GPU implementation of a linear shallow water model for massive ensemble simulations

JONSMOD 2016 Conference, Oslo, Norway

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

1 Department of Applied Mathematics, SINTEF ICT.
2 Norwegian Meteorological Institute.

Outline

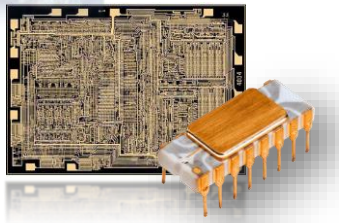
- Motivation for GPU computing
- Implementation of shallow water system on the GPU
- Summary

History lesson: development of the microprocessor 1/2

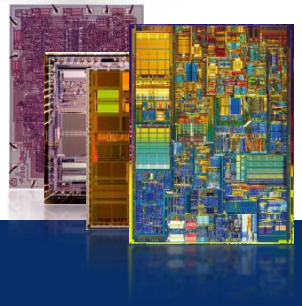
1942: Digital Electric Computer
(Atanasoff and Berry)

1947: Transistor
(Shockley, Bardeen, and Brattain)

1958: Integrated Circuit
(Kilby)



1971: Microprocessor
(Hoff, Faggin, Mazor)



1971- Exponential growth
(Moore, 1965)

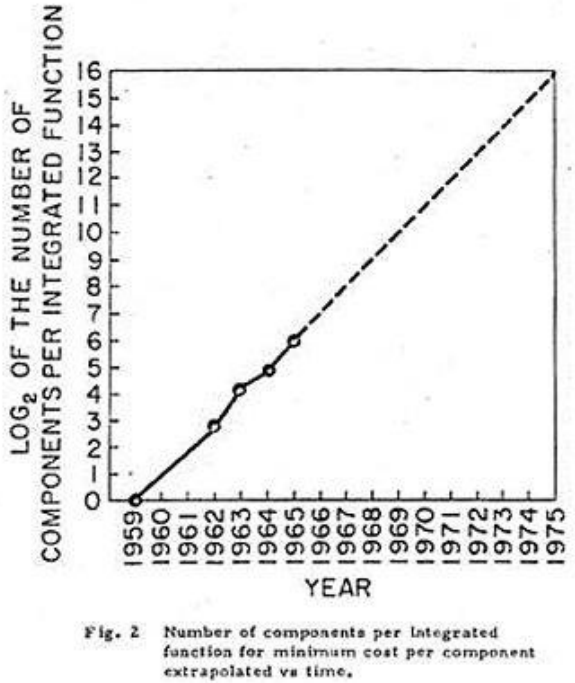
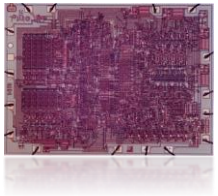


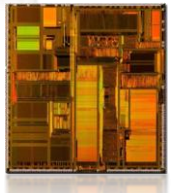
Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

History lesson: development of the microprocessor 2/2

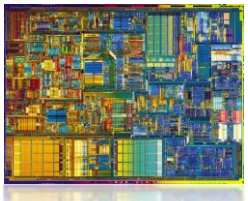


1971: 4004,
2300 trans, 740 KHz

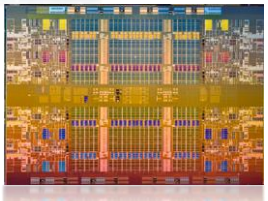
1982: 80286,
134 thousand trans, 8 MHz



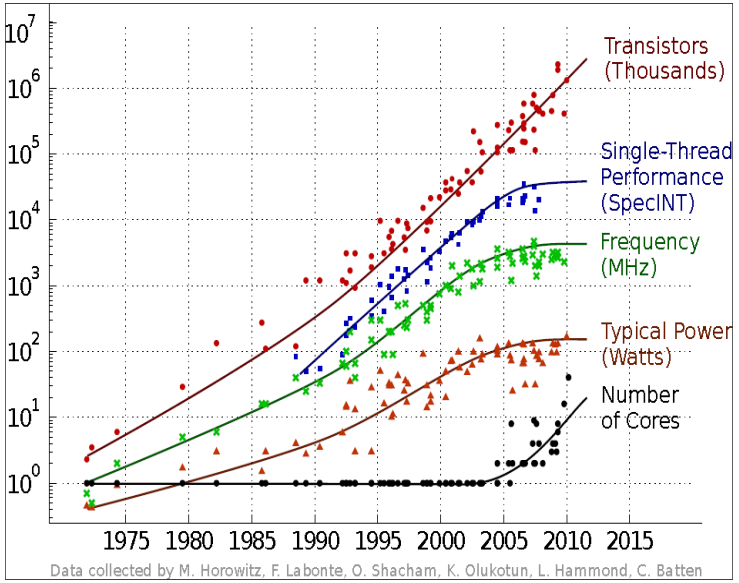
1993: Pentium P5,
1.18 mill. trans, 66 MHz



