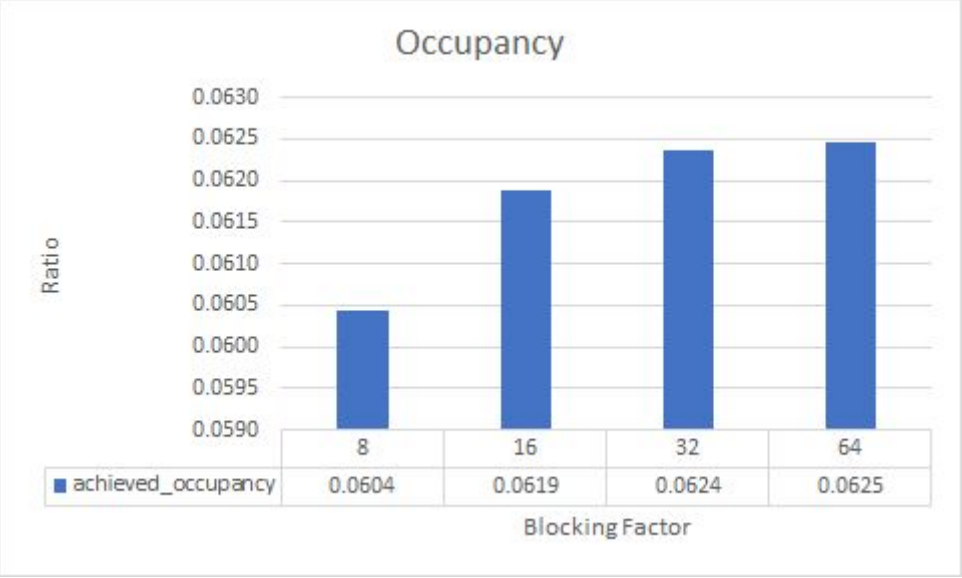
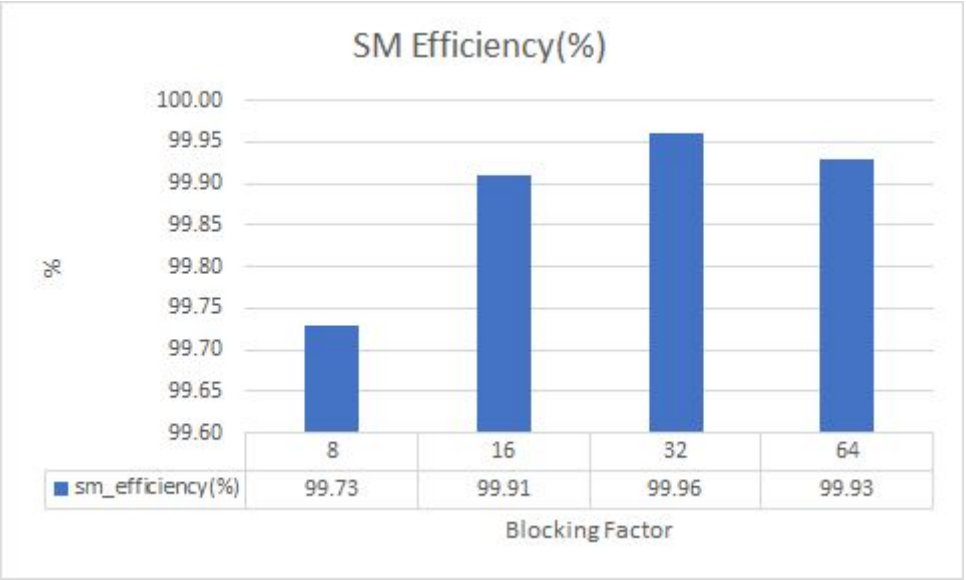
HW 4-1: Blocked All-Pairs Shortest Path¶

