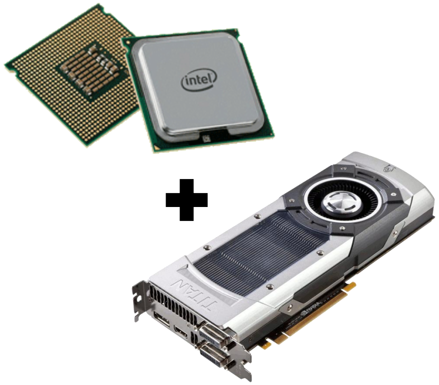
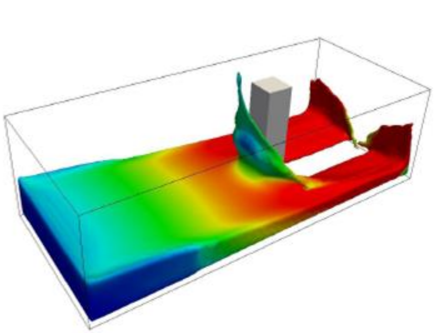
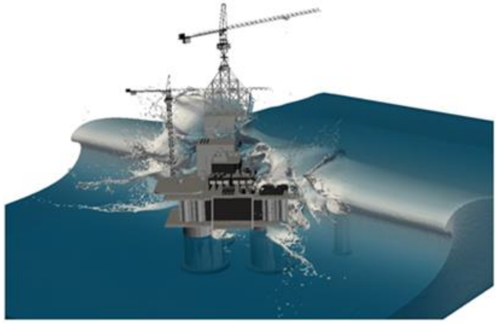
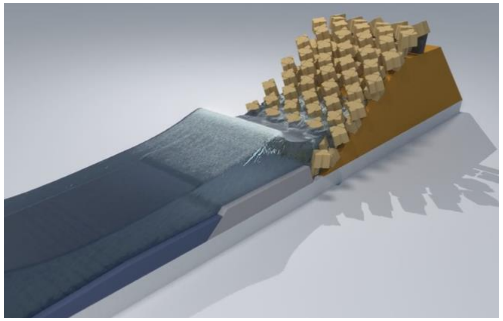
HPC for SPH methods: Multicore, GPU and multiGPU

