

Acceleration of hydraulic and environmental simulation tools using Graphical Processing Units GPUs



HPC ADMINTECH 2016

Asier Lacasta

*Grupo de Hidráulica Computacional
Universidad de Zaragoza*



**Universidad
Zaragoza**



GPU RESEARCH
CENTER

GHC @ UNIZAR-Liftec

- Developing numerical tools since 1990 for the simulation of free Surface flows
- 11 people (Profs, Post-docs, Researchers, External-collaborators, PhD...)
- + Junior researchers and students
- Industrial collaboration for developing and consultancy



Motivation

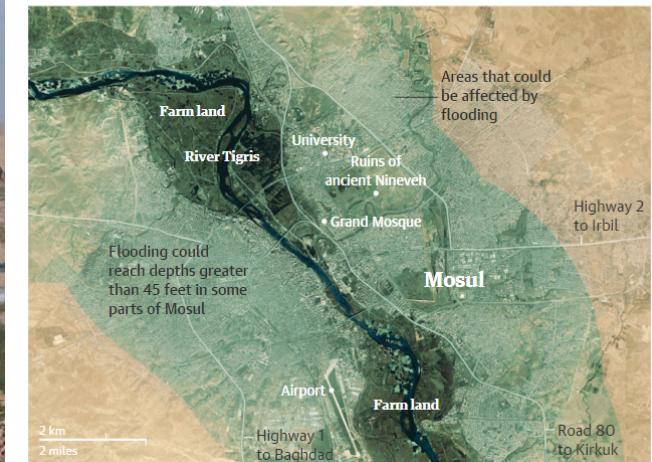
- The Guardian 03/03/2016

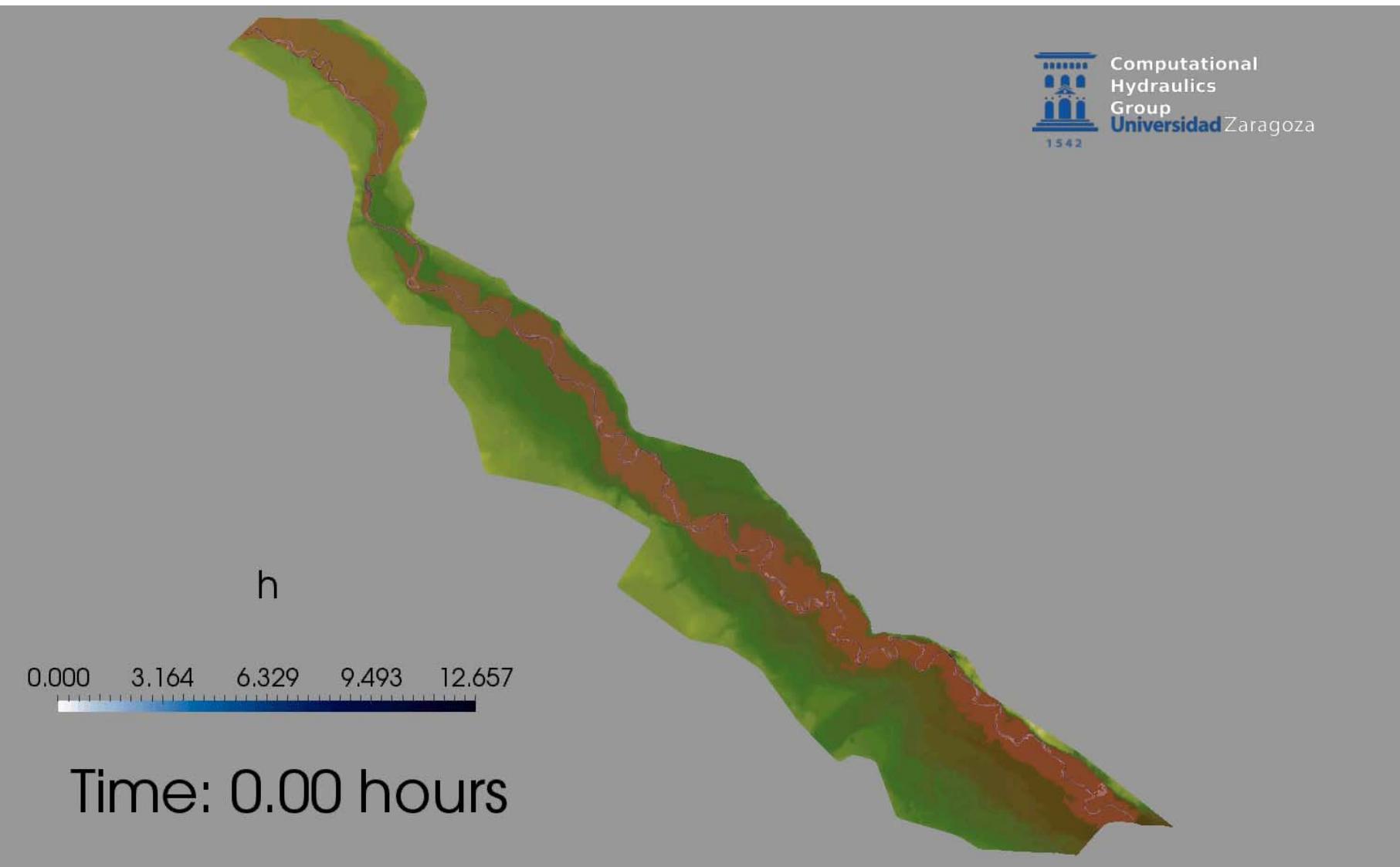


One million people along the Tigris could be at risk from a 20-metre-high flash flood if the Mosul dam collapses



Residents of Mosul have been advised to move at least 3.5 miles away from the river

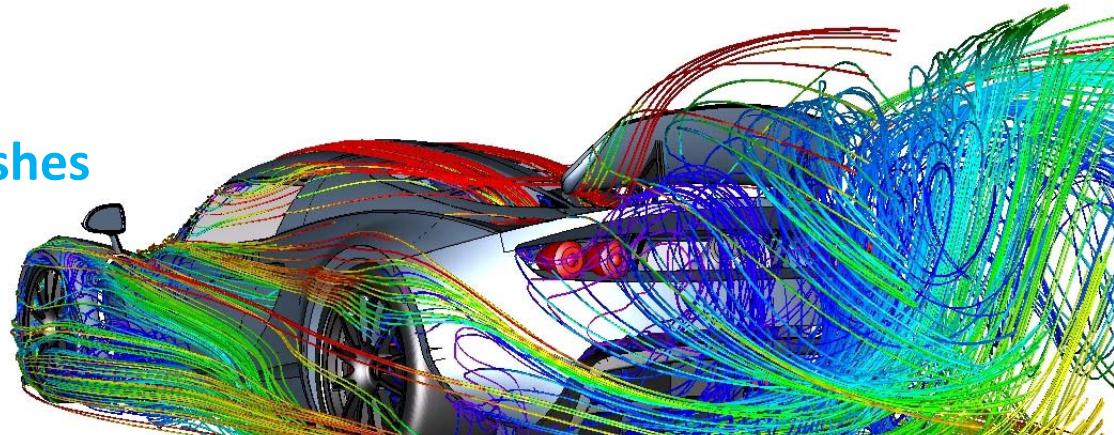
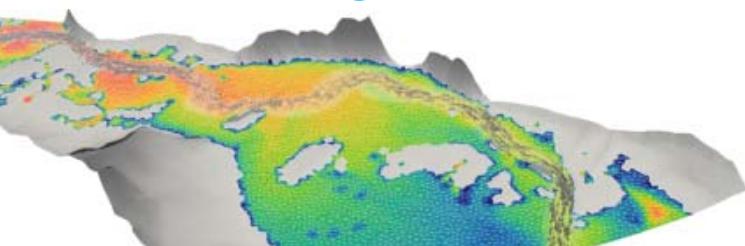






Motivation

- There are many applications in engineering that require **computational modeling**
- Some of them do not require large computational facilities
(Simplified models, steady analysis...)
- However, all the applications **related with CFD or Structural mechanics usually require high computational effort** to perform their analysis
 - **Heavy mathematical models**
 - **High-resolution computational meshes**



Our application

The **objective** is to **simulate** the **unsteady flow motion of free-Surface flows** over some Surface (e.g. Rivers, roads, mountains...)



Riverflow2D



- Riverflow2D is the commercial form of the numerical models developed at GHC.
- Consulting and software development firm
- Based in the USA
- Hydronia Europe, Spain
- R&D partnership with University of Zaragoza to develop the RiverFlow2D and OilFlow2D models
- Provides specialized professional consulting to governmental, municipal, and private clients worldwide.

Hydronia



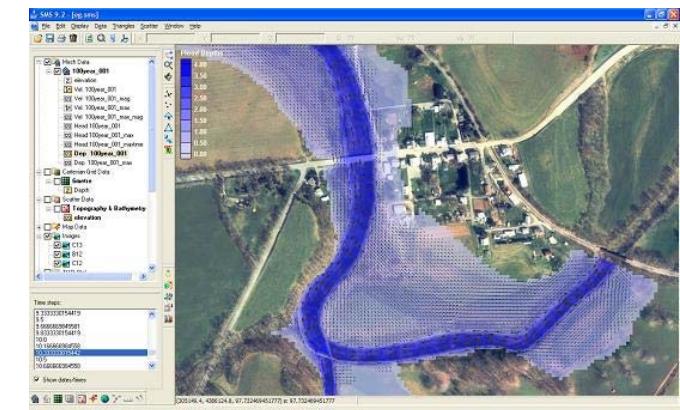
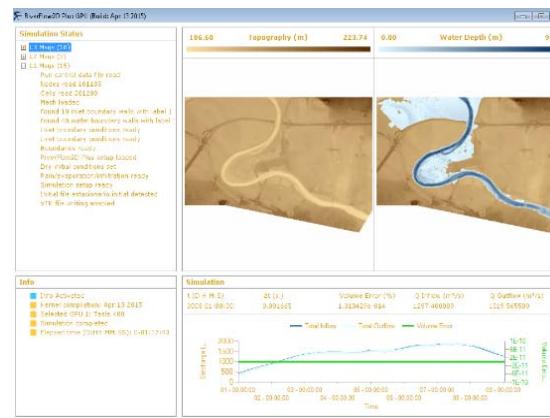
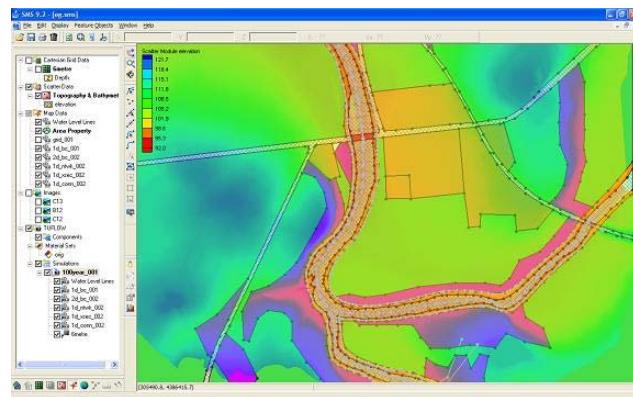
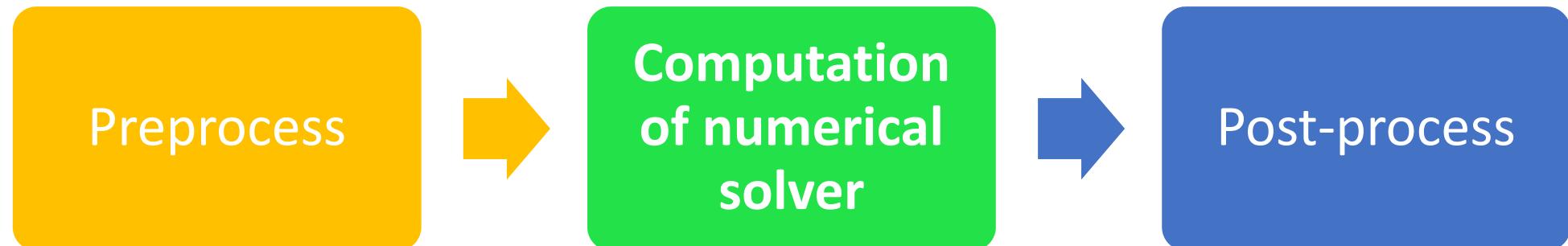
RiverFlow2D RiverFlow2D^{GPU} OilFlow2D



Motivation



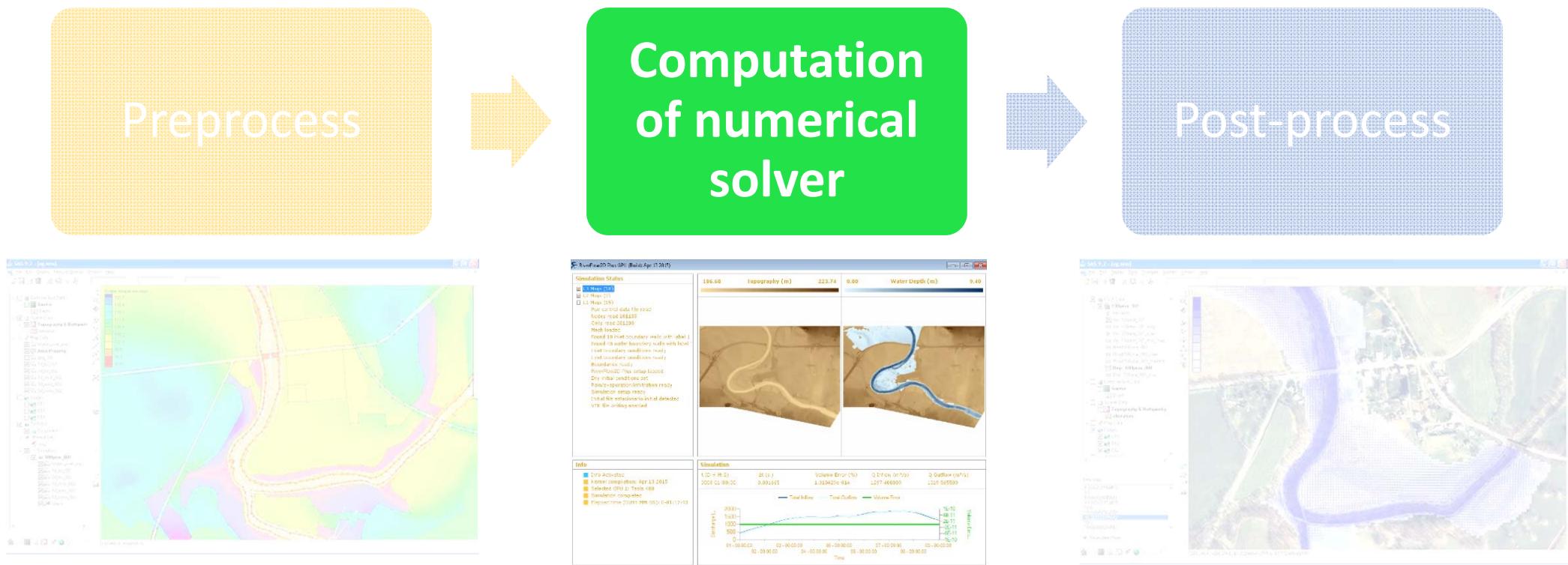
RiverFlow2D



Motivation



RiverFlow2D



Riverflow2D



- Released in 2006
- Finite Element numerical model
- Shared Memory parallelization capabilities

- Released in 2013
- Finite Volume numerical method
- Shared Memory parallelization (OpenMP)

- Released in 2014
- Finite Volume numerical engine
- Designed to be used in the last generation nVidia GPUs

GPU programming



Application of HPC

The problem can be locally stated as

```
for i=1..max  
    ...  
f[i]=fAux[i]+...  
end for
```

Application of HPC



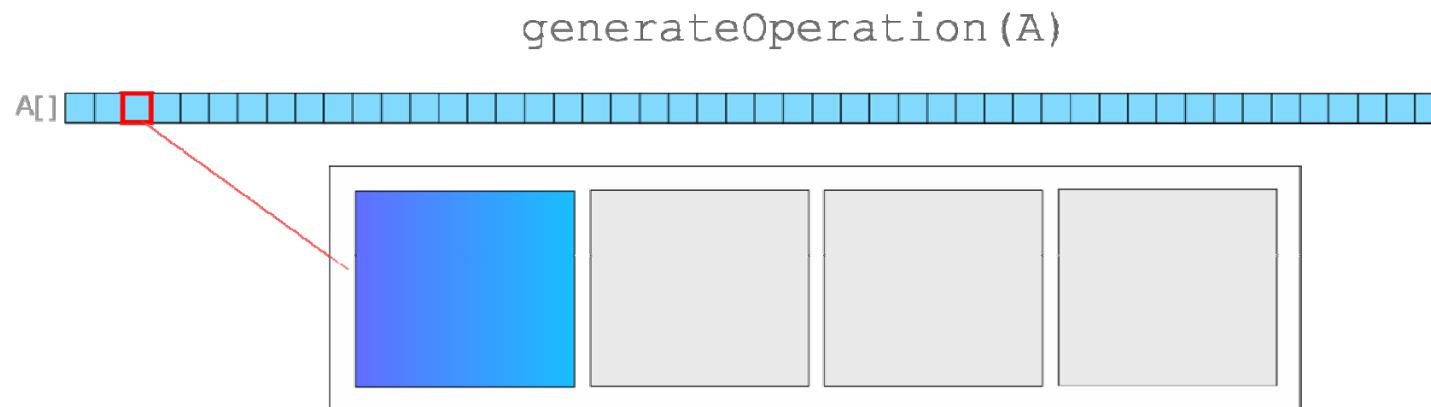
Options to operate with vector data type ([Flynn's Taxonomy](#))

- Iterative handling (Supported by every conventional processors (**SISD**))
- Multi-thread handling (Supported by processors with multi-thread capabilities (**SIMT**))
- Vectorial handling (Supported by vectorial processors (**SIMD**))
- Distributed Memory (Large computational facilities (**SPMD**))



