COMP 429/529 Parallel Programming: Project 2 Report

ENDI MERKURI, MUSTAFA ORKUN ACAR

1 Implementation

All 4 versions work correctly and the one with highest performance is the Version 3. We are participating in the competition with 621 GFlops for $N = 1024$ and $t = 500$.

1.1 Version 1: Single GPU

We divided this version into two kernels: *solveode* and *solvepde*. All references are made to global memory and reused memory locations are not stored in temporary variables.

1.2 Version 2: Kernel fusion

In this case, we have fused the kernels in the first version into one kernel to reduce the kernel launch overhead.

1.3 Version 3: Eliminating global memory references

In this version, we store the values of the reused elements in temporary variables and write to global memory only once at the end of the kernel execution.

1.4 Version 4: Using shared memory

This version creates an array in shared memory to allow faster access to the data without the need of accessing global memory. Firstly, each of the threads copies one element of data inside its data block to shared memory. After that, the threads that are at the borders of the thread block, also copy the ghost cells from global memory to shared memory. Because of these operations, one thread makes at most 5 global memory reads in order to copy the values to shared memory. Besides these global memory reads, in this version, there are 5 more shared memory reads, to read the values that we copied. On the other hand, in the third version, each threads only reads 5 elements from global memory. Because of the added shared memory reads, and the thread block synchronization overhead, our implementation of the fourth version is performing worse than that of the third version.

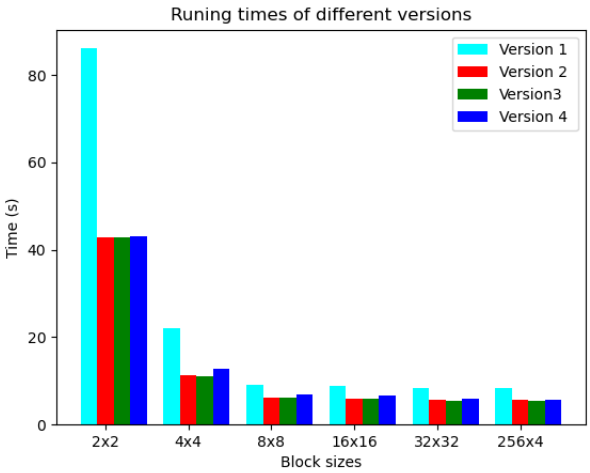
2 Experiments

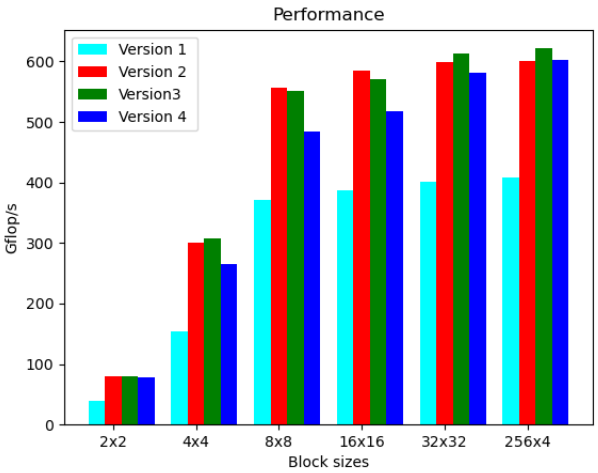
The serial version on the CPU takes 2286 s, reports 1.48 GFlops and a bandwidth of 1.7 GB/s. The first graph below presents the the running times of all 4 versions using different block sizes. We experimented with square and rectangular block sizes. The best performance we could achieve was with version 3 and blocks of 256x4 threads. The 4th version is performing slightly worse than the third one because in our implementation we are reading the same number of times from global memory as in the third version.

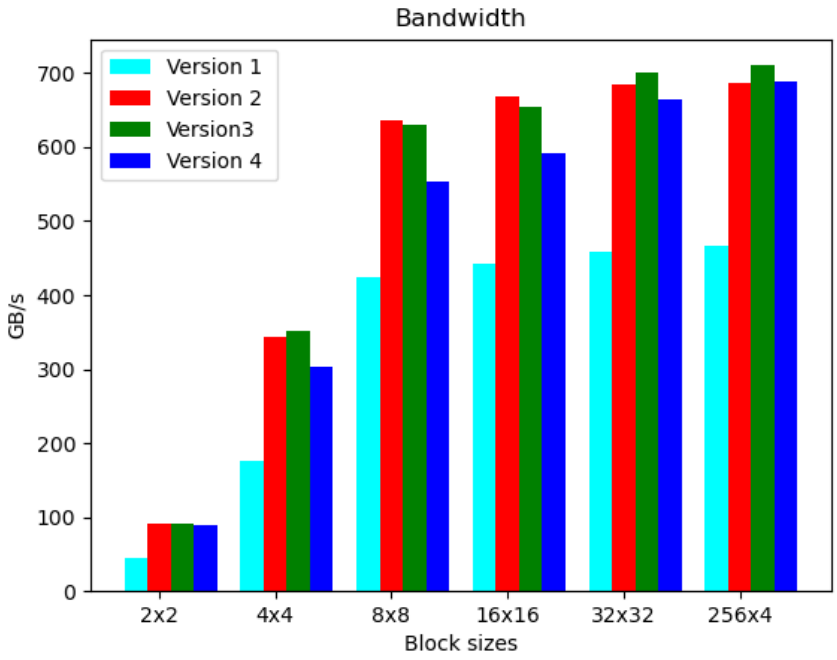
The second graph shows the GFlop rate compared to the block sizes for each of the versions. In this graph the same results are shown as in the first one.

The last graph reports the bandwidth in GB/s in relation to the block size for each of the implemented versions. The theoretical peak bandwidth of the V100 GPU is 900GB/s. We were able to achieve up to 710GB/s.

We observed the highest improvement in performance from Version 1 to Version 2 as it can be seen from the graphs below.


Runing times of different versions
Version 1
Version 2
Version3
Version 4
Time (s)
0
20
40
60
80
2x2
4x4
8x8
16x16
32x32
256x4
Block sizes


Performance
Version 1
Version 2
Version3
Version 4
Gflop/s
0
100
200
300
400
500
600
2x2
4x4
8x8
16x16
32x32
256x4
Block sizes


Bandwidth
Version 1
Version 2
Version3
Version 4
GB/s
0
100
200
300
400
500
600
700
2x2
4x4
8x8
16x16
32x32
256x4
Block sizes