GPUs for Shallow Water Simulations

André Rigland Brodtkorb[1] and Martin Lilleeng Sætra[2]

[1] SINTEF, Dept. Appl. Math., P.O. Box 124, Blindern, NO-0314 Oslo, Norway

[2] Centre of Mathematics for Applications, University of Oslo, P.O. Box 1053 Blindern, NO-0316 Oslo, Norway

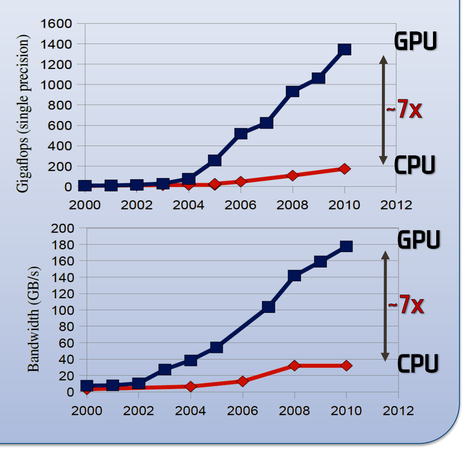
Email: Andre.Brodtkorb@sintef.no, m.l.satra@cma.uio.no,

Abstract: Graphics processing units (GPUs) have over the last decade shown to be 5-50x times faster than the CPU for a variety of algorithms [1]. They are especially well suited for explicit schemes for solving hyperbolic partial differential equations, such as the shallow water equations. We present our work on solving the shallow water equations efficiently using the GPU in this poster, which includes going from a proof-of-concept prototype to an industrially quality simulator engine.

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, three of the top five fastest supercomputers in the world use GPUs, and GPUs are also being used in a multitude of desktop software!

Wolfenstein games, 1981 to 2009

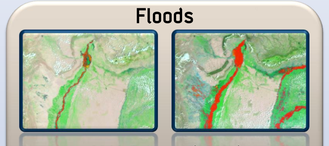


Target Application Areas

Tsunamis

2011: Japan (5321+)
2004: Indian Ocean (230 000)

Floods

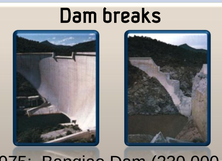


2010: Pakistan (2000+)
1931: China floods (2 500 000+)

Storm Surges

2005: Hurricane Katrina (1836)
1530: Netherlands (100 000+)

Dam breaks



1975: Banqiao Dam (230 000+)
1959: Malpasset (423)

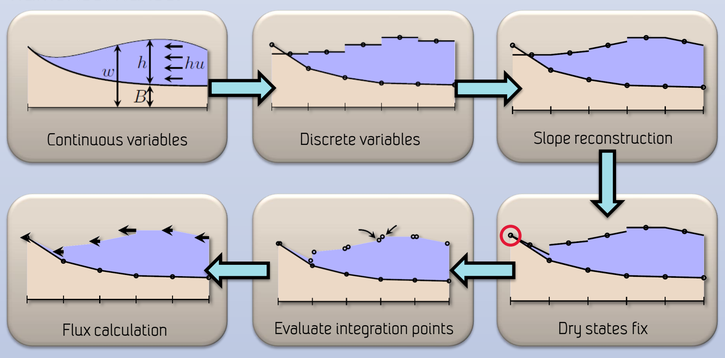
The Shallow Water Equations

The shallow water equations describe gravity-waves in a 2D free surface, under the assumption that the governing flow is horizontal.

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ -ghB_x \\ -ghB_y \end{bmatrix} + \begin{bmatrix} 0 \\ -gu\sqrt{u^2+v^2}/C_z^2 \\ -gv\sqrt{u^2+v^2}/C_z^2 \end{bmatrix}$$

Vector of Conserved variables — Flux Functions — Bed slope source term — Bed friction source term

Our target application areas include tsunamis, storm surges, floods, and dam breaks, in which handling of dry zones is an important question. We use the Kurganov-Petrova [2] scheme to compute our numerical fluxes.

