Using the GPU to Predict Drift in the Ocean

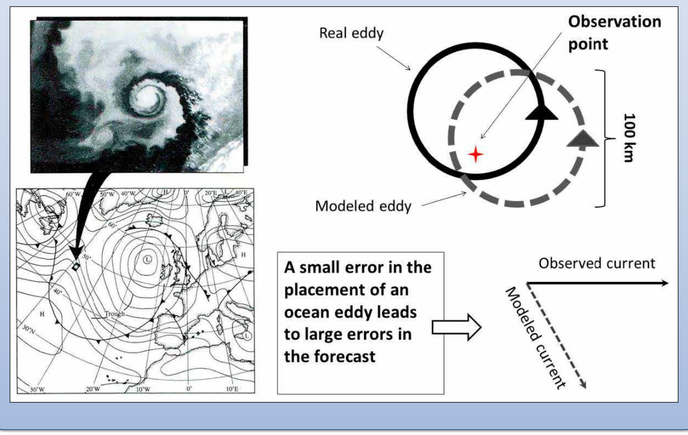
André Rigland Brodtkorb[1], Kai Håkon Christensen[2], Lars Petter Røed[2,3], and Martin Lilleeng Sætra[2,4]

[1] SINTEF, Dept. Appl. Math., P.O. Box 124 Blindern, 0314 Oslo, Norway [2] Norwegian Meteorological Institute, P.O. Box 43 Blindern, 0313 Oslo, Norway
[3] University of Oslo P.O. Box 1072 Blindern 0316 Oslo Norway [4] Westerdals Oslo ACT, P.O. Box 9215 Grønland, 0134 Oslo, Norway
Emails: Andre.Brodtkorb@sintef.no, Kai.H.Christensen@met.no, LarsPetter.Roed@met.no, Martin.L.Satra@met.no

1. Abstract: We describe the implementation of a simple numerical scheme for solving the shallow water equations on a graphics processing unit (GPU), that will be used in the further development of a massive ensemble prediction system running on the GPU. The numerical scheme has previously been used in operational forecasting, and benchmarks comparing the FORTRAN CPU version with the new GPU version have been performed. The results show that the GPU implementation gives a speed-up over the CPU of slightly more than 200 times. This is highly promising regarding the possibilities of running a large number of ensembles cost effectively on a computer and thereby increasing the usefulness of short-term ocean current forecasts and drift trajectory predictions.

2. Motivation

Although the dynamics is similar, there is a huge difference in scale between an atmospheric low pressure system (bottom left) and an oceanic low pressure system (top left), or ocean eddy, as these features are usually called. The modelled position of an atmospheric “eddy” may be wrong by 10–30 km without having any real consequence with regard to the accuracy of the weather forecast. As shown on the right, an error of similar size in the placement of an ocean eddy has a large negative impact on the ocean forecast. By utilizing the GPU’s parallel processing power, we can run very large ensembles, and get more accurate results.



3. Target Application Areas

Icebergs and Sea Ice

Oil Spills

Floating Structures

Source: Agência Brasil (CC BY 3.0 BR)

Search and Rescue

Source: Wikipedia user Jaqian (CC BY-SA 3.0)

Norwegian Meteorological Institute

SINTEF

4. Governing Equations

The shallow water equations written in flux form:

$$\partial_t \eta + \nabla_H \cdot \boldsymbol{U} = 0,$$

$$\partial_t \boldsymbol{U} + \nabla_H \cdot \left(\frac{\boldsymbol{U}\boldsymbol{U}}{H+\eta}\right) + f\boldsymbol{k}\times\boldsymbol{U} + P = \frac{\Delta\boldsymbol{\tau}}{\rho_0} + A\nabla_H^2\boldsymbol{U}, \qquad P = gH\nabla_H\eta + \frac{1}{2}g\nabla_H\eta^2 + \frac{H+\eta}{\rho_0}\nabla_H p_s.$$

Volume flux; Coriolis parameter; Wind stress and bottom stress; Eddy viscosity coefficient

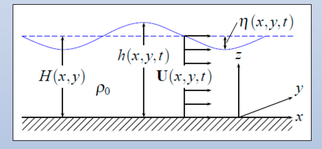
Our forward-backward numerical scheme is based on linearization around a mean:

Bottom friction; Coriolis parameter; X-component of wind shear stress

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

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

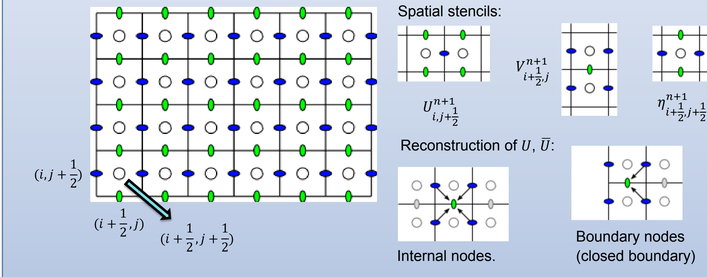
$$\eta^{n+1}_{i+\frac{1}{2},j+\frac{1}{2}} = \eta^n_{i+\frac{1}{2},j+\frac{1}{2}} - \frac{\Delta t}{\Delta x}\left[U^{n+1}_{i,j+\frac{1}{2}} - U^{n+1}_{i+1,j+\frac{1}{2}}\right] - \frac{\Delta t}{\Delta x}\left[U^{n+1}_{i,j+\frac{1}{2}} - U^{n+1}_{i+1,j+\frac{1}{2}}\right].$$



Variables with a bar, e.g., $\bar{H}$, represents reconstructed values by bilinear interpolation, see the next section.

5. Simplified Ocean Model on the GPU

The GPU is a massively parallel processor that requires different algorithms than traditional CPUs. For explicit schemes with compact stencils, we essentially have an embarrassingly parallel problem, which suits the GPU perfectly. To map the computation to the parallel architecture of the GPU, we perform domain decomposition, in which each block is computed independently. Within each block, however, we use a *kernel* with 16x16 threads that collectively computes its subdomain. A staggered grid is used, where η and H are represented in cell centres, U at vertical cell interfaces, and V at horizontal cell interfaces:



To perform a full simulation cycle, and advance the solution Δt in time, the following kernels are run:

1. Compute U
   - Use reconstructed H at U-positions
   - Reconstruct V at U-positions

2. Compute V
   - Use reconstructed H at V-positions
   - Reconstruct U at V-positions

3. Compute η
   - Use computed U and V

4. Apply boundary conditions

6. CUDA and OpenCL

The simplified ocean model has been implemented in both CUDA C/C++ and OpenCL C. This enables us to compare the implementations and run the model on many different hardware architectures. In both versions, the CPU-side code is written in C++. All benchmarking and validation results presented here are for the CUDA version. The reference CPU-implementation is written in FORTRAN.

OpenCL code example, computeEta-kernel:

```
__kernel void computeEta (
    __global const float *U,
    __global const float *V,
    __global float *eta, computeEta_args args)
{
    const int lx = get_local_id(0);
    const int ly = get_local_id(1);
    const int lnx = get_local_size(0);
    const int u_east = lx + 1 + ly * lnx;
    // [...]
    // - compute indices
    // - copy data from global memory to local memory
    // - the work group size is 16x16 work items/threads

    barrier(CLK_LOCAL_MEM_FENCE);

    eta[gid] = eta[gid]
          - args.dt / args.dx * (U_local[u_east] - U_local[u_west])
          - args.dt / args.dy * (V_local[v_south] - V_local[v_north]);
}
```

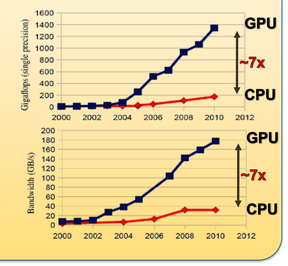
OpenCL

NVIDIA CUDA

Graphics Processing Units

Graphics processing units were originally designed for creating a 2D image on screen from a virtual 3D game world. As games have progressively become more complex, the demand for compute power has grown exponentially. With the advent of programmable GPUs around 2004, the interest in using GPUs for non-graphics applications exploded. Today, a large share of the Top 500-systems are accelerated with GPUs.

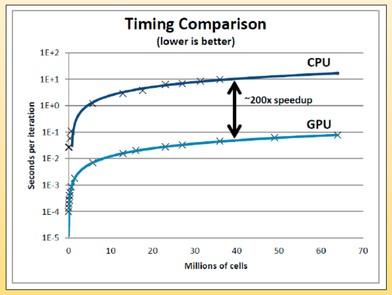
Wolfenstein games, 1981 to 2009



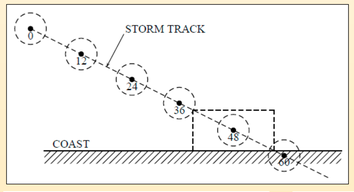
Preliminary Results

We have implemented a GPU version of a shallow water model in a rotating reference frame, and run it for nine benchmark cases. The figure shows performance comparison of the CPU and GPU runtimes for deterministic simulation runs for different problem sizes.

The GPU version is approximately 200 times faster than the CPU reference version. This is highly promising for creating large model ensembles on the GPU, perturbing forcing and initial conditions, 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. Whilst this comparison is clearly unfair (single core untuned CPU code vs. tuned GPU), it demonstrates that the problem fits very well to the GPU architecture, and shows the potential of GPU computing.



Validation Against FORTRAN CPU Reference Code



	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

Above: List of cases run. The first column describes the wind forcing. Left: Sketch of computational domain showing case 3B.