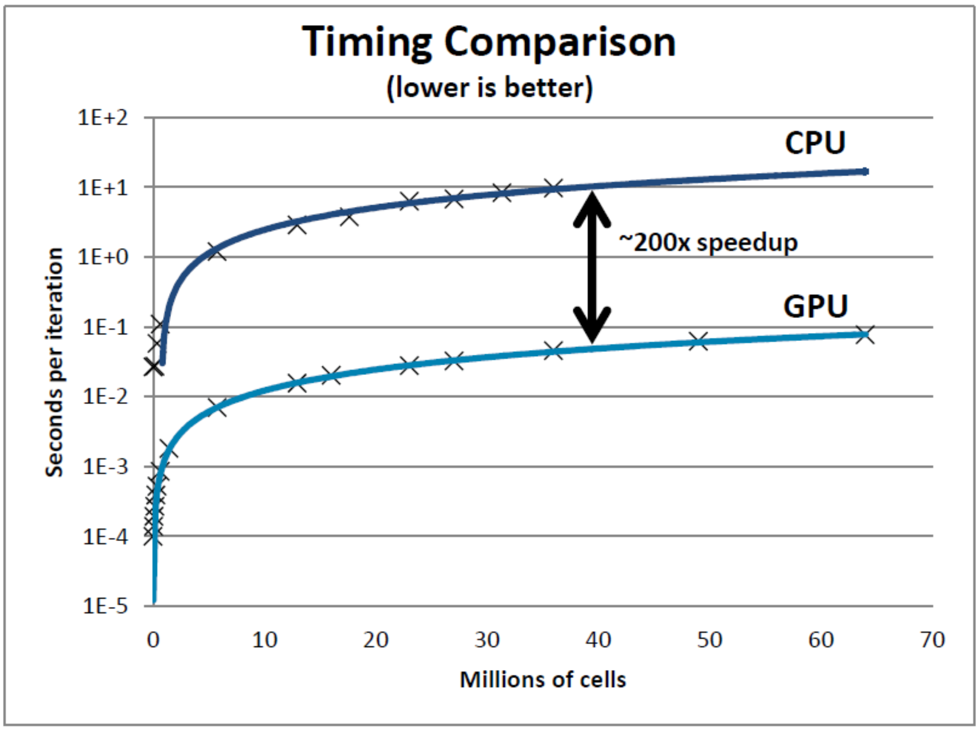
Norwegian
Meteorological
Institute

MET report

no. 27/2013
Oceanography

One-Layer Shallow Water Models on the GPU

André R. Brodtkorb[1], Trond R. Hagen[2], Lars Petter Røed[3]



[1]SINTEF IKT, Avd. for Anvendt Matematikk
[2]SINTEF IKT, Avd. for Anvendt Matematikk
[3]MET, R&D Dep.

Norwegian
Meteorological
Institute

MET report

Title One-Layer Shallow Water Models on the GPU	**Date** December 19, 2013
Section Ocean and Ice	**Report no.** 27/2013
Author(s) André R. Brodtkorb, Trond R. Hagen, Lars Petter Røed	**Classification** ○ Free [✓ ● Restricted]
Client(s) Statoil	**Client's reference** Contract: 45022367955

Abstract

We describe the implementation of a numerical scheme for solving the linear, one-layer shallow water equations on a graphics processing unit (GPU). The scheme we use and its FORTRAN code is described in some detail in *Røed* (2012a) and *Røed* (2012b) together with some benchmark cases. We have implemented this numerical scheme on a central processing unit (CPU) as well as on a GPU and run it for the these benchmark cases. The results we get show that the GPU implementation gives a speed-up over the CPU of slightly more than 200 times (cf. figure on front page). This is highly promising regarding the possibilites of running a large number of ensembles cost effectively on a computer and thereby increasing the accuracy of short-term ocean current forecasts.

Keywords

GPU, Oceanography, Numerical Modeling, Shallow Water

Disciplinary signature	Responsible signature
Lars Anders Breivik	Øystein Hov

Contents

Abstract

We describe the implementation of a numerical scheme for solving the linear, one-layer shallow water equations on a graphics processing unit (GPU). The scheme we use and its FORTRAN code is described in some detail in *Røed* (2012a) and *Røed* (2012b) together with some benchmark cases. We have implemented this numerical scheme on a central processing unit (CPU) as well as on a GPU and run it for the these benchmark cases. The results we get show that the GPU implementation gives a speed-up over the CPU of slightly more than 200 times (cf. figure on front page). This is highly promising regarding the possibilites of running a large number of ensembles cost effectively on a computer and thereby increasing the accuracy of short-term ocean current forecasts.