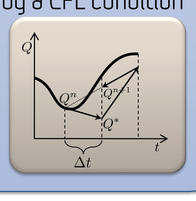


The resulting ODEs per cell are then evolved in time using a second order accurate TVD Runge-Kutta method restructed by a CFL condition

$$Q^*_{ij} = \left[Q^n_{ij} + \Delta t R(Q^n)_{ij}\right] / \left[1 + \Delta t \tilde{H}_f(Q^n_{ij})\right]$$

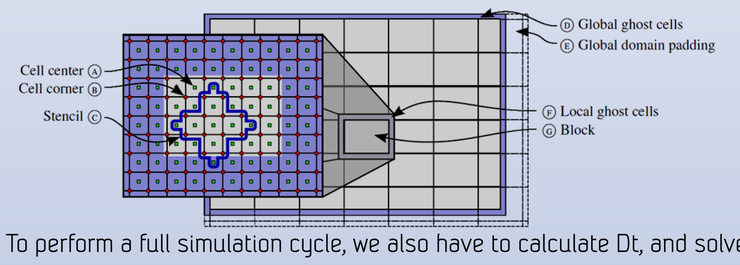
$$Q^{n+1}_{ij} = \left[\tfrac{1}{2}Q^n_{ij} + \tfrac{1}{2}\left[Q^*_{ij} + \Delta t R(Q^*)_{ij}\right]\right] / \left[1 + \tfrac{1}{2}\Delta t \tilde{H}_f(Q^*_{ij})\right]$$

$$\Delta t \leq \tfrac{1}{4}\min\left\{\Delta x / \max_\Omega |u \pm \sqrt{gh}|, \Delta y / \max_\Omega |v \pm \sqrt{gh}|\right\}$$

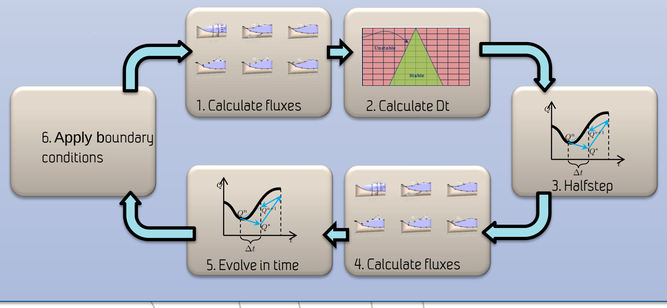


Mapping computations to the GPU

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 flux 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 16x14 threads that collectively computes its subdomain.

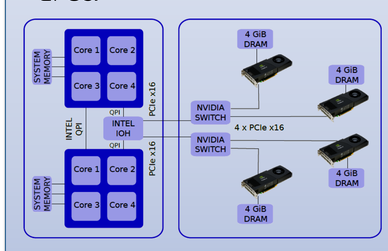


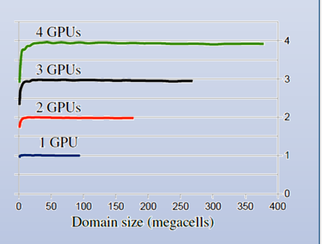
To perform a full simulation cycle, we also have to calculate Dt, and solve the ODEs for each cell.



Multi-GPU

We have extended our simulator to support multi-GPU simulations through domain decomposition. By using techniques such as ghost cell expansion to hide latencies, we are able to reach near-perfect weak and strong scaling on a single node with multiple GPUs.

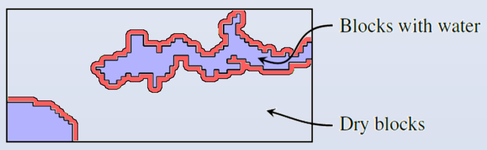




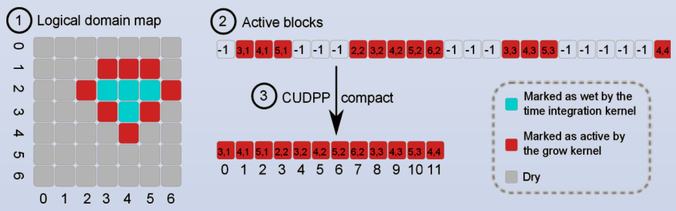
An added benefit of using multiple GPUs is that we are able to handle extremely large domains, over 350 million cells (e.g., 19000x19000 cells)

Sparse Simulations

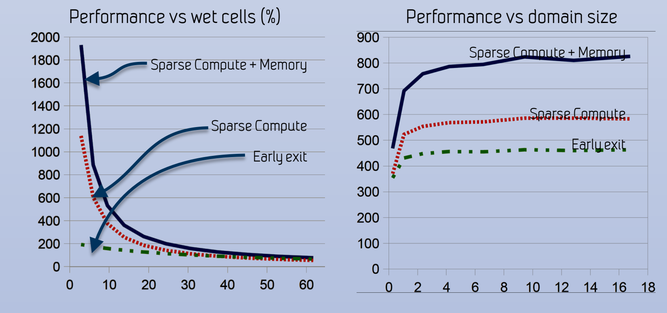
For shallow water simulations, there are often large parts of the domain without water. Our first approach to this was to use an early exit strategy. Using a map of wet blocks, we check whether or not the current block requires any computations.



