CS2523

OPERATNG SYSTEMS – II

Programming Assignment – 2

By: Devraj (ES22BTECH11011)

SYSTEM SPECIFICATIONS

Processor : 12th Gen Intel(R) Core(TM) i7-12650H 2.30 GHz

Cores : 6 P-cores, 4 E-cores

Logical Processors : 16

RAM : 16 GB

DESIGN OF THE PROGRAMS

Generating the inputs

For generating the inputs required for the programs, I have created a program named ***inputGenerator()***;

It depends on the parameter ***n***, upon which it creates a randomly generated ***n*n*** matrix having numbers from 1 to 1000 (can be changed).

It prints ***n***, ***k***, ***c***, ***bt*** and the matrix in order onto the input file to be used by the main programs.

Common things in the main programs

The main programs read the parameters ***n***, ***k***, ***c*** and ***bt*** from the input file, and then dynamically allocates memory for the two ***n*n*** matrices. Afterwards, they read the values of the matrix from the input file and put it into the 2-D matrix ***A***. Then they create the transpose of ***A*** and store it into the 2-D matrix ***A_trans***.

To store the arguments to be passed into the ***runner()*** function, they create the structure ***arguments***.

The ***dot()*** function gives the dot product of two 1-D arrays. It is used to find the elements of ***A_sq*** by multiplying a row of ***A*** with a column of ***A_trans***.

To implement multithreading, they create an array of threads, with the thread at each index having a unique set of arguments.

The thread affinities are set in the ***runner()*** function using the ***pthread_setaffinity_np()*** function. To do this, we create a CPU set with the required CPU for the thread using the ***cpu_set_t*** data type. ***CPU_ZERO()*** clears the seta and ***CPU_SET()*** sets the required CPU into the CPU set. The index of the thread in the array of threads ***tid[k]*** is passed along with the arguments of the ***runner()*** function to calculate the id of the CPU to be allotted the thread.

Only the first ***bt*** threads are allotted CPU affinities. The rest of the threads are free to run on any CPU as per the OS scheduler. This is implemented by the condition that thread affinities are set only for threads with

index/***thread_no: i < bt***.

The time required to perform the row computations is counted using the ***chrono*** header file.

Then the output matrix ***A_sq*** is printed onto the output file.

The files are closed and the memory for the arrays is deallocated and the programs end.

CHUNKS

Each thread receives a chunk (group) of adjacent rows to calculate. The chunks are of uniform size ***n/k***.

There are ***k*** chunks and hence each chunk is allotted a thread. So, each thread finds ***n/k*** rows of ***A_sq***.

Therefore, each thread receives rows ***i*n/k*** to ***(i+1)*n/k -1***

where ***i*** goes from 0 to ***k***-1.

The arguments passed to the ***runner()*** function are :

1. ***start***- starting index(row number) of the chunk.
2. ***end*** – ending index(row number) of the chunk.
3. ***thread_no*** – the index of the thread in the array of threads.

Experiment 1:

The CPU to be allotted a thread is set in the given manner:

The i_{th} ***b*** (= ***k***/***c***) threads have an affinity for the CPU with id ***i mod c***.

The time of execution is counted from the creation of the threads till they finish joining.

Experiment 2:

The CPU to be allotted a thread is set in the given manner:

The i_{th} thread has an affinity for the CPU with id ***i mod*** (***c***/***2***).

The time of execution is counted for the computation of the rows allotted the thread within the ***runner()*** function and stored in a global array: ***times[k]*** . Using this array we find the average execution time for the bounded and unbounded threads.

MIXED

Each thread receives a set of rows of size ***n/k***. In this case, the index of the rows received by a thread differ by the number of threads ***k***,

i.e., each thread receives row number ***i***, ***i+k***, ***i+2*k***, ….. , ***i + (n/k -1)*k***

where ***i*** can go from 0 to ***k***-1.

The arguments passed into the ***runner()*** function are:

1. ***start*** - the start index of the set of rows
2. ***max*** – the max index a row can have+1 (***n***)
3. ***jump*** – the number by which the index of rows jumps within a thread
4. ***thread_no*** – the index of the thread in the array of threads.

Experiment 1:

The CPU to be allotted a thread is set in the given manner:

The i_{th} ***b*** (= ***k***/***c***) threads have an affinity for the CPU with id ***i mod c***.

The time of execution is counted from the creation of the threads till they finish joining.