Application of HPC

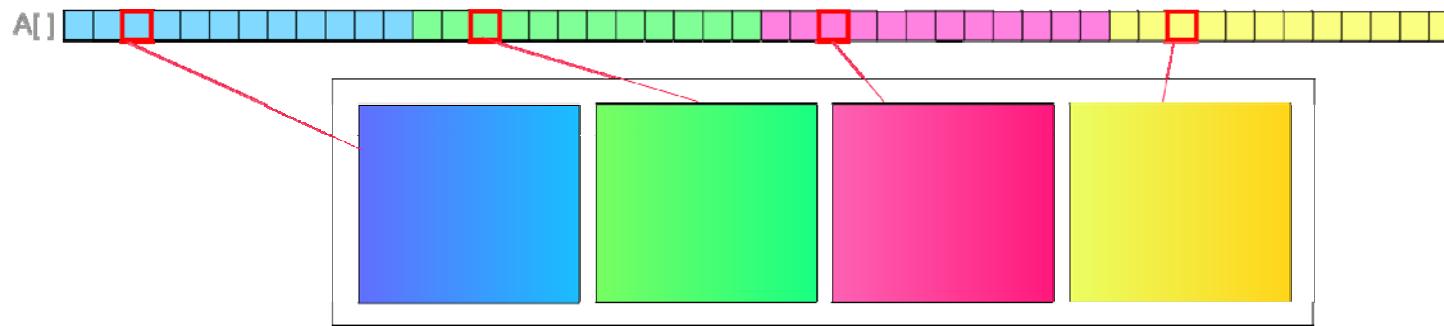
- Iterative handling (Supported by every conventional processors (**SISD**))
 - Elements are processed one by one





Application of HPC

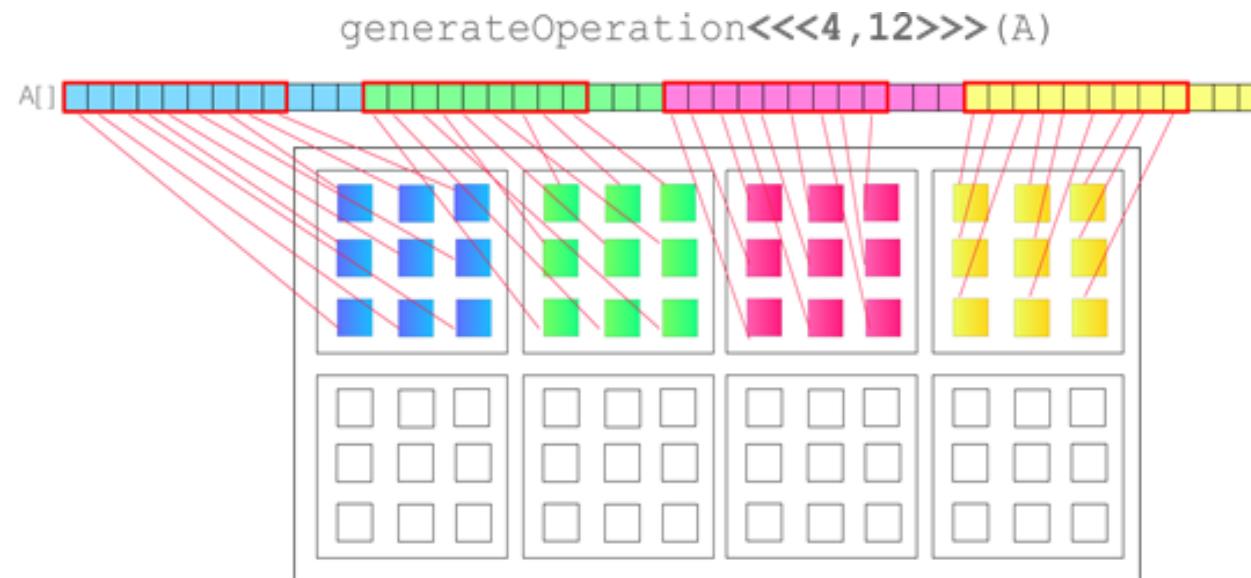
- Multi-thread handling (Supported by processors with multi-thread capabilities (**SIMT**))
 - Subsets of m elements are simultaneously processed





Application of HPC

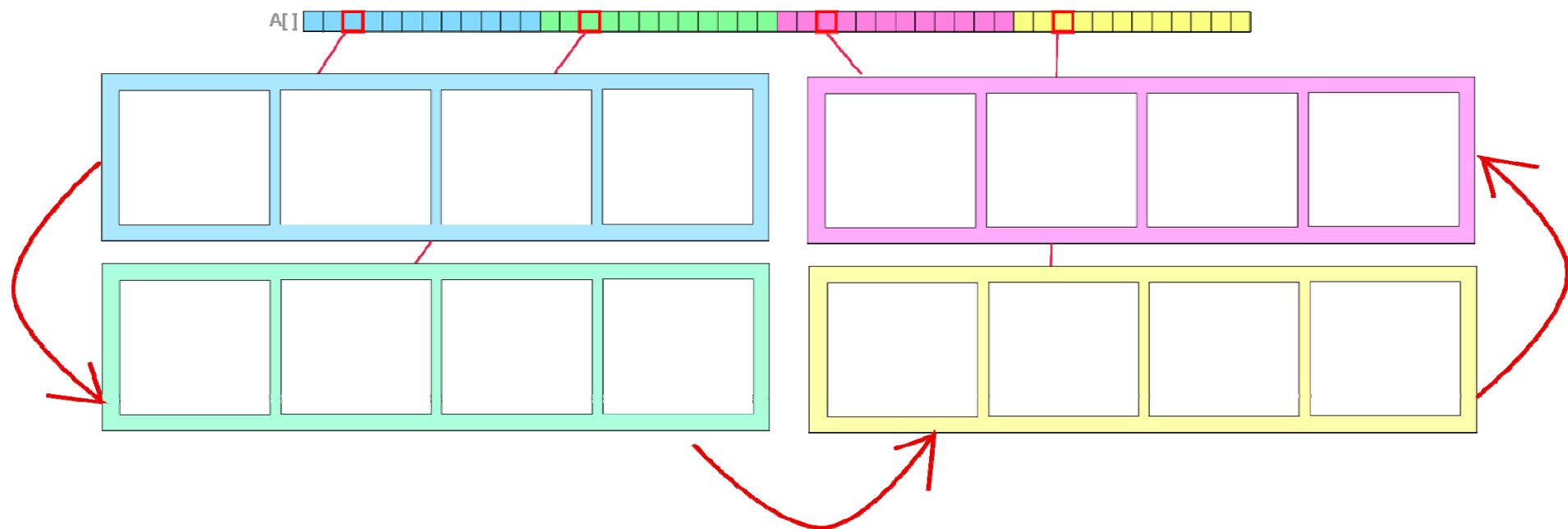
- Vectorial handling (Supported by vectorial processors (**SIMD**))
 - n elements are simultaneously processed.



Application of HPC



- Distributed Memory (Large computational facilities (**SPMD**)))
 - The computational load is explicitly distributed along the system



Application of HPC



- Iterative handling (Supported by every conventional processors (**SISD**))
 - Elements are processed one by one
- Multi-thread handling (Supported by processors with multi-thread capabilities (**SIMT**))
 - Subsets of m elements are simultaneously processed
- Vectorial handling (Supported by vectorial processors (**SIMD**))
 - n elements are simultaneously processed.
- Distributed Memory (Large computational facilities (**SPMD**)))
 - The computational load is explicitly distributed along the system



GPU Implementation

```
for i=1..max
```

```
    . . .
```

```
    f[i]=fAux[i]+...
```

```
end for
```



GPU Implementation

The whole 1024 iterations must be executed...

Iteración 1	Iteración 33
Iteración 2	Iteración 34
Iteración 3	Iteración 35
Iteración 4	Iteración 36
Iteración 5	Iteración 37
Iteración 6	Iteración 38
Iteración 7	Iteración 39
Iteración 8	Iteración 40
Iteración 9	Iteración 41
Iteración 10	Iteración 42
Iteración 11	Iteración 43
Iteración 12	Iteración 44
Iteración 13	Iteración 45
Iteración 14	Iteración 46
Iteración 15	Iteración 47
Iteración 16	Iteración 48
Iteración 17	Iteración 49
Iteración 18	Iteración 50
Iteración 19	Iteración 51
Iteración 20	Iteración 52
Iteración 21	Iteración 53
Iteración 22	Iteración 54
Iteración 23	Iteración 55
Iteración 24	Iteración 56
Iteración 25	Iteración 57
Iteración 26	Iteración 58
Iteración 27	Iteración 59
Iteración 28	Iteración 60
Iteración 29	Iteración 61
Iteración 30	Iteración 62
Iteración 31	Iteración 63
Iteración 32	Iteración 64

■ ■ ■

Iteración 961	Iteración 993
Iteración 962	Iteración 994
Iteración 963	Iteración 995
Iteración 964	Iteración 996
Iteración 965	Iteración 997
Iteración 966	Iteración 998
Iteración 967	Iteración 999
Iteración 968	Iteración 1000
Iteración 969	Iteración 1001
Iteración 970	Iteración 1002
Iteración 971	Iteración 1003
Iteración 972	Iteración 1004
Iteración 973	Iteración 1005
Iteración 974	Iteración 1006
Iteración 975	Iteración 1007
Iteración 976	Iteración 1008
Iteración 977	Iteración 1009
Iteración 978	Iteración 1010
Iteración 979	Iteración 1011
Iteración 980	Iteración 1012
Iteración 981	Iteración 1013
Iteración 982	Iteración 1014
Iteración 983	Iteración 1015
Iteración 984	Iteración 1016
Iteración 985	Iteración 1017
Iteración 986	Iteración 1018
Iteración 987	Iteración 1019
Iteración 988	Iteración 1020
Iteración 989	Iteración 1021
Iteración 990	Iteración 1022
Iteración 991	Iteración 1023
Iteración 992	Iteración 1024



GPU Implementation

The whole 1024 ~~iterations~~ must be executed...

...but they are required to be executed at the same time

Iteración 1	Iteración 33
Iteración 2	Iteración 34
Iteración 3	Iteración 35
Iteración 4	Iteración 36
Iteración 5	Iteración 37
Iteración 6	Iteración 38
Iteración 7	Iteración 39
Iteración 8	Iteración 40
Iteración 9	Iteración 41
Iteración 10	Iteración 42
Iteración 11	Iteración 43
Iteración 12	Iteración 44
Iteración 13	Iteración 45
Iteración 14	Iteración 46
Iteración 15	Iteración 47
Iteración 16	Iteración 48
Iteración 17	Iteración 49
Iteración 18	Iteración 50
Iteración 19	Iteración 51
Iteración 20	Iteración 52
Iteración 21	Iteración 53
Iteración 22	Iteración 54
Iteración 23	Iteración 55
Iteración 24	Iteración 56
Iteración 25	Iteración 57
Iteración 26	Iteración 58
Iteración 27	Iteración 59
Iteración 28	Iteración 60
Iteración 29	Iteración 61
Iteración 30	Iteración 62
Iteración 31	Iteración 63
Iteración 32	Iteración 64

■ ■ ■

Iteración 961	Iteración 993
Iteración 962	Iteración 994
Iteración 963	Iteración 995
Iteración 964	Iteración 996
Iteración 965	Iteración 997
Iteración 966	Iteración 998
Iteración 967	Iteración 999
Iteración 968	Iteración 1000
Iteración 969	Iteración 1001
Iteración 970	Iteración 1002
Iteración 971	Iteración 1003
Iteración 972	Iteración 1004
Iteración 973	Iteración 1005
Iteración 974	Iteración 1006
Iteración 975	Iteración 1007
Iteración 976	Iteración 1008
Iteración 977	Iteración 1009
Iteración 978	Iteración 1010
Iteración 979	Iteración 1011
Iteración 980	Iteración 1012
Iteración 981	Iteración 1013
Iteración 982	Iteración 1014
Iteración 983	Iteración 1015
Iteración 984	Iteración 1016
Iteración 985	Iteración 1017
Iteración 986	Iteración 1018
Iteración 987	Iteración 1019
Iteración 988	Iteración 1020
Iteración 989	Iteración 1021
Iteración 990	Iteración 1022
Iteración 991	Iteración 1023
Iteración 992	Iteración 1024



GPU Implementation

Conceptually sets of 32 elements are established → warp

Iteración 1	Iteración 33
Iteración 2	Iteración 34
Iteración 3	Iteración 35
Iteración 4	Iteración 36
Iteración 5	Iteración 37
Iteración 6	Iteración 38
Iteración 7	Iteración 39
Iteración 8	Iteración 40
Iteración 9	Iteración 41
Iteración 10	Iteración 42
Iteración 11	Iteración 43
Iteración 12	Iteración 44
Iteración 13	Iteración 45
Iteración 14	Iteración 46
Iteración 15	Iteración 47
Iteración 16	Iteración 48
Iteración 17	Iteración 49
Iteración 18	Iteración 50
Iteración 19	Iteración 51
Iteración 20	Iteración 52
Iteración 21	Iteración 53
Iteración 22	Iteración 54
Iteración 23	Iteración 55
Iteración 24	Iteración 56
Iteración 25	Iteración 57
Iteración 26	Iteración 58
Iteración 27	Iteración 59
Iteración 28	Iteración 60
Iteración 29	Iteración 61
Iteración 30	Iteración 62
Iteración 31	Iteración 63
Iteración 32	Iteración 64

• • •

Iteración 961	Iteración 993
Iteración 962	Iteración 994
Iteración 963	Iteración 995
Iteración 964	Iteración 996
Iteración 965	Iteración 997
Iteración 966	Iteración 998
Iteración 967	Iteración 999
Iteración 968	Iteración 1000
Iteración 969	Iteración 1001
Iteración 970	Iteración 1002
Iteración 971	Iteración 1003
Iteración 972	Iteración 1004
Iteración 973	Iteración 1005
Iteración 974	Iteración 1006
Iteración 975	Iteración 1007
Iteración 976	Iteración 1008
Iteración 977	Iteración 1009
Iteración 978	Iteración 1010
Iteración 979	Iteración 1011
Iteración 980	Iteración 1012
Iteración 981	Iteración 1013
Iteración 982	Iteración 1014
Iteración 983	Iteración 1015
Iteración 984	Iteración 1016
Iteración 985	Iteración 1017
Iteración 986	Iteración 1018
Iteración 987	Iteración 1019
Iteración 988	Iteración 1020
Iteración 989	Iteración 1021
Iteración 990	Iteración 1022
Iteración 991	Iteración 1023
Iteración 992	Iteración 1024



GPU Implementation

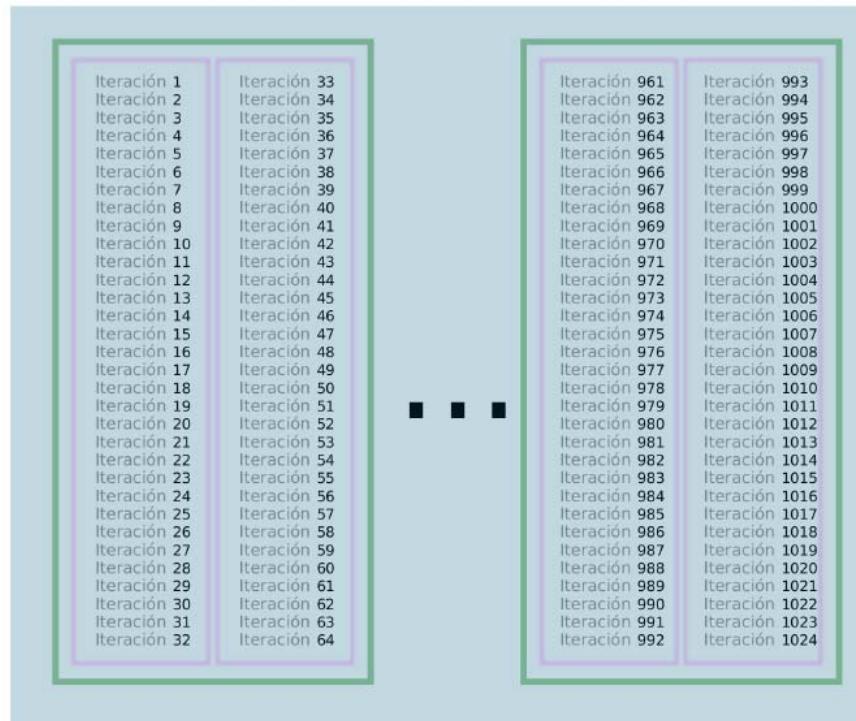
Warps are grouped in sets of 2 → **blocks**





GPU Implementation

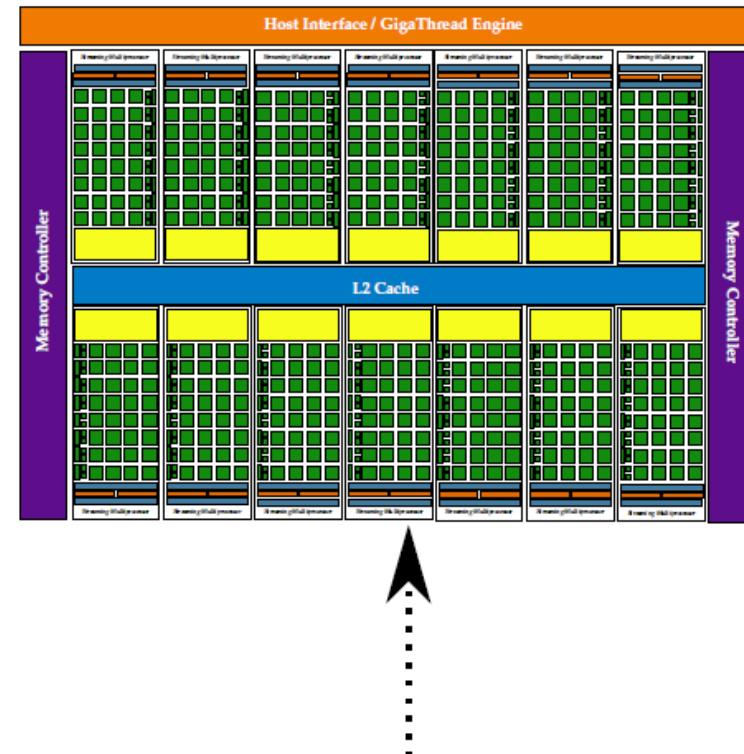
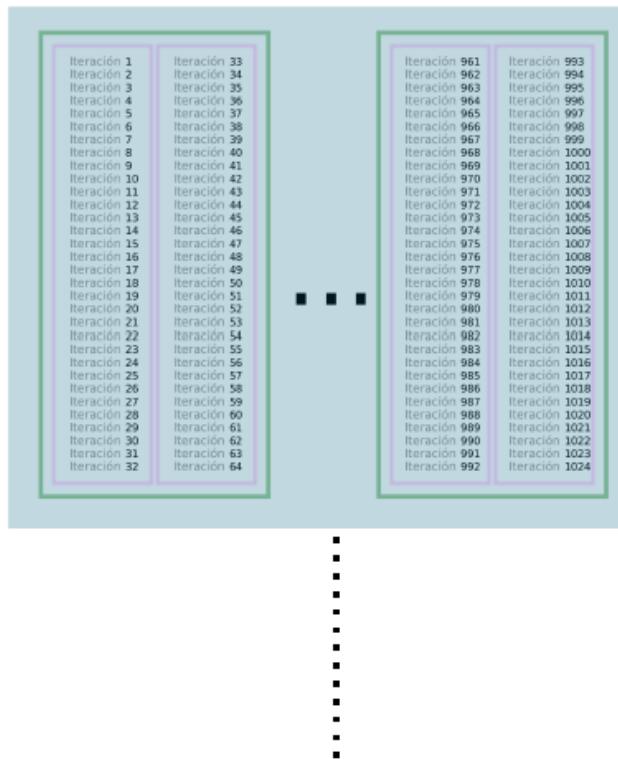
Conceptually All the blocks are regrouped → grid