1 Introduction

Today's oceanographic forecasts are based on highly complex numerical models capturing to a high degree of accuracy the different physics that dominates oceanic motion. While these models are accurate in a statistical sense, they often exhibit large errors regarding ocean current forecasts. This is mostly due to large uncertainties in the knowledge of the initial state, but also to a lesser degree the forcing (atmospheric input, river discharges, lateral input at open boundaries, etc.). An alternative to running a single highly complex model, is to run an ensemble of simpler models differing only in their initial state, and select those that match observations best for the forecast. This approach requires a large number of runs, and thus high performance of the simpler model.

In *Røed* (2012a), a set of ocean models and numerical schemes are presented. The presented equations are thought to capture varying degrees of the physics that dominate short term ocean currents. The models presented are a one-layer model, in which the ocean is modeled with uniform density throughout the water column, a 2-layer model, in which the ocean is modeled as a bottom layer with one density and an upper or top layer with a second density, and finally a $1\frac{1}{2}$-layer model, which can be thought of as a simplification of the 2-layer model in which the bottom layer is assumed to be very deep compared to the upper layer. It should be emphasized that the equations to solve are basically the same for these three models, consisting of what is commonly referred to as the rotating, shallow water equations. Two numerical schemes are presented for the one-layer model. The first scheme uses a linearization of the equations assuming that the surface elevation and thereby the currents are small. The second scheme keeps the non-linear nature of the equations. In *Røed* (2012b), a Fortran reference implementation of the linear model is presented together with nine benchmark cases and their results.

The graphics processing unit (GPU) is the device used to render the image on screen. The GPU has over the last decade been used in a variety of different non-graphics applications, and shown to outperform the traditional CPU by 5-50 times *Brodtkorb et al.* (2010). The non-rotating, shallow water equations have also been shown to be very well suited for implementation on the GPU *Brodtkorb et al.* (2011).

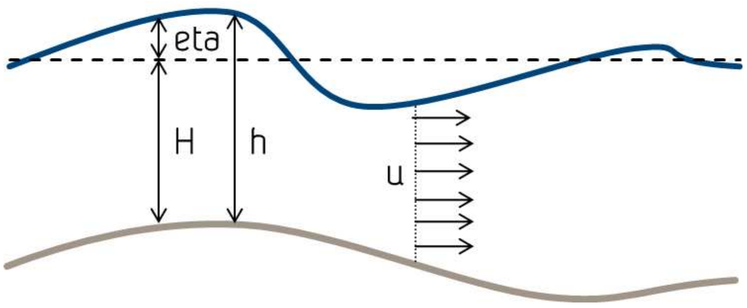


Figure 1: Sketch of the variables in the one-layer linearized model.

The report is organized as follows. We present the numerical scheme in Section 2, while we present the GPU implementation of the scheme in Section 3. Furthermore, in Section 4, we present a verification of the GPU solutions against the CPU solutions for the nine benchmark cases presented in *Røed* (2012b). Here we show that most of them provides results agreeing within floating point precision (10^{-6}). In Section 5 we investigate the perfomance ratio and show that the GPU implementation outperforms the CPU solver 200 times. Finally Section 6 offers a short summary.

2 Numerical Scheme

In this section, we briefly review the numerical scheme for the linear one-layer shallow water model. For a more thorough discussion, we refer the reader to *Røed* (2012a,b). The numerical scheme is based on linearization around a mean depth, so that for a spatial coordinate (x, y) we have that the water depth is $h(x, y, t) = H(x, y) + \eta(x, y, t)$ where $\eta(x, y, t)$ is sea surface deviation away from the equilibrium depth $H(x, y)$, $U(x, y, t)$ is the depth averaged velocity in the x-direction, and $V(x, y, t)$ is the depth averaged velocity in the y-direction (Figure 1).

