Suitability of Shallow-Water solving methods for GPU acceleration

Floris Buwalda

Delft University of Technology

May 13, 2019

Main Research questions

Which numerical method is best suited for solving the shallow water equations on a GPU in terms of versatility, robustness and speedup?

Subquestions:

1. Explicit vs implicit methods
2. Viability of software package solutions
3. Suitability for FORTRAN/Deltares
4. Possible use of GPU Tensor cores
5. 32 vs 64 bit precision tradeoffs

Literature Research questions

1. What are the SWE and which form are we going to solve?
2. What discretization method exist and which is most suitable?
3. Which time integration methods exist and are suitable?
4. What linear solvers exist and are suitable for GPU implementation?
5. What GPU architecture aspects will need to be taken into consideration?

What are the SWE and which form are we going to solve?

$$\frac{\partial H}{\partial t} = -\frac{\partial}{\partial x}(Hu_x) - \frac{\partial}{\partial y}(Hu_y)$$

$$\frac{\partial u_x}{\partial t} = -\frac{\partial u_x}{\partial x}u_x - \frac{\partial u_x}{\partial y}u_y - g\frac{\partial H}{\partial x} - \frac{gu_x\|\mathbf{u}\|}{C^2H^2} + \nu\left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2}\right)$$

$$\frac{\partial u_x}{\partial t} = -\frac{\partial u_y}{\partial x}u_x - \frac{\partial u_y}{\partial y}u_y - g\frac{\partial H}{\partial y} - \frac{gu_y\|\mathbf{u}\|}{C^2H^2} + \nu\left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2}\right)$$

What are the SWE and which form are we going to solve?

$$
\begin{aligned}
\frac{\partial H}{\partial t} &= -\frac{\partial}{\partial x}\left(Hu_x\right) - \frac{\partial}{\partial y}\left(Hu_y\right) \\
\frac{\partial u_x}{\partial t} &= -\frac{\partial u_x}{\partial x}u_x - \frac{\partial u_x}{\partial y}u_y - g\frac{\partial H}{\partial x} - \frac{gu_x||\mathbf{u}||}{C^2H^2} + \nu\left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2}\right) \\
\frac{\partial u_x}{\partial t} &= -\frac{\partial u_y}{\partial x}u_x - \frac{\partial u_y}{\partial y}u_y - g\frac{\partial H}{\partial y} - \frac{gu_y||\mathbf{u}||}{C^2H^2} + \nu\left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2}\right)
\end{aligned}
$$

Most terms are non-linear!

Full set is Parabolic, without viscosity term it becomes Hyperbolic.

What are the SWE and which form are we going to solve?

Linearised system:

$$\frac{\partial \mathbf{u}}{\partial t} = A\frac{\partial \mathbf{u}}{\partial x} + B\frac{\partial \mathbf{u}}{\partial y} + C\mathbf{u}$$

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ H \end{bmatrix} \quad A = \begin{bmatrix} U_x & 0 & g \\ 0 & U_x & 0 \\ Z & 0 & U_x \end{bmatrix} \quad B = \begin{bmatrix} U_y & 0 & 0 \\ 0 & U_y & g \\ 0 & Z & U_y \end{bmatrix} \quad C = \begin{bmatrix} c & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

A, B, C: $3N \times 3N$ matrices with diagonal $N \times N$ matrices as entries.

What are the SWE and which form are we going to solve?

Stelling & Duinmeijer:

$$\frac{\partial \mathbf{u}_{n+1}}{\partial t} = A\frac{\partial \mathbf{u}_{n+\theta}}{\partial x} + B\frac{\partial \mathbf{u}_{n+\theta}}{\partial y} + C\mathbf{u}_{n+1} + D$$

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ H \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & g \\ 0 & 0 & 0 \\ H'_n & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & g \\ 0 & H'_n & 0 \end{bmatrix} \quad C = \begin{bmatrix} c_f \frac{\|\mathbf{u}_n\|}{H'_n} & 0 & 0 \\ 0 & c_f \frac{\|\mathbf{u}_n\|}{H'_n} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$D \in \mathbb{R}^{3N\times 1} = \begin{bmatrix} \left(u'_x\right)^n \frac{\partial \mathbf{u}_x^n}{\partial x} + \left(u'_y\right)^n \frac{\partial \mathbf{u}_x^n}{\partial y} \\ \left(u'_x\right)^n \frac{\partial \mathbf{u}_y^n}{\partial x} + \left(u'_y\right)^n \frac{\partial \mathbf{u}_y^n}{\partial y} \\ 0 \end{bmatrix}$$

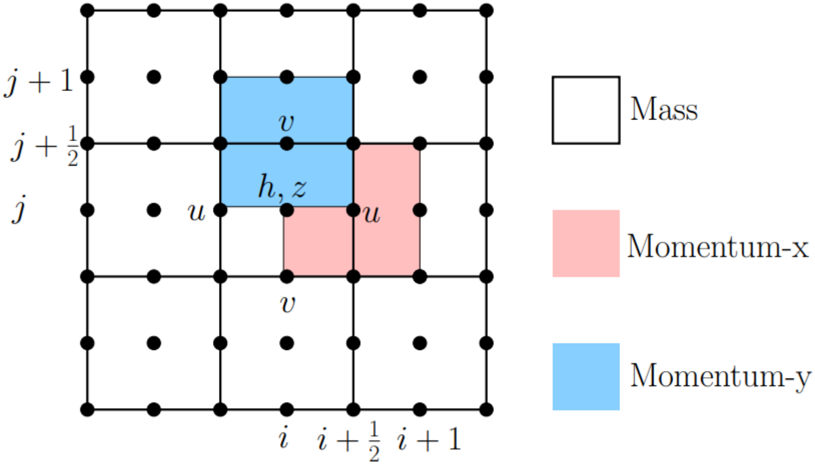
What discretization method exist and which is most suitable?