2000: Pentium 4,
42 mill. trans, 1.5 GHz



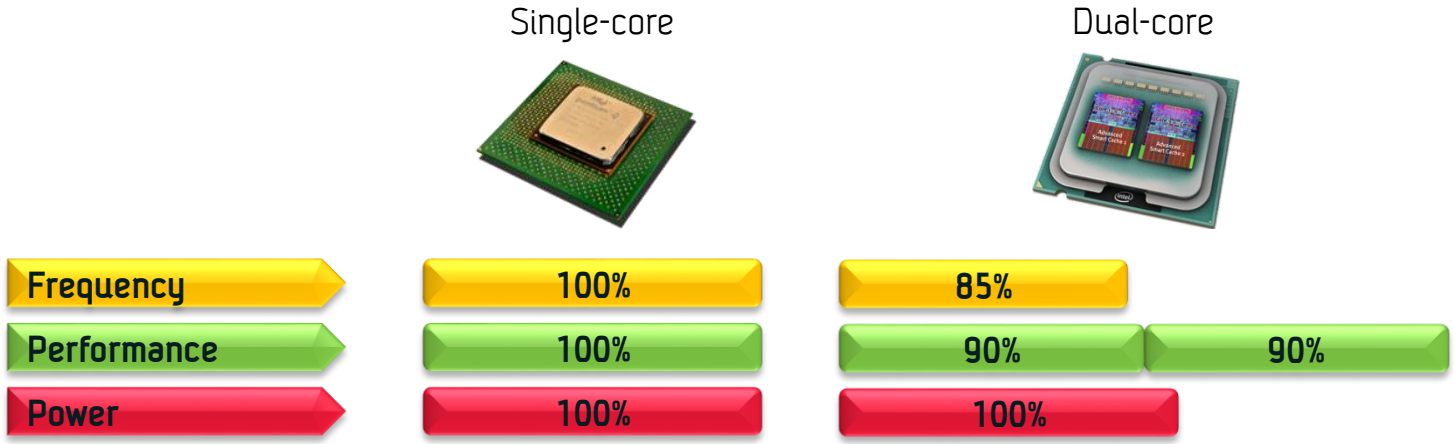
2010: Nehalem
2.3 bill. Trans, 8 cores, 2.66 GHz



Why Parallelism?

The power density of microprocessors
is proportional to the clock frequency cubed:[1]