GPU Implementation



The **grid** contains all elements to be processed in the GPU





GPU Implementation

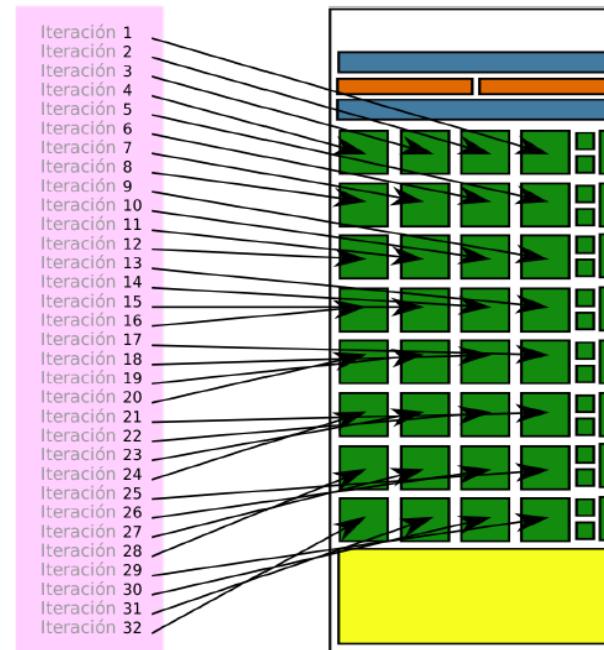
The **block** includes all the elements to be processed in a Streaming Multiprocessor (SM/SMx)





GPU Implementation

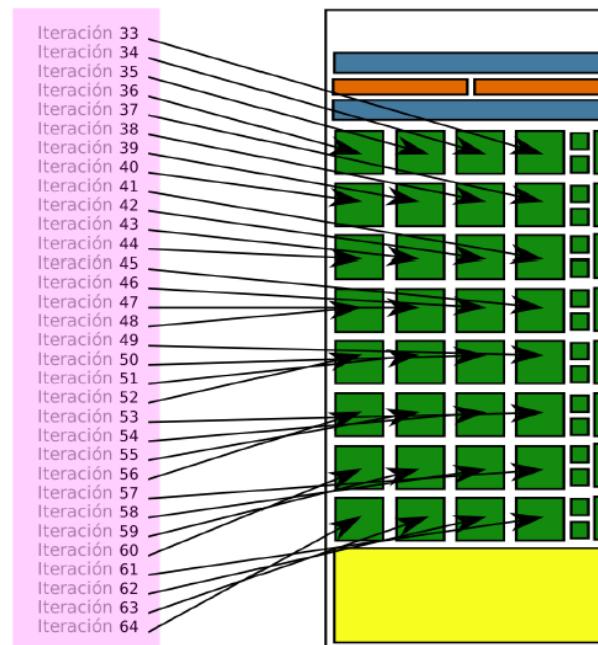
...Particularly, **block** is executed in sets of 32 elements (**warps**).
First a **warp**



GPU Implementation



...and after the former one ends, the second **warp** is executed



GPU Implementation



This is where **thread** appears

- Each **thread** can establish its unique identifier within the **block**



GPU Implementation

This is where **thread** appears

- Each **thread** can establish its unique identifier within the **block**
- Each **block** can establish its identifier within the **grid**



GPU Implementation

This is where **thread** appears

- Each **thread** can establish its unique identifier within the **block**
- Each **block** can establish its identifier within the **grid**

```
i = threadIdx.x + (blockIdx.x * blockDim.x)
```

Our model designed for GPUs



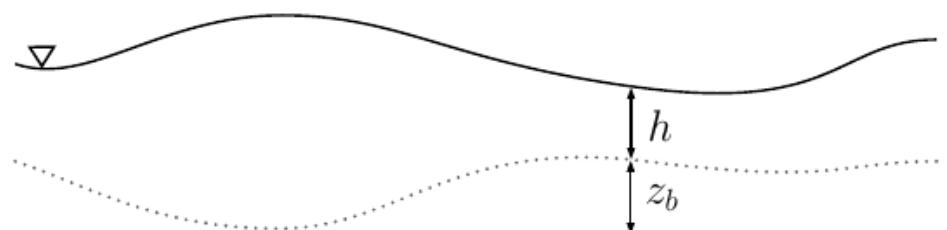
Some maths... (1/4)

We are interested in the simulation of the **Shallow Water Equations**. They represent the flow dynamics in a fluid (sometimes, but not necessarily, a free surface)

$$W_h : \frac{\partial h}{\partial t} + \frac{\partial(hu)}{\partial x} + \frac{\partial(hv)}{\partial y} = i_h$$

$$W_{q_x} : \frac{\partial(hu)}{\partial t} + \frac{\partial}{\partial x} \left(hu^2 + \frac{1}{2} gh^2 \right) + \frac{\partial(huv)}{\partial y} = -gh(S_{0x} - S_{fx})$$

$$W_{q_y} : \frac{\partial(hv)}{\partial t} + \frac{\partial(huv)}{\partial x} + \frac{\partial}{\partial y} \left(hv^2 + \frac{1}{2} gh^2 \right) = -gh(S_{0y} - S_{fy})$$

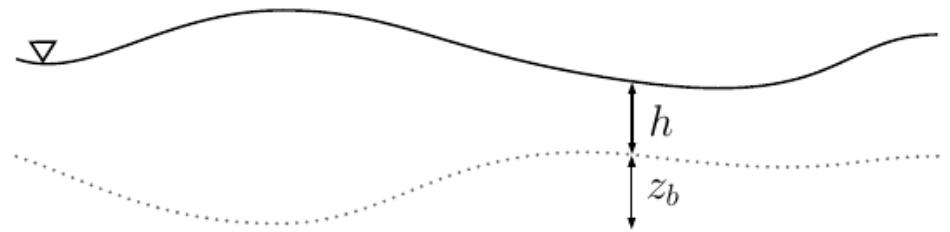




Some maths... (2/4)

We are interested in the simulation of the **Shallow Water Equations**. They represent the flow dynamics in a fluid (sometimes, but not necessarily, a free surface)

- It accounts for friction losses
- Bed-slope influence

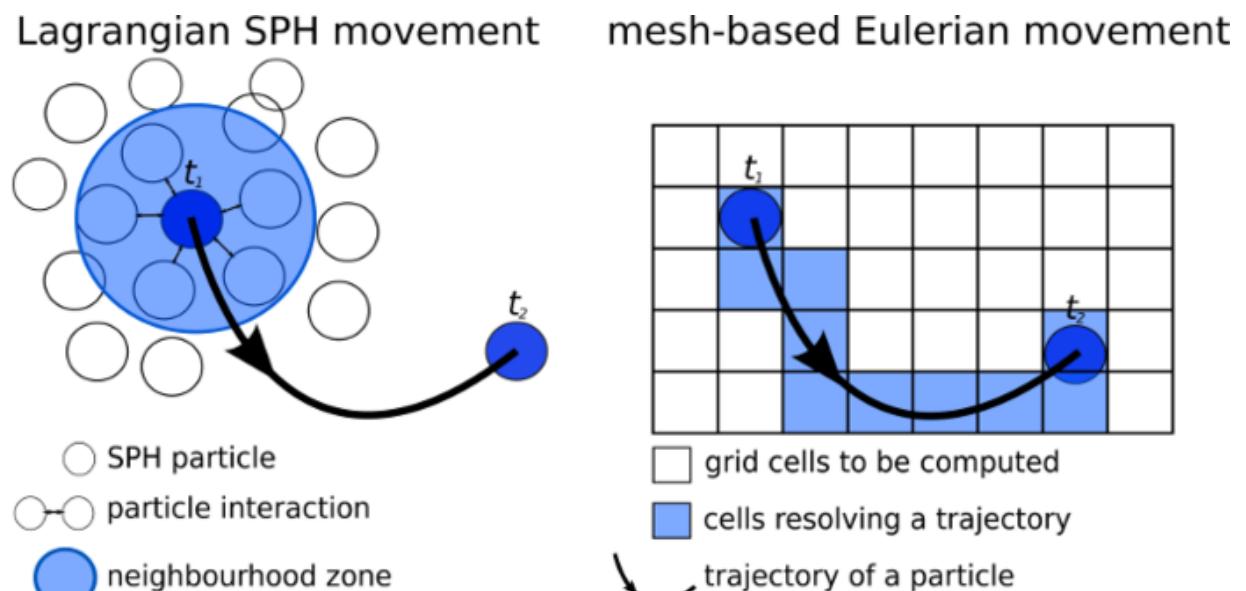


$$S_{0x} = -\frac{\partial z_b}{\partial x} \quad S_{0y} = -\frac{\partial z_b}{\partial y} \quad S_{fx} = \frac{n^2 u \sqrt{u^2 + v^2}}{h^{4/3}}, \quad S_{fy} = \frac{n^2 v \sqrt{u^2 + v^2}}{h^{4/3}}$$



Quality of the results

- There are many numerical factors that matter when solving these equations
 - Numerical solver Eulerian/Lagrangian

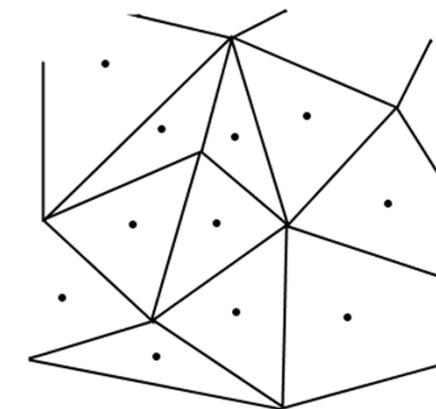
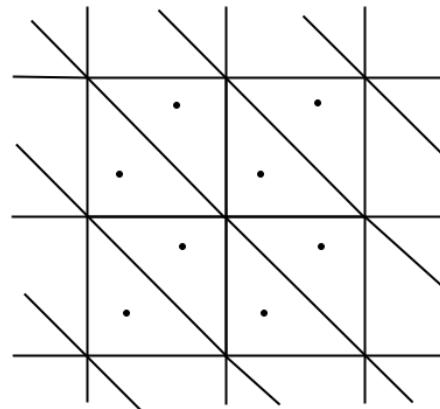
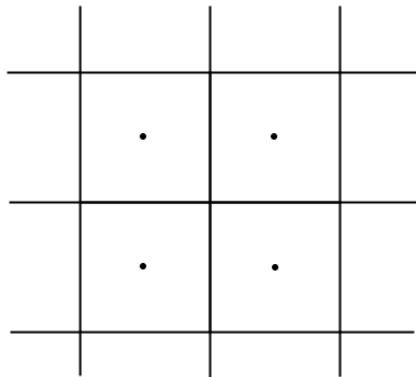


<https://www2.uibk.ac.at/>



Quality of the results

- There are many numerical factors that matter when solving these equations
 - Numerical solver Eulerian/Lagrangian
 - Type of mesh: Structured/Unstructured, Triangular/Quadrilateral





Quality of the results

- There are many numerical factors that matter when solving these equations
 - Numerical solver Eulerian/Lagrangian
 - Type of mesh: Structured/Unstructured, Triangular/Quadrilateral
 - Temporal integration Implicit/Explicit

$$f_i^{n+1} = f_j^{n+1} + \dots$$

$$f_i^{n+1} = f_j^n + \dots$$



Some maths... (3/4)

This mathematical model **cannot be analitically solved** and then, a numerical solver is required (FOU scheme)

They are written compactly as

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{E} = \mathbf{R}$$

$$\mathbf{F} = \left(q_x, \frac{q_x^2}{h} + \frac{1}{2} gh^2, \frac{q_x q_y}{h} \right)^T, \quad \mathbf{G} = \left(q_y, \frac{q_x q_y}{h}, \frac{q_y^2}{h} + \frac{1}{2} gh^2 \right)^T$$



Some maths... (4/4)

It is integrated over each computational cell, and using Gauss theorem

$$\frac{\partial}{\partial t} \int_{\Omega_i} \mathbf{U} d\Omega + \oint_{\Gamma} \mathbf{E} n d\Gamma = \int_{\Omega_i} \mathbf{R} d\Omega$$

After applying some algebra as well as Roe's linearization

$$\mathbf{U}_i^{n+1} A_i = \mathbf{U}_i^n A_i - \sum_{k=1}^{NE} \sum_{m=1}^3 (\tilde{\lambda}^{-\theta} \alpha \tilde{\mathbf{e}})^m k l_k \Delta t$$

$$\Delta t = CFL \min_k \left\{ \frac{\min(\chi_i, \chi_j)}{\max_m |\lambda_k^m|} \right\} CFL \leq 0.5$$

J. Murillo, P. Garc'ia-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. International Journal for Numerical methods in Fluids, 54(5):543-590, 2007.

J. Murillo and P. Garc'ia-Navarro. 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. J. Comput. Phys. 229, 11 (June 2010), 4327-4368.



Some maths... (4/4)

It is integrated over each computational cell, and using Gauss theorem

$$\frac{\partial}{\partial t} \int_{\Omega_i} \mathbf{U} d\Omega + \oint_{\Gamma} \mathbf{E} n d\Gamma = \int_{\Omega_i} \mathbf{R} d\Omega$$

After applying some algebra as well as Roe's linearization

$$\mathbf{U}_i^{n+1} A_i = \mathbf{U}_i^n A_i - \sum_{k=1}^{NE} \sum_{m=1}^3 (\tilde{\lambda}^{-\theta} \alpha \tilde{\mathbf{e}})^m k \Delta t$$

EDGE BASED SOLVER

$$\Delta t = CFL \min_k \left\{ \frac{\min(\chi_i, \chi_j)}{\max_m |\lambda_k^m|} \right\} \quad CFL \leq 0.5$$

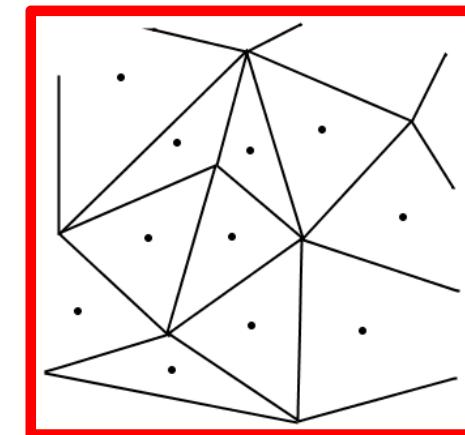
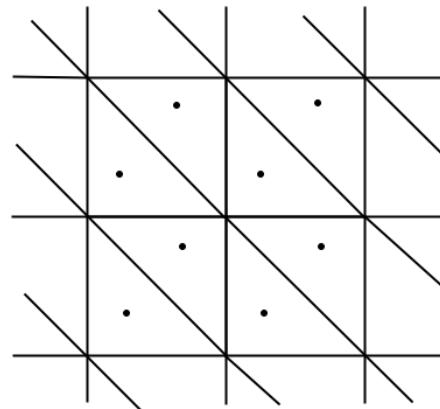
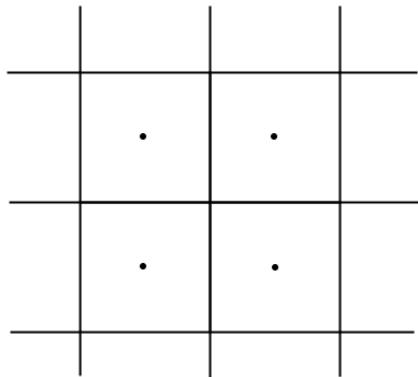
J. Murillo, P. Garc'ia-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. International Journal for Numerical methods in Fluids, 54(5):543-590, 2007.

J. Murillo and P. Garc'ia-Navarro. 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. J. Comput. Phys. 229, 11 (June 2010), 4327-4368.



Quality of the results

- There are many numerical factors that matter when solving these equations
 - Numerical solver **Eulerian**/Lagrangian
 - Temporal integration Implicit/**Explicit**
 - Type of mesh: Structured/**Unstructured, Triangular**/Quadrilateral





Quality of the results

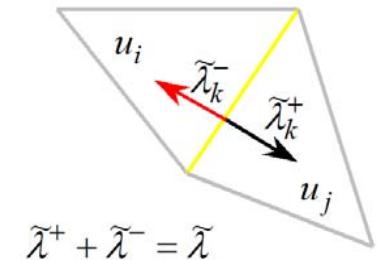
- There are many numerical factors that matter when solving these equations
 - Numerical solver
Eulerian/Lagrangian
 - Time integration
Implicit/**Explicit**
 - Type of mesh:
Structured/**Unstructured**,
Triangular/Quadrilateral
- Pros:
 - Information only depends on previously computed information
 - The numerical solver is conservative (no mass generation/loss)
- Cons
 - Slow convergence (time-step restriction for numerical stability)
 - Mesh dependency
 - Unstructured information



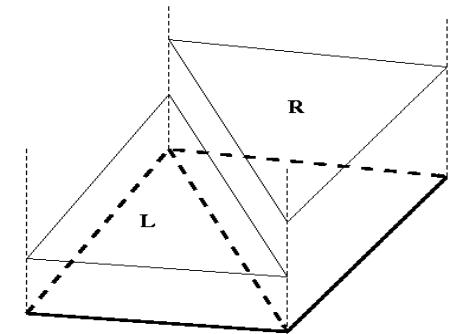
GPU Implementation

The operations required in our solver are...

$$\mathbf{U}_i^{n+1} A_i = \mathbf{U}_i^n A_i - \sum_{k=1}^{NE} \sum_{m=1}^3 (\tilde{\lambda}^- \theta \alpha \tilde{\mathbf{e}})^m \mathbf{l}_k \Delta t$$



$$\Delta t = CFL \min_k \left\{ \frac{\min(\chi_i, \chi_j)}{\max_m |\lambda_k^m|} \right\} CFL \leq 0.5$$



J. Murillo, P. Garc'ia-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. International Journal for Numerical methods in Fluids, 54(5):543-590, 2007.

