

CS 5220

Project 2 - Shallow Water Simulation

Marc Aurele Gilles (mtg79)
Sheroze Sherifdeen(mss385)

October 18, 2015

1 Introduction

Define structured grid computations
Define shallow water simulations
Math overview and associated abstractions in code

2 Design Decisions

The following sections describe the implementation changes from the original code found at <https://github.com/cornell-cs5220-f15/water>.

2.1 Memory Layout

The original solution a two dimensional vectors of 3-vectors to represent U_t , $F(U)_x$, and $G(U)_y$. During each time step, the solution accesses each element in the 2-D grid sequentially. Then, the `vector<vector<real>>` representation leads to memory accesses that are not local spatially.

Therefore, our solution chooses to use 3 separate two dimensional vectors per objects, U_t , $F(U)_x$, and $G(U)_y$. Thus, we are required in general to perform three loop iterations in place of a single loop in the original solution. But this approach leverages spatial locality, especially in `compute_step` and `limited_derivs` functions.

2.2 Vectorization

By observing the profiling information, we noticed that the original solution spends majority of its computational time in the functions `limited_derivs`, `compute_step` and `compute_fg_speeds`. By adopting the newer memory layout, we enabled spatially local memory accesses. We were also able to decompose `for` loops in the solution to improve vectorization. Refer to `ipo_out_vectorization.optrpt` in <https://github.com/sheroze1123/water/tree/vectorization> for more information.

In `compute_fg_speeds`, we performed two separate loops to compute flux and wave speeds. The flux computation and the wave speed computation for the complete grid is not handled by the `Physics` class. We used `#pragma simd` directives to instruct the compiler the ability to vectorize

these computations. The compiler was successfully able to vectorize these functions with an estimated potential speedup of 6.7.

`limited_derivs` uses the `limdiff` function in `minmod.h`. To improve the vectorization of this computation, we changed the implementation of `limdiff` in the following ways.

1. `limdiff` now performs the computation on the complete grid instead of at one grid point.
2. `limdiff` was decomposed as `limdiff_x` and `limdiff_y` to perform the limiter along the x dimension and the y dimension separately while still retaining unit stride.

2.3 Parallelization

The updated memory layout is amenable to parallelization via OpenMP, particular during the costly operations of `limited_derivs` and `compute_step`.

At the beginning of a call to `limited_derivs` we initialize a team of threads using `#pragma omp parallel`. Each application of the limiter to the components of U_t , $F(U)_x$, and $G(U)_y$ can be computed in parallel since we removed the data dependencies by creating 3 separate two dimensional arrays per component. We use the `#pragma omp for` directive to parallelize the grid computation of each component.

In `compute_step`, we initialize a team of threads using `#pragma omp parallel`. The predictor step is now performed in parallel. Then, we create a barrier before proceeding to the corrector step since the the corrector step needs information from the predictor. After computing the corrector step, we perform a barrier before finally copying back to storage in parallel. The performance increase we observed via parallelization reached up to 3+ speedup. Results are plotted in section 3.2.

2.4 Domain Decomposition

3 Analysis

3.1 Profiling

The following time profiles were obtained on a 200x200 grid by advancing 100 frames.

3.1.1 Original solution

We began the optimization by analyzing the time profiles of the original code. The following table shows the top 4 functions by CPU time.

Function	Module	CPU Time	Spin Time	Overhead Time
Central2D<Shallow2D, MinMod<float>>::limited_derivs	shallow	2.529 s	0 s	0 s
Central2D<Shallow2D, MinMod<float>>::compute_step	shallow	1.210 s	0 s	0 s
Central2D<Shallow2D, MinMod<float>>::compute_fg_speeds	shallow	0.426 s	0 s	0 s
_IO_file_xsputn	libc -2.12.so	0.027 s	0 s	0 s
_IO_fwrite	libc -2.12.so	0.025 s	0 s	0 s

3.1.2 Memory Layout Update

3.1.3 Vectorization

Profiling of vectorization shows good improvements in performance, especially in the `limited_derivs` and `compute_fg_speeds` functions, but a reduction in performance in the `compute_step` func-

tion. The repetition of `limdiff_x` and `limdiff_y` is due to separation of components and axes in `limited_derivs`.

Function	Module	CPU Time	Spin Time	Overhead Time
<code>compute_step</code>	<code>shallow</code>	2.136 s	0s	0s
<code>limited_derivs</code>	<code>shallow</code>	0.448 s	0s	0s
<code>compute_fg_speeds</code>	<code>shallow</code>	0.362 s	0s	0s
<code>limdiff_y</code>	<code>shallow</code>	0.204 s	0s	0s
<code>limdiff_x</code>	<code>shallow</code>	0.201 s	0s	0s
<code>limdiff_x</code>	<code>shallow</code>	0.200 s	0s	0s
<code>limdiff_y</code>	<code>shallow</code>	0.200 s	0s	0s
<code>limdiff_x</code>	<code>shallow</code>	0.197 s	0s	0s
<code>limdiff_x</code>	<code>shallow</code>	0.192 s	0s	0s
<code>limdiff_y</code>	<code>shallow</code>	0.192 s	0s	0s
<code>limdiff_x</code>	<code>shallow</code>	0.188 s	0s	0s
<code>limdiff_y</code>	<code>shallow</code>	0.186 s	0s	0s
<code>limdiff_y</code>	<code>shallow</code>	0.184 s	0s	0s
<code>vector<float, allocator<float>>::operator []</code>	<code>shallow</code>	0.062 s	0s	0s
<code>vector<float, allocator<float>>::operator []</code>	<code>shallow</code>	0.061 s	0s	0s
<code>vector<float, allocator<float>>::operator []</code>	<code>shallow</code>	0.051 s	0s	0s

3.1.4 Parallelization

3.1.5 Domain Decomposition

3.2 Scaling Study

3.2.1 Strong Scaling Study

Using a 500x500 grid and 100 frames, we observe the speedup with respect to the number of threads in our parallel implementation.