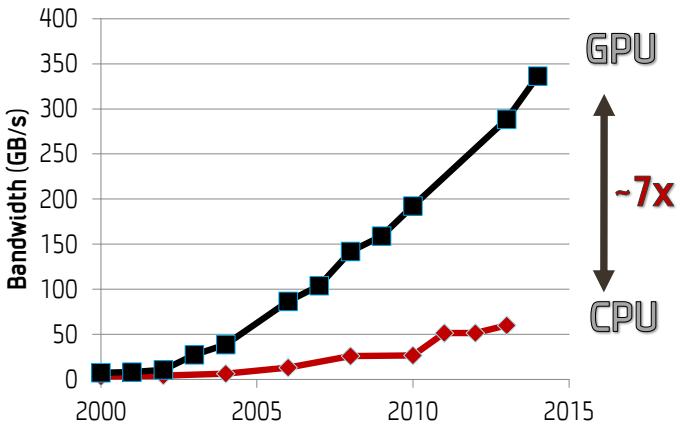
$$P_d \propto f^3$$

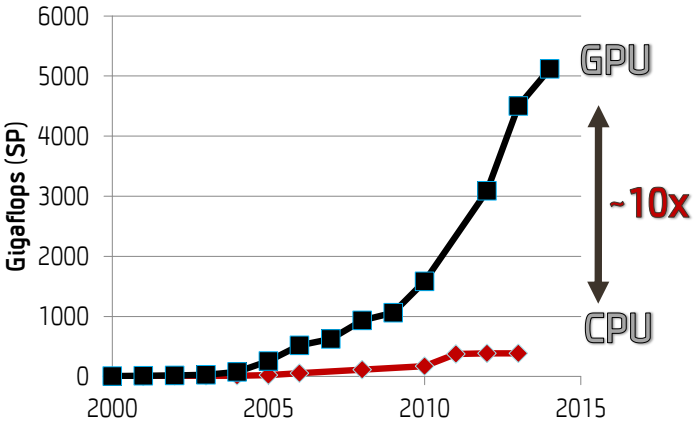


[1] Brodtkorb et al. State-of-the-art in heterogeneous computing, 2010

Massive Parallelism: The Graphics Processing Unit

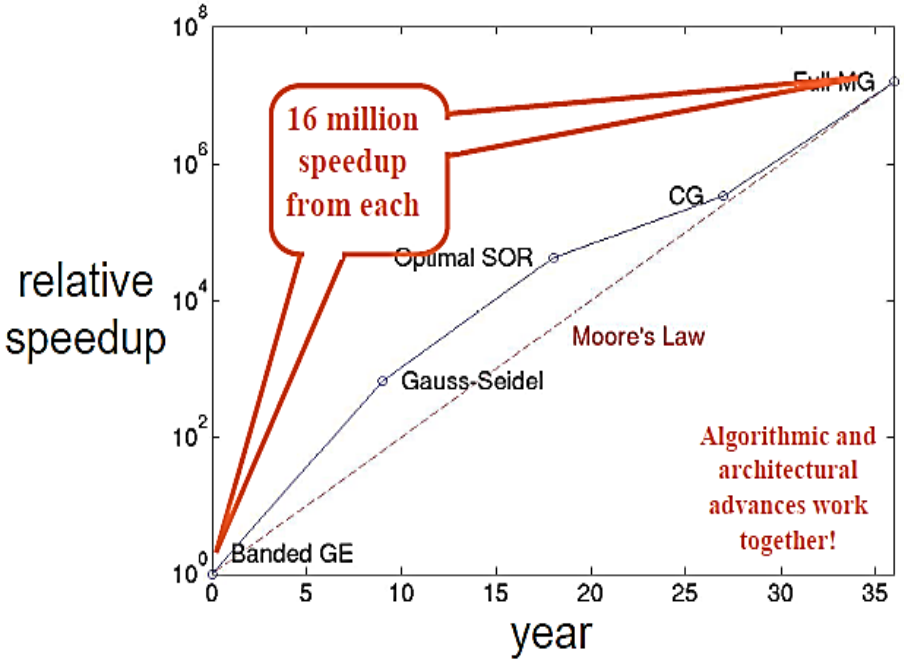
- Up-to 5760 floating point operations in parallel!
- 5-10 times as power efficient as CPUs!





Why care about computer hardware?

- The key to performance, is to consider the full algorithm and architecture interaction.
- A good knowledge of both the algorithm and the computer architecture is required.

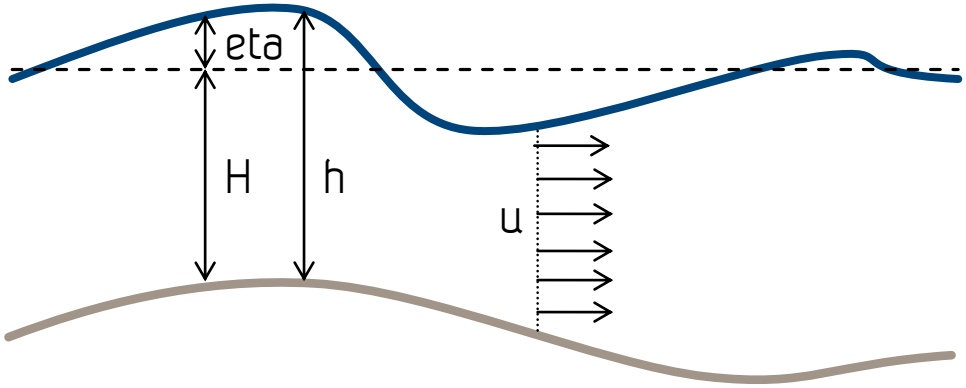


Graph from David Keyes, Scientific Discovery through Advanced Computing, Geilo Winter School, 2008

Work performed

- Numerical scheme implemented on the GPU
  - Bathymetry source terms
  - Wind source terms
  - Semi implicit (backward) friction source terms
  - Open and closed boundaries
- Implementation compared against reference FORTRAN implementation
- Assessment and comparison of performance

Mathematical Model



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

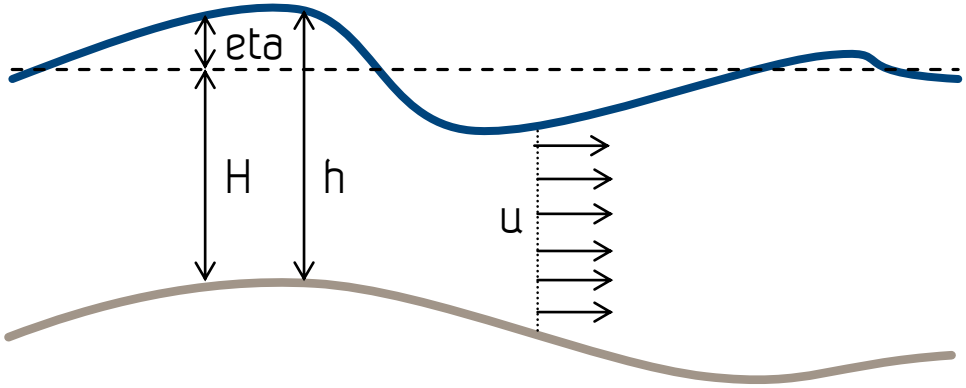
$$\partial_t \mathbf{U} + \nabla_H \cdot \left(\frac{\mathbf{U}\mathbf{U}}{H+\eta}\right) + f\mathbf{k} \times \mathbf{U} + \mathbf{P} = \frac{\Delta \boldsymbol{\tau}}{\rho_0} + A\nabla_H^2 \mathbf{U},$$

Coriolis

Bottom and wind stress

Eddy viscosity parameter

Linearized model



$$\partial_t U - fV = -gH\partial_x \eta + \frac{\tau_s^x - \tau_b^x}{\rho_0},$$

$$\partial_t V + fU = -gH\partial_y \eta + \frac{\tau_s^y - \tau_b^y}{\rho_0},$$

$$\partial_t \eta = -\partial_x U - \partial_y V,$$

Discretized Equations (Numerical scheme)