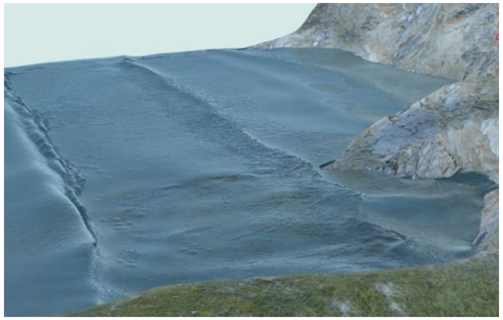
cpu gpu
DualSPHysics







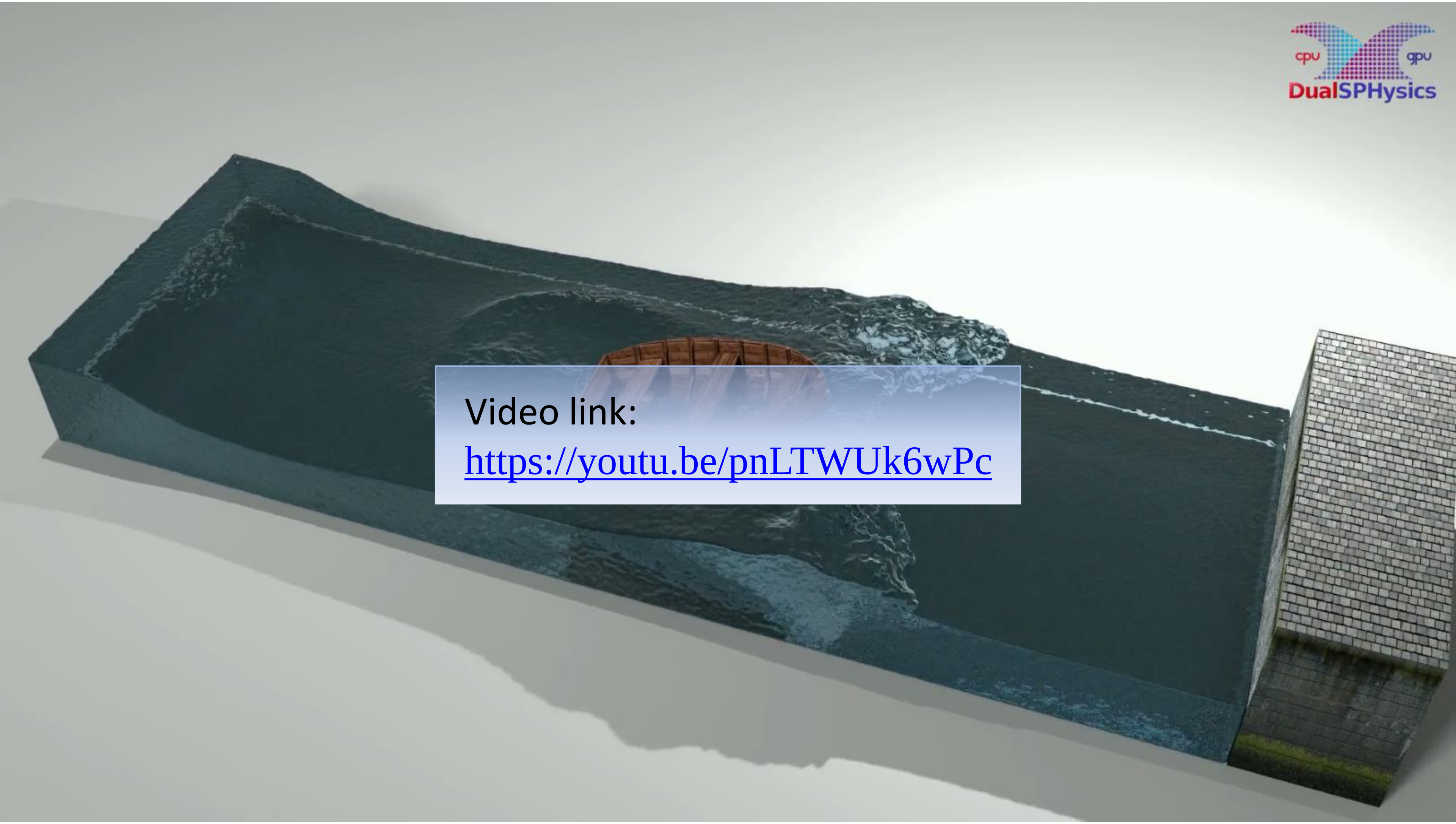




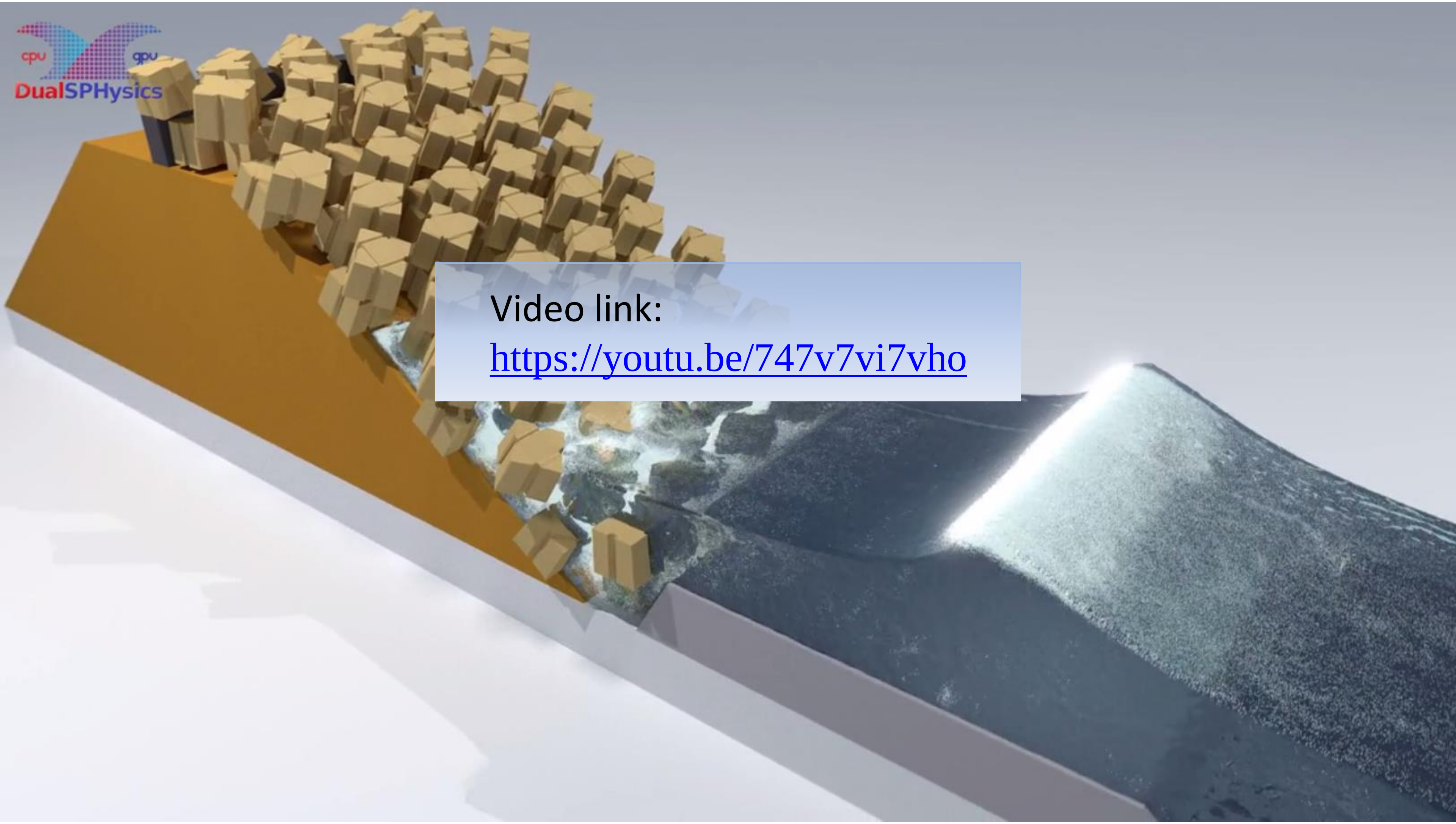
Universidade Vigo

MANCHESTER 1824
The University of Manchester

José Domínguez, A. Mokos, B.D. Rogers,
A.J.C. Crespo and M. Gómez-Gesteira



Video link:
https://youtu.be/pnLTWUk6wPc



Video link:
https://youtu.be/747v7vi7vho

Outline

1. Introduction
   - **1.1. Smooth Particle Hydrodynamics**
   - **1.2. Why is SPH too slow?**
   - **1.3. High Performance Computing (HPC)**
   - **1.4. DualSPHysics project**
2. DualSPHysics implementation
   - 2.1. Implementation in three steps
   - 2.2. Neighbour list approaches
3. CPU acceleration
4. GPU acceleration
   - 4.1. Parallelization problems in SPH
   - 4.2. GPU optimisations
5. Multi-GPU acceleration
   - 5.1. Dynamic load balancing
   - 5.2. Latest optimisations in Multi-GPU
   - 5.3. Large simulations
   - 5.4. Future improvements
6. New Multi-GPU approach

1.1. Smooth Particle Hydrodynamics

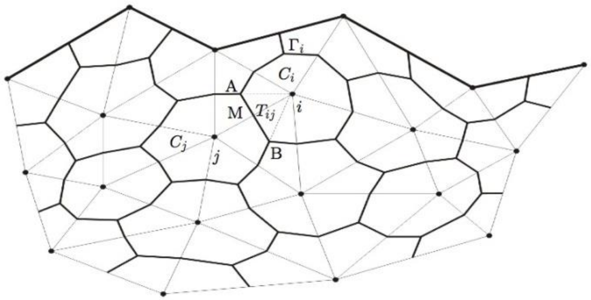
PHYSICAL GOVERNING EQUATIONS

EULERIAN DESCRIPTION
(spatial description)