The numerical scheme uses a staggered grid so that the different variables are placed one half grid length apart as illustrated in Figure 2. For simplicity we

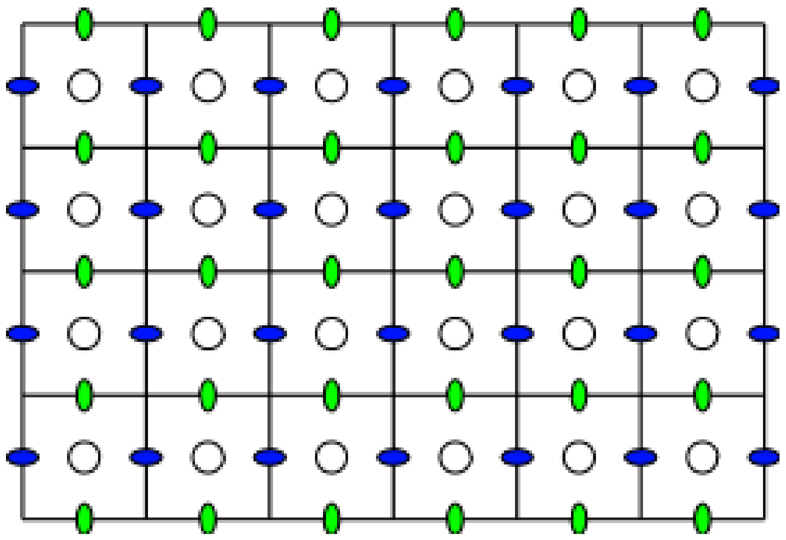


Figure 2: Staggered grid in which the different variables are at different spatial locations. The water disturbance, or sea level, η, is the circle at cell centers, and the velocities, U and V, are placed at cell intersections along the x- and y-axis respectively.

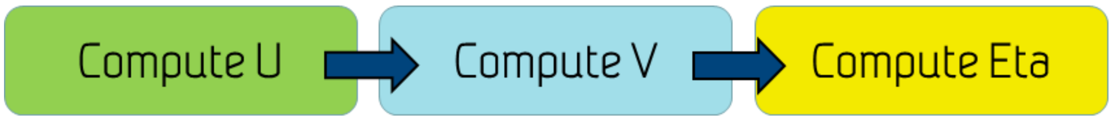


Figure 3: A full simulation cycle. First, the values of U at the next timestep are computed. These are then used in the computation of V at the next timestep, and finally we update η using U and V at the next timestep.

define the following notation for the different variables:

$$\mu_{i+1/2,j+1/2} = \mu\left(x_{i+1/2}, y_{j+1/2}\right) = \mu\left((i+1/2)\Delta x, (j+1/2)\Delta y\right), \tag{1}$$

$$U_{i,j+1/2} = U\left(x_i, y_{j+1/2}\right) = U\left(i\Delta x, (j+1/2)\Delta y\right), \tag{2}$$

$$V_{i+1/2,j} = V\left(x_{i+1/2}, y_j\right) = V\left((i+1/2)\Delta x, j\Delta y\right). \tag{3}$$

We define $\mu_{i+1/2,j+1/2}$ to be at the *center* of cell (i, j), $U_{i,j+1/2}$ to be at the intersection between cell (i, j) and $(i+1, j)$, and $V_{i+1/2,j}$ to be at the intersection between cell (i, j) and $(i, j+1)$ (Figure 1). In addition, we define $\bar{V}_{i,j+1/2}$ to be the reconstructed value of V at the location of $U_{i,j+1/2}$.