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

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

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

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

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

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

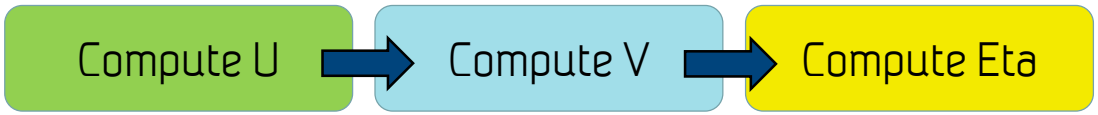
$$P^n_{i+1/2,j} = g\bar{H}_{i+1/2,j}\frac{\eta^n_{i+1/2,j+1/2} - \eta^n_{i+1/2,j-1/2}}{\Delta x},$$

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

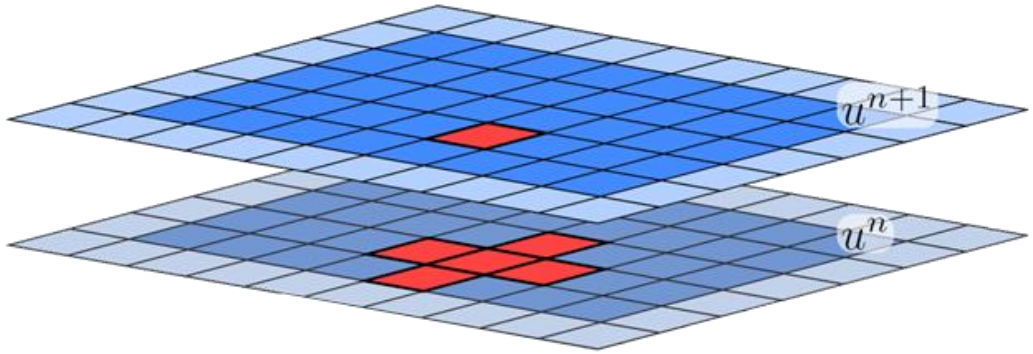
$$\eta^{n+1}_{i+1/2,j+1/2} = \eta^n_{i+1/2,j+1/2} - \frac{\Delta t}{\Delta x}\left[U^{n+1}_{i,j+1/2} - U^{n+1}_{i+1,j+1/2}\right] - \frac{\Delta t}{\Delta y}\left[V^{n+1}_{i+1/2,j} - V^{n+1}_{i+1/2,j+1}\right].$$

Implementation

- The numerical scheme computes U, V, and Eta after each other for each time step

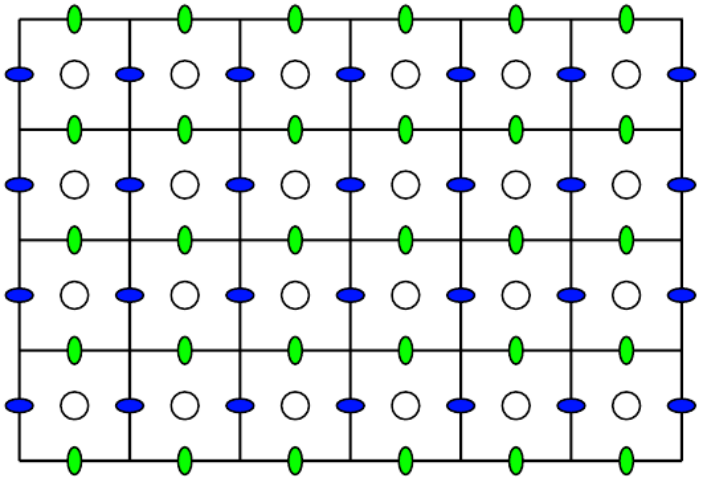


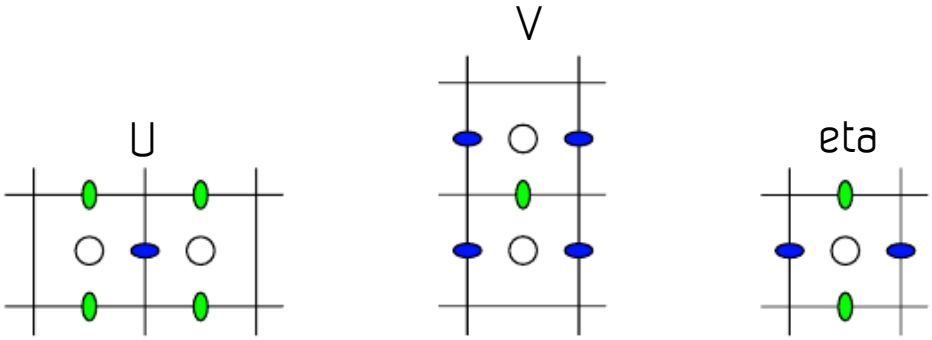
- Computing U, V, and Eta is done with *CUDA Kernels*
- A kernel is a GPU program that executes in a data-parallel fashion:
  All cells in the domain are computed simultaneously!