A large improvement to this approach is to use *sparse* simulations, in which we only perform calculations on parts of the domain with water.



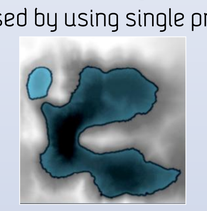
We do this by performing compaction on wet blocks map, to get a list of blocks containing water. We then only launch blocks for the blocks with water. We have also developed this strategy to only store data for wet blocks as well, thus decreasing the memory footprint.

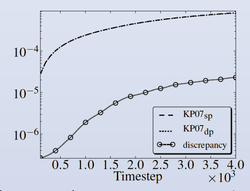


As illustrated in the benchmarking of an idealized circular dam break, our sparse simulation holds excellent performance for typical simulation domains, with a great improvement over the early exit strategy.

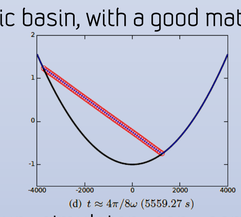
Accuracy

It is beneficial for GPUs to use single precision, as it executes faster, and requires less memory. Our results show that the inherent modelling errors in our scheme shadow any error caused by using single precision.

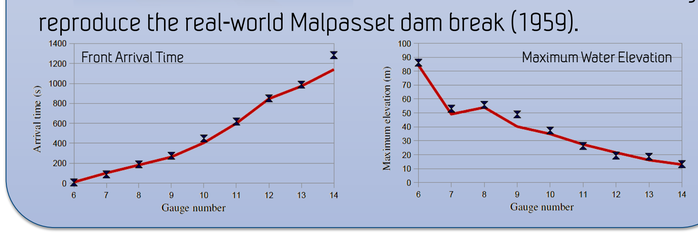




To validate that our simulator produces the correct results, we have compared our numerical results with the analytical solutions of oscillations in a parabolic basin, with a good match.

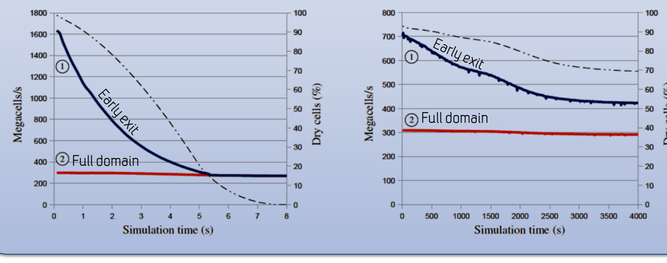


We have further verified that our simulator can correctly reproduce the real-world Malpasset dam break (1959).



Performance

Reporting performance for a simulator is difficult, because timings are highly problem dependent. For the Malpasset dam break case, we can simulate 30 minutes in 11 seconds at up-to 700 megacells / second with the early exit strategy, and for a synthetic dam break, we reach up-to 1600 megacells / second.



References:

[1] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik and O. O. Storaasli, State-of-the-Art in Heterogeneous Computing, Scientific Programming, 18(1) (2010), pp. 1--33

[2] A. Kurganov and G. Petrova. A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. Communications in Mathematical Sciences, 5:133–160, 2007.

Further Reading:

• A. R. Brodtkorb, T. R. Hagen, K.-A. Lie and J. R. Natvig, Simulation and Visualization of the Saint-Venant System using GPUs, Computing and Visualization in Science, special issue on Hot topics in Computational Engineering, 13(7), (2011), pp. 341--353, DOI: 10.1007/s00791-010-0149-x.

• A. R. Brodtkorb, M. L. Sætra, and M. Altinakar, Efficient Shallow Water Simulations on GPUs: Implementation, Visualization, Verification, and Validation, Computers & Fuids, 55, (2011), pp 1--12. DOI: 10.1016/j.compfluid.2011.10.012.

• M. L. Sætra and A. R. Brodtkorb, Shallow Water Simulations on Multiple GPUs, Proceedings of the Para 2010 Conference Part II, Lecture Notes in Computer Science 7134 (2012), pp 56--66, Springer.

• M. L. Sætra, Sparse Grid Shallow Water Simulation on GPUs, technical report, 2012.

•Videos: http://youtube.com/babrodtk