The numerical scheme consists of three stages, as illustrated in Figure 3. We

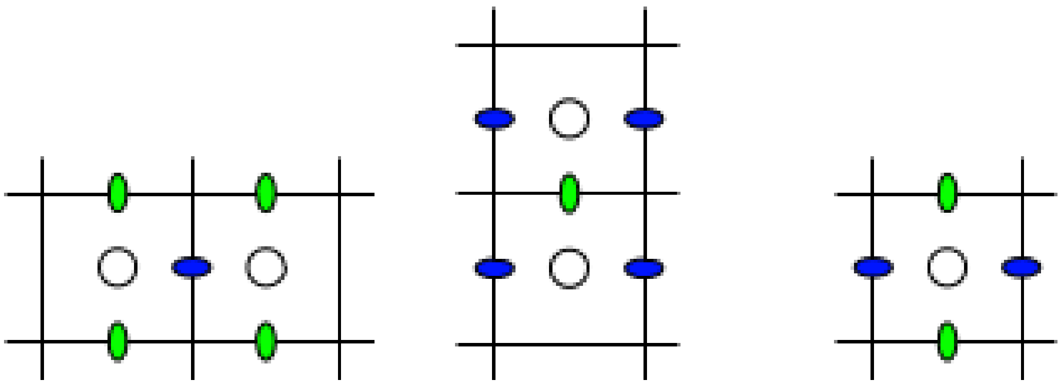

(a) $U^{n+1}_{i,j+1/2}$ (b) $V^{n+1}_{i+1/2,j}$ (c) $\eta^{n+1}_{i+1/2,j+1/2}$

Figure 4: Computational stencils used to compute U, V, and η.

start by computing the U component of the numerical momentum (Figure 4):

$$U^{n+1}_{i,j+1/2} = \frac{1}{B_{i,j+1/2}} \left[U^n_{i,j+1/2} + \Delta t \left(f\bar{V}^n_{i,j+1/2} - P^n_{i,j+1/2} + X^{n+1}_{i,j+1/2} \right) \right], \tag{4}$$

in which

$$B_{i,j+1/2} = \left(1 + \frac{R\Delta t}{\bar{H}_{i,j+1/2}} \right), \tag{5}$$

$$P^n_{i,j+1/2} = g\bar{H}_{i,j+1/2} \frac{\eta^n_{i+1/2,j+1/2} - \eta^n_{i-1/2,j+1/2}}{\Delta x}, \tag{6}$$

$$X^{n+1}_{i,j+1/2} = \frac{1}{\rho_0} [\tau^x_s]^{n+1}_{i,j+1/2}. \tag{7}$$

Here, B is due to the semi-implicit handling of the linear bottom friction source term, and R is the friction coefficient. Similarly, we have that τ^x_s is the x component of the wind shear stress. We can compute the V momentum similarly, but using the newly computed U momentum values (Figure 4):

$$V^{n+1}_{i+1/2,j} = \frac{1}{B_{i+1/2,j}} \left[V^n_{i+1/2,j} + \Delta t \left(f\bar{U}^{n+1}_{i+1/2,j} - P^n_{i,j+1/2} + Y^{n+1}_{i+1/2,j} \right) \right], \tag{8}$$

in which

$$B_{i+1/2,j} = \left(1 + \frac{R\Delta t}{\bar{H}_{i+1/2,j}} \right), \tag{9}$$

$$P^n_{i+1/2,j} = g\bar{H}_{i+1/2,j} \frac{\eta^n_{i+1/2,j+1/2} - \eta^n_{i+1/2,j-1/2}}{\Delta x}, \tag{10}$$

$$Y^{n+1}_{i+1/2,j} = \frac{1}{\rho_0} [\tau^x_s]^{n+1}_{i+1/2,j}. \tag{11}$$

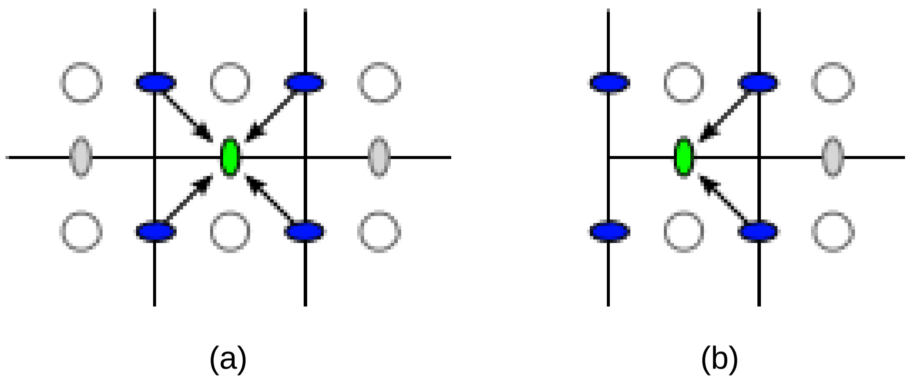


Figure 5: Reconstruction of variables: (a) The reconstruction of the velocity components U at the location of velocity components V for internal nodes. (b) The reconstruction of the velocity components U at the location of velocity components V for boundary nodes when the boundary is closed. The reconstruction of V at U locations is similar.

