CME 213 - Parallel Computing
homework 4

Leon H. Kloker
Institute for Computational and Mathematical Engineering
CME 213, Stanford University
leonkl@stanford.edu

Question 1

```
Order:  8, 4096x4096, 2000 iterations
time (ms) GBytes/sec
CPU 55876.7 43.2366
[==========] Running 3 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 3 tests from DiffusionTest
[ RUN ] DiffusionTest.GlobalTest
Order:  8, 4096x4096, 2000 iterations
time (ms) GBytes/sec
Global 17706.1 136.446
L2Ref LInf L2Err
0.054971 0 0
[ OK ] DiffusionTest.GlobalTest (27131 ms)
```

Question 2

```
[ RUN ] DiffusionTest.BlockTest
Order:  8, 4096x4096, 2000 iterations
time (ms) GBytes/sec
Block 754.633 3201.45
L2Ref LInf L2Err
0.054971 0 0
[ OK ] DiffusionTest.BlockTest (10046 ms)
```

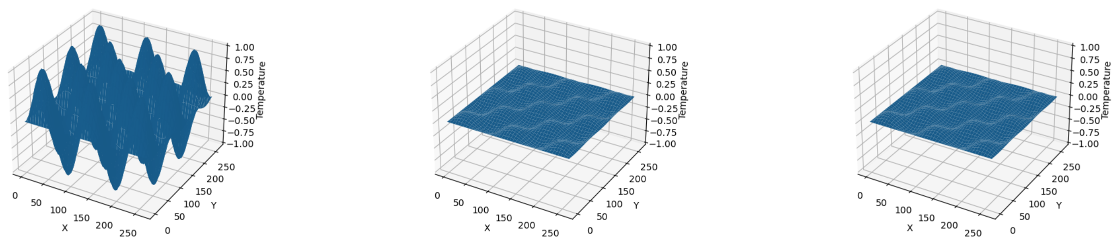

Figure 1: From left to right: Temperature field at iteration 0, 1000 and 2000.

Question 3

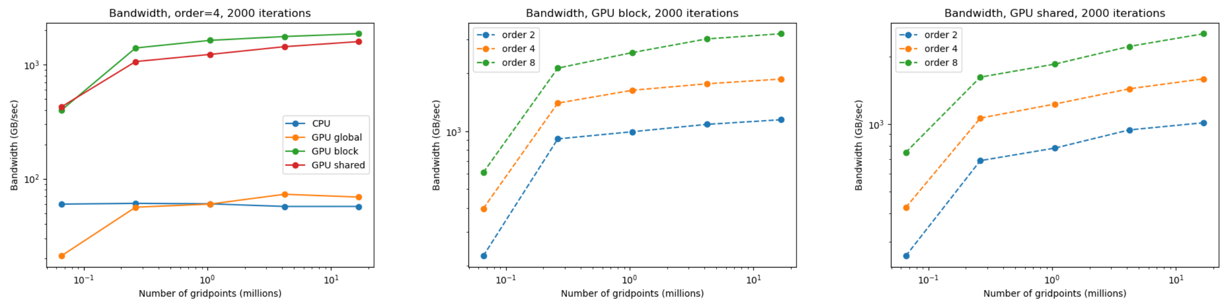

Figure 2: From left to right: 1) Bandwidth over the number of gridpoints for the CPU, GPU global, GPU block and GPU shared memory implementations. 2) Bandwidth over the number of gridpoints for the GPU block implementation for order 2, 4 and 8. 3) Bandwidth over the number of gridpoints for the GPU shared memory implementation for order 2, 4 and 8.

Question 4

The GPU block and shared kernels have a far better performance than the GPU global kernel and CPU for all grid sizes as well as for all orders of the finite differences. The block implementation has a slight edge over the shared implementation for large grid sizes, whereas the shared implementation is slightly better for a small grid of 256x256.

We also expect the global GPU kernel to perform worse than the other two as the memory access in this implementation is not coalesced as the stencil always needs access to the temperature value above and below which are far away from each other in the array.

In the block implementation, the temperature values above, below and even left and right are at least partially reused by the threads in the same block which leads to less memory access that is wasted.

The shared memory implementation first loads all the values needed for the calculation of thread block into shared memory and then makes use of the low latency when the derivatives are evaluated, also leading to a high bandwidth.

When the order is increased, every stencil needs access to more neighbouring gridpoints. As these values have already been loaded into either cache or shared memory in the block and shared implementation, there is not a lot of additional global memory access needed, such that the overall bandwidth increases as visible in figure 2.

Moreover, when the problem size is increased, the proportion of the computational overhead is decreased and the bulk behaviour of the kernel dominates the overall bandwidth. We can see that this leads to an increase in bandwidth for the GPU block and shared implementation, whereas the bandwidth of the GPU global implementation increases only slightly.

Question 5

```
[ RUN ] DiffusionTest.SharedTest
Order:  8, 4096x4096, 2000 iterations
time (ms) GBytes/sec
Shared 969.025 2493.14
L2Ref LInf L2Err
0.054971 0 0
[ OK ] DiffusionTest.SharedTest (9919 ms)
```