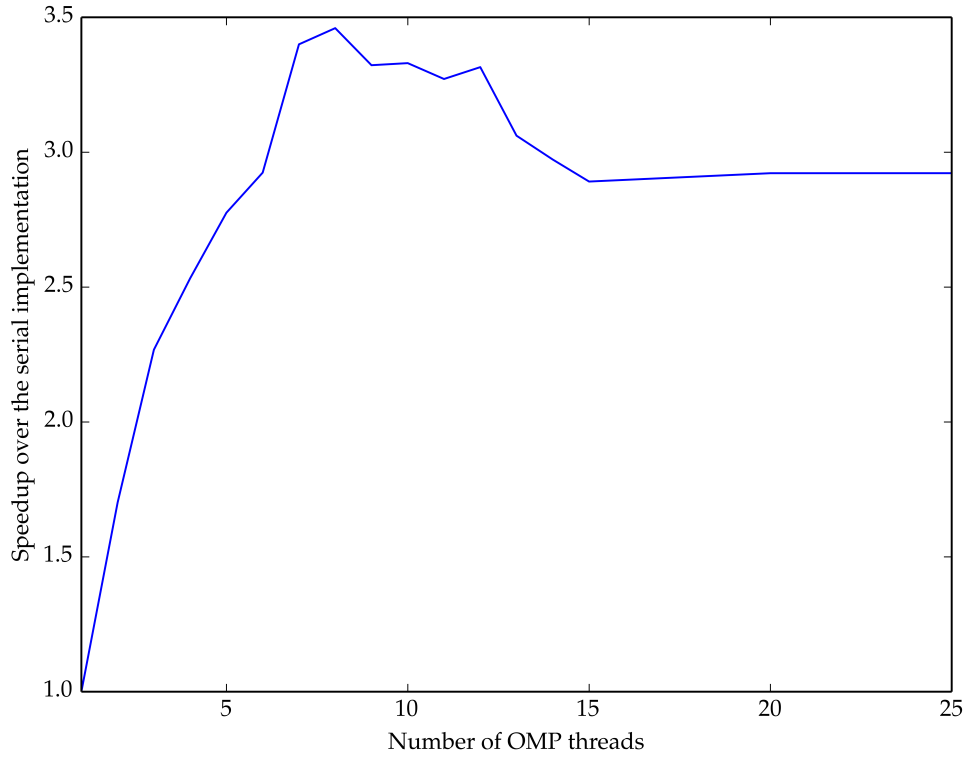


Figure 1: Speedup as a function of the number of threads

3.2.2 Weak Scaling Study

We vary the threads but keep the problem size/thread constant.

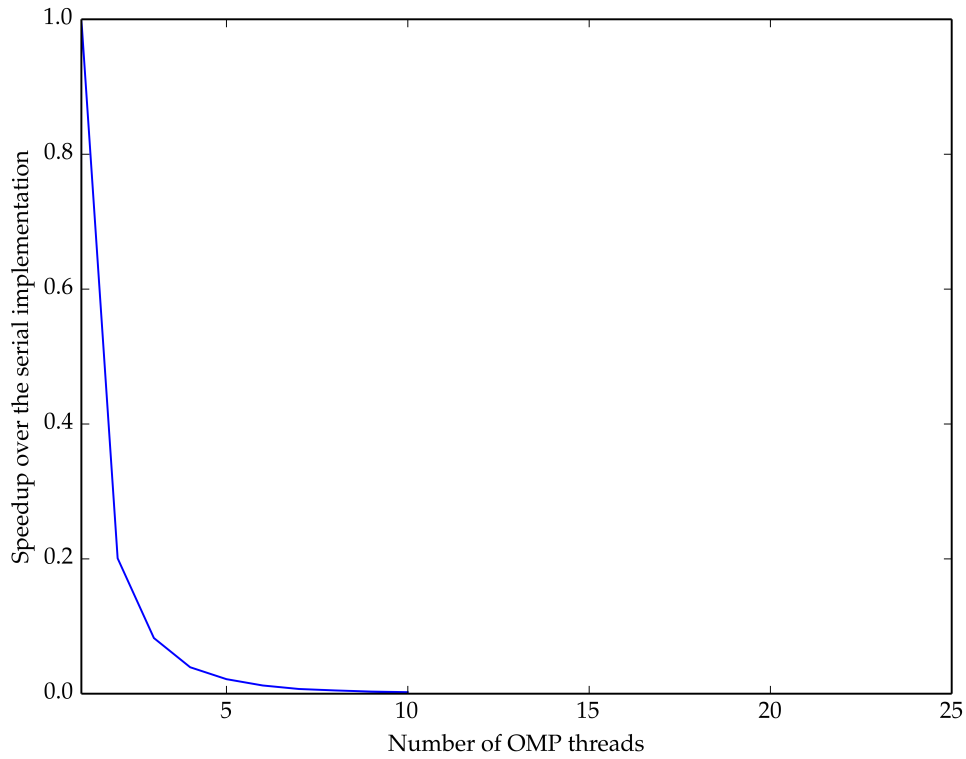


Figure 2: Speedup as a function of the number of threads

References

- [1] Data Alignment to Assist Vectorization. (n.d.). Retrieved September 30, 2015, from <https://software.intel.com/en-us/articles/data-alignment-to-assist-vectorization>