106033233 資工21 周聖諺

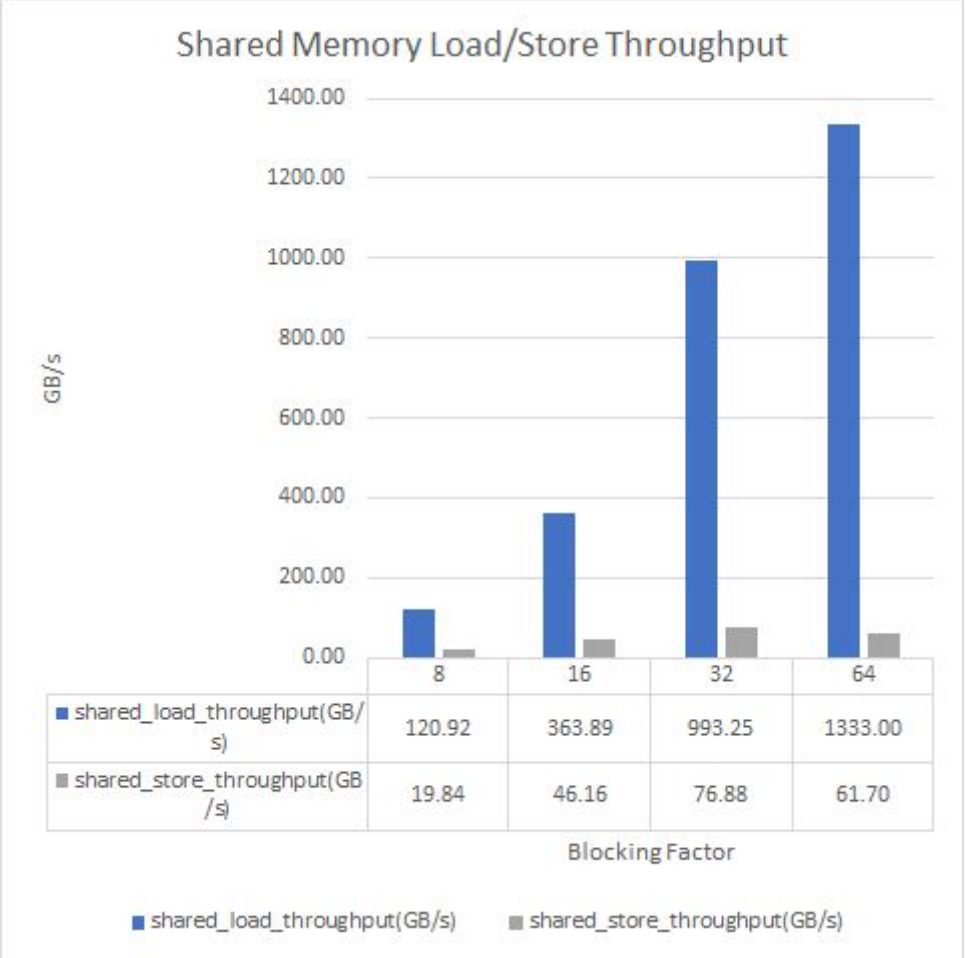
1. Implementation
    a. Shared Memory: Load block i-j, i-k, j-k, to share memory before path relaxing.
    b. Stream: Asynchronous execution of phase 2. It means it relaxes the path of the row and the column.
    c. Linear Address: I expand the shared memory into a linear array to keep column-major accessing.
    d. Huge kernels instead of small ones: The phase3 kernel relaxes the path of the whole matrix in phase 3 instead of relaxing the paths of 4 sub-matrices individually.
    e. Dynamic grid dimension: Assign 1 SM block for each block.
    f. Reverse row and column of block i-k: since phase3 will access the column of the block i-k, I reverse the column and the row to keep the row-major memory accessing and avoid bank conflict.
2. Profiling Results

