IntroPDP: Assignment 0

Shriram Ravi
shriramr
shriramr@buffalo.edu
50419944

Q1 – A brief:

We're implementing Stencil pattern on Z = A X K. Hence, I reduced (sum) the columns of A and the rows of K, and multiplied respective rows and columns, which gives sum of Zij. Not novel but a nimble solution to make the matrix multiplication calculation with a smaller number of steps.

Q2 – Performance Discussion:

Sequential:

- The implemented logic scaled up well across different sizes

Parallel:

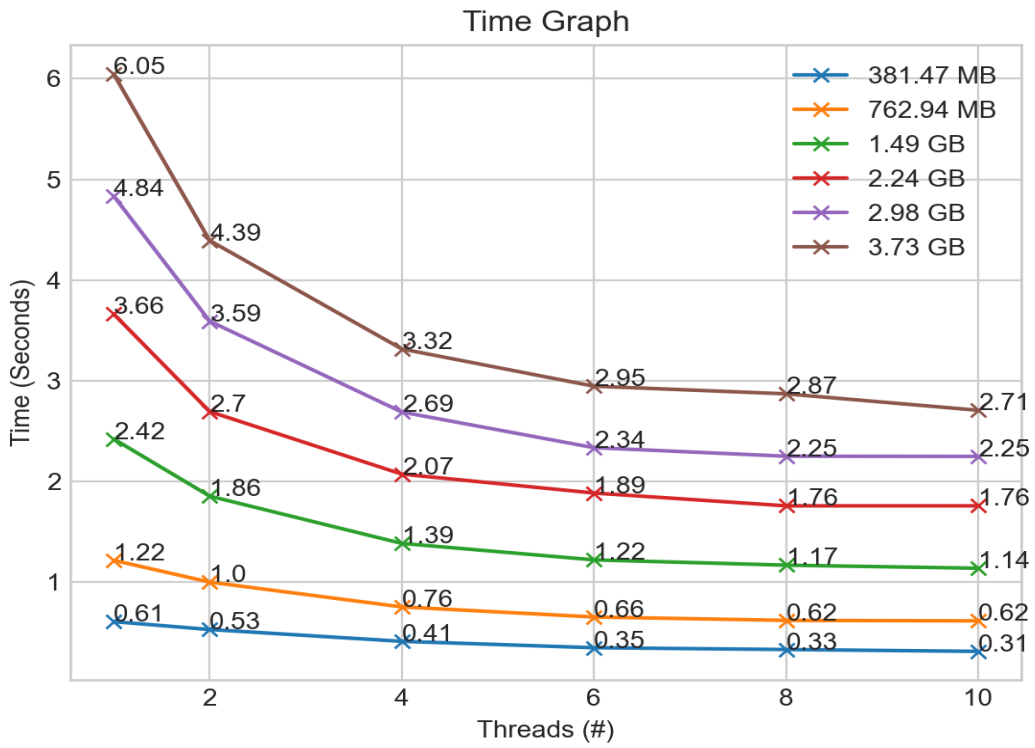
- The implemented logic scaled up decently with increase in number of threads for different sizes, with the OpenMP.
- The OpenMP constructs used in my implementation are ***parallel for*** along with ***static scheduling and collapse.***
- Below are the Table for observed time values with Time, Speedup and Efficiency graphs

Calculated times:

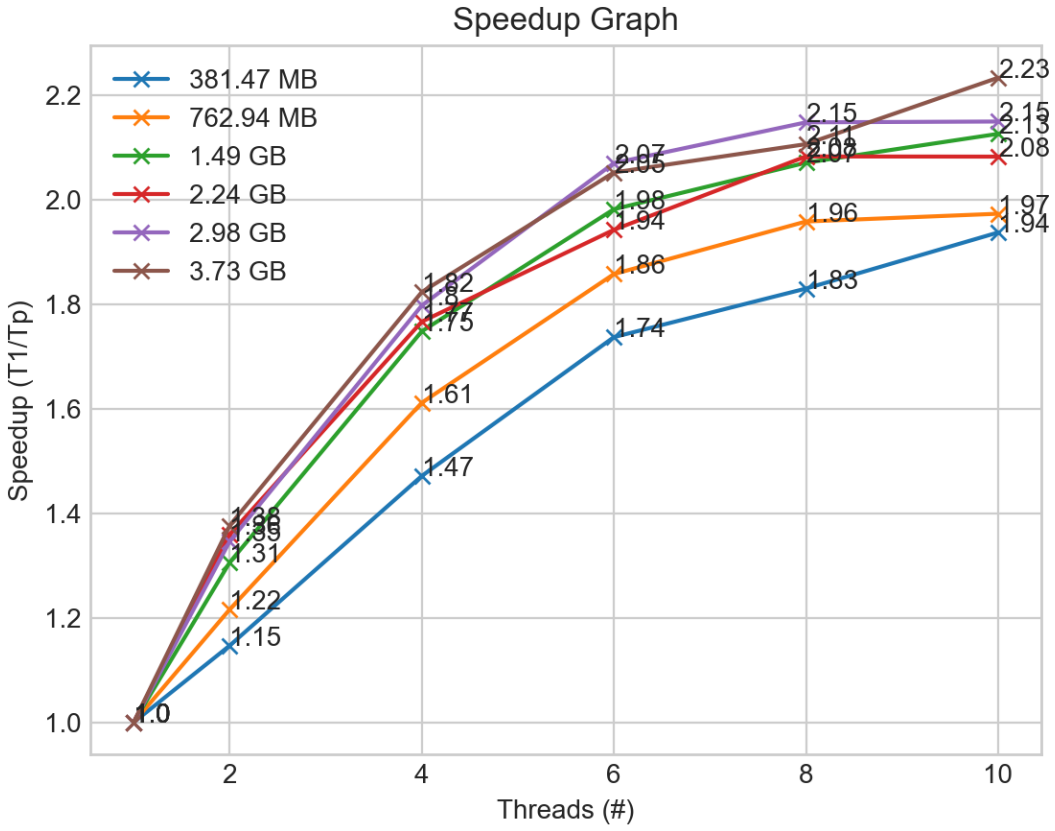
Size of matrix (* 100000) \ Threads	1	2	4	6	8	10
1000	0.61006	0.53189	0.41461	0.35123	0.3334	0.31494
2000	1.21897	1.00237	0.75648	0.65623	0.62253	0.61784
4000	2.42373	1.85599	1.38657	1.22316	1.1706	1.14019
6000	3.66437	2.6955	2.07441	1.88672	1.75956	1.7599
8000	4.83652	3.59255	2.69085	2.33679	2.25196	2.25028
10000	6.04682	4.39295	3.31671	2.94696	2.87119	2.70795

- Speedup goes max up to 2.23 times of the sequential runtime
- Efficiency goes max up to 68.82% while using 2 threads
- This isn't the case of the solution with best efficiency
- As the number of threads increases, the efficiency decreases but there's a speed up
- I suspect the dip in efficiency is because of swapping of memory to and from the cache for large amounts of data.
- With the increase in number of threads, there will also be thread-synchronization overheads.