J. Murillo and P. Garc'ia-Navarro. 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. J. Comput. Phys. 229, 11 (June 2010), 4327-4368.

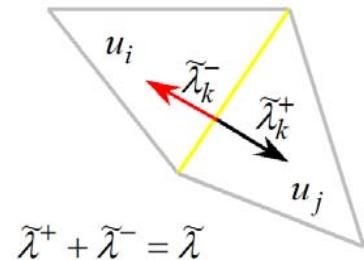
GPU Implementation



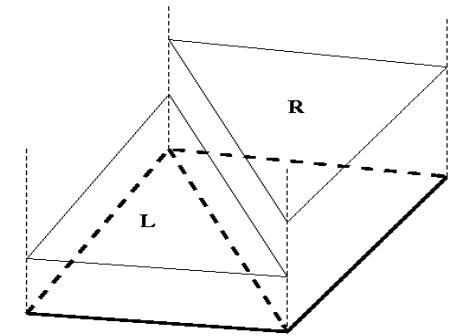
The operations required in our solver are ...

$$\mathbf{U}_i^{n+1} A_i = \mathbf{U}_i^n A_i - \sum_{k=1}^{NE} \sum_{m=1}^3 (\tilde{\lambda}^- \theta \alpha \tilde{\mathbf{e}})^m / k \Delta t$$

1



$$\Delta t = CFL \min_k \left\{ \frac{\min(\chi_i, \chi_j)}{\max_m |\lambda_k^m|} \right\} CFL \leq 0.5$$



J. Murillo, P. García-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. International Journal for Numerical methods in Fluids, 54(5):543-590, 2007.

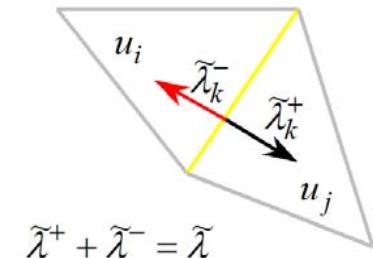
J. Murillo and P. García-Navarro. 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. J. Comput. Phys. 229, 11 (June 2010), 4327-4368.

GPU Implementation

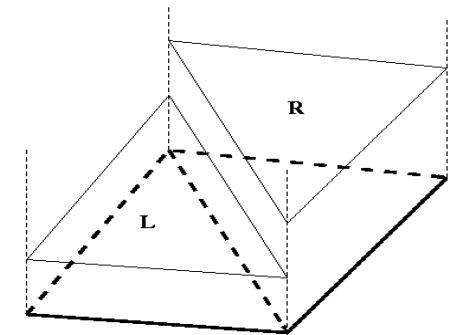


The operations required in our solver are ...

$$\mathbf{U}_i^{n+1} A_i = \mathbf{U}_i^n A_i - \sum_{k=1}^{NE} \sum_{m=1}^3 (\tilde{\lambda}^- \theta \alpha \tilde{\mathbf{e}})^m k I_k \Delta t$$



$$\Delta t = CFL \min_k \left\{ \frac{\min(\chi_i, \chi_j)}{\max_m |\lambda_k^m|} \right\} CFL \leq 0.5$$



J. Murillo, P. García-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. International Journal for Numerical methods in Fluids, 54(5):543-590, 2007.

J. Murillo and P. García-Navarro. 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. J. Comput. Phys. 229, 11 (June 2010), 4327-4368.

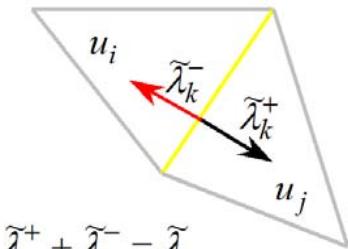
GPU Implementation



The operations required in our solver are ...

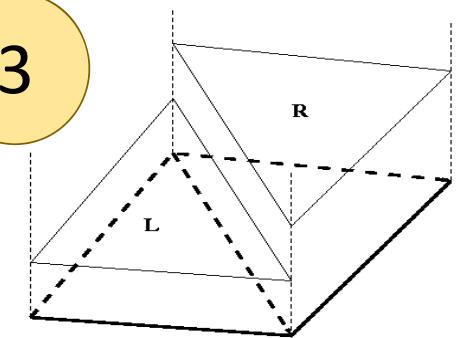
$$\mathbf{U}_i^{n+1} A_i = \mathbf{U}_i^n A_i - \sum_{k=1}^{NE} \sum_{m=1}^3 (\tilde{\lambda}^- \theta \alpha \tilde{\mathbf{e}})^m \mathbf{l}_k \Delta t$$

1
2



$$\Delta t = CFL \min_k \left\{ \frac{\min(\chi_i, \chi_j)}{\max_m |\lambda_k^m|} \right\} CFL \leq 0.5$$

3



J. Murillo, P. Garc'ia-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. International Journal for Numerical methods in Fluids, 54(5):543-590, 2007.

J. Murillo and P. Garc'ia-Navarro. 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. J. Comput. Phys. 229, 11 (June 2010), 4327-4368.

GPU Implementation



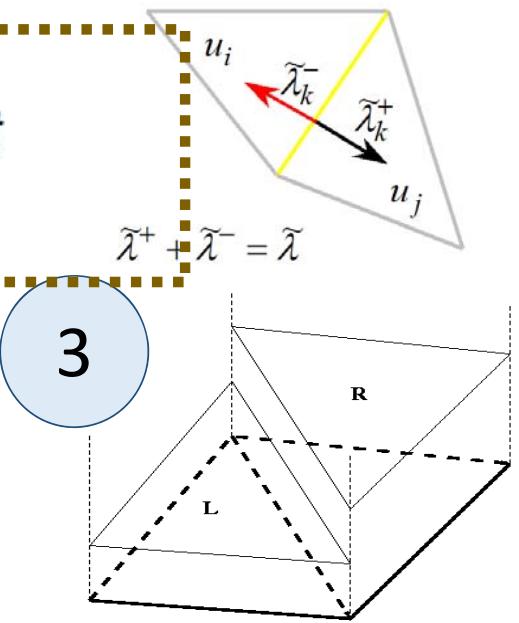
The operations required in our solver are ...

4

$$\mathbf{U}_i^{n+1} A_i = \mathbf{U}_i^n A_i - \sum_{k=1}^{NE} \sum_{m=1}^3 (\tilde{\lambda}^- \theta \alpha \tilde{\mathbf{e}})^m \mathbf{l}_k l_k \Delta t$$

2

$$\Delta t = CFL \min_k \left\{ \frac{\min(\chi_i, \chi_j)}{\max_m |\lambda_k^m|} \right\} CFL \leq 0.5$$

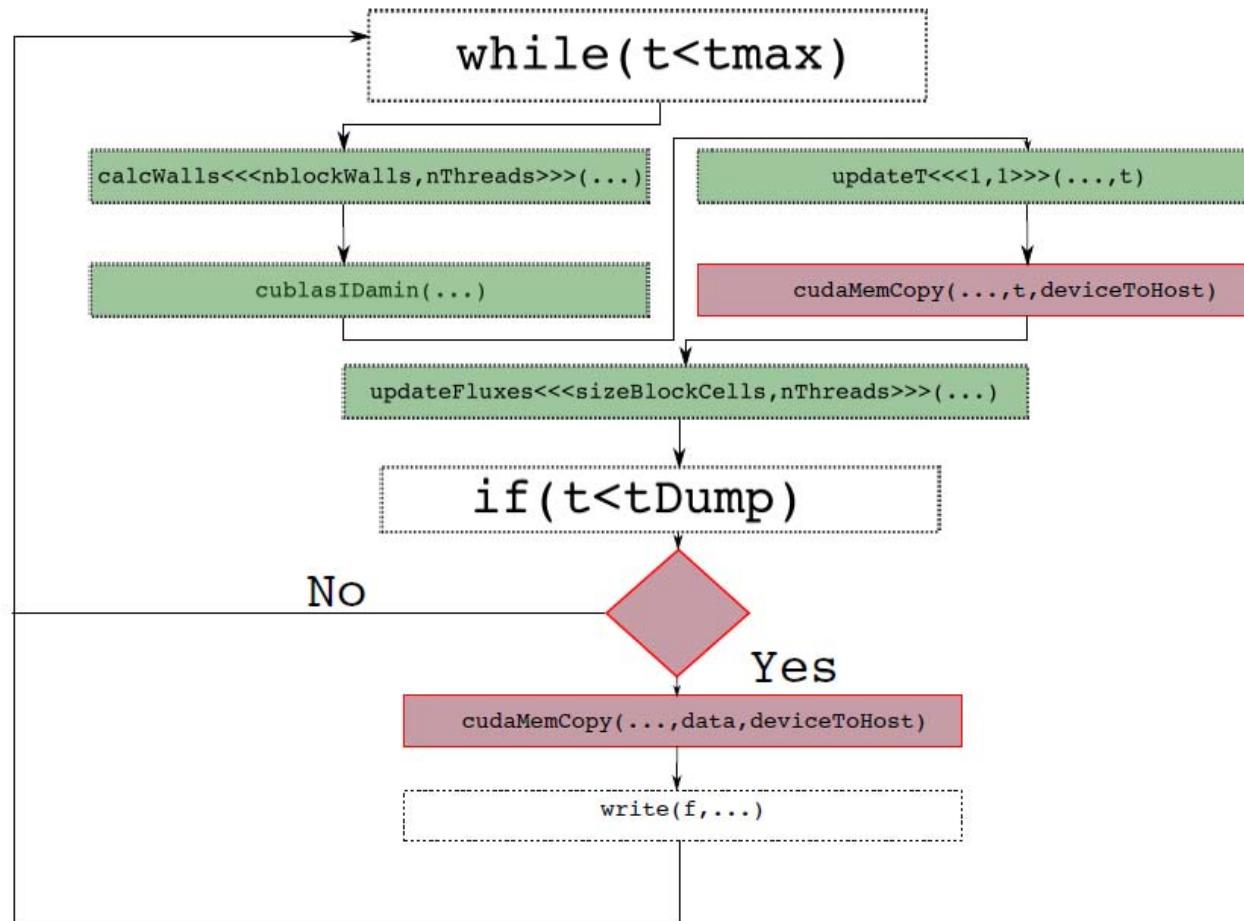


J. Murillo, P. García-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. International Journal for Numerical Methods in Fluids, 54(5):543-590, 2007.

J. Murillo and P. García-Navarro. 2010. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. J. Comput. Phys. 229, 11 (June 2010), 4327-4368.



GPU Implementation





GPU Implementation

```
2 // Configuration of the parameters
3 threads=512;
4 wallBlocks=nWall/threads;
5 cellBlocks=nCell/threads;
6 while(t<tmax){
7     // Calculate the fluxes
8     calculateWallFluxes<<<wallBlocks,threads,0,executionStream>>>(...);
9     // Stablish the minimum dt obtaining the ID of the
10    // minimum dt
11    // (*) Explained at section 4.3
12    cublasIdamin(...,nWall,vDt,1,id);
13    // And assign it
14    newDt<<<1,1,0,executionStream>>>(dt,vDt,id);
15    // Update the elapsed time (in GPU)
16    updateT<<<1,1,0,executionStream>>>(cuda_t,dt);
17    // Retrieves the value of t to CPU
18    cudaMemcpy(t,cuda_t,sizeof(double),cudaMemcpyDeviceToHost);
```



GPU Implementation

```
19 // Update the cell values
20 assignFluxes<<<cellBlocks,threads,0,executionStream>>>(...);
21 // Verify if it is neccessary to dump data and
22 // if it is neccessary, process it.
23 // (*) Detailed in section 4.4
24 if(t<t_dump){
25     // Copy of cell variables to a GPU
26     // stored buffer
27     cudaMemcpy(..., cudaMemcpyDeviceToDevice);
28     // Stablishing the barrier to ensure the copy of the
29     // data to the buffer
30     cudaStreamSynchronize(copyStream);
31     // Copy the data to the CPU buffer
32     cudaMemcpyAsync(..., cudaMemcpyDeviceToHost,copyStream);
33     // Create another stream in order to be which controls
34     // the disk-transfer
35     pthread_create( &diskThread, ...);
36 }
```



GPU Implementation

Listing 3.1: Simple 1D transport equation in C

```
1 void upwindStepGPU(double *fn,double *fnmas1){  
2     int i;  
3     for (i=1; i<1/DELTA_X; i++) {  
4         fnmas1[i]=fn[i]+c*DELTA_T*(fn[i-1]-fn[i])/(DELTA_X);  
5     }  
6 }
```

Listing 3.2: Simple 1D transport equation in CUDA

```
1 __global__ void upwindStepGPU(double *fn,double *fnmas1)  
2 {  
3     // Point to the data  
4     unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;  
5     if(i<MAX){  
6         fnmas1[i]=fn[i]+c*DELTA_T*(fn[i-1]-fn[i])/(DELTA_X);  
7     }  
8 }
```



GPU Implementation

Reduction of each operation comparing Shared-memory parallelization against Tesla c2075 GPU Implementation





GPU Implementation

One additional detail: **Single or double precision?**

$$h_j^{***} = h_j^n - \alpha_k^1 + \left(\frac{\beta}{\tilde{\lambda}} \right)_k^1 \geq 0$$

Contributions of this (**very important**) term

α	-7.0000000000000018e-03
β	-1.8340340645691451e-03
λ	2.6200486636702064e-01
$\epsilon \propto -\alpha_k^1 + (\beta/\tilde{\lambda})_k^1$	-8.6736173798840354e-19
h_i^{n+1}	9.7990000000000005e-06

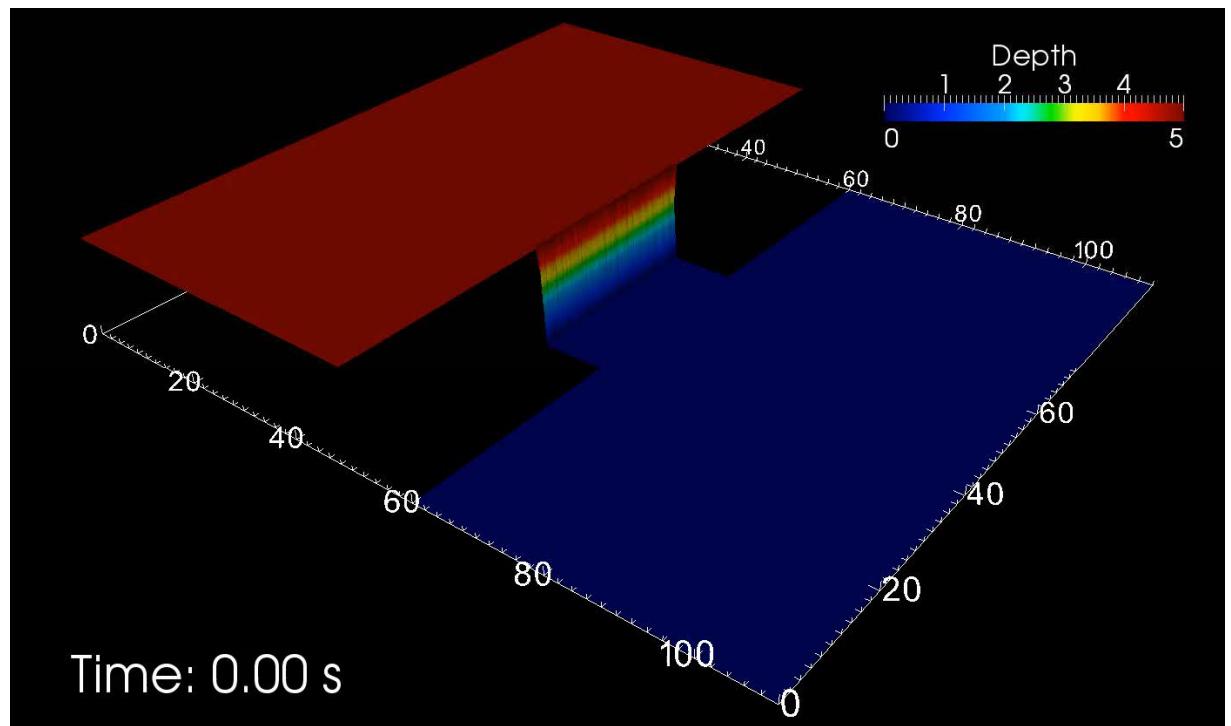
GPU Implementation



In 2012, Lacasta et al.
performs the simulation of a
5-0 dambreak problem.

