to create node required to store information about on what data a particular thread has to work upon.

4.3 global variables used

- n:Represents the size of the square matrices involved in the matrix multiplication.
- k:Denotes the number of threads used for parallelizing matrix multiplication.
- p;Represents the total number of elements in the result matrix (n * n).
- result,resultm,resultnew:Matrices used to store the results of matrix multiplication using different techniques or strategies.
- given: Input matrices for matrix multiplication
- code: Input matrices for matrix multiplication
- cores:stores input for number of cores

- bt:stores input for number of bounded threads.
- executiontimeschunk and executiontimesmixed arrays to store execution times for each thread.

4.4 chunk and mixed functions

These functions are directly called by main function.these functions further create pthreads,decide and allocate data on which they have to work and call functions for calculating data/part of matrix which a particular thread works upon.these input parameters required by each thread to work upon are starting point(row numbers) of thread,how many rows to compute,A.P difference etc. These functions passes nodes as input required to multiplychunk and multiplymixed function so that they can operate on that,it then sums times to execute each thread seperately for bounded and non-bounded and finds its average,further,then output result in other file after joining pthreads.

- variables used
  - pthread threads[n]: Array to store pthread identifiers.
  - struct node* temp[n]: Array of node pointers to store thread-specific information.
  - rem:to know how many rows of matrix needs to be adjusted due to not proper division of total number of rows and number of threads.
  - t=n/k Variable to store the number of rows each thread should process in an even division scenario.
  - ofstream: Output file stream for storing the resulting matrix.
  - timeb,timenonb: variable to sum exxecution times of threads.
- Difference in techniques
  - in chunk,serial rows are appointed to threads
  - in mixed rows appointed to each thread is not serial,so their staring point is i.unlike chunk.

4.5 multiplymixed and multiplychunk functions

These functions is individually called by every thread and calculates rows on which it has to operate from node given as input.node has every information required from its operation. It then stores element of resultant matrix in seperate matrix.multiplymixed function implements a mixed strategy for matrix multiplication. It leverages a struct, node, to pass essential parameters such as the number of rows, starting index, and the increment factor k,which is number of threads.these factors are used to know starting point,ending point and increase factor during thread executing multiplication multiplychunk function adopts a chunk-based strategy for matrix multiplication. Similar to multiplymixed, it utilizes the node struct to pass parameters,number of continuos rows are calculated by threads. This function computes matrix products for a specific chunk of rows in serial, promoting parallelization in large-scale matrix operations.scheduling is done on thread number which is stored in node of each thread.for thread number less than bt,continuos threads are assigned to cores depending on it,scheduling is done on the basis of thread number,we calculate how many bounded threads each assigned core should get which is bt/cores then assign threads on basis of their thread number.i/(bt/cores) will assign core number according to their i value.continuos threads will be assigned same cores.start and end time of each thread is stored seperately and then used to calculate execution time.later,it is saved in global variable at index equal to its thread number.

- Variables in multiplychunk and multiplymixed
  - numrows variable Represents the number of rows that the thread should process.
  - s variable Represents the starting index for the rows processed by the thread.
  - e variable Represents the ending index for the rows processed by the thread.
  - k variable Represents number of threads in execution if multiplication.
  - sum variable Accumulates the result of matrix multiplication for a specific element in the result matrix

  - c variable Counter variable used in the while loop to iterate over elements in a row during matrix multiplication
  - i thread number
  - startseconds,endseconds,durationcc to calculate time

- Difference in approach of both processes: rows calculated in multiply chunk are in serial,so for looping over rows we have increment factor of just one,but for looping over rows in mixed technique we have increment factor of k.

- Inbuilt functions used

  - gettimeofday,chrono to note timme
  - cpuzero Clears set, so that it contains no CPUs.
  - cpuset to add cpu in the set of cpu's to which a given task would work upon.
  - schedsetaffinity to set hard affinity of cores for each thread.

