CME213 Spring 2023 Homework 4

Ines Dormoy – idormoy@stanford.edu

May 25, 2023

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Question 1

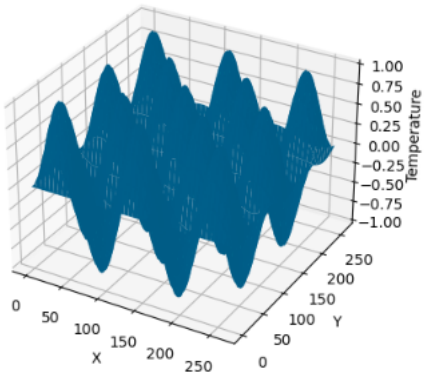

Figure 1: Plot for 0 iterations with 256x256 grid size and 8 order

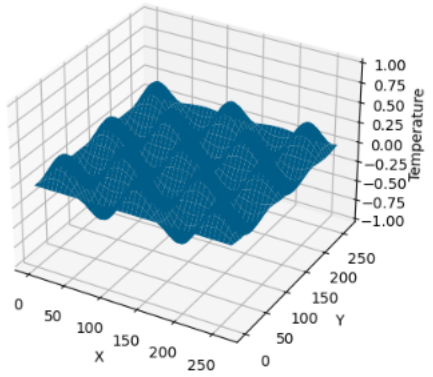

Figure 2: Plot for 1000 iterations with 256x256 grid size and 8 order

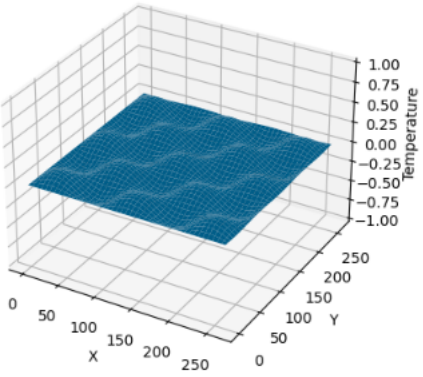

Figure 3: Plot for 2000 iterations with 256x256 grid size and 8 order

Question 3.1

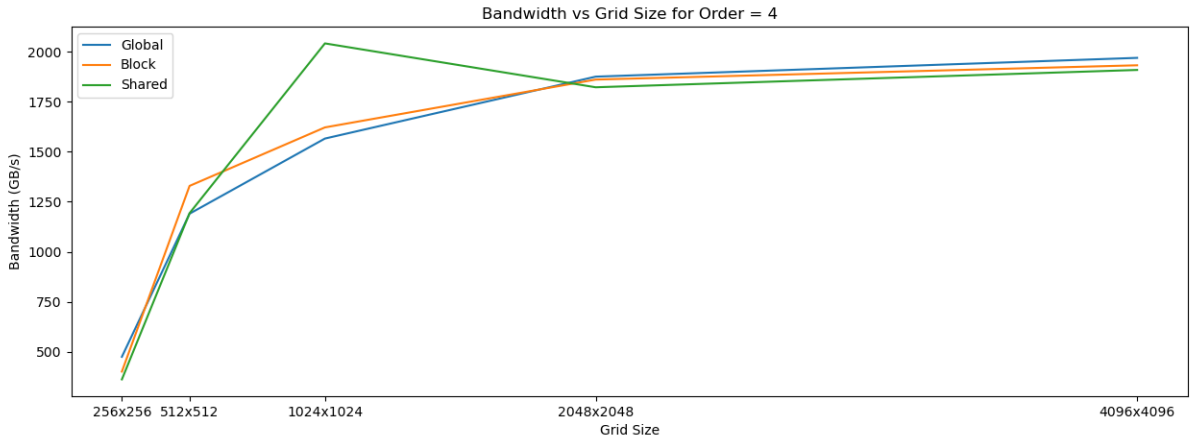

Figure 4: Bandwidth for the 2 different algorithms, order 4

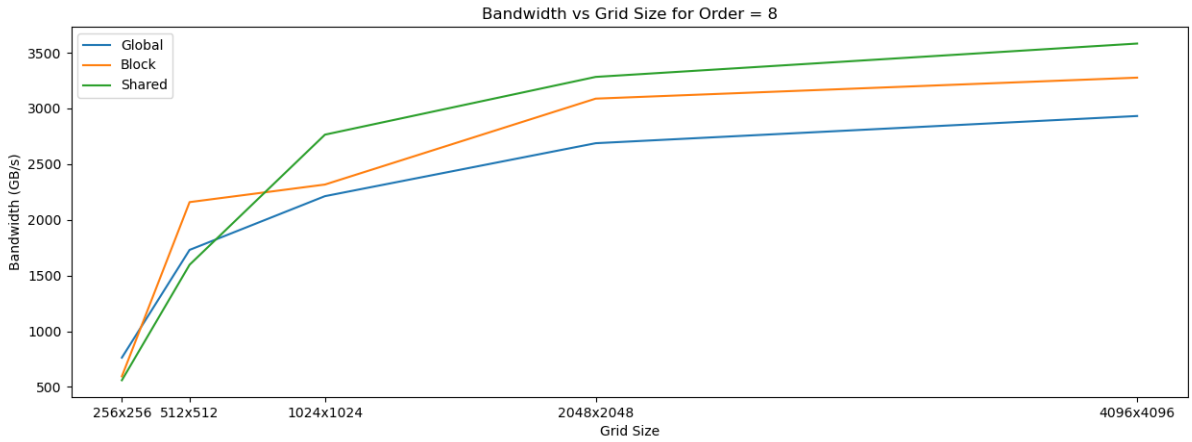

Figure 5: Bandwidth for the 2 different algorithms, order 8

Global vs Block Interestingly, the superiority of block implementation in terms of bandwidth is not consistent. When considering an order of 8, the block implementation consistently outperforms or matches the global implementation. However, for an order of 4, the global implementation proves to be better when working with grid sizes larger than 1024. This observation can be explained by the fact that increasing the value of numYPerStep (in this case, I chose 8 as the hyperparameter) actually hampers performance. With numYPerStep computations being performed by a single thread, the number of operations increases, potentially leading to decreased efficiency. Nevertheless, the performance benefits from improved memory access, as a thread accessing coalesced locations while performing numYPerStep operations optimizes memory utilization. Thus, there exists a trade-off between the number of operations performed (compute bound) and the efficiency of memory access (memory bound). For an order of 8, the block implementation is superior due to

its increased memory access, which benefits from performing numYPerStep computations per thread. However, for an order of 4, the focus shifts towards compute-bound operations, favoring the global implementation.

Shared When considering an order of 8, the shared implementation consistently proves to be the most efficient for grid sizes larger than 1024. However, for smaller grid sizes, the overhead involved in setting up the shared memory becomes a limiting factor, preventing the shared algorithm from outperforming the block and global algorithms. On the other hand, when working with an order of 4, the shared algorithm outperforms the other algorithms for a grid size of 1024. Nevertheless, its performance gradually declines starting from 2048, with all three algorithms exhibiting similar performance beyond that point. This can be attributed to the fact that the problem size is not sufficiently large to yield significant differences among the algorithms, and the results heavily rely on the choice of hyperparameters and grid implementation.