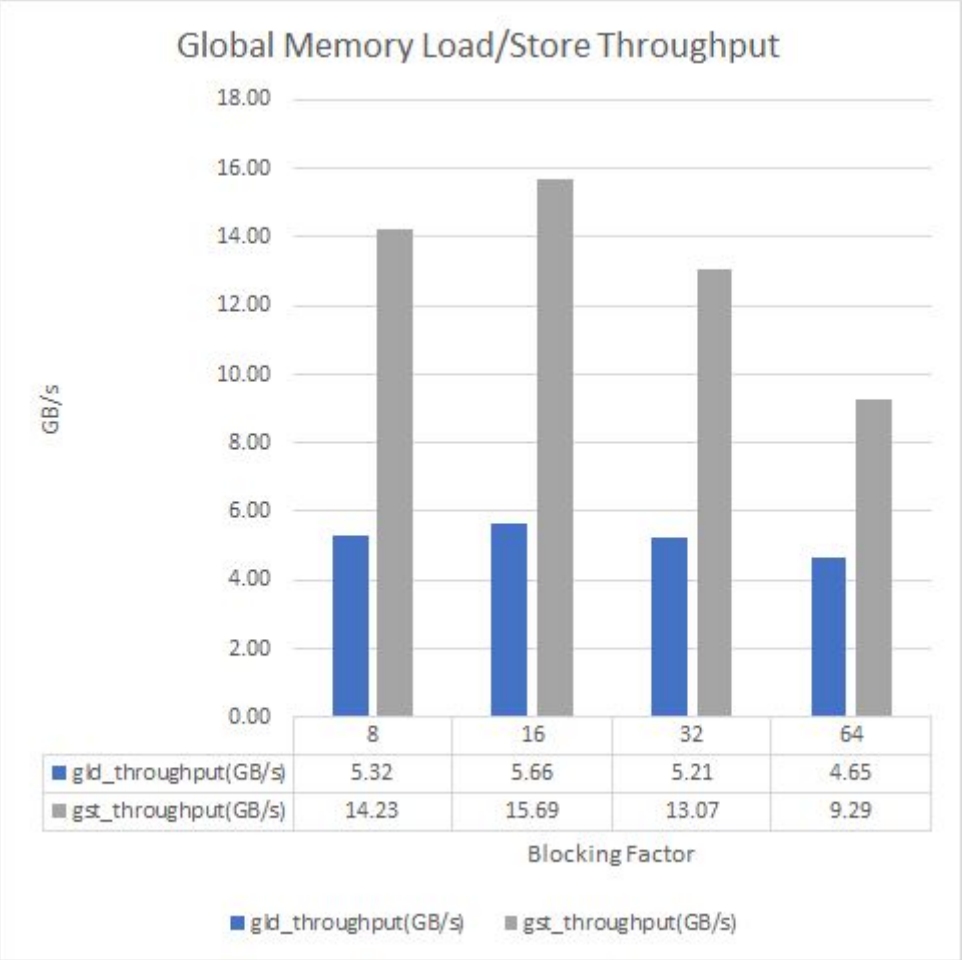
The biggest kernel of the program is 'phase3_cal_cuda' which calculates the phase3 of the block floyd warshall algorithm.





As the chart shows, the occupancy of the kernel is very low. However, the SM efficiency is very high, almost reaching 100%. The program might always execute several warps but it doesn't launch enough warps to fill the SMs.





The throughput of the shared memory gets higher while the blocking factor becomes larger. In the meantime, the throughput of the global memory gets lower.

