Simulation of Shallow Water systems using GPUs

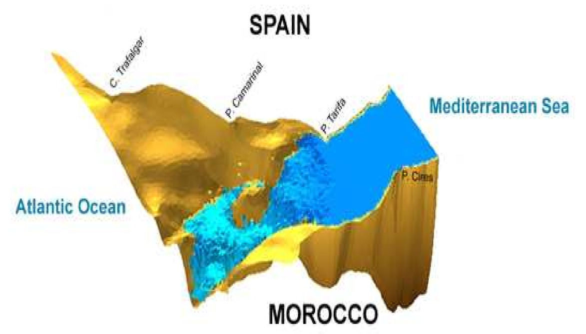
M. J. Castro, M. Lastra, J. M. Mantas, C. Ureña

Depto. de Análisis Matemático
Facultad de Ciencias
Univ. de Málaga. 29071. Málaga
castro@anamat.cie.uma.es

Depto. de Lenguajes y Sistemas Informáticos
E.T.S. Ingeniería Informática y Telecomunicaciones
Univ. de Granada. 18071 Granada
mlastral@ugr.es jmmantas@ugr.es curena@ugr.es

Introduction

- **Goal**: Efficient Simulation of one or two layer fluids that can be modeled by the shallow water systems.
- **Applications**: simulation of rivers, channels, oceanic flows, ...



- **Problem**: Very long lasting simulations in big computacional domains require extremely efficient high performance solvers.
- **Cost effective solution**: To exploit the parallel processing power of modern Graphics Processing Units (GPUs) to speedup the numerical solution of the model.
  - Modern GPUs offer over 100 processing units optimized for performing floating point operations.
  - We need to adapt the calculations and the data domain of the numerical algorithm to the graphics processing pipeline.

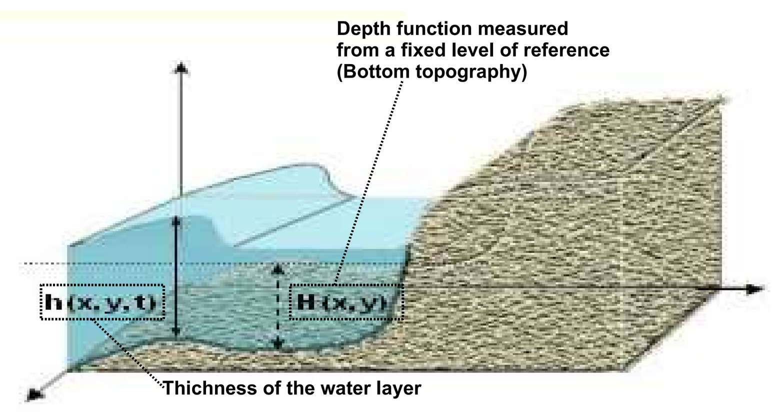
1

Mathematical model: One layer shallow system

PDE system which models a shallow layer of fluid that occupies a bounded subdomain $D \subset \mathbb{R}^2$ under the influence of the gravitational acceleration g.

$$\begin{cases} \dfrac{\partial h}{\partial t} + \dfrac{\partial q_x}{\partial x} + \dfrac{\partial q_y}{\partial y} = 0 \\ \dfrac{\partial q_x}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_x^2}{h} + \dfrac{g}{2}h^2\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_x q_y}{h}\right) = gh\dfrac{\partial H}{\partial x} \\ \dfrac{\partial q_y}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_x q_y}{h}\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_y^2}{h} + \dfrac{g}{2}h^2\right) = gh\dfrac{\partial H}{\partial y} \end{cases}$$

Problem: To study the time evolution of $W(x, y) = [h, q_x, q_y]^T$ fulfilling the Shallow water equations.

