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

JONSMOD 2016 Conference, Oslo, Norway

André R. Brodtkorb¹, Lars Petter Røed²

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)

1956

2000

Exponential growth (Moore, 1965)
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: \[ P_d \propto f^3 \]

---

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.
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

\[
\partial_t U + \nabla_H \cdot \left( \frac{UU}{H+\eta} \right) + f k \times U + P = \frac{\Delta \tau}{\rho_0} + A' \nabla_H^2 U,
\]

- **Coriolis**
- **Bottom and wind stress**
- **Eddy viscosity parameter**

\( H \) and \( h \) denote the depth of the water column and the water depth, respectively. \( u \) represents the wind stress.
Linearized model

\[
\begin{align*}
\partial_t U - fV &= -gH \partial_x \eta + \frac{\tau_s^x - \tau_b^x}{\rho_0}, \\
\partial_t V + fU &= -gH \partial_y \eta + \frac{\tau_s^y - \tau_b^y}{\rho_0}, \\
\partial_t \eta &= -\partial_x U - \partial_y V,
\end{align*}
\]
Discretized Equations (Numerical scheme)

\[
U_{i,j+1/2}^{n+1} = \frac{1}{B_{i,j+1/2}} \left[ U_{i,j+1/2}^n + \Delta t \left( fV_{i,j+1/2}^n - P_{i,j+1/2}^n + X_{i,j+1/2}^{n+1} \right) \right],
\]
\[
B_{i,j+1/2} = \left( 1 + \frac{R\Delta t}{\bar{H}_{i,j+1/2}} \right),
\]
\[
P_{i,j+1/2}^n = g\bar{H}_{i,j+1/2} \frac{\eta_{i+1/2,j+1/2}^n - \eta_{i-1/2,j+1/2}^n}{\Delta x},
\]
\[
X_{i,j+1/2}^{n+1} = \frac{1}{\rho_0} \left[ \tau_s^{x} \right]_{i,j+1/2}^{n+1}.
\]

\[
V_{i+1/2,j}^{n+1} = \frac{1}{B_{i+1/2,j}} \left[ V_{i+1/2,j}^n + \Delta t \left( fU_{i+1/2,j}^{n+1} - P_{i+1/2,j}^n + Y_{i+1/2,j}^{n+1} \right) \right],
\]
\[
B_{i+1/2,j} = \left( 1 + \frac{R\Delta t}{\bar{H}_{i+1/2,j}} \right),
\]
\[
P_{i+1/2,j}^n = g\bar{H}_{i+1/2,j} \frac{\eta_{i+1/2,j+1/2}^n - \eta_{i+1/2,j-1/2}^n}{\Delta x},
\]
\[
Y_{i+1/2,j}^{n+1} = \frac{1}{\rho_0} \left[ \tau_s^{y} \right]_{i+1/2,j}^{n+1}.
\]

\[
\eta_{i+1/2,j+1/2}^{n+1} = \eta_{i+1/2,j+1/2}^n - \frac{\Delta t}{\Delta x} \left[ U_{i,j+1/2}^{n+1} - U_{i+1,j+1/2}^{n+1} \right] - \frac{\Delta t}{\Delta y} \left[ V_{i+1/2,j}^{n+1} - V_{i+1/2,j+1}^{n+1} \right].
\]
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

```c
__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

<table>
<thead>
<tr>
<th></th>
<th>Closed boundary</th>
<th>Open boundary</th>
<th>Open boundary with shelf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Along Shore</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
</tr>
<tr>
<td>Bell Shaped Along Shore</td>
<td>2A</td>
<td>2B</td>
<td>2C</td>
</tr>
<tr>
<td>Moving Cyclone</td>
<td>3A</td>
<td>3B</td>
<td>3C</td>
</tr>
</tbody>
</table>

- 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 $1 \times 10^{-6}$
- 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)
Moving Cyclone Wind Stress

- Maximum difference is $10^{-2}$, $40^{-2}$ and $50^{-3}$.
- 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

![Timing Comparison Graph]

- Timing Comparison (lower is better)
- CPU
- GPU
- ~200x speedup

Millions of cells vs. Seconds per iteration

Technology for a better society
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
  - Documentation of simple ocean models for use in ensemble predictions
    Part II: Benchmark cases, L. P. Røed, 2012