Computational Stencils

- The computational stencils are compact
- The computational stencils make the computation of each grid cell independent of all other cells
- This gives a numerical scheme that is highly suitable for implementation on the GPU





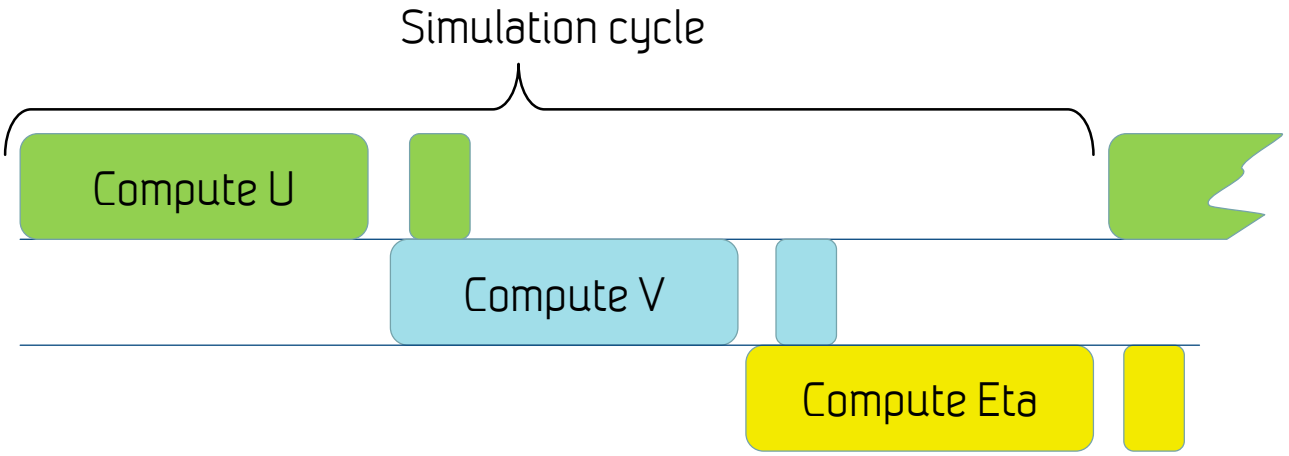
Implementation

- Our CUDA kernel is a function that is executed for each cell in the domain in parallel

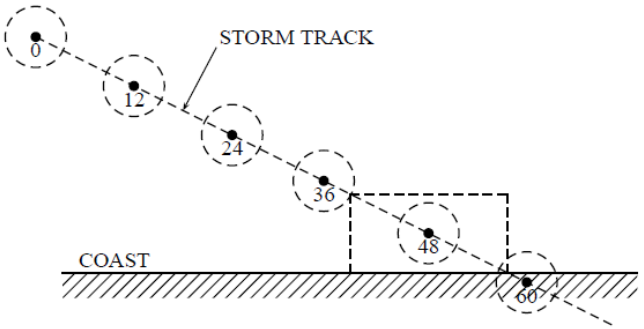
```
__global__ void computeUKernel(const ForwardBackwardLinearParameters params_,
                               const ForwardBackwardLinearCUDAData data_,
                               const float t_) {

    //Data indexing variables
    const unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
    const 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) );
}
```

Implementation

- In addition to U, V, and Eta, we need to compute "external" solutions for the open boundary conditions
- To do this efficiently, we introduce task parallelism:
  the external U for the next time step is calculated simultaneously as V
  the external V for the next time step is calculated simultaneously as Eta
  …



Validation Cases

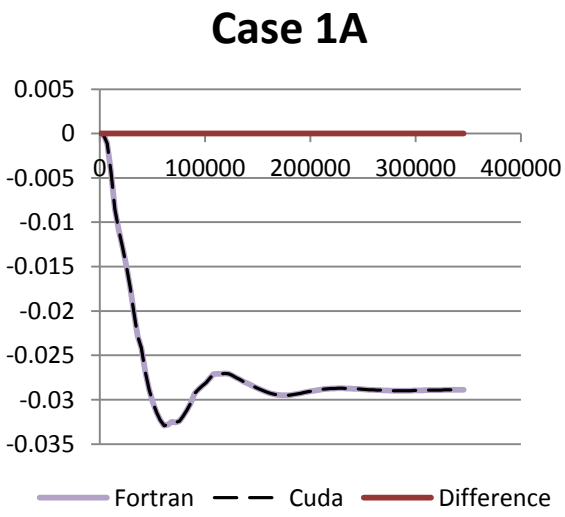


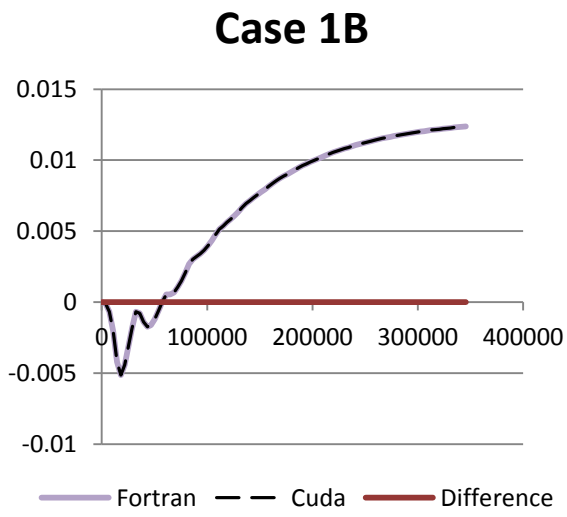
	Closed boundary	Open boundary	Open boundary with shelf
Uniform Along Shore	1A	1B	1C
Bell Shaped Along Shore	2A	2B	2C
Moving Cyclone	3A	3B	3C

