Operating Systems–2: Spring 2024
Programming Assignment 2 : Thread Affinity

- My laptop specifications:
  - 16 cpus.
  - 12th Gen Intel Core i5-1240P processor. And supports Hyper-Threading technology with 16 processors.
  - And each core have ability to handle to threads.

Low level design of my codes:

File:assign_chunk_src_co22btech11008.cpp, takes input N,K,C,BT and matrix A from input file (inp.txt).

This code creates K threads. And divide N rows among these K threads firstly. Then depending upon the value of BT it binds some threads to some no. of cores.

Ex. N=1024,K=32,C=16,BT=4;

Then each thread will execute (1024/32)32 rows multiplications. (Assigning rows to threads is done in a chunk manner. Like the first 1024/32 to first thread, next 32 to second thread and so on..)

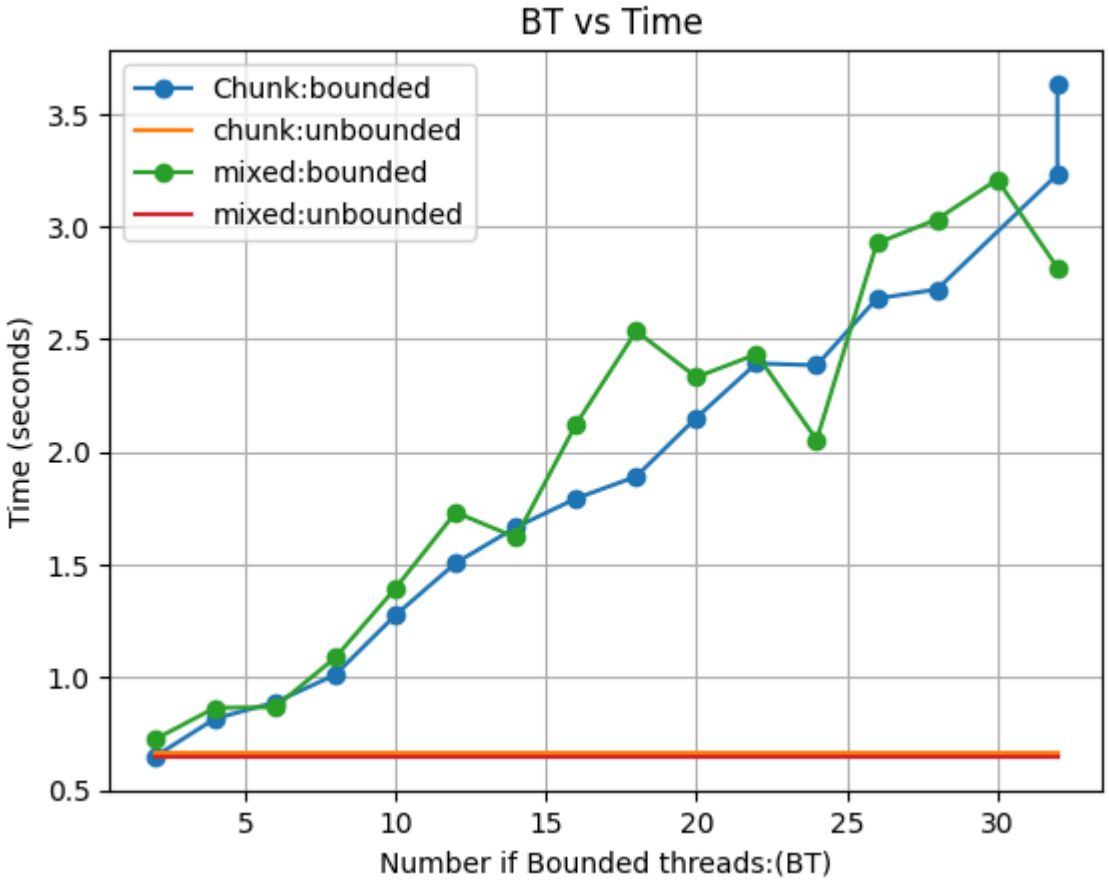
Now. as BT=4. then the first four threads will be bound to the first four cores. And the remaining 28 threads will be scheduled by OS scheduler.

file:assign_mixed_co22btech11008.cpp

Same thing happens in file assign_mixed_co22btech11008.cpp . Only assigning of rows is different .Here, the rows of the C matrix are evenly distributed among the threads. Thread1 will be responsible for the following rows of the C matrix: 1, k+1, 2*k+1 ....,. Similarly,
thread2 will be responsible for the following rows of the C matrix: 2, k+2, 2*k+2 ....,.
This pattern continues for all the threads.
Ex: N=10, k=4
So p=N/K=2 and . 4 threads.
So thread1 will do calculations for 1, 5 and 9.
Thread2 —> 2 , 6 and 10
thread 3—>3 and 7
Thread 4—>4and ,8

- **1st experiment**:

  Total Time vs Number of Bounded Threads, BT.



Experiment at N=1024,K=32, C=16,
And increasing the value of BT from 0,2 ,4,6 8…up to 32.
Time taken to complete multiplication is shown on the Y axis. And the number of threads bounded shown on the x axis.

index	chunk_data	mixed_data
0	0.66277	0.649938
2	0.647944	0.72443
4	0.816554	0.864806
6	0.889252	0.867364
8	1.01156	1.08763
10	1.27597	1.39542
12	1.50694	1.73197
14	1.66584	1.62058
16	1.79347	2.12253
18	1.89106	2.53621
20	2.15193	2.33191
22	2.39306	2.43366
24	2.38455	2.05376
26	2.68073	2.9261
28	2.72124	3.03229
30	3.23142	3.20862
32	3.63279	2.81806

In Graph:

Here:

Orange line is the chunk method at BT=0 that means.all threads will be scheduled by os. No threads are unbounded. In this case. Average time of execution for K=32.N=1024 , C=16 is 0.66277 seconds.