Finally, we may compute the sea surface deviation using the newly computed U and V momenta (see also Figure 4):

$$\begin{aligned}\eta^{n+1}_{i+1/2,j+1/2} = \eta^{n}_{i+1/2,j+1/2} &- \frac{\Delta t}{\Delta x}\left[U^{n+1}_{i,j+1/2} - U^{n+1}_{i+1,j+1/2}\right] \\ &- \frac{\Delta t}{\Delta y}\left[V^{n+1}_{i+1/2,j} - V^{n+1}_{i+1/2,j+1}\right].\end{aligned} \tag{12}$$

The numerical scheme consisting of (4), (8) and (12) requires evaluation of the variable at other places than where it is defined, in particular the terms marked with an overbar, e.g., $\bar{U}^{n+1}_{i+1/2,j}$ in (8). We therefore need to reconstruct their values using linear interpolation as illustrated in Figure 5. Note that for nodes close to the edge, this reduces to a linear interpolation. Thus we get

$$\bar{U}^n_{i+1/2,j} = \begin{cases} \frac{1}{2}\left[U^n_{i+1,j-1/2} + U^n_{i+1,j+1/2}\right], & i = 0 \\ \frac{1}{4}\left[U^n_{i,j-1/2} + U^n_{i,j+1/2} + U^n_{i+1,j-1/2} + U^n_{i+1,j+1/2}\right], & i \in [1, nx-2] \\ \frac{1}{2}\left[U^n_{i,j-1/2} + U^n_{i,j+1/2}\right], & i = nx-1 \end{cases} \tag{13}$$

$$\bar{V}^n_{i,j+1/2} = \begin{cases} \frac{1}{2}\left[V^n_{i-1/2,j+1} + V^n_{i+1/2,j+1}\right], & i = 0 \\ \frac{1}{4}\left[V^n_{i-1/2,j} + V^n_{i+1/2,j} + V^n_{i-1/2,j+1} + V^n_{i+1/2,j+1}\right], & j \in [1, ny-2] \\ \frac{1}{2}\left[V^n_{i-1/2,j} + V^n_{i+1/2,j}\right], & i = ny-1 \end{cases} \tag{14}$$

$$\bar{H}_{i,j+1/2} = \frac{1}{2}(H_{i-1/2,j+1/2} + H_{i+1/2,j+1/2}), \tag{15}$$

$$\bar{H}_{i+1/2,j} = \frac{1}{2}(H_{i+1/2,j-1/2} + H_{i+1/2,j+1/2}). \tag{16}$$

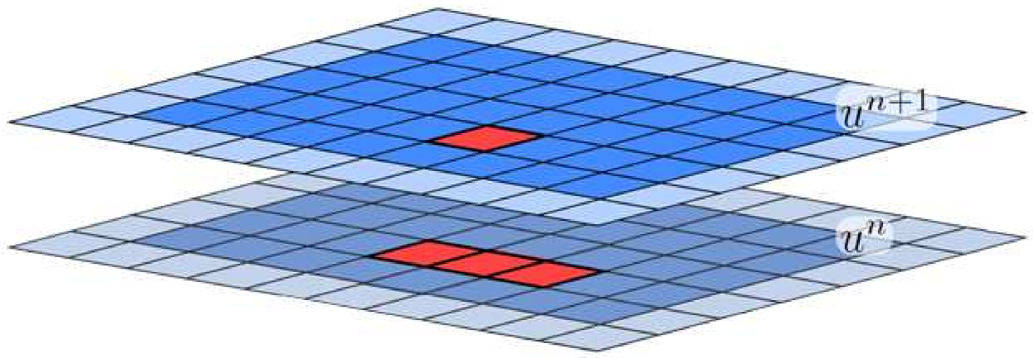


Figure 6: Computational domain for two timesteps. The values at timestep $n + 1$ can be computed from the values at timestep n independently for each cell (illustrated here for a three-point stencil).

3 GPU Implementation

The numerical scheme detailed in the previous section is essentially a stencil computation. This means that the value of cell i, j at the next timestep can be computed independently of all other cells as illustrated in Figure 6. Such stencil computation suits the GPU perfectly, as the GPU is a highly parallel processor.

Our implementation of the numerical scheme results in three different *kernels* that compute U, V, and η, respectively. Each of these kernels run after each other sequentially (Figure 3), but exploit the inherent parallelism of the stencil computations within each kernel. One of these kernels is outlined in Listing 1. This kernel is essentially just like a regular CPU function, only that it executes in parallel for all data elements, operates on data that is located on the GPU, and stores the results also on the GPU. Furthermore, there are some extra variables that determine which data element the function is being executed for.