Tesla c2075 vs Trombon (7 x
Intel i7 CPU 860@ 2.80 GHz
10 GFLOPS/Core)

	28-Core [◊]	1-Core	GPU
Cells		106648	
CFL		0.9	
t_n		400.0	
h_0		5 - 0	
Comp. Load (s.)	363.2	9383.83	250.79
Watts	1470	210	215
S_{up}	25.84	37.41	



GPU Implementation



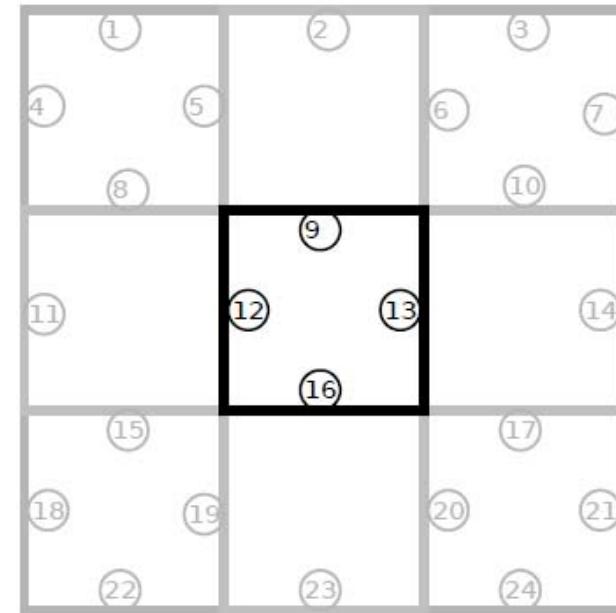
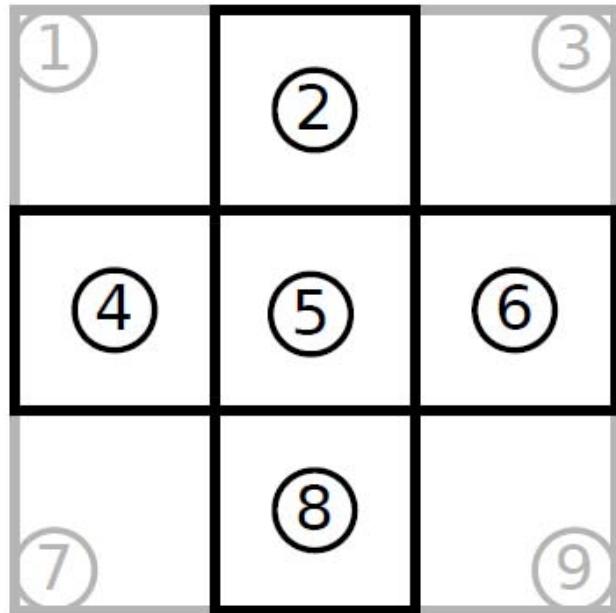
SIMT sounds really promising **but...**

- warps ‘do not like’ to perform different instructions (A.k.A. Thread Divergence).
 - *In other words, control flow patterns (if... else...) may reduce the GPU performance!!*
- Global memory is read in chunks of 128 bytes.
 - *What if memory accesses are not consecutive?*



GPU Implementation

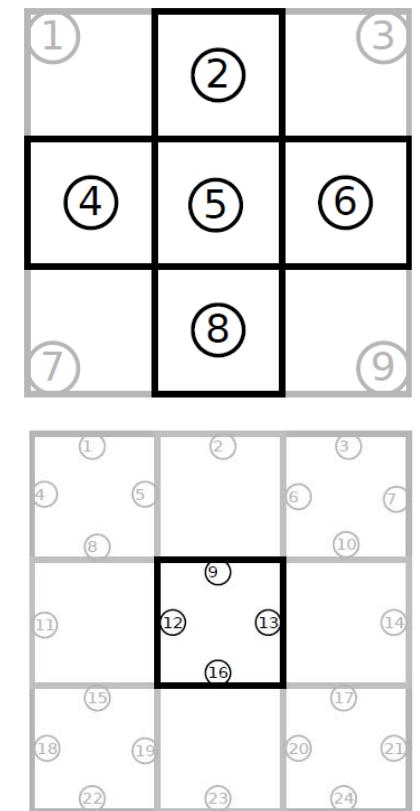
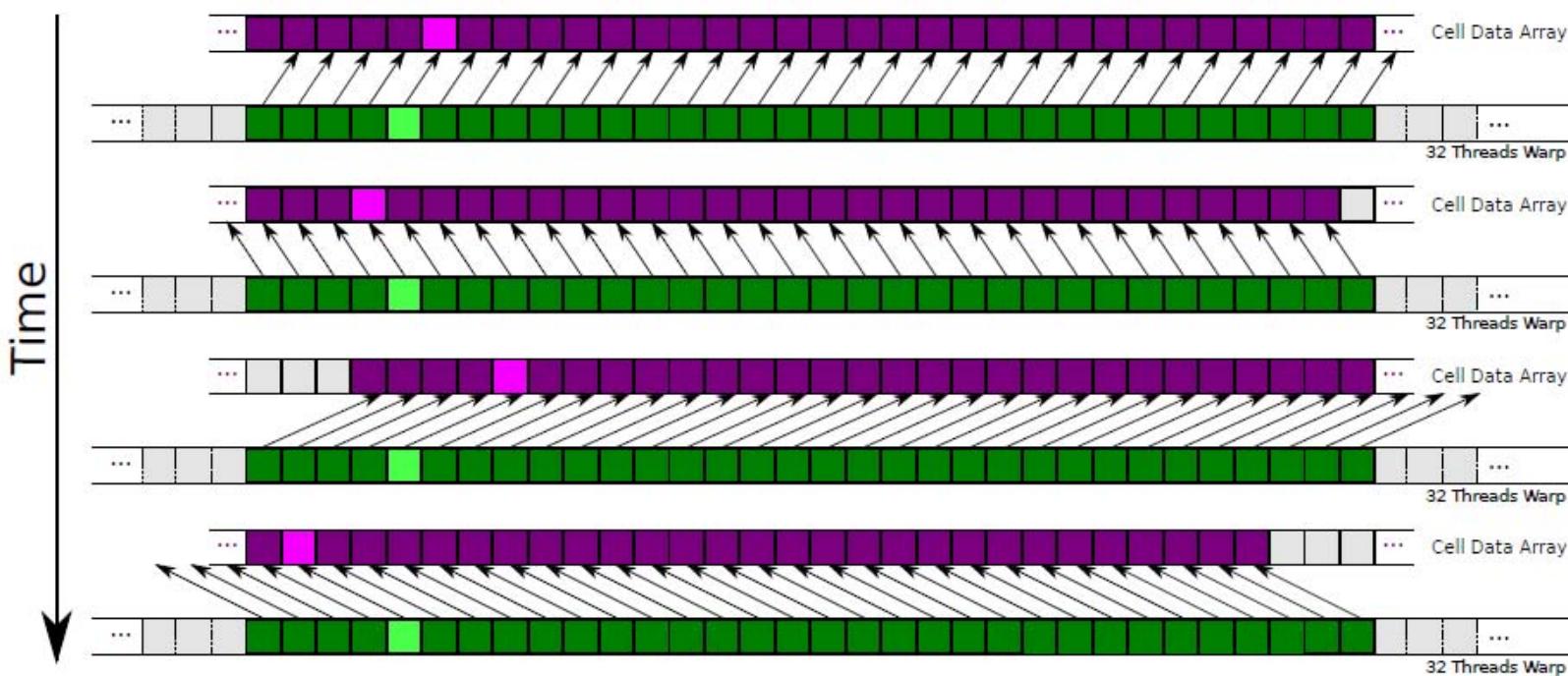
- GPUs were originally designed for dealing with “pixels”



GPU Implementation



- Which generates misaligned but coalesced Access pattern to compute the flush variation following NSEW scheme



GPU Implementation

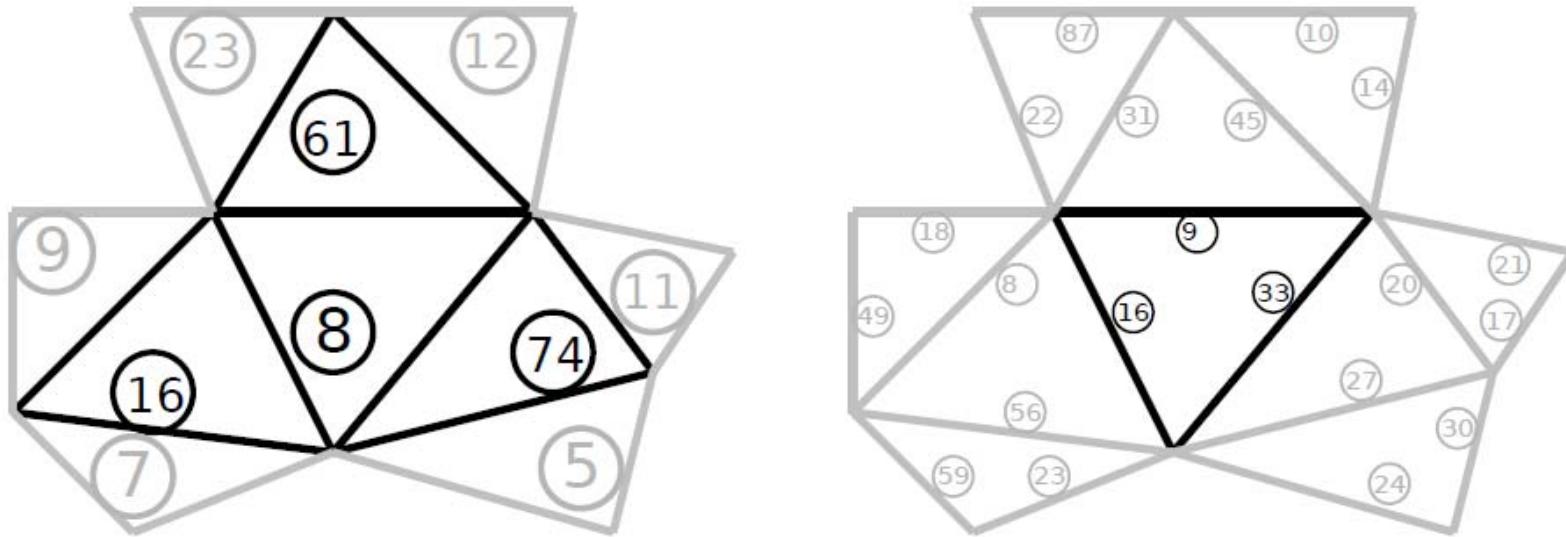


- There are many papers about computing CFD solvers on GPUs. However, most of them deal with structured info
 - Thibault, J. C., & Senocak, I. (2009, January). CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows. In *Proceedings of the 47th AIAA aerospace sciences meeting* (pp. 2009-758).
 - Sætra, M. L., Brodtkorb, A. R., & Lie, K. A. (2015). Efficient GPU-implementation of adaptive mesh refinement for the shallow-water equations. *Journal of Scientific Computing*, 63(1), 23-48.
 - Liang, Q., & Smith, L. S. (2015). A high-performance integrated hydrodynamic modelling system for urban flood simulations. *Journal of Hydroinformatics*, 17(4), 518-533.
 - Zhai, J., Yuan, L., Liu, W., & Zhang, X. (2015). Solving the Savage–Hutter equations for granular avalanche flows with a second-order Godunov type method on GPU. *International Journal for Numerical Methods in Fluids*, 77(7), 381-399.
 - Smith, L. S., Liang, Q., & Quinn, P. F. (2015). Towards a hydrodynamic modelling framework appropriate for applications in urban flood assessment and mitigation using heterogeneous computing. *Urban Water Journal*, 12(1), 67-78.
 - Vacondio, R., Aureli, F., Ferrari, A., Mignosa, P., & Dal Palù, A. (2016). Simulation of the January 2014 flood on the Secchia River using a fast and high-resolution 2D parallel shallow-water numerical scheme. *Natural Hazards*, 80(1), 103-125.
 - ...



GPU Implementation

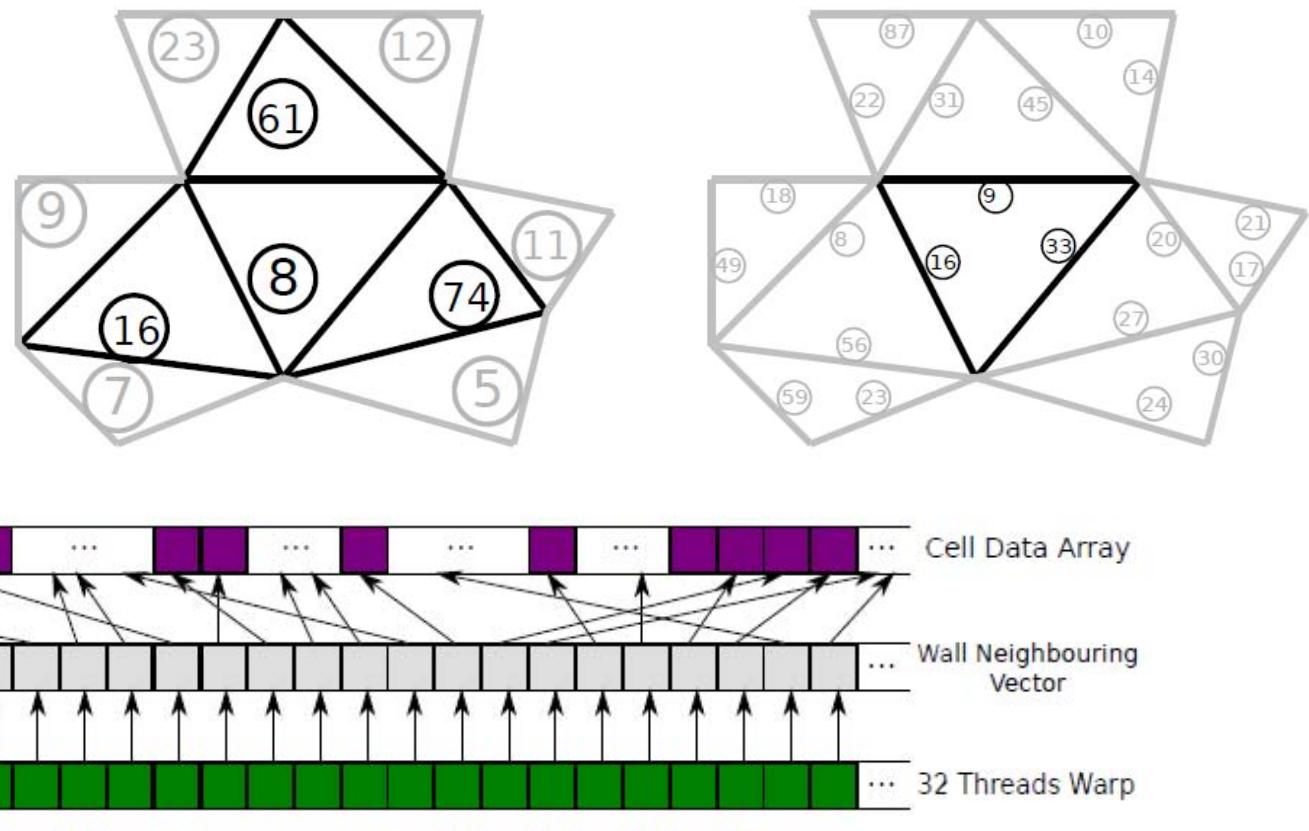
- In order to make feasible the implementation on GPUs of unstructured grids solvers, some aspects have to be adopted



GPU Implementation



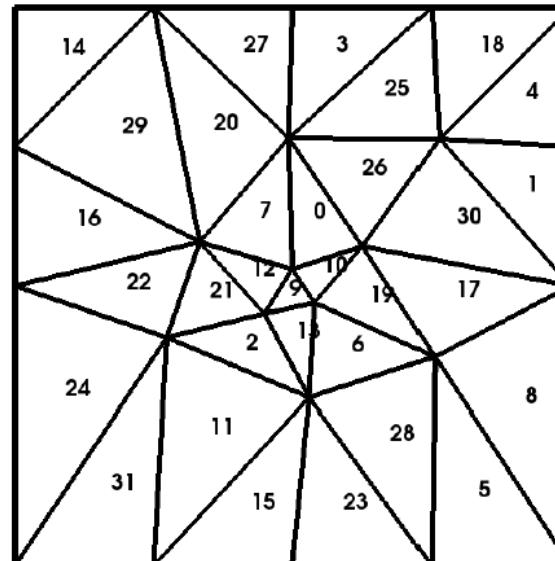
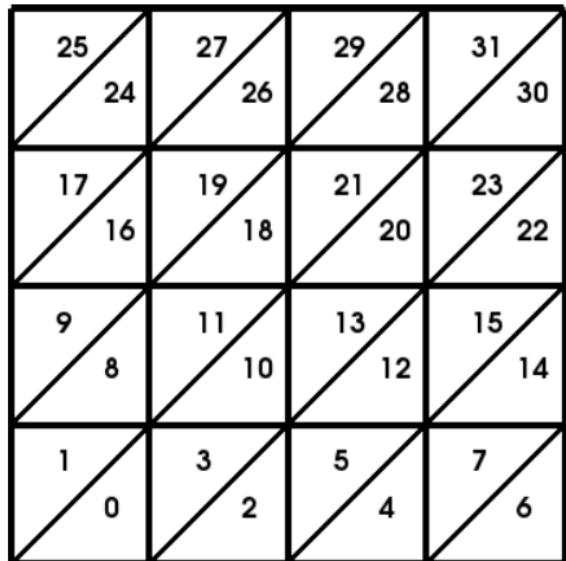
- Uncoalesced access pattern to get W data (Stored by cell). Processing wall 9 is light coloured when it access to cell 8 ($i=9$, $c1=8$)





GPU Implementation

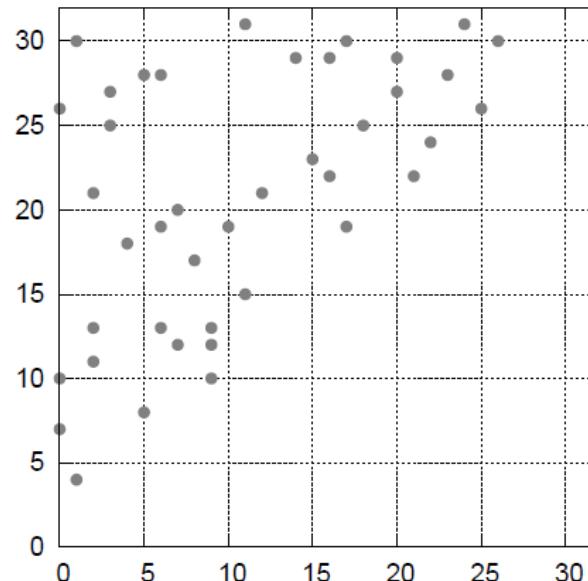
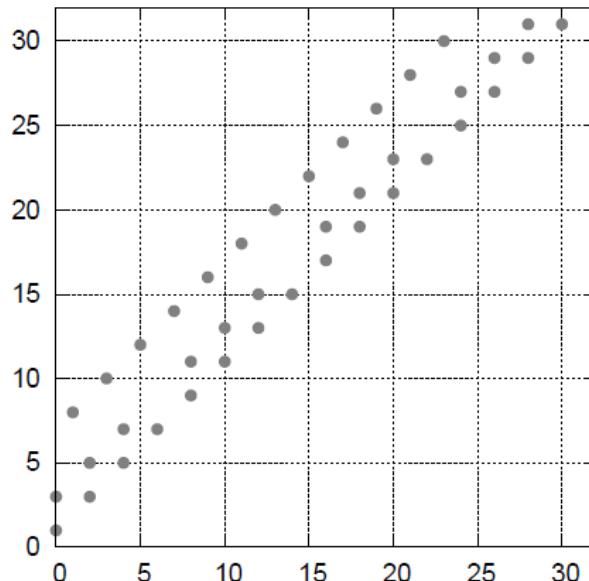
- Some improvements can be done on the mesh to improve the coalescence