5 Graphs

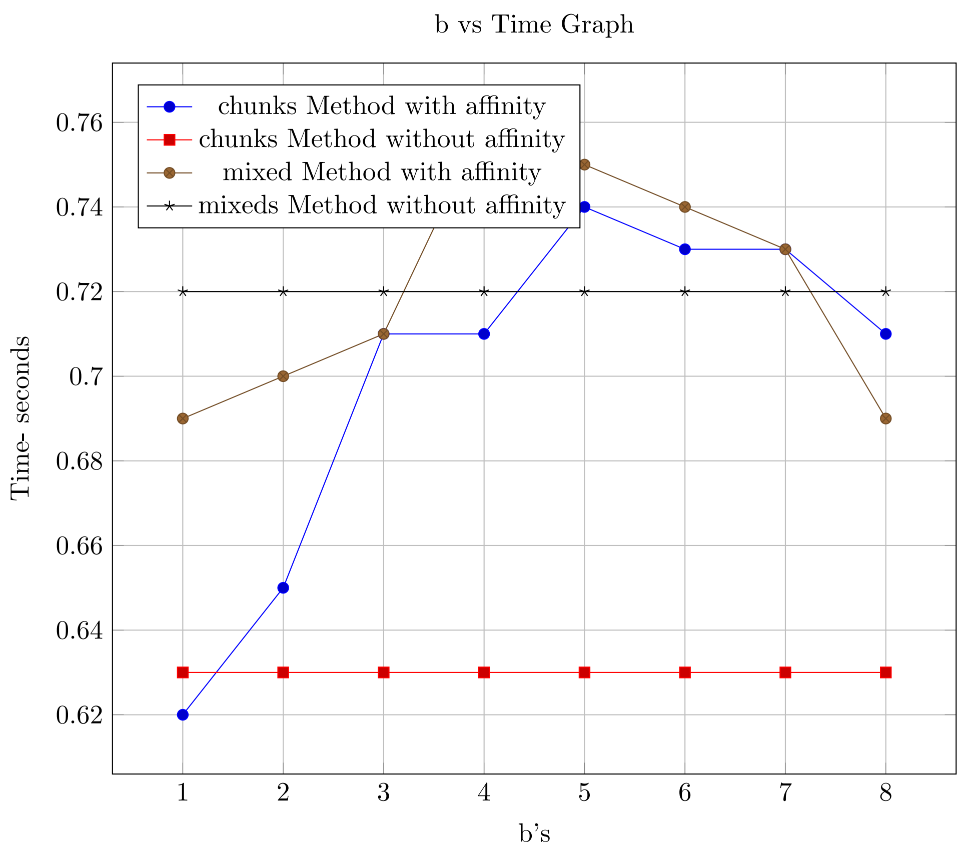

Figure 1: N=1024,K=32,C=8,b=K/C

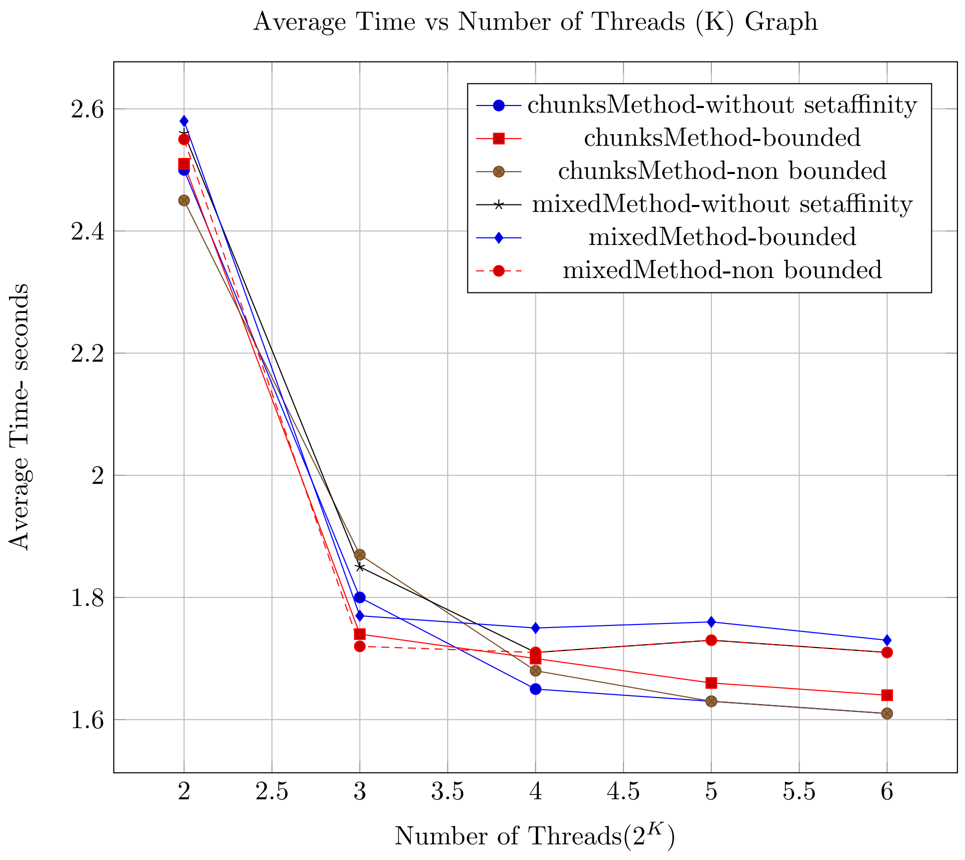

Figure 2: N=1024,C=4,FOR SETAFFINITY BOUNDED THREADS IS K/2

6 output files

- output for chunk method is saved with name "chunk.output"
- output for mixed method is saved with name "mixed.output"

7 Results

7.1 Experiment 1

- We assumed no affinity for one curve each for mixed and chunk,so we see 2 howizontal lines in graph
- initially increasing 'b' increases time,but it does not increase for higher b's,rather decreases or remains stagnant.
- Generally mixed is taking more time compared to its chunk counterpart.

7.2 Experiment 2

- General trend for most of curves shows as we increase number of threads,execution time decreases or remains constant,minute or negligible increase is seen in some curves for higher number of threads
- For most of observations,mixed observation for time is higher than compared to chunk part
- for without bounded to cores curve,general trend of initially decreasing then saturating is seen as increasing threads
- Generally mixed is taking more time compared to its chunk counterpart.

- Bounded method is taking generally more time than non bounded
- for higher number of threads,execution time for bounded and non-bounded is nearly same compared to not binded to core,but bounded is slight higher
- rough trend observed is almost similar for chunk and mixed method

8 Analysis and Conclusions

- As number of threads increases,work is divided between them so decrease in execution time is generally seen with increasing threads.it later saturates as waiting time increases execution time.
- Bounded threads has more execution time than non bounded as schedulling is left on OS for non-bounded thread,it uses cores more efficienlty as it does not wait to get particular core but uses which is available.
- average time for bounded and non bounded threads is little less than total time.there are 2 reasons for it one is the position used to calculate time for complete process is not direct sum of threads but also includes some little other processes,second reason is core is assigned to work upon more than one thread simulateously so many threads are executed parallely
- Bounded threads sometimes incures non-uniform work load balance on core,it is also a reason for its higher execution time.
- benefit of bounded thread is caching,but it has overhead of waiting time and work balance,there is trade-off between this advantage and disadvantage
- We successfully implemented affinity for cores in this code and observed results for it
- Through bounded threads,we aimed to enhance caching,which we observed that it works at some places,but there is its tradeoff with waiting time and work balance.
- The comparison between executions with and without affinity suggests that setting affinity can have a significant impact on execution time. However, the actual effect varies depending on the specific conditions and configurations.
- Chunk method assigns contiguous rows to each thread, possibly leading to better cache utilization and reduced overhead compared to the mixed method, where thread assignments are non-contiguous.
- output files can be obtained in same directory with above mentioned naming conventions