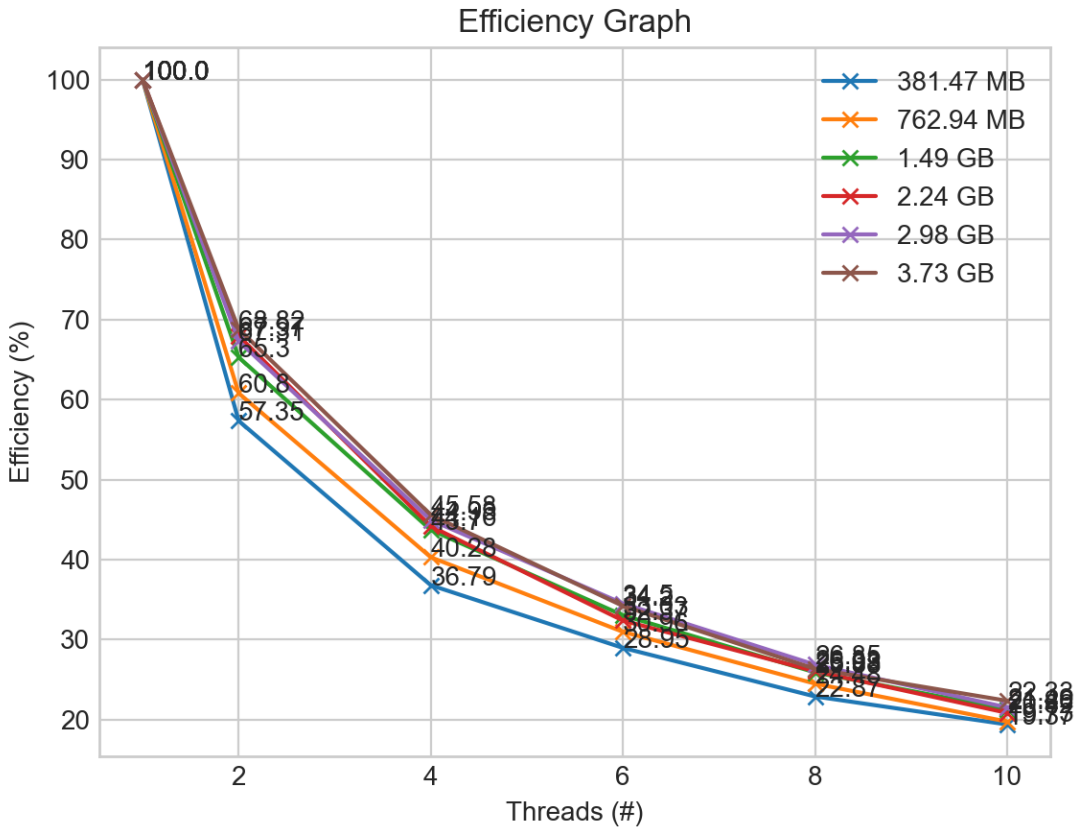
Time graph:



Speedup graph:



Efficiency graph:



Code Scalability:

Is it scalable?
Yes, this is scalable. As the number of threads increases, the time taken to arrive at the solution decreases.

Is this efficient?
No, this is not efficient. This is a straightforward solution for this problem which can be reduced even further.

How can we improve?
The nested loops to identify the index can be unfolded to a single loop which can be easily parallelized by OpenMP constructs. I've implemented that solution and attached the graphs below (Check Experiment section and attached log. This has not been submitted for grading since it had issues with correctness and lesser effective, convoluted sequential solution)

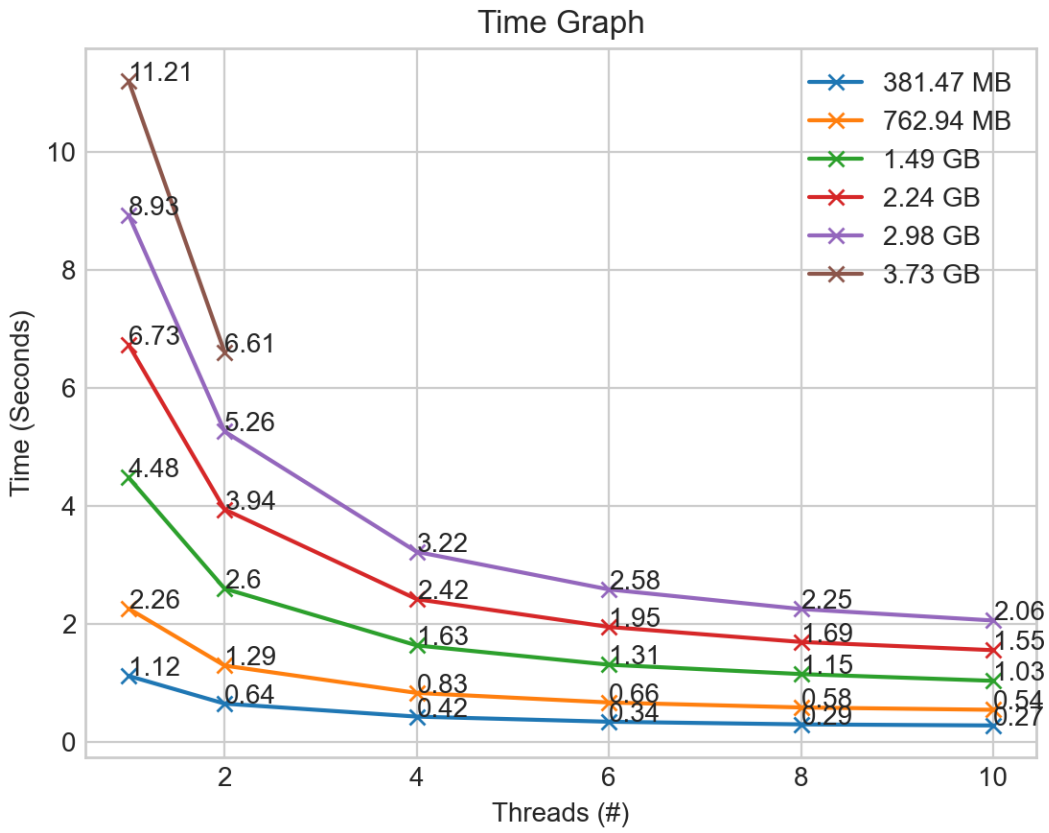
Is it always scalable?
Not always. As the efficiency graph suggests, this solution has poor efficiency after 10 threads. This can be due to the overheads and read/write to memory.
Also, it is comparatively less efficient with smaller sizes. This can be due to the thread-synchronization overhead which overshadows any performance gain.