Question 3.2

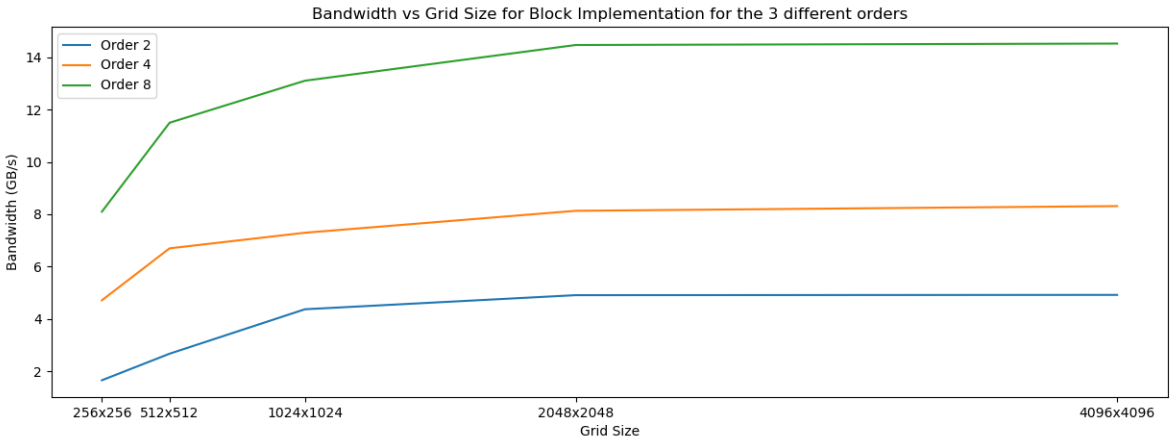

Figure 6: Bandwidth for the 3 different orders, block implementation

We observe that the bandwidth increases with the order as more coalesced accesses are made in the memory and the memory bound decreases.

Question 3.3

As for the block algorithm, the bandwidth increases with the order in the case of the shared memory algorithm. The performance drops between grid sizes 1024 and 2048, probably because of less compatible sizes of the shared memory implementation with 2048 grid sizes.

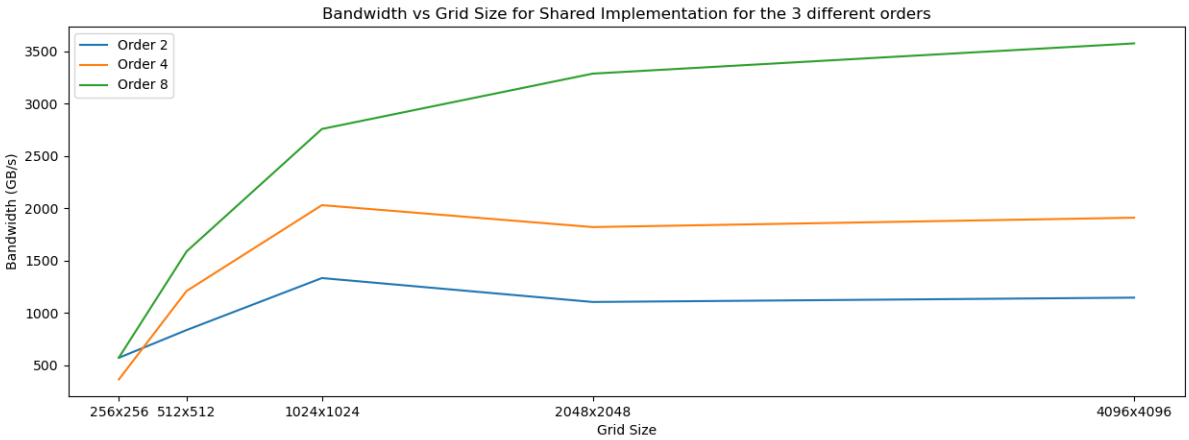

Figure 7: Bandwidth for the 3 different orders, shared implementation

Question 4

Kernel In general, the shared kernel demonstrates superior performance for an order of 8. However, when considering an order of 4, all three kernels exhibit similar performance, indicating that the choice of hyperparameters heavily influences the results.

Performance differences among kernels In theory, the shared algorithm should outperform both the block and global algorithms. This trend is indeed observed for the order 8 stencil. However, for smaller stencils, the advantages offered by the shared and block algorithms are less pronounced.

Performance differences varying the order For all algorithms, increasing the order leads to higher bandwidth due to increased cache hits. Accessing elements multiple times from the cached memory enhances performance.

Performance differences varying the problem size As the problem size increases, the bandwidth also improves. This is primarily because a greater number of threads are utilized, enabling simultaneous computation of more operations. The performance continues to increase until it reaches a plateau, reflecting the maximum compute capacity.