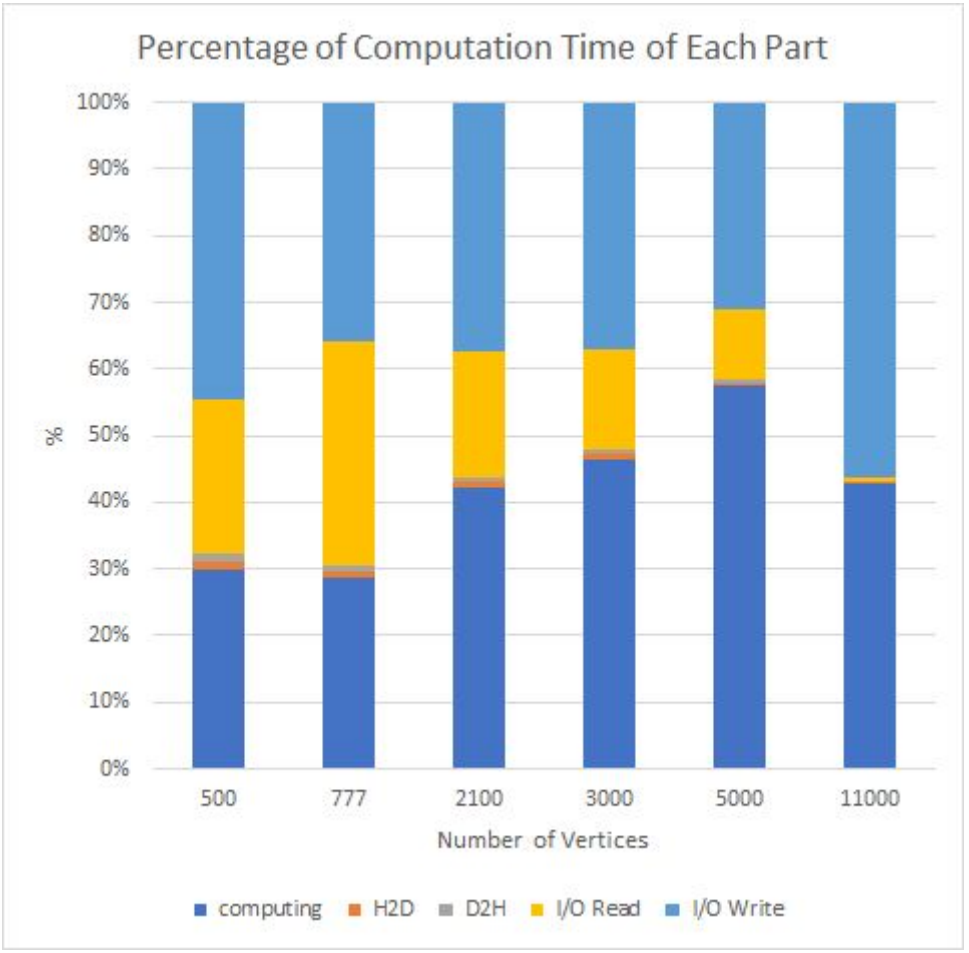
3. Experiment & Analysis

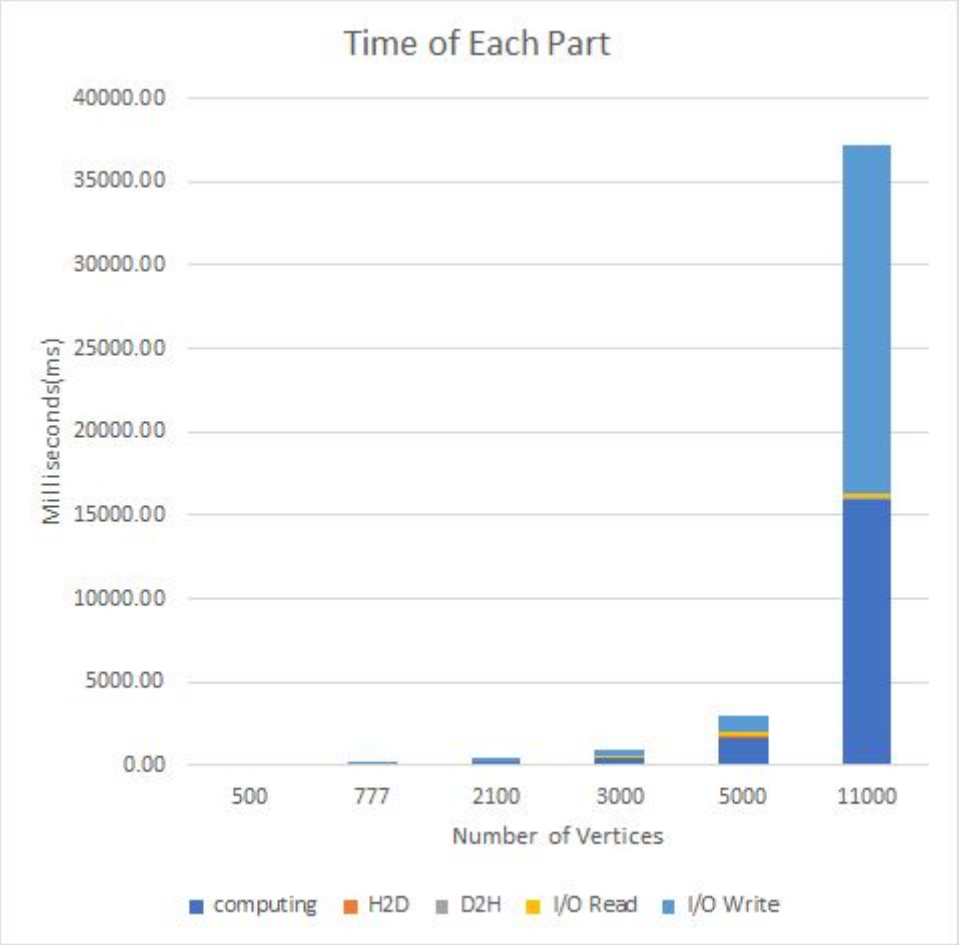
a. System Spec

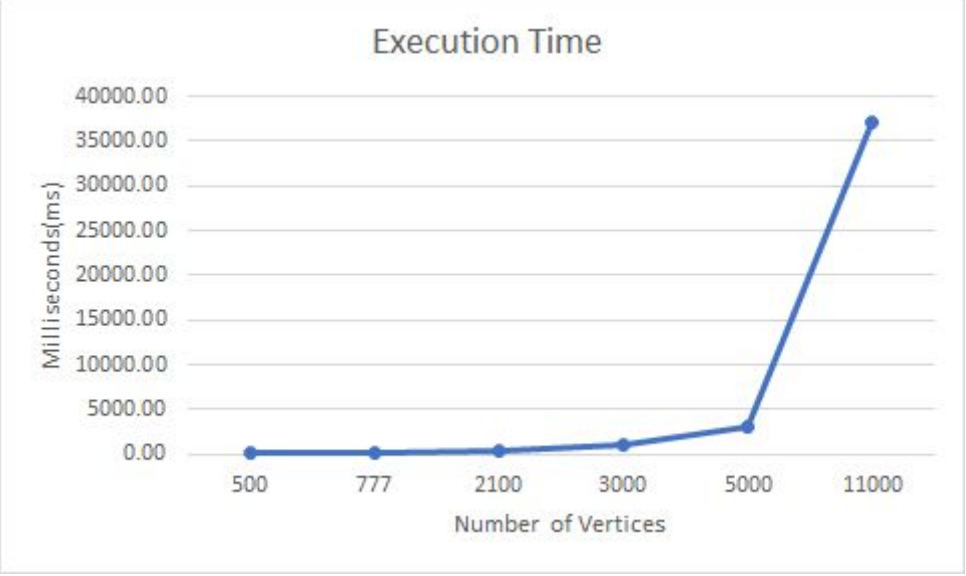
Hades, Gefore 1080TI * 1 with 8119MB ram

b. Time Distribution

I used the 'c10.1' 'c15.1' 'c17.1' 'c18.1' 'c20.1' 'p11k1' as test cases for the following result.


Percentage of Computation Time of Each Part
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
%
500
777
2100
3000
5000
11000
Number of Vertices
computing
H2D
D2H
I/O Read
I/O Write