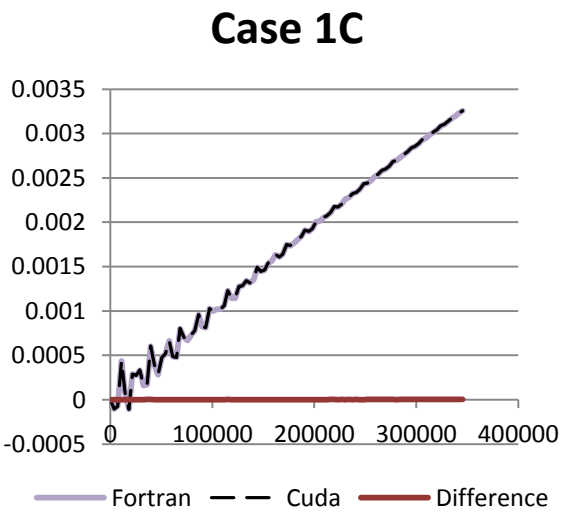
- Nine benchmark cases used to check if the implementation can reproduce the results of the original FORTRAN code
  - Three different types of wind forces (uniform, bell shaped, and a cyclone)
  - Two types of boundaries (open and closed)
  - Two types of bathymetries (flat, and with shelf)
- Difference measured for time series for each case
- Results are visually identical, and show the same dynamics

Uniform Along Shore Wind Stress

- Maximum absolute difference throughout the simulation was 1e-6
  - This is to be expected for single precision simulations



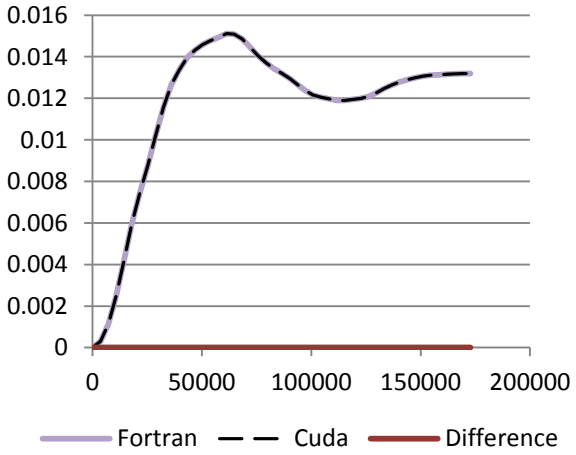




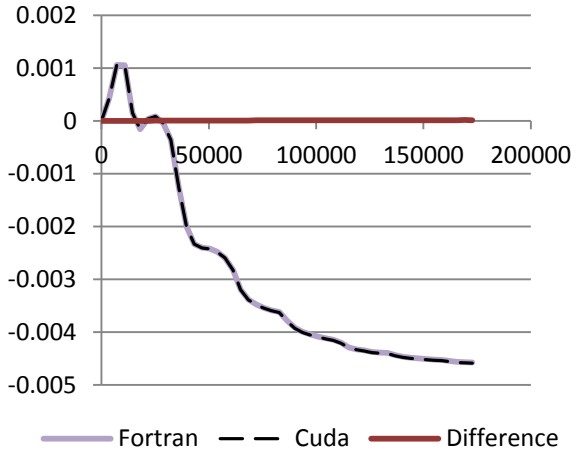
Bell Shaped Along Shore Wind Stress

- Maximum difference for the different cases is 0.0, 1e-5 and 4e-6, respectively
  - Cases B and C run 1920 time steps, which gives a very small error per time step, but still too large.
  - The most probable cause for the discrepancy is differences in the implementation of the open boundaries (closed boundaries give no difference)