GPU Implementation

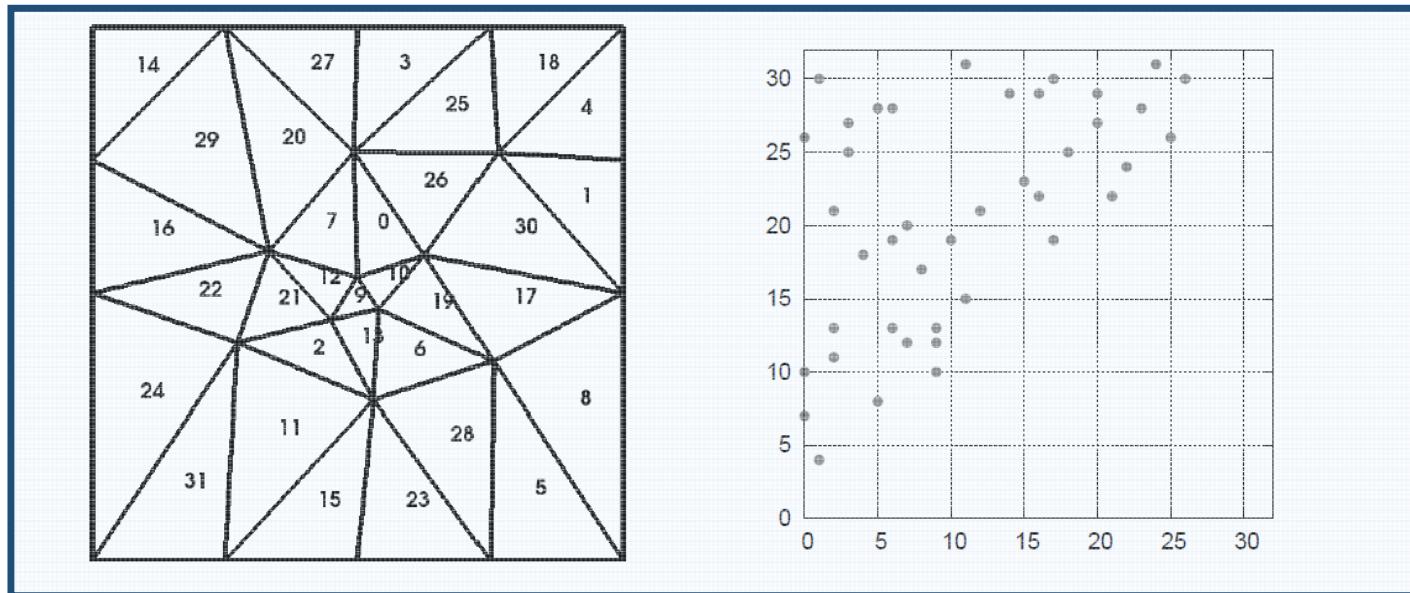
- Connectivity matrix of the unstructured mesh shows disordered patterns





GPU Implementation

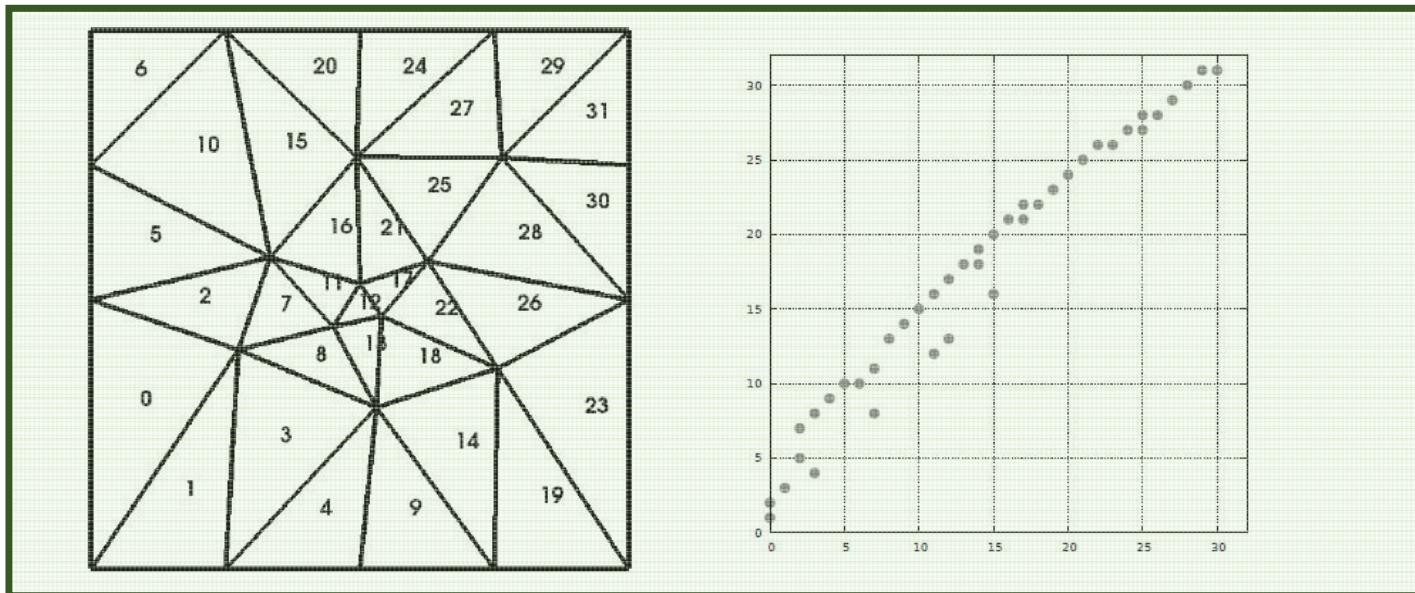
- Reverse Cuthill-McKee (RCM) algorithm improves the bandwidth of a matrix, so it can be applied over the connectivity matrix in order to generate an optimal ordering





GPU Implementation

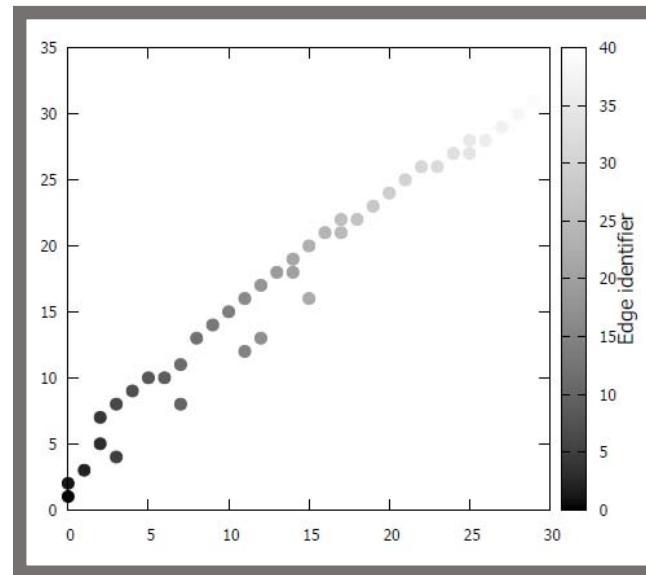
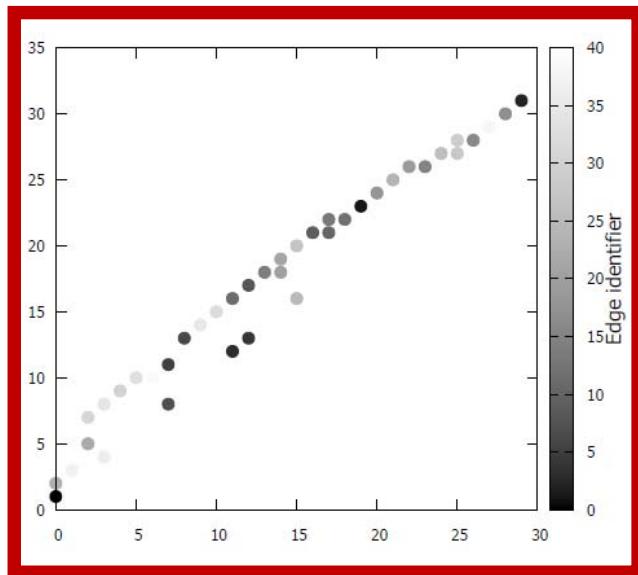
- Reverse Cuthill-McKee (RCM) algorithm improves the bandwidth of a matrix, so it can be applied over the connectivity matrix in order to generate an optimal ordering





GPU Implementation

- RCM improves the cells accessing. However, it is an edge-based solver and some operations require that consecutive edges contain similar cell IDx

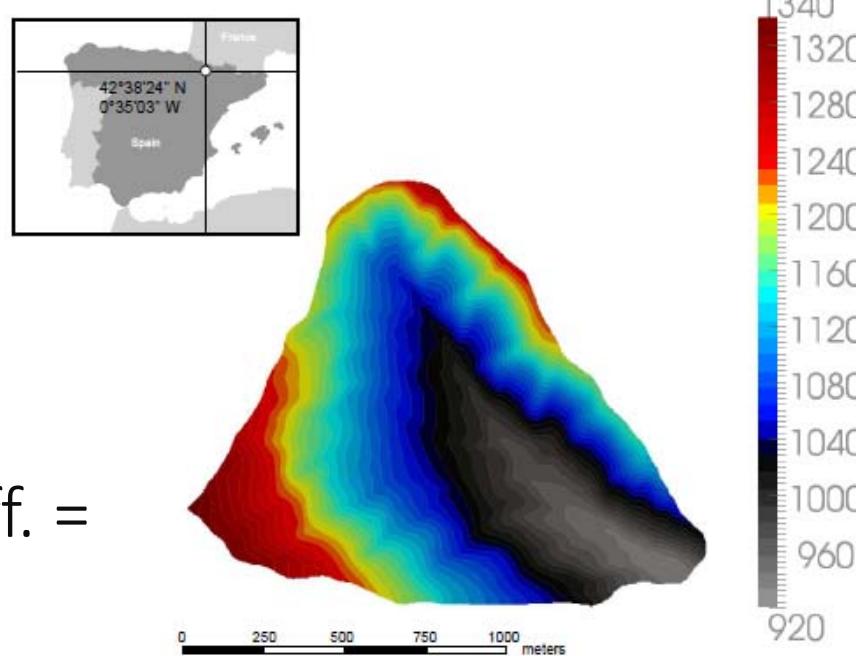




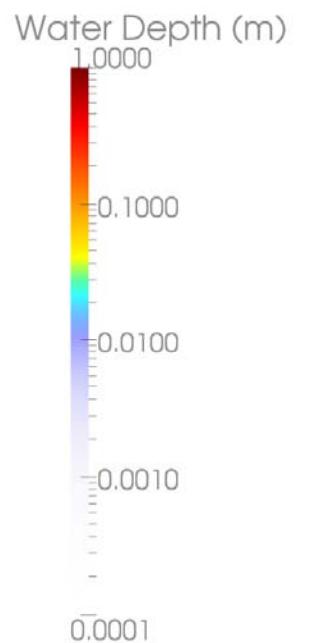
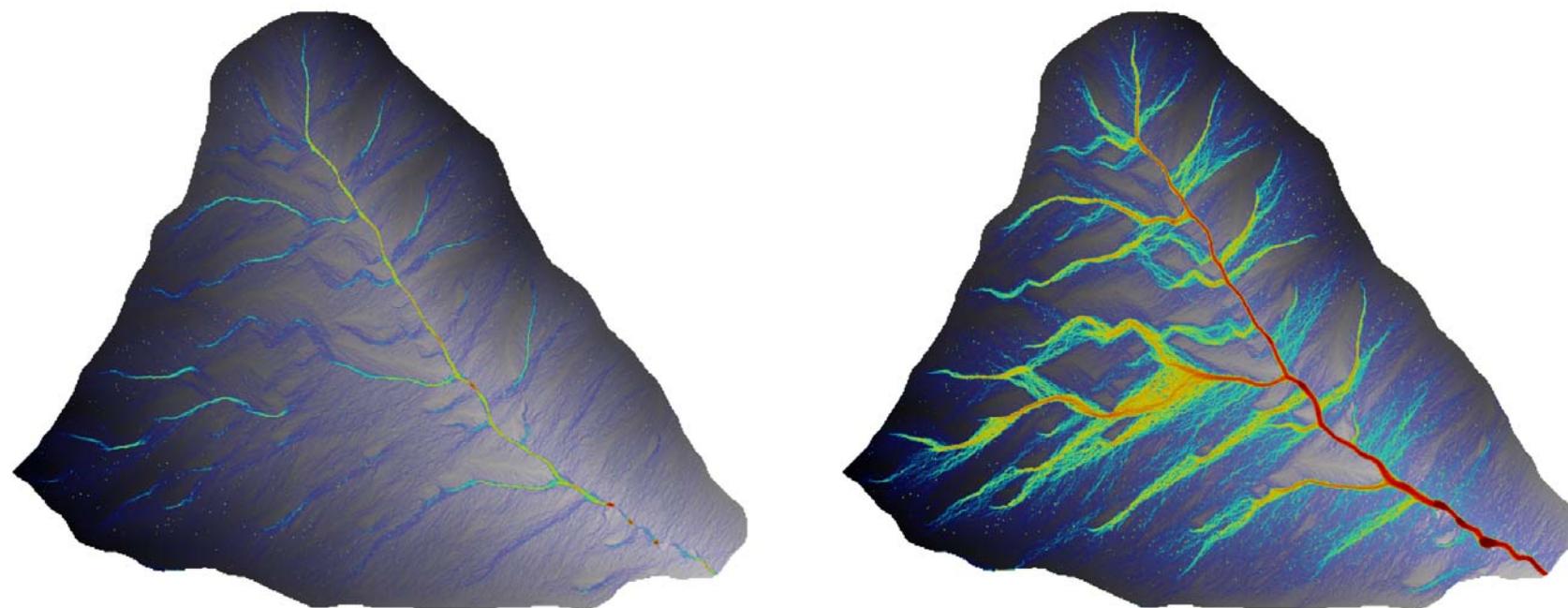
Effect of mesh ordering

Arnás rainfall simulation

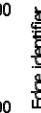
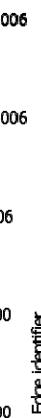
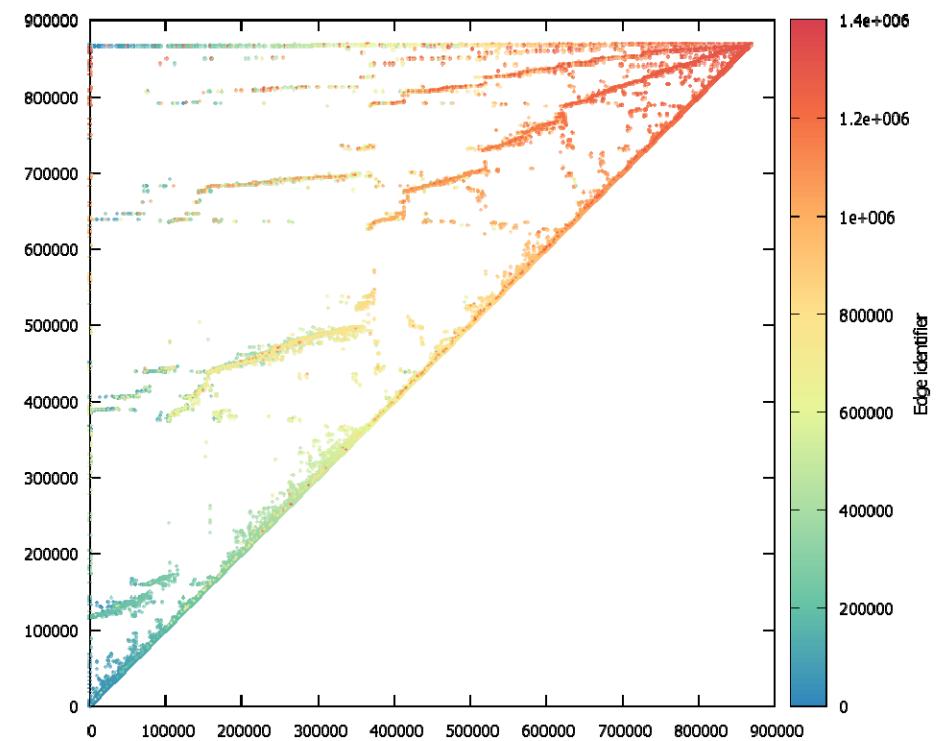
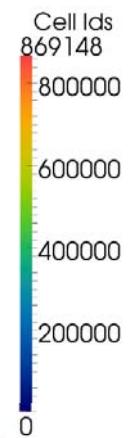
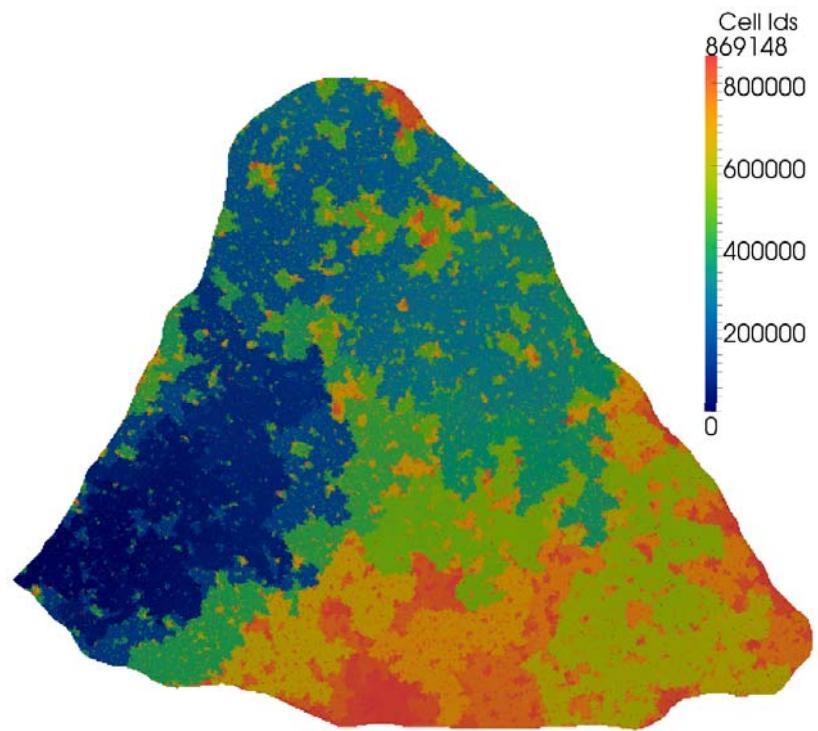
- 8 hours rainfall event
- >800000 cells
- 0.9 CFL condition
- Free outflow OBC
- n Manning's Roughnes coeff. = 0.05792662