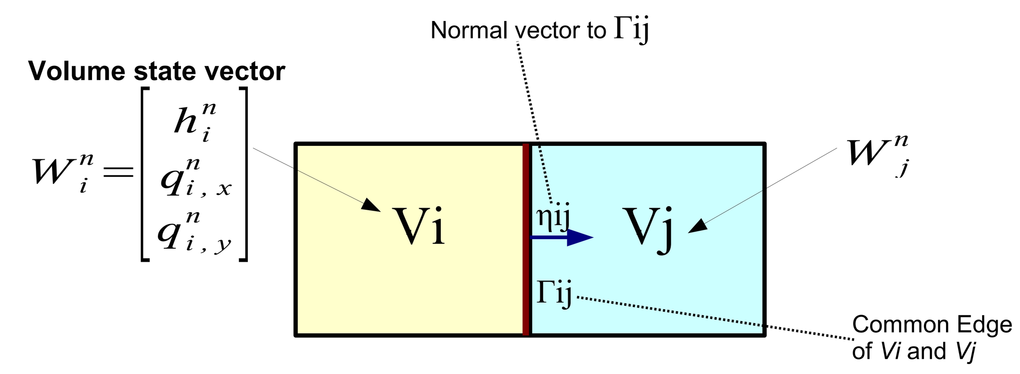


$[q_x(x,y,t),\ q_y(x,y,t)]$ = depth-averaged velocity of the layer at (x,y) and time t

2

Numerical Scheme

- The shallow water system is discretized by means of a Finite Volume scheme.
- Domain D is divided in M finite volumes (closed polygons): $V_i \subset \mathbb{R}^2$, $i = 1, \ldots, M$, with area $| V_i |$.



$$W_i^n = \begin{bmatrix} h_i^n \\ q_{i,x}^n \\ q_{i,y}^n \end{bmatrix}$$

Time stepping

$$W_i^0 \xrightarrow{\Delta t^0} W_i^1 \xrightarrow{\Delta t^1} \ldots \rightarrow W_i^n \xrightarrow{\Delta t^n} W_i^{n+1} \rightarrow \ldots$$

$$W_i^{n+1} = W_i^n - \frac{\Delta t^n}{| V_i |} \sum_{j \in Neighbors_i} | \Gamma_{ij} | \, F_{ij}^n$$

3

Numerical Scheme (2)

$$F_{ij}^n = P_{ij}^n \left[A_{ij}^n (W_j^n - W_i^n) - S_{ij}^n (H_j - H_i) \right].$$

$$P_{ij}^n = \frac{1}{2} K_{ij}^n \cdot \left[I - sgn(D_{ij}^n) \right] \cdot (K_{ij}^n)^{-1}$$

where $A_{ij}^n \in \mathbb{R}^{3\times3}$ and $S_{ij}^n \in \mathbb{R}^3$ depends on W_i^n and W_j^n. D_{ij}^n is a diagonal matrix whose coefficients are the eigenvalues of A_{ij}^n and the columns of $K_{ij}^n \in \mathbb{R}^{3\times3}$ are the associated eigenvectors.

Computation of Δt^n

$$\Delta t^n = \min_{i=1,\ldots,M} \left\{ \left[\frac{\sum_{j \in Neighbors_i} | \Gamma_{ij} | \, \| D_{ij}^n \|_\infty}{2\gamma \, | V_i |} \right]^{-1} \right\}$$