- Finite differences ← easiest
- Finite volumes
- Finite elements

Grid choice:

- Arakawa C-grid ← best way to avoid odd-even decoupling
- Collocated grid

Arakawa C-grid



Which time integration methods exist and are suitable?

Explicit

- Euler forward ← starting point
- Runge-Kutta 4

Implicit:

- Euler backwards
- Crank-Nicholson
- Theta method ← suggested by Stelling & Duinmeijer
- Alternating direction implicit

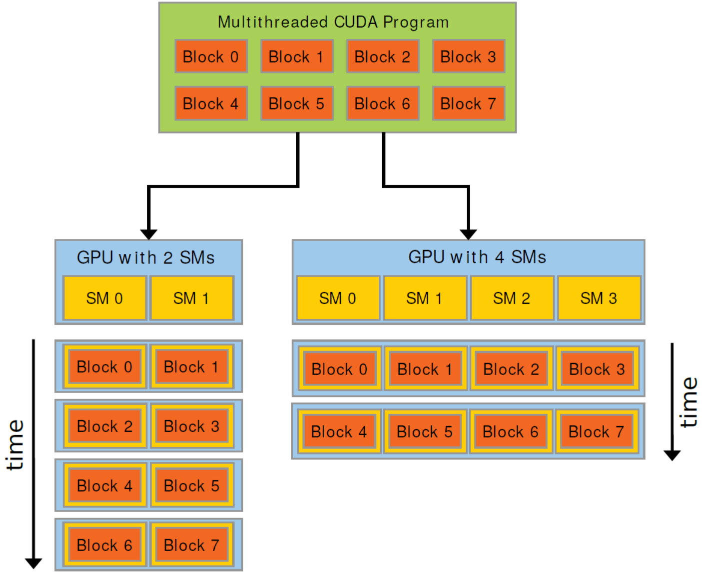
What GPU architecture aspects will need to be taken into consideration?

Nvidia vs AMD?:

- AMD: Cheap High-bandwidth memory, cheap double precision
- Nvidia: Industry standard CUDA platform
- OpenCL is an Open Standard and easily portable but less mature
- Computations will be done on a single Nvidia 2080 Ti GPU
- FP32 performance: 13.45 Teraflops
- FP64 performance: 420 Gflops (1:32)
- Expensive Scientific Computing cards have better double precision, error correcting memory and more memory bandwidth.

What GPU architecture aspects will need to be taken into consideration?

- An Nvidia GPU consists of Streaming multiprocessors that execute blocks of 32 parallel threads sequentially


Multithreaded CUDA Program
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 2 SMs
SM 0
SM 1
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
time
GPU with 4 SMs
SM 0
SM 1
SM 2
SM 3
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
time

What GPU architecture aspects will need to be taken into consideration?

Four main types of memory are available to a GPU program:

- registers: very fast on-chip memory accessible to a single thread
- Shared memory: very fast on-chip memory accessible to all threads in a block
- Device memory: slower memory accessible to all threads in a program
- Host memory: very slow memory accessible to GPU and CPU

GPU computations are often memory-bound so memory management is key.

What linear solvers exist and are suitable for GPU implementation?

- Basic iterative methods: (relaxed) Jacobi & Gauss-Seidel
- Direct solution methods; LU/Cholesky decomposition
- Preconditioned Conjugate gradient
- Multigrid

What linear solvers exist and are suitable for GPU implementation?

- Conjugate gradient is a clear winner as the Stelling & Duinmeijer scheme is SPD.
- Main focus is likely to find an effective preconditioner for CG.
- Other methods mentioned are good options for a preconditioner.

Possible use of Tensor cores

- Tensor cores are extremely efficient at performing dense low-precision matrix-matrix multiplication.
- Most linear solvers perform sparse matrix-vector multiplication which is not suitable
- LU/Cholesky factorization can formulated to contain matrix-matrix multiplications

Test problems

Two proposed problems:

1. Closed (zero-flux boundary) square domain with a non-uniform initial water level
2. Closed square domain with linearly increasing bathymetry and a Dirichlet b.c.

Conclusion

We have successfully defined the scope of the project:

- Stelling & Duinmeijer scheme
- Finite differences discretization on staggered structured grid
- Program will be built in CUDA and run on a 2080 Ti GPU
- Explicit time integration using Euler forward
- Implicit time integration using theta method
- Solve implicit linear system using Conjugate Gradient
- Find an appropriate preconditioner for Conjugate Gradient
- Test solver packages for comparison
- Test additional methods