Optimization and Vectorization – Assignment 2 Report

Students: Gianmarco Picarella (2713810), Joel Brieger (1550942)

Data layout used:

- ***Task 1,2:*** For the SIMD tasks, we used an odd-even subdivision of the entire grid along the horizontal axis. First, we converted the layout from an AoS to a SoA; then, we subdivided each property into two separate arrays based on the location of the point along the x axis. The rest lengths are grouped by direction in groups of 4 (4 right, 4 left, 4 bottom, 4 top) in an array with its size being a multiple of 16.
- ***Task 3,4:*** For the GPGPU tasks, we used a SoA data layout and a list of indices (more on that in the next section) to drive the computation and constrain it based on the data dependencies for each grid point.

Completed Tasks:

- ***Task 1,2:*** Given our subdivision between odd and even points, we perform a two pass (odd/even) update for each row in the grid using SIMD to fetch 4 grid points and then process 4 neighbours along the same direction each time. This way, we never process the same point simultaneously. We access the neighbours in a convenient way by shifting along the odd/even arrays: for example, the right neighbours for 4 points at odd positions [j, j+1, j+2, j+3] can be easily collected by shifting the array of even positions to the right by 1 [j+1, j+2, j+3, j+4] and so on.
- ***Task 3,4:*** In the GPGPU task we process all the points along the same diagonal simultaneously. We start from the top-left diagonal and process all the grid. Each diagonal waits for the previous one to be done before executing. We did not reorganize the data in a diagonal fashion but just provided a list of 256*2 rows (the number of diagonals in a squared matrix with side 256) and 256 columns (the longest diagonal possible) to the GPU. The row with index j contains the list of indices representing all the elements in the diagonal with index j starting from the top-left corner of the grid. We run a kernel with 256x1 work items and a work group of 256x1 elements. Each work-item will process all the valid indices found at its threadIdx. Of course, especially in the first and last diagonals, most threads will be idle, waiting for the working threads to be done. A huge speed-up is obtained around the middle point of the grid where almost all threads are occupied.

Benchmarks:

- ***Baseline***: Task 1: 205186 ns, Task 2: 6122566 ns
- ***SIMD***: Task 1: 102303 ns (2x speedup), Task 2: 2920272 ns (2.1x speedup)
- ***GPGPU***: Task 1: 7446 ns (27.5x speedup), Task 2: 13132 ns (466x speedup without measuring the device2cpu final copy)