Scalable in what sense?
As we increase the number of threads, the amount of time decreased. This is strongly scalable, but not with much efficiency.
This is not weekly scalable, which is apparent from the main diagonal in the table

What's causing it not to scale?
The nested loops with collapse in my use doesn't seem much effective.
Tried using tasks and task loop. It gave the same results as this.
Unfolding the loops into one gave better results.

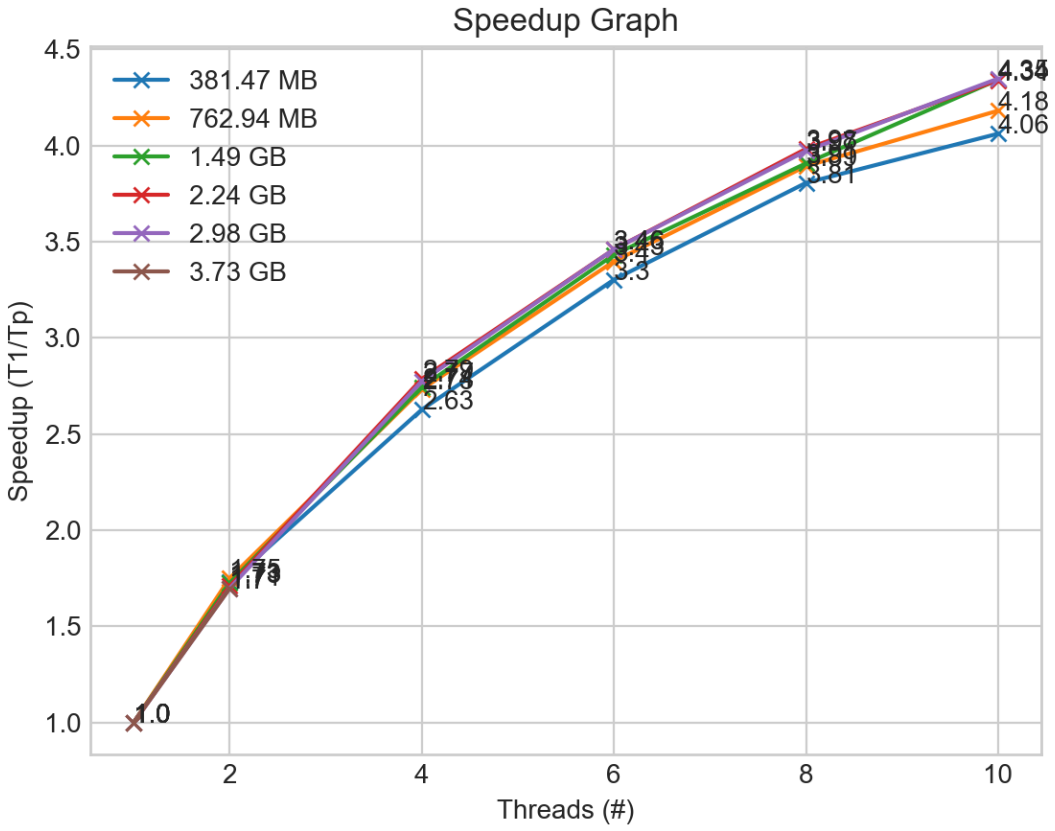
Experiments (Not for grading, supports few above mentioned claims):