Experiment 2:

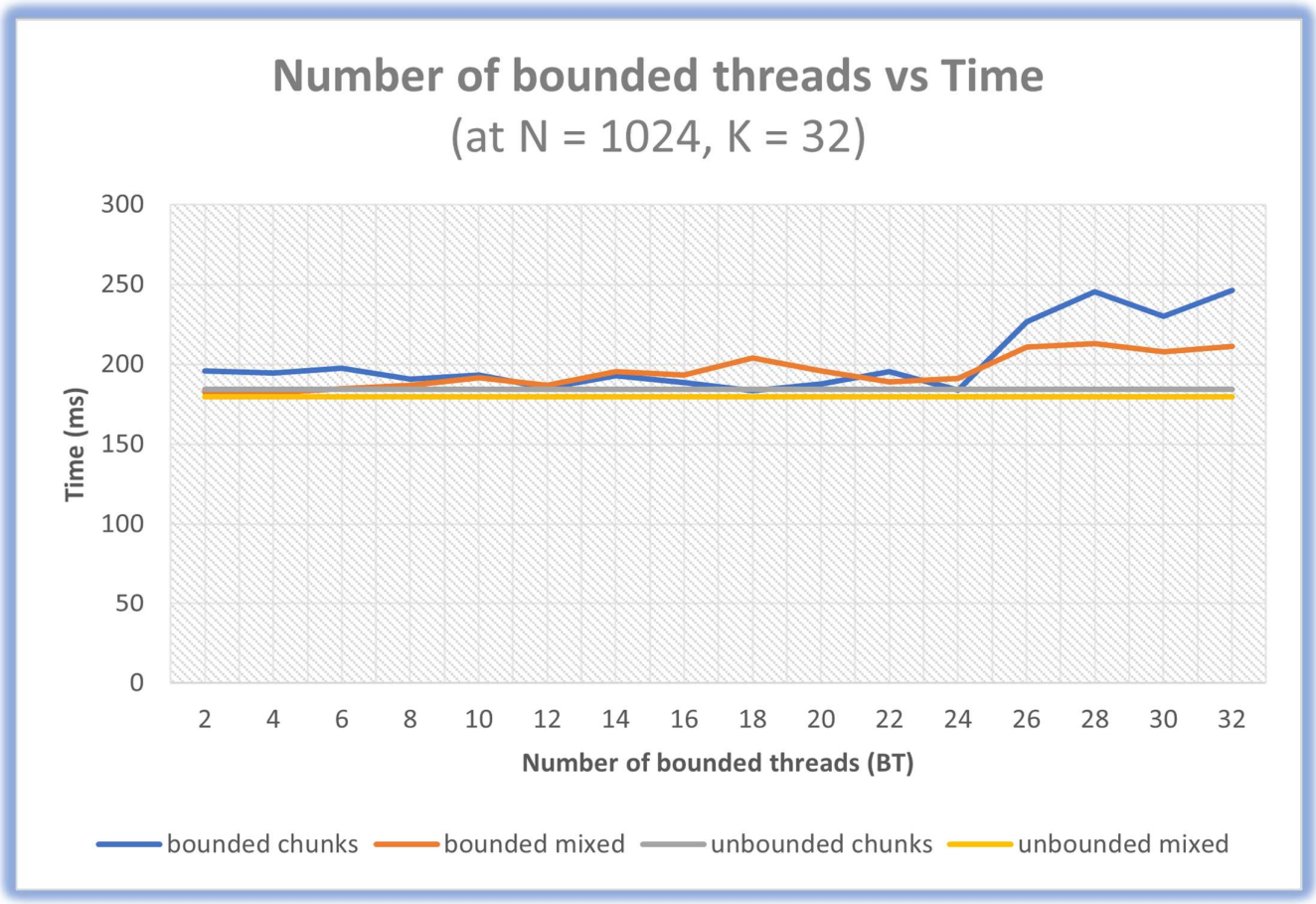
The CPU to be allotted a thread is set in the given manner:

The i_{th} thread has an affinity for the CPU with id ***i mod*** (***c***/***2***).

The time of execution is counted for the computation of the rows allotted the thread within the ***runner()*** function and stored in a global array: ***times[k]*** . Using this array we find the average execution time for the bounded and unbounded threads.