In *Røed* (2012b), there are several boundary conditions that are outlined. The open boundaries require an external solution that is used to represent what the solution is outside the computational domain. We need to compute this external solution simultaneously with the internal solution. The most efficient way to do this, is to compute it simultaneously as we solve for the internal domain, as shown in Figure 7. We do this by computing the external solution of U at the next timestep simultaneously as we compute the internal solution of V. This hides the

Listing 1: GPU kernel that computes U at the next timestep.

```
//GPU Kernel that evolves U in time
__global__ void computeUKernel(Parameters params_,
                               Data data_,
                               float t_) {

      //Data indexing variables
      unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
      unsigned int j = blockIdx.y*blockDim.y + threadIdx.y;

      [...] //Read input data, compute stresses, etc.

      //Store result to main GPU memory
      data_.U[j][i] = B*(U_current +
                         params_.dt*(params_.f*V_m + P + X));
}
```

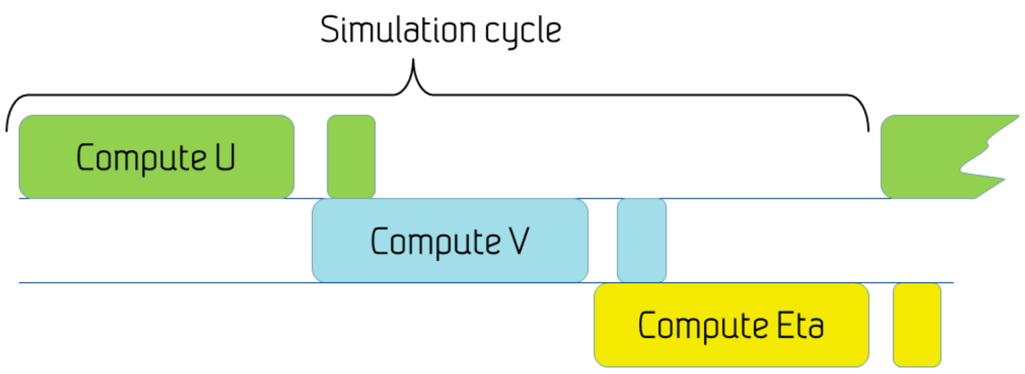


Figure 7: Simulation cycle with external solution.

overhead of computing this external solution, which otherwise could be considered a relatively expensive function on the GPU.

4 Verification and accuracy

We have run the nine benchmark cases as defined in *Røed* (2012b), and compared our results against the results produced by the Fortran reference code. The benchmark cases are summarized in Table 1.

	Closed boundary	**Open boundary**	**Open boundary with shelf**
Uniform Along Shore	1A	1B	1C
Bell Shaped Along Shore	2A	2B	2C
Moving Cyclone	3A	3B	3C

Table 1: List of cases run. First column describes the wind forcing.

All cases have one boundary closed (wall), which is used to represent the coast line, and the remaining boundaries vary with the case. The closed boundary cases (cases 1-3 A) use a closed (wall) boundary condition, and the open boundary cases (cases 1-3 B) use an open boundary condition in which the solution outside the domain is prescribed by the external solution, and a numerical sponge is used to impose these conditions. The open boundary cases with shelf (cases 1-3 C) have a varying bathymetry that is used to represent a continental shelf off the coast and an open boundary condition.

In addition to varying the boundary conditions and bathymetry, the wind stress term is also varied. Cases 1 A-C use a uniform wind stress, cases 2 A-C use a spatially varying wind stress, and cases 3 A-C use a spatially and temporally varying wind stress mimicking a moving cyclone or low pressure system.