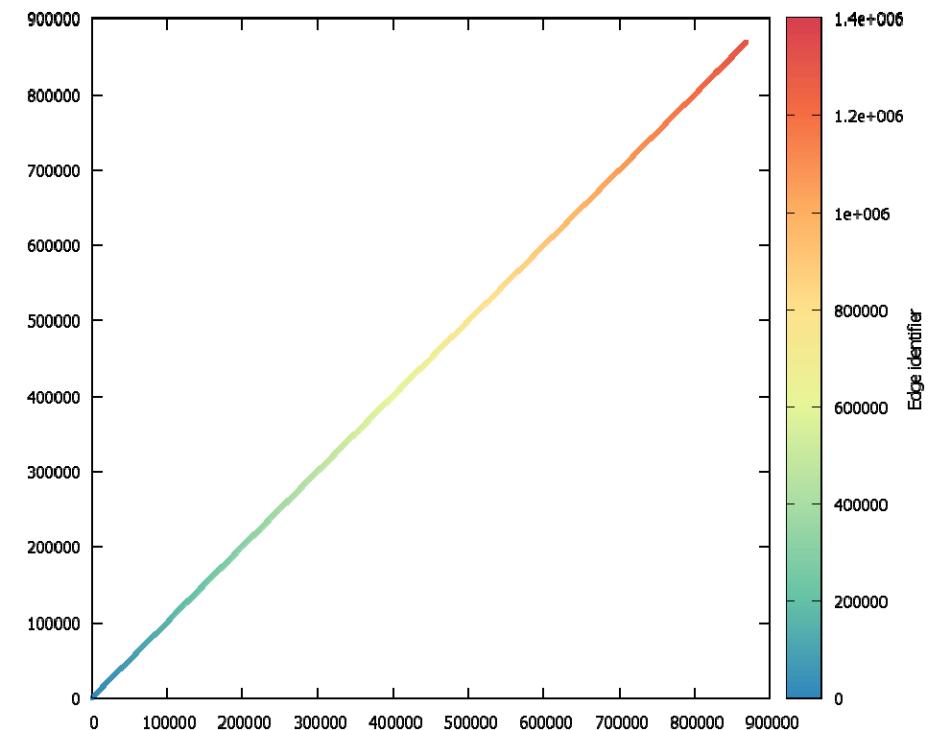
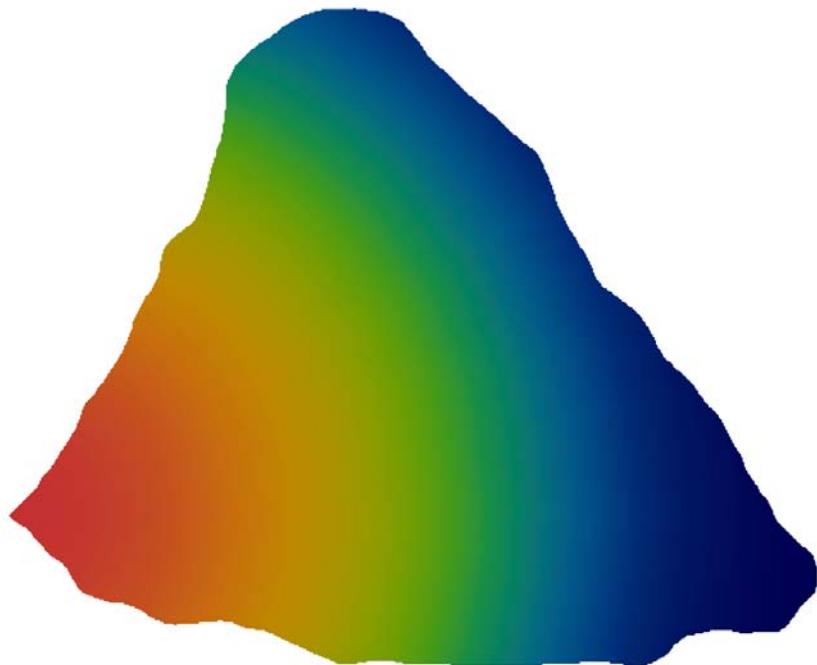
Two snapshots



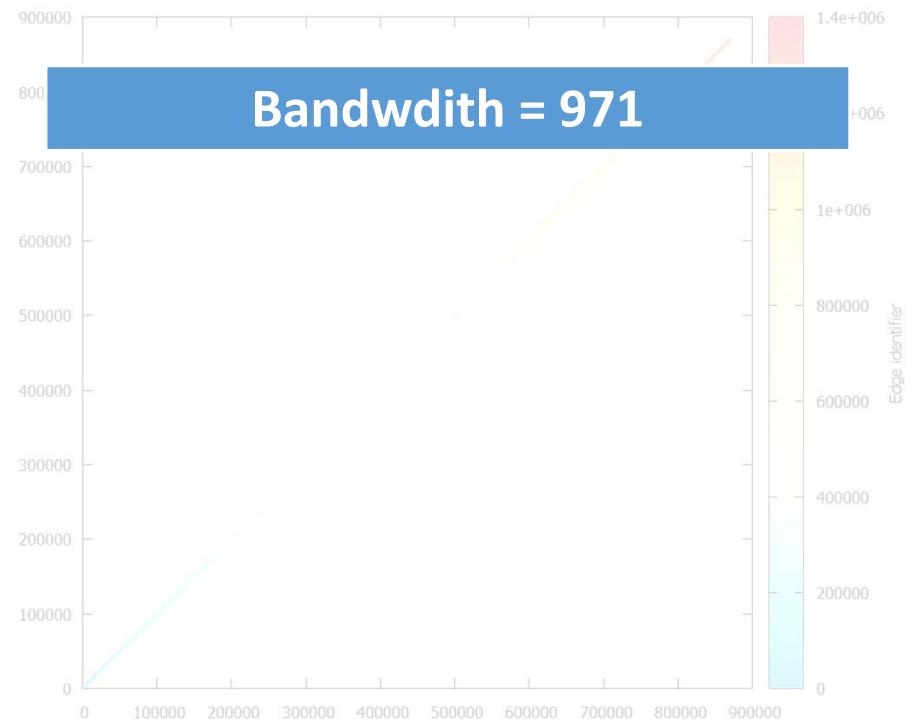
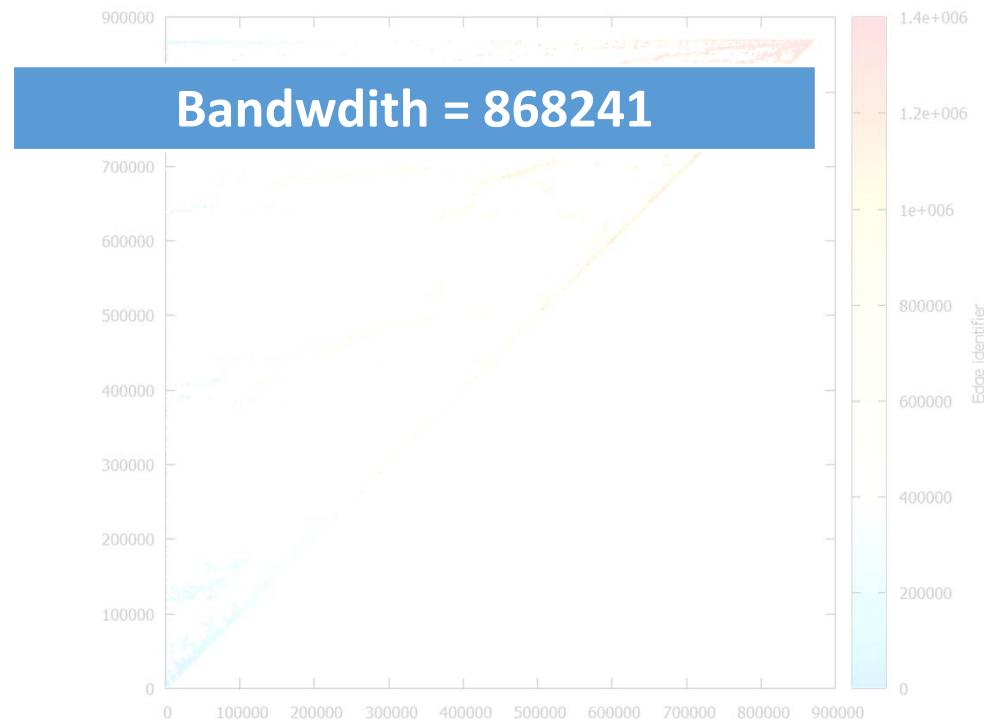
Disordered connectivity matrix



Ordered connectivity matrix

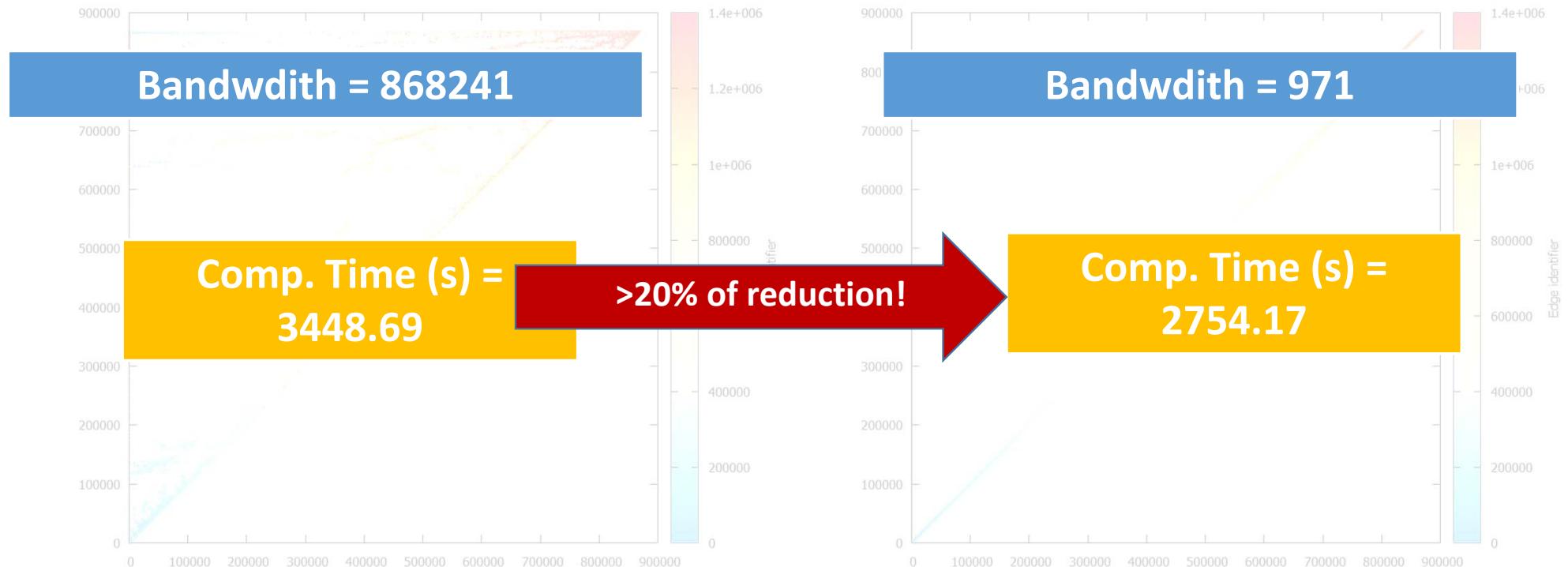


Effect of mesh ordering





Effect of mesh ordering



Test cases



Large-scale modeling

4 applications that require HPC (or at least, some acceleration...)

- Ebro river 2013 flooding
- Tous Dam-breach
- Experiments of Landslides & their simulation
- Chemical species transport



Large-scale modeling

Ebro river flooding

March 2015 floodings

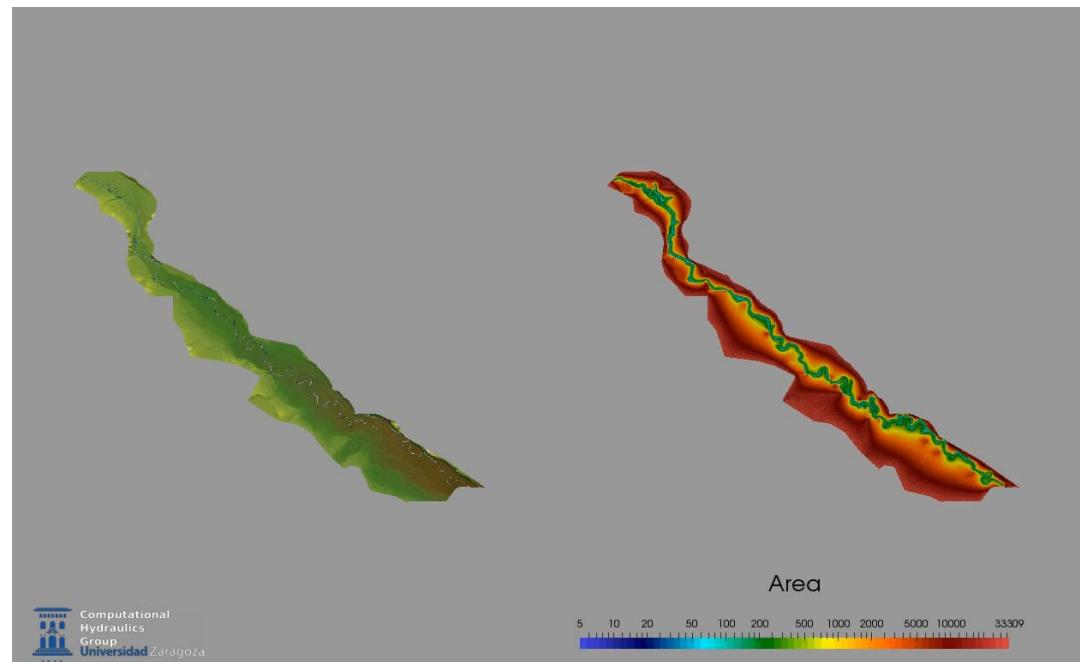
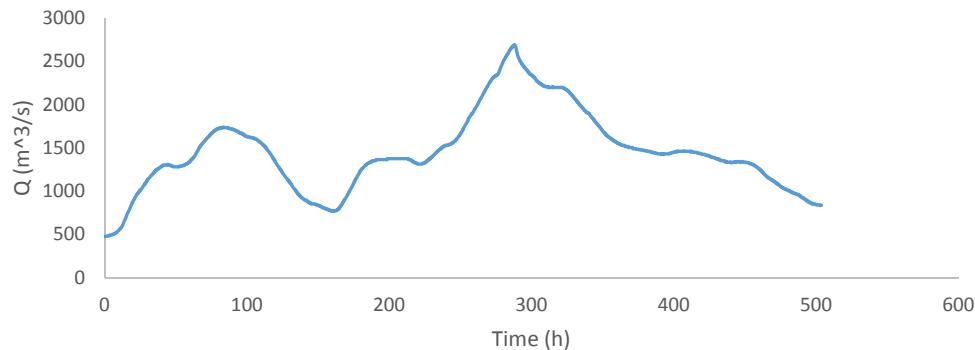
25 M€ economic losses

Castejon (Navarra)-Zaragoza (Aragón)

87 kms (125 kms river reach)

Non-Uniform mesh 867000 cells

21 Days event

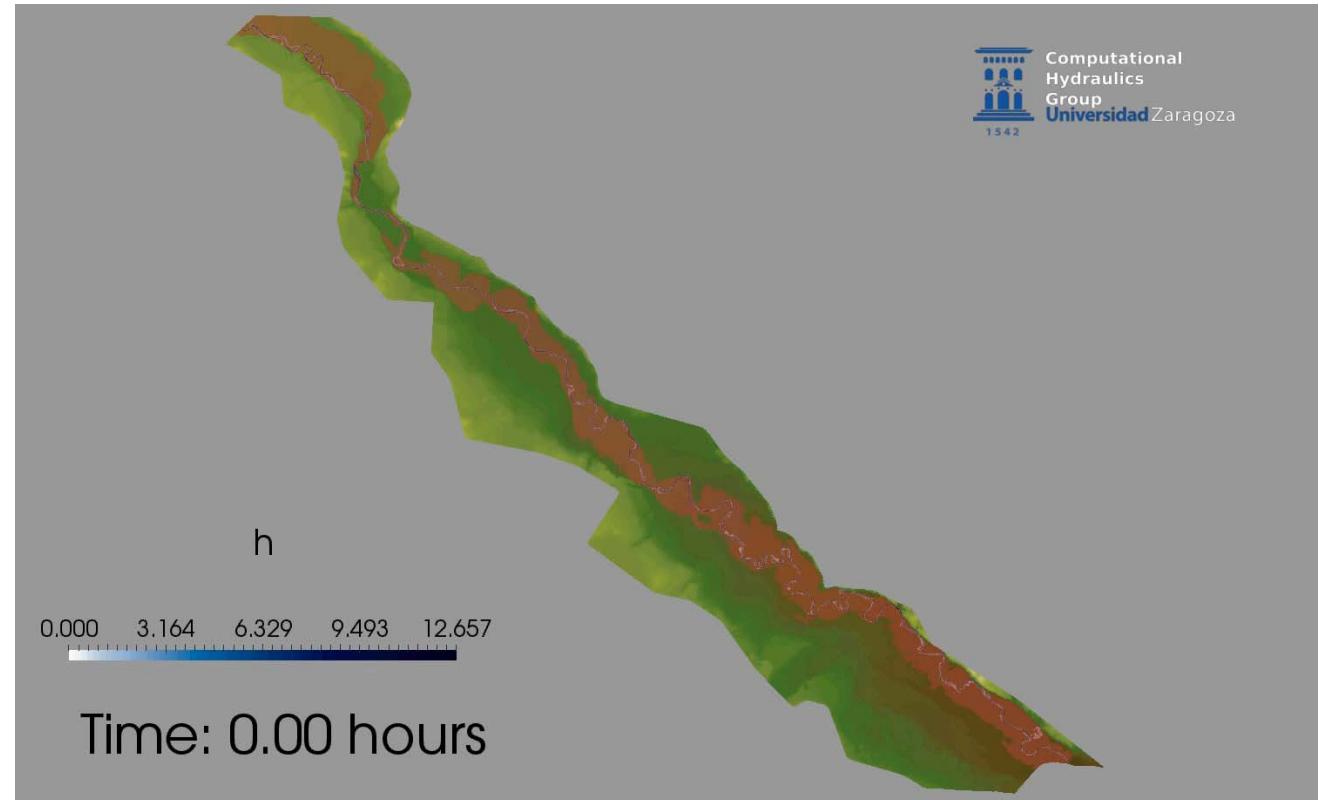


Large-scale modeling



Ebro river flooding

The simulation of 21 days event took 17 hours using an NVIDIA Titan Black GPU



Large-scale modeling

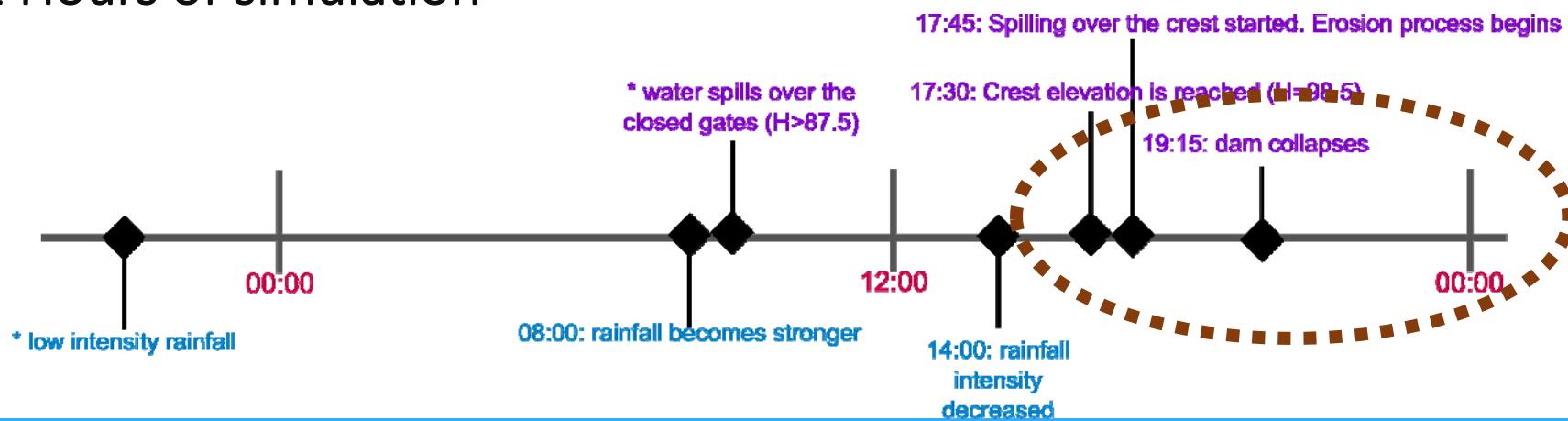
Tous dam-breach

1982 Tous dam (Valencia) disaster

10 km

300000 cells (Refined mesh)