Performance of the programs

Time vs Number of Bounded Threads, BT

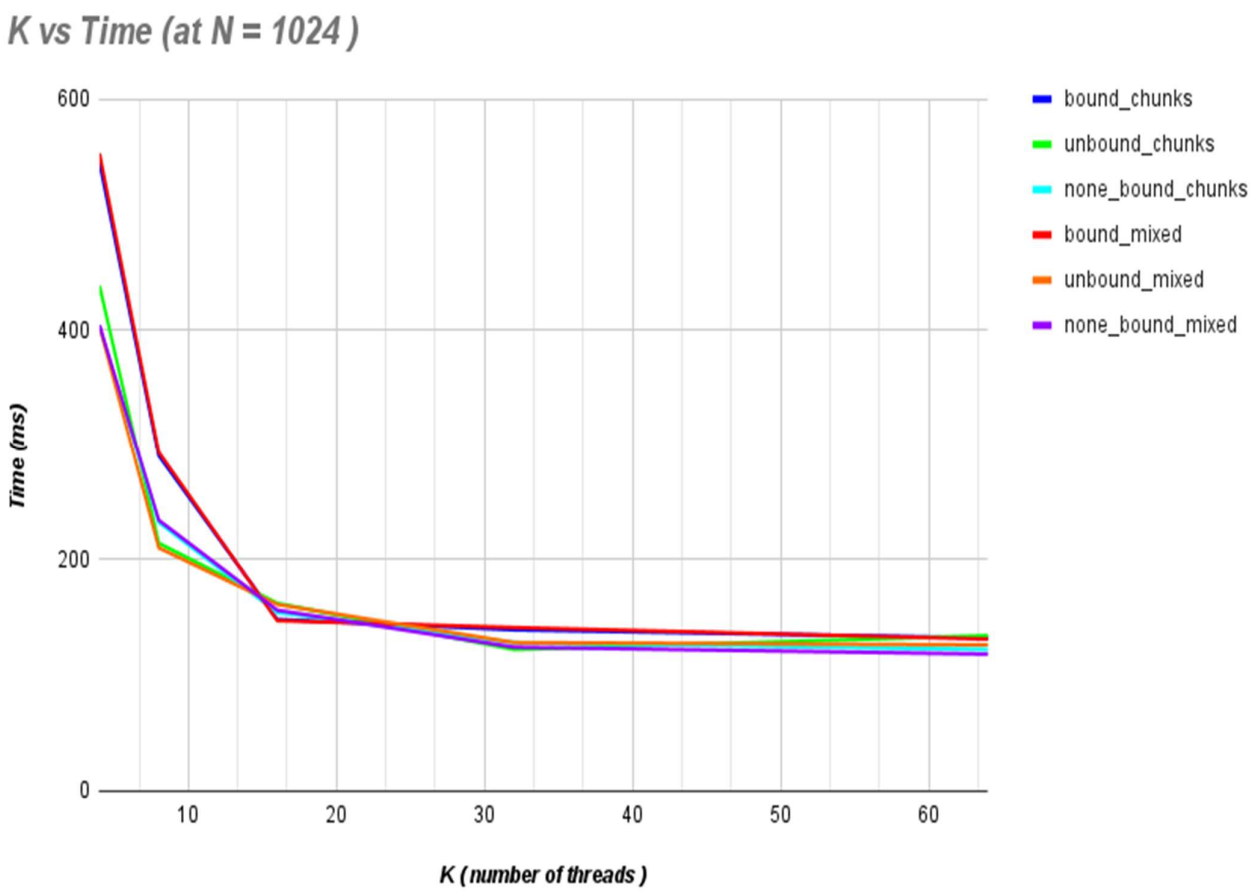


As we can see, as we increase the number of bounded threads the time of execution increases considerably for both the chunks and the mixed algorithms.

This could be due to the overhead caused by context switches that occur when the recently created thread is running on a CPU allotted by the OS scheduler and is then switched to another CPU when its affinity is set. This leads to an increase in the total time taken.

The times of execution are between 170ms to 250ms for all the points in the x-axis.

Time vs Number of threads, K



In this graph also, we can see that the times for bound threads are considerably higher compared to those of the unbound threads. This deviation is higher when the total number of threads is on the lower side.

The reason for this might be the same as in the previous graph. The context switch times after setting the affinities add up and increase the execution time.

The times for the ***chunks*** and ***mixed*** algorithms are almost the same, with the ***mixed*** algorithm performing slightly better for each case (bound, unbound, none bound).

The average times are around 550ms for ***k***=4 and decrease exponentially to around 125ms for ***k***=64 for each curve.