These benchmark cases are designed to test that the different parts of the implementation work as expected, and all of our results are visually identical to the reference, showing the same dynamics. Figure 8 show our results as plots of time series of a single spatial location as the simulation progresses. All of our results are within floating point precision (10^{-6}) of the reference solution, except for cases 3 B and 3 C. These two cases have a slight difference, but show the same dynamics as the reference. The only difference in these two cases from the others, is that they have a temporally and spatially varying wind stress term in the

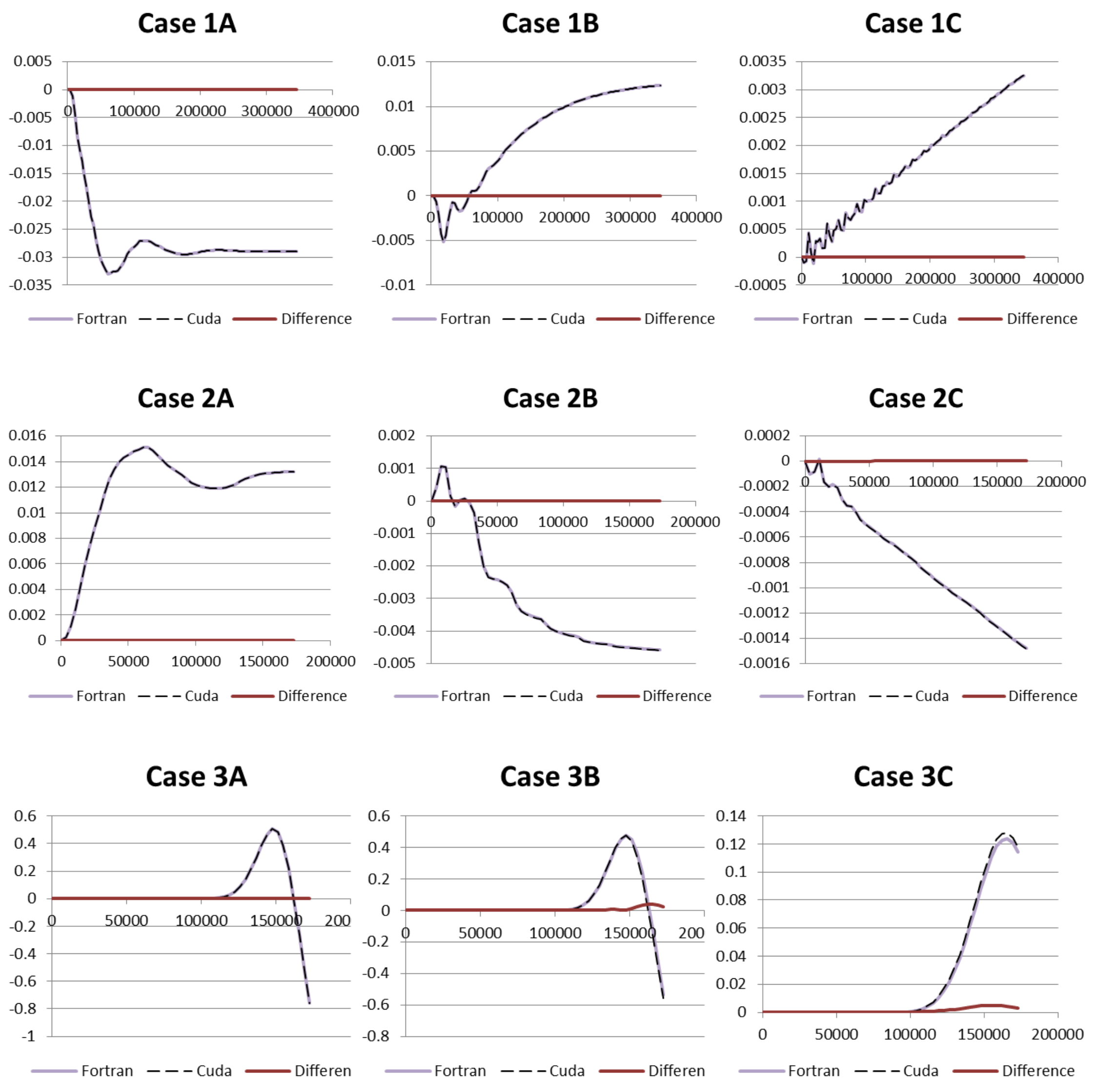


Figure 8: Comparison of reference Fortran results and Cuda GPU results. The simulation results are identical to within floating point precision, except for cases 3B and 3C. The discrepancies between the reference and GPU solution in this case comes from different interpretations of the moving wind stresses for the external solution.

external solution (used for the open boundary conditions). From this we conclude that the discrepancy is due to a different interpretation of the temporal-spatial location of the wind stress term in the external solution, and that the boundary conditions and numerical scheme give the expected results.