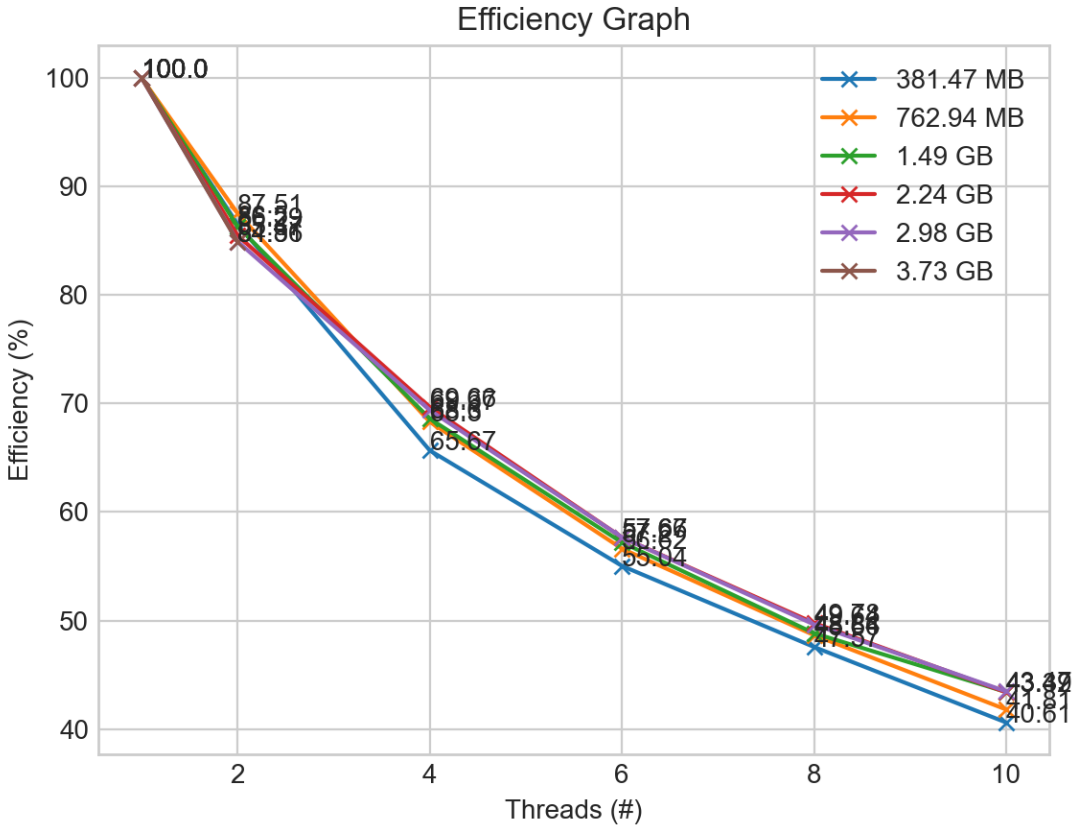
Time graph:
This is when the nested loops used in my solution are unfolded into a single loop.



Speed and Efficiency Graphs:

When the nested loops are unfolded the speedup gets 2x and the efficiency goes up by 20% for while using 10 threads.


Speedup Graph
381.47 MB
762.94 MB
1.49 GB
2.24 GB
2.98 GB
3.73 GB
Speedup (T1/Tp)
Threads (#)
1.0
2.63
3.3
3.81
4.06
4.18


Efficiency Graph
381.47 MB
762.94 MB
1.49 GB
2.24 GB
2.98 GB
3.73 GB
Efficiency (%)
Threads (#)
100.0
65.67
55.04
47.57
40.61