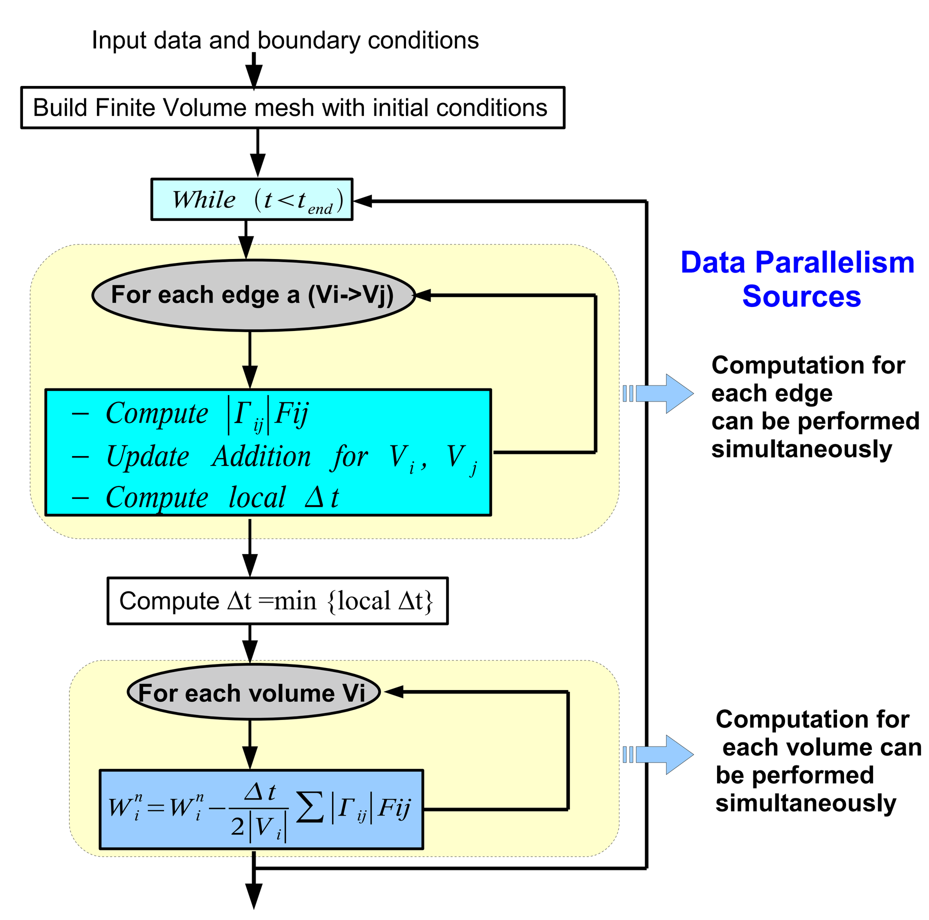
where $0 < \gamma \geq 1$.

Remarks:

- High arithmetic intensity and locality (the computation for each edge or volume only depends on data from neighbour volumes).
- High degree of potential data parallelism.

4

Numerical Scheme (3)

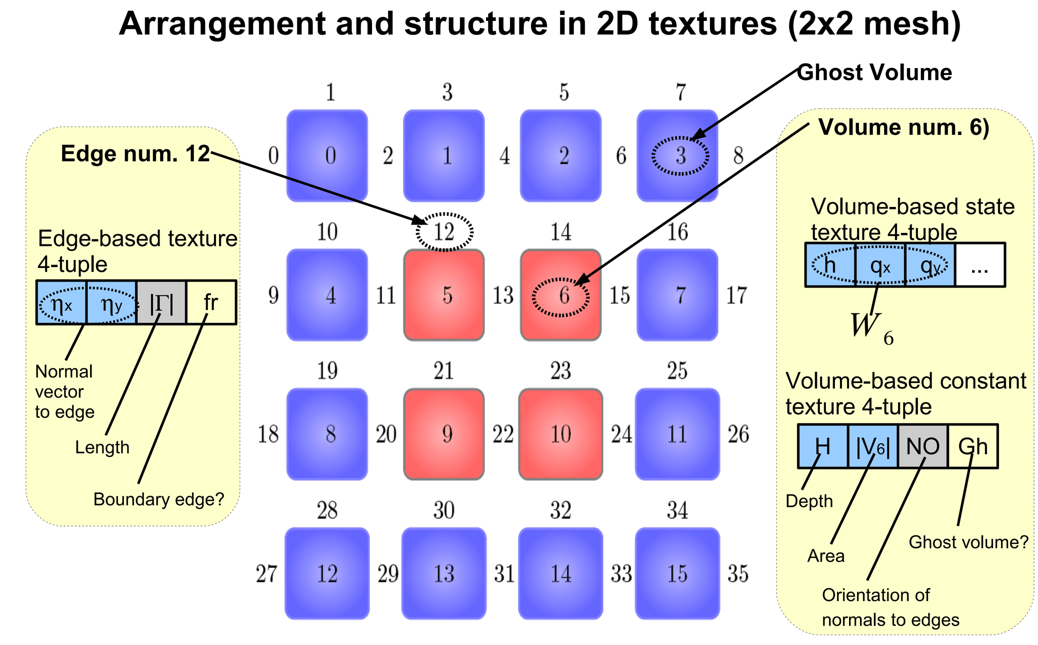


5

Data Storage in GPU

Information about volumes and edges must be stored as 2D textures (it allows the storage of $n \times m$ floating point 4-tuples):

- Two textures to store volume-based information (one 4-tuple per volume): one stores the values of $W(x, y, t)$ for each volume and the other stores constant data associated to each volume.
- One texture to store edge-based information (4-tuple per edge).