5 Performance

We assess the performance and efficiency of our GPU implementation of the numerical scheme by investigating the run-time as a function of domain size. As the exact same operations are performed regardless of the initial conditions, we find this a good measure. We run Case 1A with a varying grid size, and record the run-time. The benchmark is run on an Intel Core i7-2600K @ 3.7 GHz equipped with 8 GB RAM and an NVIDIA GeForce 480 GTX @ 1.4 GHz graphics card. This is a commodity level CPU and commodity level GPU in a comparable price segment. The CPU code is compiled with the "-O3" optimization flag using the g95 Fortran compiler, and the GPU code is compiled using CUDA 4.1 in Visual Studio 2010 using standard "Release" build settings.

Figure 9 shows our performance results. The Fortran solution is only able to run simulation cases up-to roughly 40 million cells, while the GPU version was able to run over 65 million cells. The least squares approximation of a linear function to the measured run-times gives us an expected speed-up of over 200 times for reasonably sized domains. This is highly promising with regard to the idea of running a large number of ensembles, as the GPU presumably will be able to run two hundred times as many scenarios as the CPU within the same time frame.

6 Summary

We have implemented a GPU version of a rotating, linear one-layer, shallow water model, and run it for nine benchmark cases. We show that the GPU implementation is roughly 200 times faster than the CPU reference, while still producing as accurate results. This is highly promising for running an ensemble of ocean current predictions on the GPU which differ only in their initial state, since the GPU provides the possibility of running 200 ensemble members within the same time frame as it takes to run one ensemble member on the CPU.

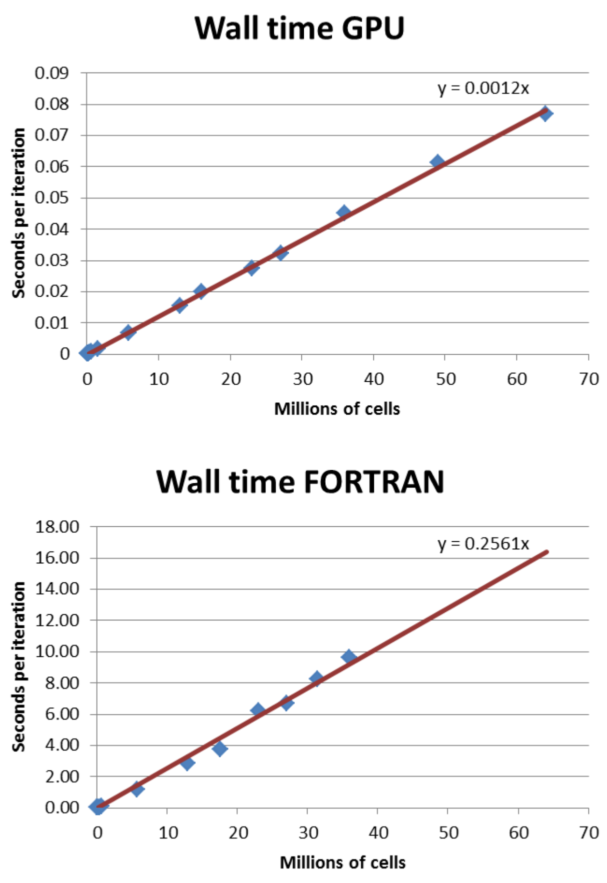


Figure 9: Performance comparison of the CPU and GPU runtimes for different problem sizes. The actual runtimes are marked as blue diamonds, and the red line is a least squares approximation of the data using a linear functional. The GPU version is approximately 200 times faster than the CPU version.

References

Brodtkorb, A. R., C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. Storaasli (2010), State-of-the-art in heterogeneous computing, *Scientific Programming*, *18*(1), 1 – 33.

Brodtkorb, A. R., M. L. Sætra, and M. Altinakar (2011), Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation, *Computers & Fuids*, *55*, 1–12, doi:10.1016/j.compfluid.2011.10.012.

Røed, L. P. (2012a), Documentation of simple ocean models for use in ensemble predictions. Part I: Theory, *Tech. Rep. 3/2012*, Norwegian Meteorological Institute.

Røed, L. P. (2012b), Documentation of simple ocean models for use in ensemble predictions. Part II: Benchmark Cases, *Tech. Rep. 5/2012*, Norwegian Meteorological Institute.