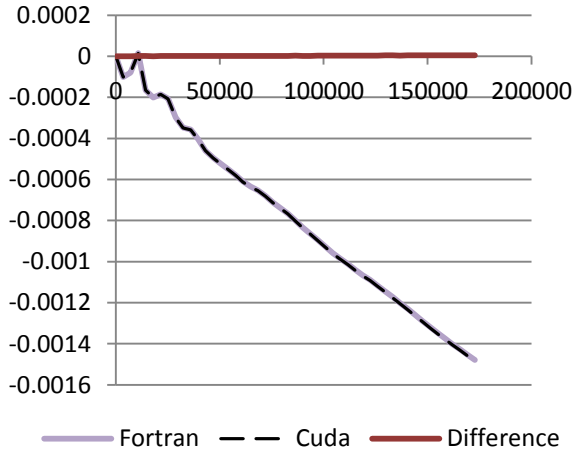
Case 2A



Case 2B

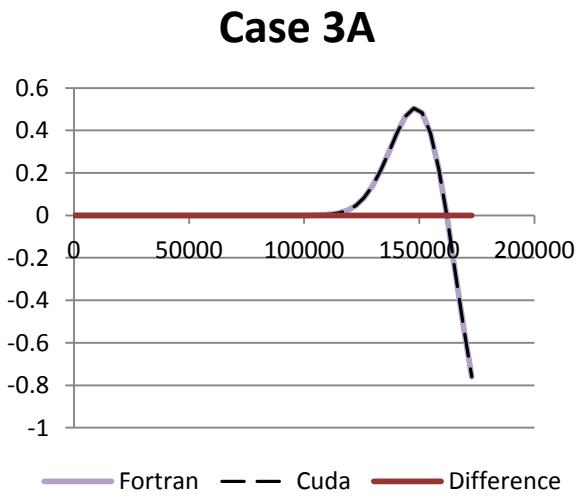


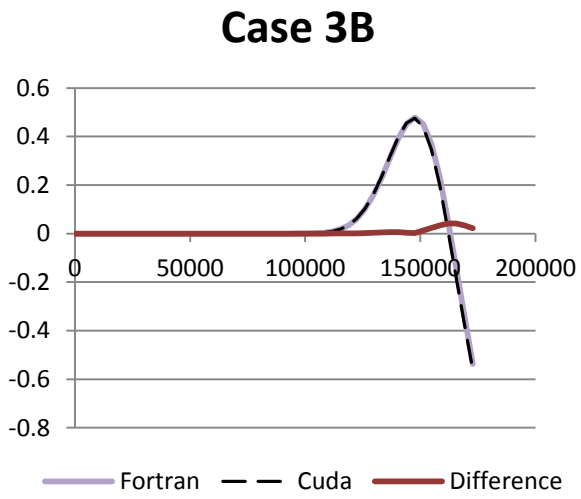
Case 2C

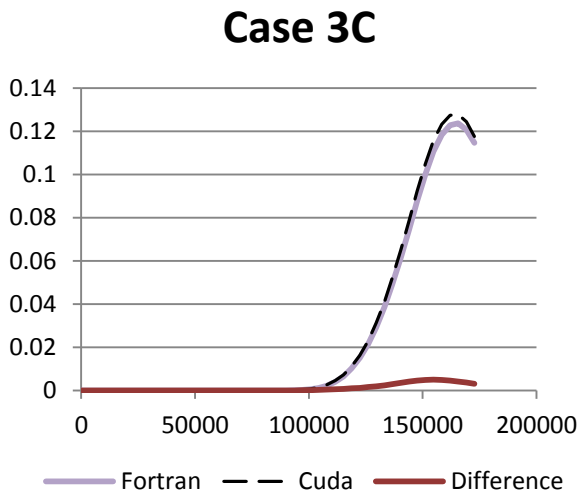


Moving Cyclone Wind Stress

- Maximum difference is 1e-2, 4e-2 and 5e-3.
- Most probable cause for discrepancy is different handling of open boundaries
- The physics is still captured in all models







Summary accuracy

- Results are identical closed boundaries for all wind stress types
  - Negligible discrepancies that are well within the errors imposed by floating point (1e-6)

- Differences in implementation of open boundaries gives rise to discrepancies
  - Is highly probable that identical handling of open boundaries will give results within single precision errors
  - Uniform along-shore wind with open boundaries gives identical results to within single precision

Performance Assessment

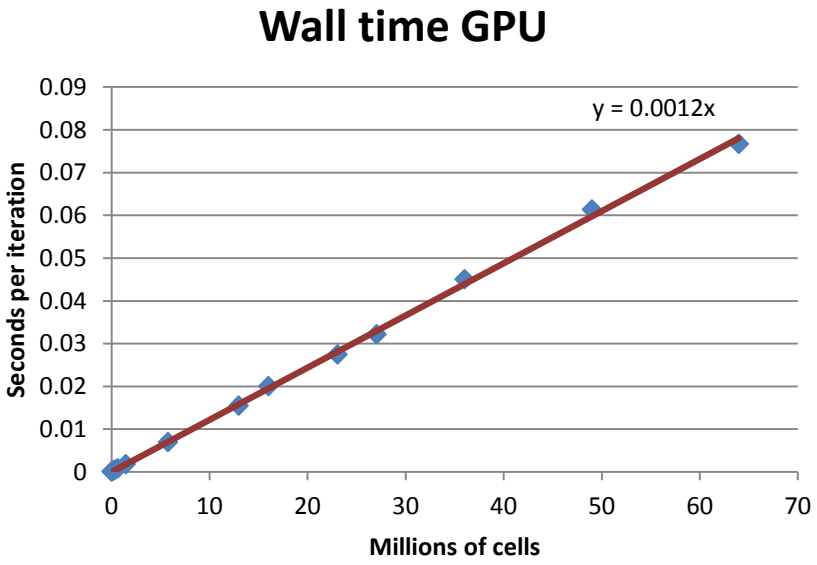
- The GPU implementation is efficient but not optimized
  - The right choices have been made (such as accessing memory by rows and not columns, etc.)
  - No further hand optimizations performed