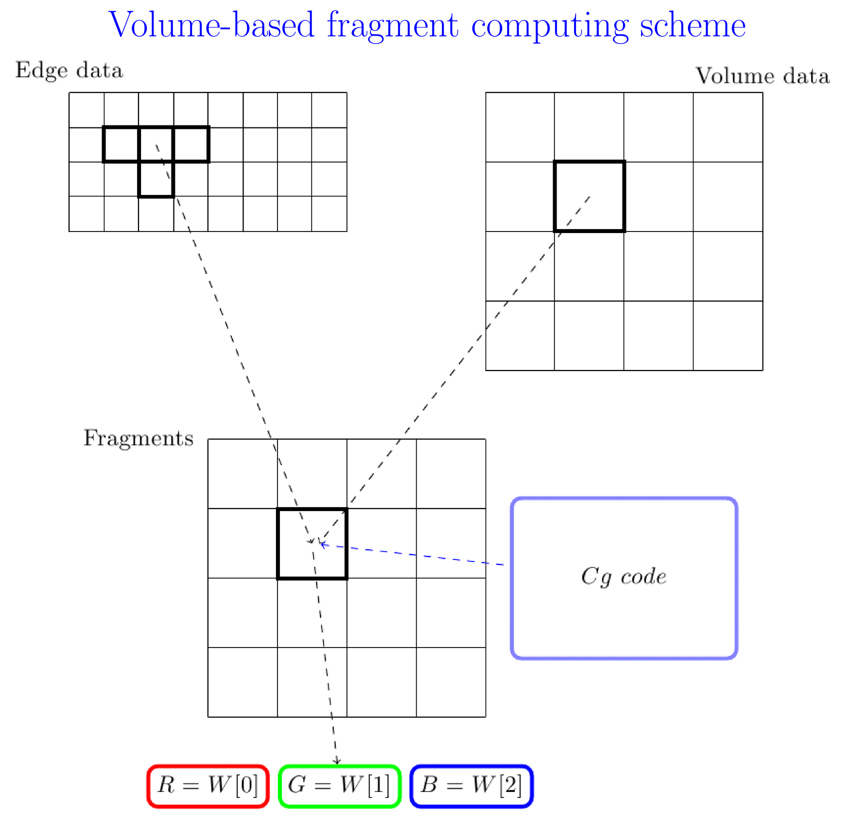


6

Computing Units in the GPU

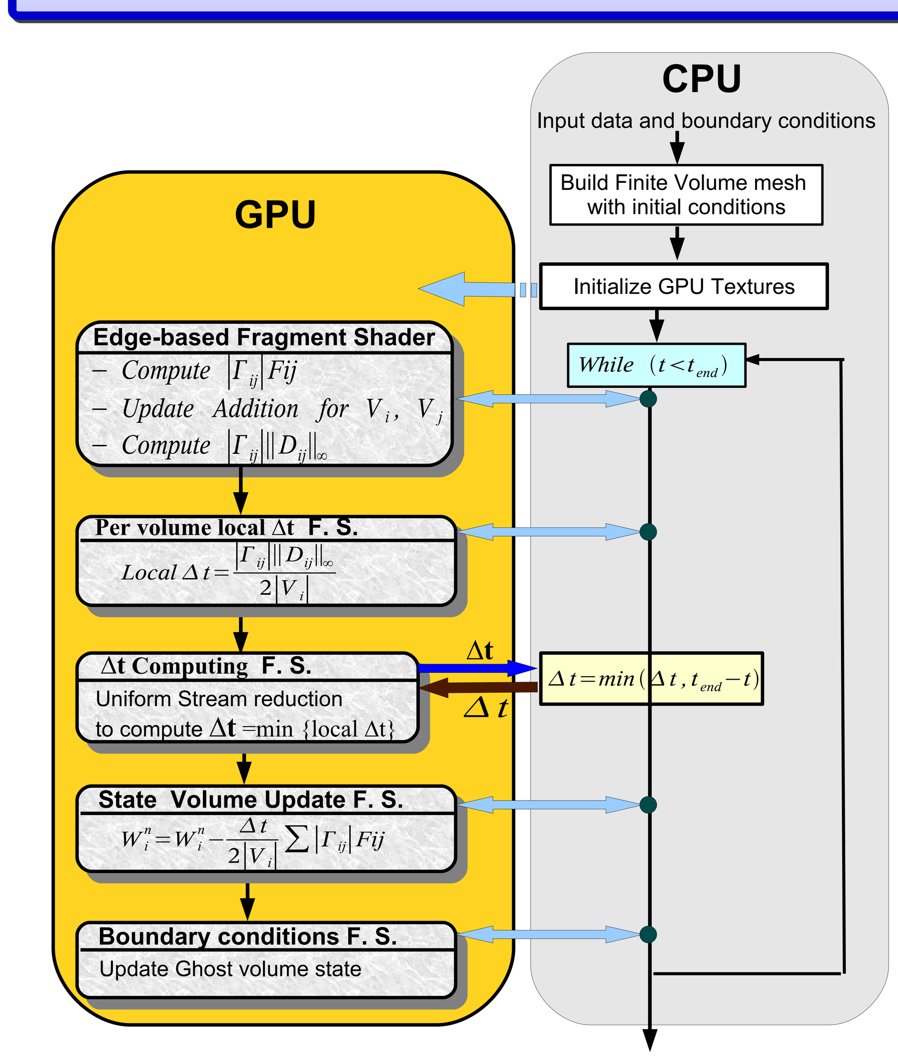
- Each computing step must be performed in a data parallel fashion following a *fragment shader* written in Cg.
- The same code is applied to each fragment (volume or edge) and other textures can be accessed to obtain input data.

Volume-based fragment computing scheme



7

Computing Phases in GPU

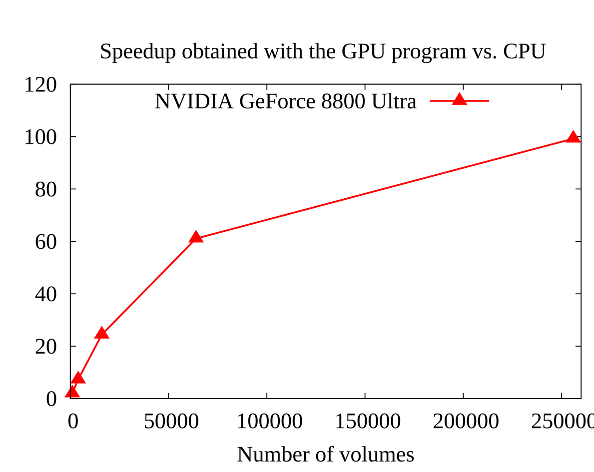


8

Performance Results

- $1\ m \times 10\ m$ rectangular channel with $H(x, y) = 1 - \cos(2\pi x)/2$.
- Time interval $[0, 5]$ with $\gamma = 0.9$ and wall boundary conditions.
- $W_i^0(x, y) = [H(x, y) + 2(x < 5), \quad 0, \quad 0]^T$.
- **CPU**: Intel Xeon Nocona 2.66 Ghz. SSE-optimized code (Intel IPP 4.1). Intel C++ comp. em64t extension (-O2).
- **GPU**: NVIDIA GeForce 8800 Ultra. OpenGL + Cg Code.

Mesh size	T_{CPU}	T_{GPU}
100 x 10	1.05	0.53
200 x 20	8.09	1.11
400 x 40	64.23	2.63
800 x 80	510.6	8.36
1600 x 160	4046.58	40.79



Conclusion: Simulations on an NVIDIA GeForce 8800 Ultra GPU (about 700 dollars) are found to be up to two orders of magnitude faster than the SSE-optimized CPU version of the code.

9