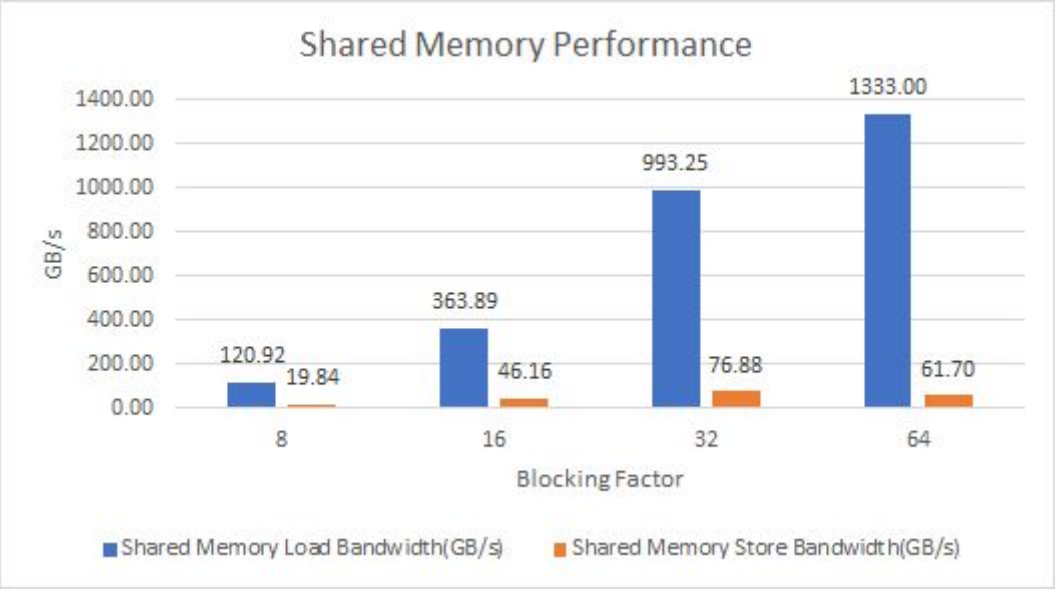


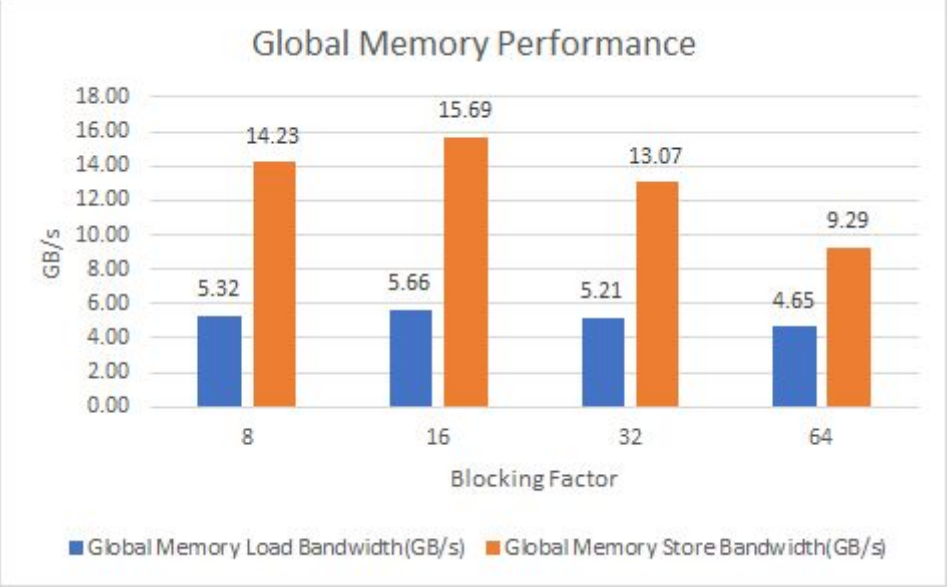
It is trivial that phase 3 and I/O writing consumes the most of the time(more than 75%) and phase 1 and 2 only take 25% of the time or less. The time of I/O writing takes more and more time while the input size gets larger. It even surpasses the time of computation. However,

the time of writing I/O is very weird with 11000 vertices that it grows over 20 times than the one with 5000 vertices.

c. Blocking Factor

I use c21 as the test case for the following result.

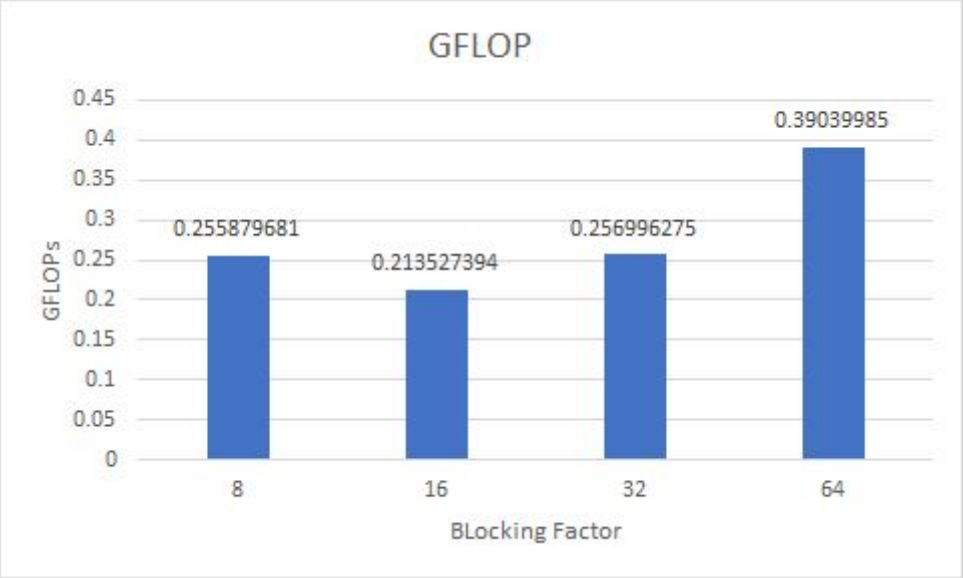




Due to Hades getting something wrong, I cannot profile the metric inst_integer and calculate the GFLOPs.

As for the throughput of shared memory and global memory, we can see a general trend that shared memory is much faster than global memory and memory load is faster than memory store. While the block factor grows, the kernel requires more shared memory to cache the state of the block and it requires fewer global memory loads / stores. However, the store throughput is much lower than the load throughput. It seems to be weird. It means that the bottleneck of the kernel is slow shared memory storing.

Since there is a time limitation, I use C10 as the test case.



It seems weird that the GFLOPs become lower with 16, 32 blocking factors but it is trivial that it has the highest GFLOPs at blocking factor 64.

d. Optimization

After using shared memory and stream, the performance of the program has a significant improvement. However, it seems that the shared memory version still has some issues. It should be faster.

e. Block Dimension

The block dimension is a critical factor for the performance. The best result is 2D block (8, 64) with blocking factor 64. When I change the dimension to (32, 32) or (16, 32) or any other dimensions, it gets much slower than the best one. Although the number of threads are the same.

4. Experience & conclusion

The homework is quite tricky and it needs lots of research and experiments to optimize the performance. It seems that the program still has some issues. It should be faster.