Red line is a mixed method at BT=0. and we can see it is taking minimum timing. Among all methods.

Blue line is a chunk bounded to 16 cores. At Bt=4, 4 threads to cores are bound to 4 cores. Remaining scheduled by os.

Green line is mixed bounded . 32 threads are bounded to 16 cores. At Bt=8, 8 threads to cores are bound to 8 cores. Remaining scheduled by os.

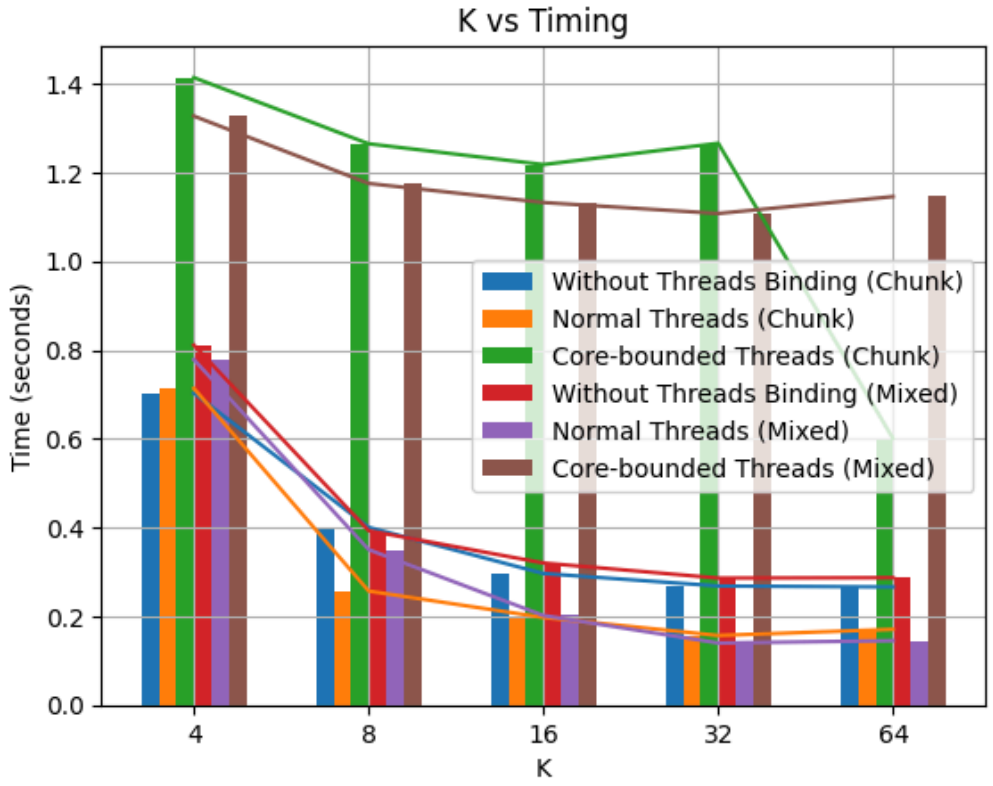
Comparing timing:
Time increases as no. of bounded threads increasing in both cases . and when no thread is bounded, time is minimum that is shown by horizontal lines.

Conclusion:

- Binding of threads decreases efficiency of the program.
- The efficiency decrease is evident from the increasing execution time when a higher number of threads are bound to coreS.
- In summary, the analysis suggests that, in this scenario, thread binding has a negative impact on program efficiency, and leaving threads unbounded initially results in faster execution times.

Experiment 2:

K	Without Threads Binding (Chunk)	Normal Threads (Chunk)	Core-bounded Threads (Chunk)	Without Threads Binding (Mixed)	Normal Threads (Mixed)	Core-bounded Threads (Mixed)
4	0.702867	0.713629	1.41396	0.810991	0.778427	1.327301
8	0.400449	0.256818	1.264835	0.392908	0.350219	1.175452
16	0.296389	0.197316	1.218	0.320068	0.201746	1.132502
32	0.268282	0.15673	1.265977	0.286154	0.139751	1.107753
64	0.266041	0.170609	0.600633	0.286984	0.145008	1.145801



.

Here :

All bars are decreasing with an increasing number of threads. That we can predict also. At K=32 and K=64 bars are almost the same as , my laptop has 16 cpus only and every cpu can handle 2 threads. So after 16 any increase in the number of threads will not increase performance much.

Chunk:
blue :Average time execution without threads being bound to cores - Chunk algorithm (as done
in the assignment1 experiments)

Green :Average time execution of Core-bounded threads - Chunk algorithm

Orange :Average time execution of Normal threads - Chunk algorithm

Here trend followed for most of K:
blue<= orange<=green.

Mixed:
red:Average time execution without threads being bound to cores - Mixed algorithm (as done
in the assignment1 experiments)

brown:Average time execution of Core-bounded threads - Mixed algorithm
Purple :Average time execution of Normal threads - Mixed algorithm

Here trend followed for most of K:
purple<= red <=brown.

Overall trend mostly:
blue<=purple<=orange<= red <=brown<=green

Conclusion:

- For the given experimental conditions, leaving threads unbound from specific cores generally leads to better performance in terms of execution time.
- The Chunk algorithm tends to exhibit lower execution times without thread binding compared to the Mixed algorithm.
- Core binding, especially in the Mixed algorithm, tends to have a noticeable impact on performance, resulting in higher execution times.