12 Hours of simulation

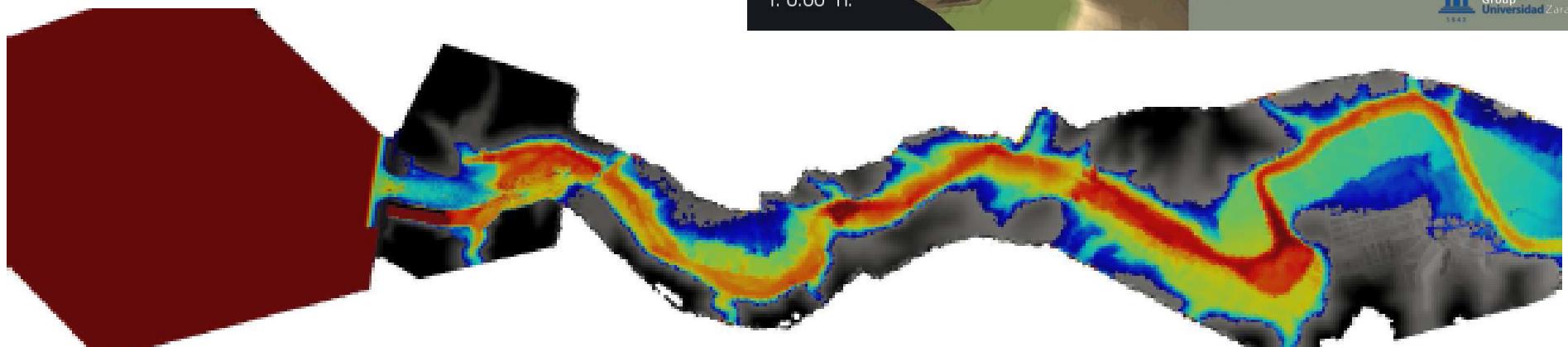
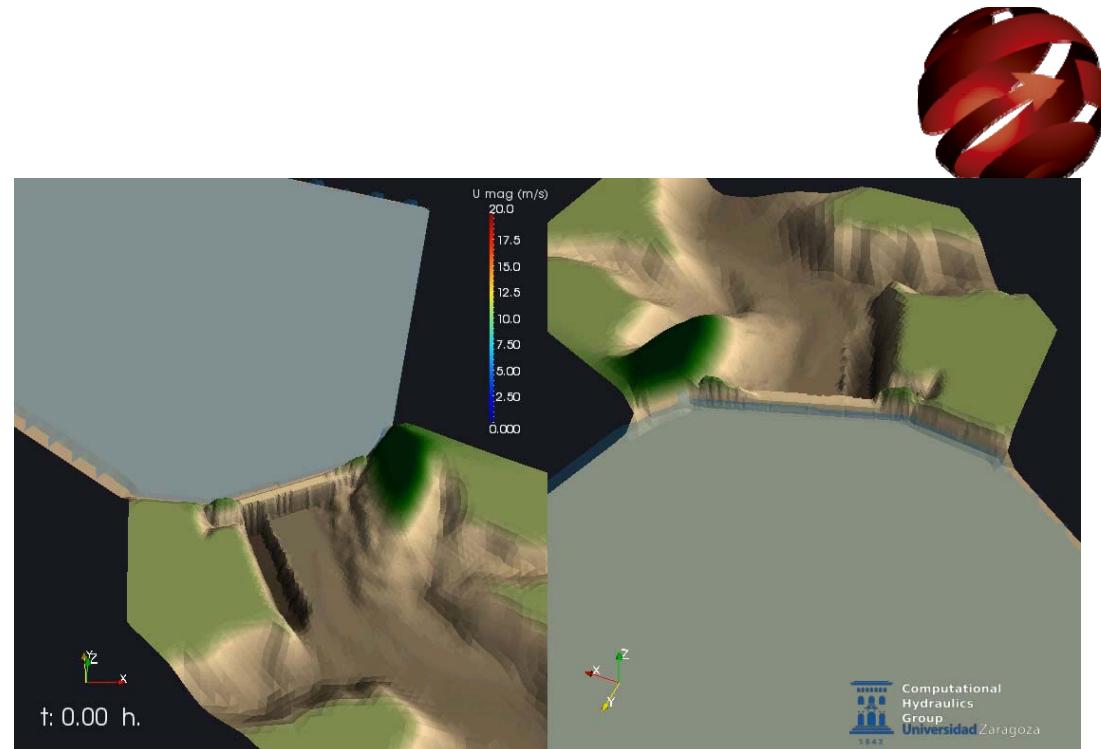


Large-scale modeling

Tous dam-break

CPU vs. GPU

Intel Core i7 3770K	NVIDIA Titan Black
207 Hours 7'	8 Hours 7'
SpeedUp	25.25

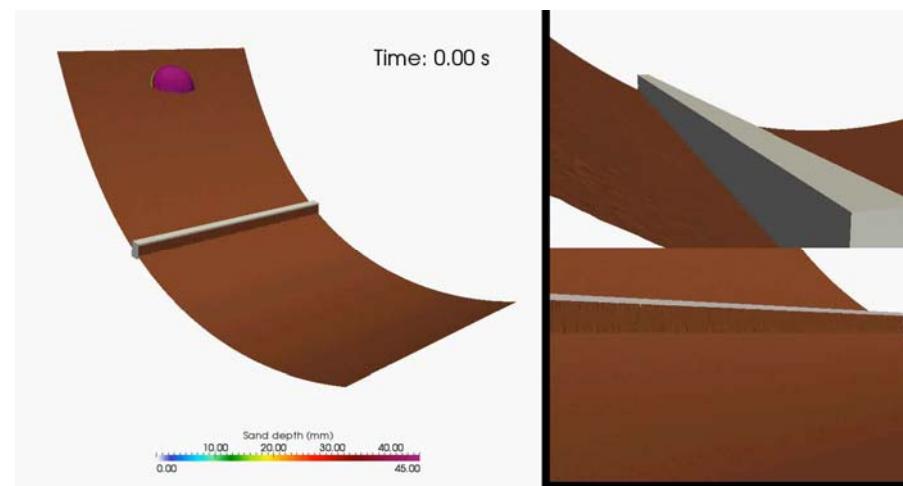


Large-scale modeling

Experimental landslides

Experimental set-up for the validation of the numerical model for landslides applications

- Kinect device + OpenNI + (Nestk & PCL (point-cloud library) libs)



Large-scale modeling

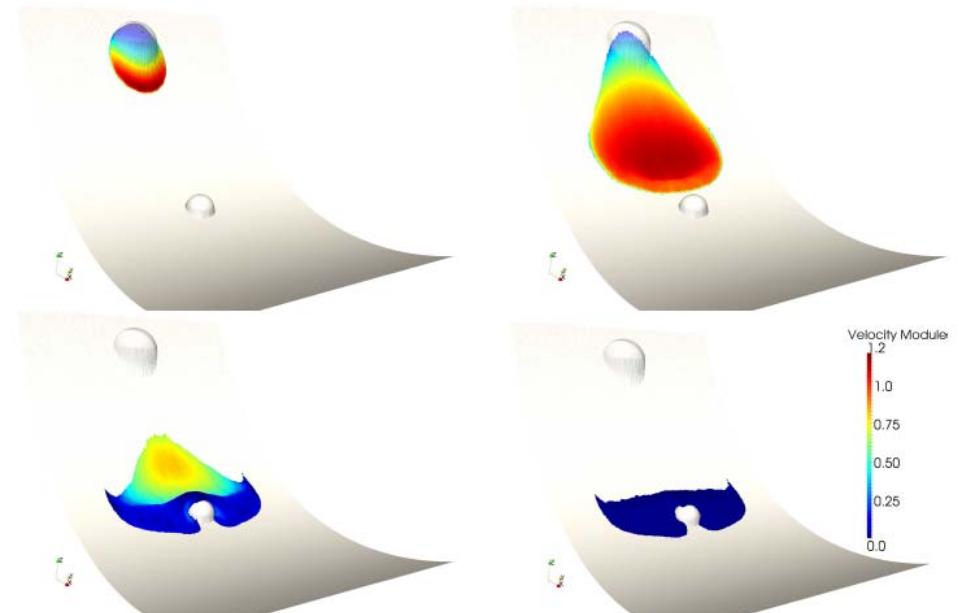


Experimental landslides

One of the experimental cases was simulated in order to compare CPU and GPU implementation and the RCM ordering

Intel Core i7 3770 k @ 3.50 GHz

NVIDIA Tesla c2075



Number of cells	Sequential	4-Cores	Sup	GPU Std	Sup	GPU+RCM	Sup
319354	191.15	81.63	2.34	5.48	34.88	3.83	49.96



Large-scale modeling

Solute transport

This is a live DEMO of the capability that the GPU has for accelerating the computation of these models

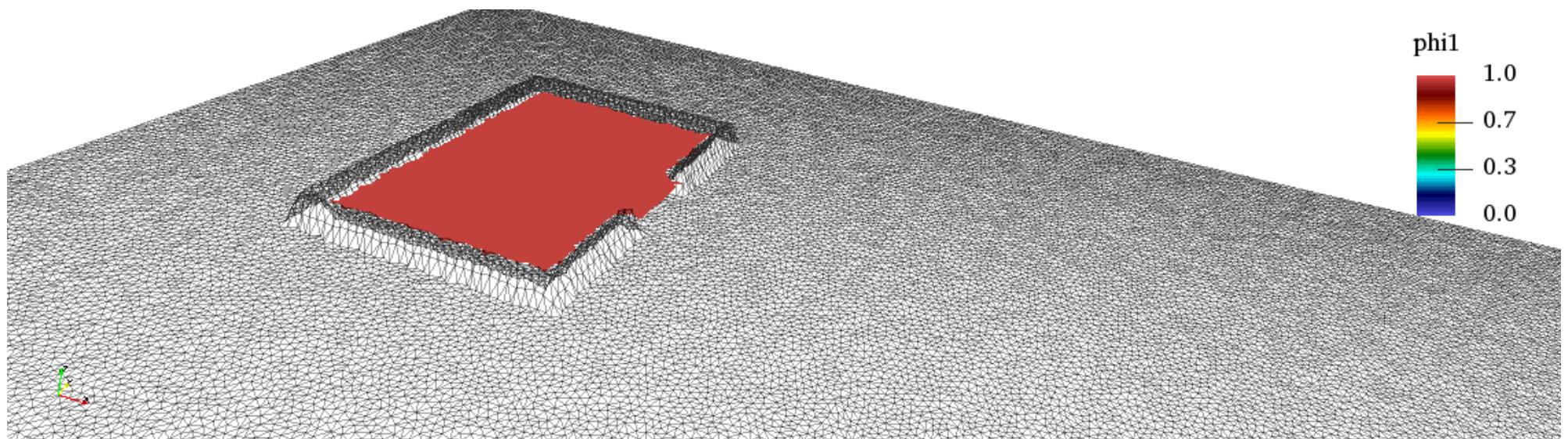




Large-scale modeling

Solute transport

This is a live DEMO of the capability that the GPU has for accelerating the computation of these models



Conclusions



- **GPU computing** can accelerate your models largely
- **Many simulations can be done in a Desktop-type computer** thanks to these devices
- The **performance** reached by these devices **scales depending on the implementation**. Special care must be paid on details!
- It is worthwhile analysing the possible application **of the Many-core model** in your application
- From the **academic point of view**, the advance that implies the GPUs on research is really **promising**. However, **commercial software** requires **additional effort from companies** when introducing the advances that GPU implies.

Acceleration of hydraulic and environmental simulation tools using Graphical Processing Units GPUs



HPC ADMINTECH 2016

Asier Lacasta

*Grupo de Hidráulica Computacional
Universidad de Zaragoza*



**Universidad
Zaragoza**



GPU RESEARCH
CENTER



Some maths...

We are interested in the simulation of the Shallow Water Equations. They represent the flow dynamics in a fluid (sometimes, but not necessarily, a free surface)



$$W_h : \frac{\partial h}{\partial t} + \frac{\partial(hu)}{\partial x} + \frac{\partial(hv)}{\partial y} = i_h$$

$$W_{q_x} : \frac{\partial(hu)}{\partial t} + \frac{\partial}{\partial x} \left(hu^2 + \frac{1}{2} gh^2 \right) + \frac{\partial(huv)}{\partial y} = -gh(S_{0x} - S_{fx})$$

$$W_{q_y} : \frac{\partial(hv)}{\partial t} + \frac{\partial(huv)}{\partial x} + \frac{\partial}{\partial y} \left(hv^2 + \frac{1}{2} gh^2 \right) = -gh(S_{0y} - S_{fy})$$

Some maths...

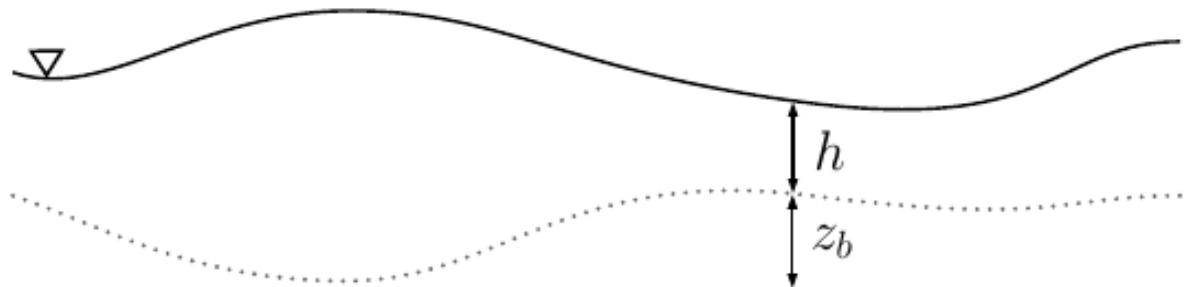


We are interested in the simulation of the Shallow Water Equations

$$W_h : \frac{\partial h}{\partial t} + \frac{\partial(hu)}{\partial x} + \frac{\partial(hv)}{\partial y} =$$

$$W_{q_x} : \frac{\partial(hu)}{\partial t} + \frac{\partial}{\partial x} \left(hu^2 + \frac{1}{2} gh^2 \right) + \frac{\partial(huv)}{\partial y} = -gh(S_{0x} - S_{fx})$$

$$W_{q_y} : \frac{\partial(hv)}{\partial t} + \frac{\partial(huv)}{\partial x} + \frac{\partial}{\partial y} \left(hv^2 + \frac{1}{2} gh^2 \right) = -gh(S_{0y} - S_{fy})$$



Why HPC matters?



- Our numerical tools require large computational effort to perform their calculations



Motivation



Natural disasters like flooding events, landslides, contaminant in rivers or the breach in mining reservoirs are often considered as unavoidable events

Motivation



2015-2016 UK floodings

- **5 December:** Storm Desmond brings more than a month's rain to parts of Cumbria, leading to flooding in Carlisle and other towns
- **12 December:** River levels remain high and more than 70 flood warnings are issued amid more heavy rain
- **22 December:** Communities in Cumbria flood again - some for the third time in less than a month
- **25 December:** More than 100 flood alerts and warnings are issued across England and Wales as Storm Eva brings torrential rain
- **26 December:** Residents in West Yorkshire and Lancashire are evacuated from their homes and flooding hits Leeds, Greater Manchester and York
- **27 December:** Police in York advise hundreds of people to evacuate their homes as 24 severe flood warnings remain in place in the north-east of England
- **30 December:** Storm Frank hits, with Scotland and parts of northern England worst affected by fresh flooding.
- **4 January:** Heavy rain in Scotland causes yet more flooding, particularly in the North East. Residents are evacuated and some roads collapse.



The Telegraph

Overview



The objective is to simulate the unsteady flow motion of free-Surface flows over Surface (e.g. Rivers, roads, mountain)

