Shallow Water: Initial Report

Group Number: 2
Members: Nimit Sohoni (nss66), Edward Tremel (ejt64), Ian Vermeulen (iyv2)

Optimizations used or attempted:

Restructuring and annotation to enable greater vectorization, informed by results from profiling
Parallelization with OpenMP

More details:

1. Vectorization
    a. Used compiler directives such as '#pragma ivdep' to encourage the compiler to vectorize loops and ignore possible dependencies that did not actually exist.
    b. Attempted to restructure the code to make better use of memory locality and vectorization.
2. Parallelization
    a. Split the simulation grid into rectangular blocks based on the number of threads available.
        i. For an a-by-b grid with p threads, the block size would be floor(a/√p)-by-floor(b/√p). Some cells may not covered by any blocks, so those are processed separately. However, the number of these cells should be small relative to the entire grid size, so should not affect performance too much.
        ii. We will also have k layers of ghost cells for each block, so that each thread can compute k time steps before communication is ncessary. k is a parameter that we will vary to figure out which value gives the best performance.

Results:

When we ran the amplxe-cl profiling tool on the default dam_break test case, we found that by far the bottleneck was the limited_derivs function.

```
amplxe: Executing actions 50 % Generating a report         Function     Module    CPU Time    Spin Time      Overhead Time
-------------------------------------------------------  ------------  --------  ---------  -----------     -------------
Central2D<Shallow2D, MinMod<float>>::limited_derivs         shallow           1.148s           0s                 0s
Central2D<Shallow2D, MinMod<float>>::compute_step           shallow           0.659s           0s                 0s
Central2D<Shallow2D, MinMod<float>>::compute_fg_speeds      shallow           0.230s           0s                 0s
[Outside any known module]                                  [Unknown]         0.021s           0s                 0s
_IO_file_xsputn                                             libc-2.12.so      0.015s           0s                 0s
_IO_fwrite                                                  libc-2.12.so      0.010s           0s                 0s
Central2D<Shallow2D, MinMod<float>>::solution_check         shallow           0.004s           0s                 0s
SimViz<Central2D<Shallow2D, MinMod<float>>>::write_frame  shallow             0.004s           0s                 0s
Central2D<Shallow2D, MinMod<float>>::Central2D              shallow           0.003s           0s                 0s
Central2D<Shallow2D, MinMod<float>>::run                    shallow           0.001s           0s                 0s
std::array<float, (unsigned long)3>::operator[]             shallow           0.001s           0s                 0s
Central2D<Shallow2D, MinMod<float>>::offset                 shallow           0.001s           0s                 0s
do_lookup_x                                                 ld-2.12.so        0.001s           0s                 0s
```

After we made some changes to improve vectorization, we were able to speed up the compute_step and compute_fg_speeds functions slightly:

```
amplxe: Executing actions 50 % Generating a report          Function     Module    CPU Time    Spin Time      Overhead Time
--------------------------------------------------------  ------------  --------  ---------  -----------     -------------
Central2D<Shallow2D, MinMod<float>>::limited_derivs          shallow           1.149s           0s                  0s
Central2D<Shallow2D, MinMod<float>>::compute_step            shallow           0.548s           0s                  0s
Central2D<Shallow2D, MinMod<float>>::compute_fg_speeds       shallow           0.130s           0s                  0s
[Outside any known module]                                   [Unknown]         0.017s           0s                  0s
_IO_file_xsputn                                              libc-2.12.so      0.012s           0s                  0s
_IO_fwrite                                                   libc-2.12.so      0.010s           0s                  0s
Central2D<Shallow2D, MinMod<float>>::solution_check          shallow           0.005s           0s                  0s
SimViz<Central2D<Shallow2D, MinMod<float>>>::write_frame   shallow             0.005s           0s                  0s
Central2D<Shallow2D, MinMod<float>>::Central2D               shallow           0.002s           0s                  0s
Central2D<Shallow2D, MinMod<float>>::run                     shallow           0.002s           0s                  0s
std::array<float, (unsigned long)3>::operator[]              shallow           0.002s           0s                  0s
Central2D<Shallow2D, MinMod<float>>::offset                  shallow           0.001s           0s                  0s
do_lookup_x                                                  ld-2.12.so        0.001s           0s                  0s
```

As can be seen, both the compute_step and compute_fg_speeds functions take about 0.1 seconds less time after vectorization is used. These numbers were not just coincidences or artifacts, but held up after repeated trials.

The #pragma ivdep directive was helpful in loops where vectorization was only being held back by the compiler's prior inability to assume that the memory locations being modified and the memory locations being read did not overlap. However, in many cases, such as the limited_derivs function, the loops still did not vectorize, because they would actually have become slower than the serial version due to memory access patterns that disrupted cache locality. Therefore, it would be necessary to change the structure of the code to enable access patterns making use of cache locality. For instance, instead of using vectors of 'vecs' for u, f, g, and so on, we could instead store all this data in a flattened (single-dimensional) data structure and change our 'offset' function to convert three input numbers into an offset within this array, similar to what Professor Bindel did in his C code. We may use valarrays for this purpose. Indeed, for the purposes of this submission, we opted to instead go with the C code provided by Professor Bindel, to benchmark our parallelization, and meanwhile simultaneously another team member would work on getting the C++ code properly vectorized.

Parallelization results:

Unfortunately, we were unable to finish implementing the parallelization before the assignment was due.

Future steps:

We will finish restructuring the C++ code to attempt to get it close to the performance of the C code. We will finish implementing the grid parallelization scheme, and will tune parameters to find out the optimal number of layers of ghost cells per block.