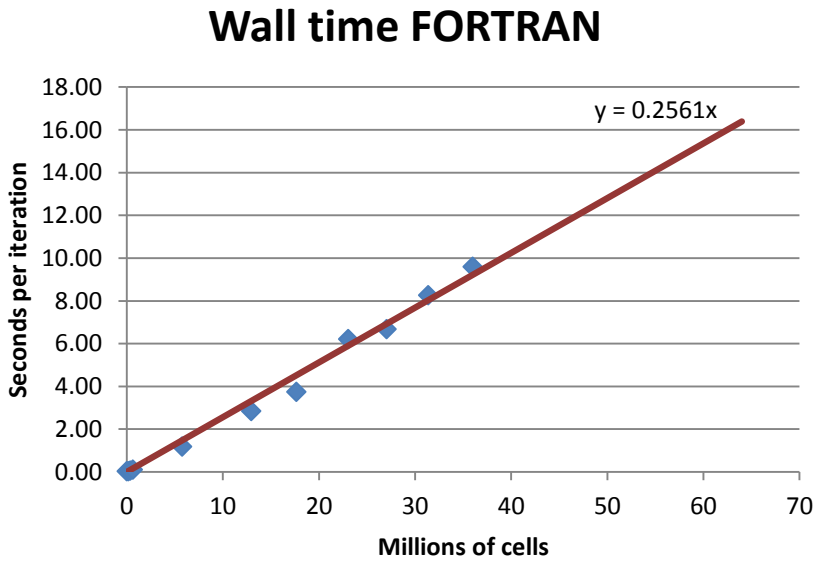
- FORTRAN code compiled with g95 on Ubuntu with "-O3" optimization flag
- CUDA code compiled with CUDA 4.1 and Visual Studio 2010 using standard "release" build settings

- Benchmark run on
  - Intel Core i7-2600k @ 3.7 GHz
  - 8 GiB RAM
  - NVIDIA GeForce 480 GTX GPU @ 1.4 GHz (price today ~2000 NOK)

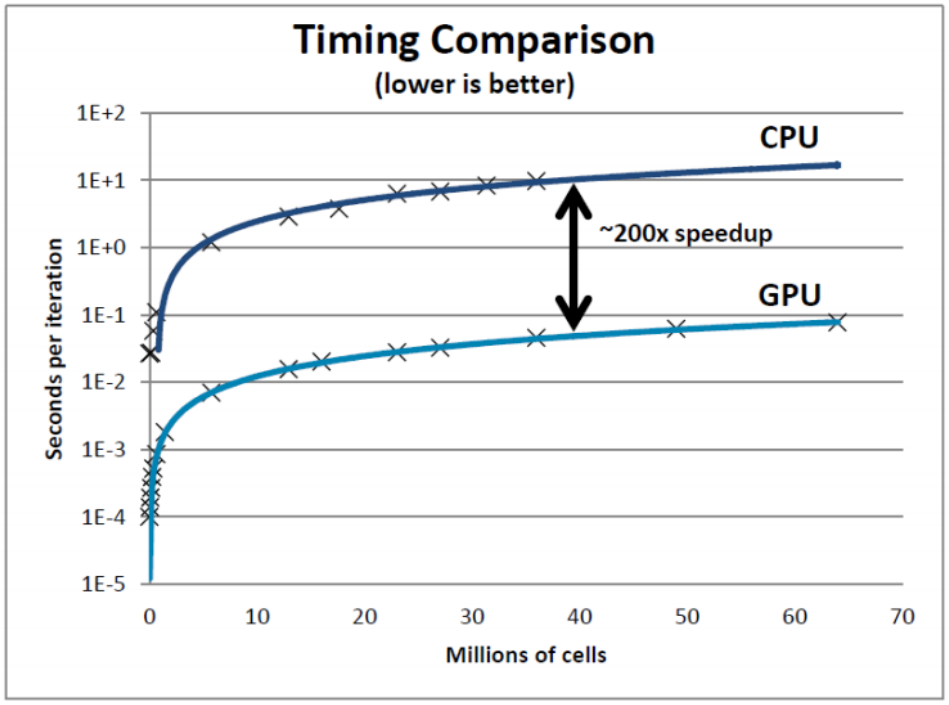
Performance Assessment

- Benchmark run for different (square) domain sizes, and wall clock time measured
- FORTRAN could not go above ~40 million cells
- GPU implementation is roughly 213 times faster than FORTRAN
- Please note: Fortran code is *not* optimized, whilst GPU code is optimized





Performance Assessment



Suitability for Ensemble Methods

- The GPU implementation is O(100) times faster than the FORTRAN implementation
- This enables running large domains, or running many domains simultaneously
- This suits simulation of massive ensembles very well

References and acknowledgements

- Project team consisting of Lars Petter Røed, Kai Christiansen, Göran Boström, Trond Hagen, Yvonne Gusdal.

- Main references:

  - Documentation of simple ocean models for use in ensemble predictions Part I: Theory, **L. P. Røed**, 2012
  - Documentation of simple ocean models for use in ensemble predictions Part II: Benchmark cases, **L. P. Røed**, 2012
  - One-Layer Shallow Water Models on the GPU, **A. R. Brodtkorb**, **T. R. Hagen**, **L. P. Røed**, 2013
  - State-of-the-Art in Heterogeneous Computing, **A. R. Brodtkorb**, **C. Dyken**, **T. R. Hagen**, **J. M. Hjelmervik** and **O. O. Storaasli**, 2010