LAGRANGIAN DESCRIPTION
(material description)

COMPUTATIONAL METHODS

GRID-BASED METHODS



SPH

MESHFREE METHODS

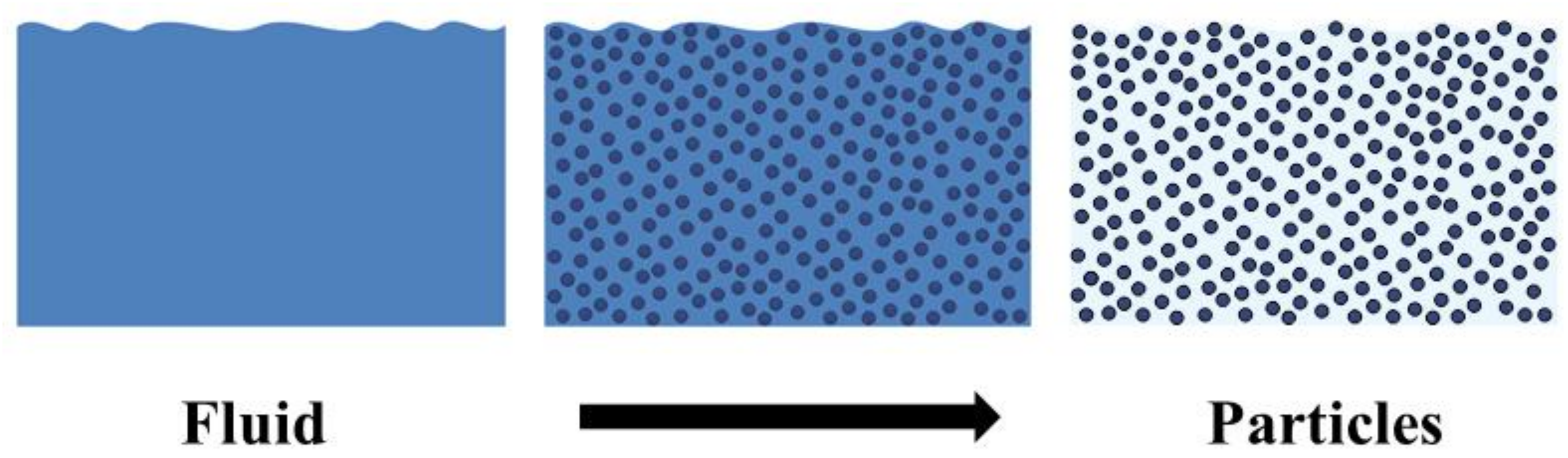
MESHFREE PARTICLE METHODS
(particle represents a part of
the continuum domain)

SMOOTHED PARTICLE HYDRODYNAMICS

1.1. Smooth Particle Hydrodynamics

SPH method was invented for astrophysics during the seventies, but now it is applied in many different fields including fluid dynamics and solid mechanics.

Fluid is represented using particles which move according to the governing dynamics.



Comparing to grid-based methods, SPH interactions are carried out between a given particle and its moving neighbours. Thus, these **neighbours are unknown since they change at each instant.**

1.1. Smooth Particle Hydrodynamics

SPH method was invented for astrophysics during the seventies, but now it is applied in many different fields including fluid dynamics and solid mechanics.

Fluid is represented using particles which move according to the governing dynamics.

SPH is particularly suited to describe a variety of **free-surface flows**:

- Wave propagation over a beach.
- Plunging breakers.
- Wave-structure interactions.
- Solid bodies impacting on water surface.
- Dam breaks.

1.2. Why is SPH too slow?

Drawbacks of SPH:

- SPH presents a **high computational cost** that increases when increasing the number of particles.

+

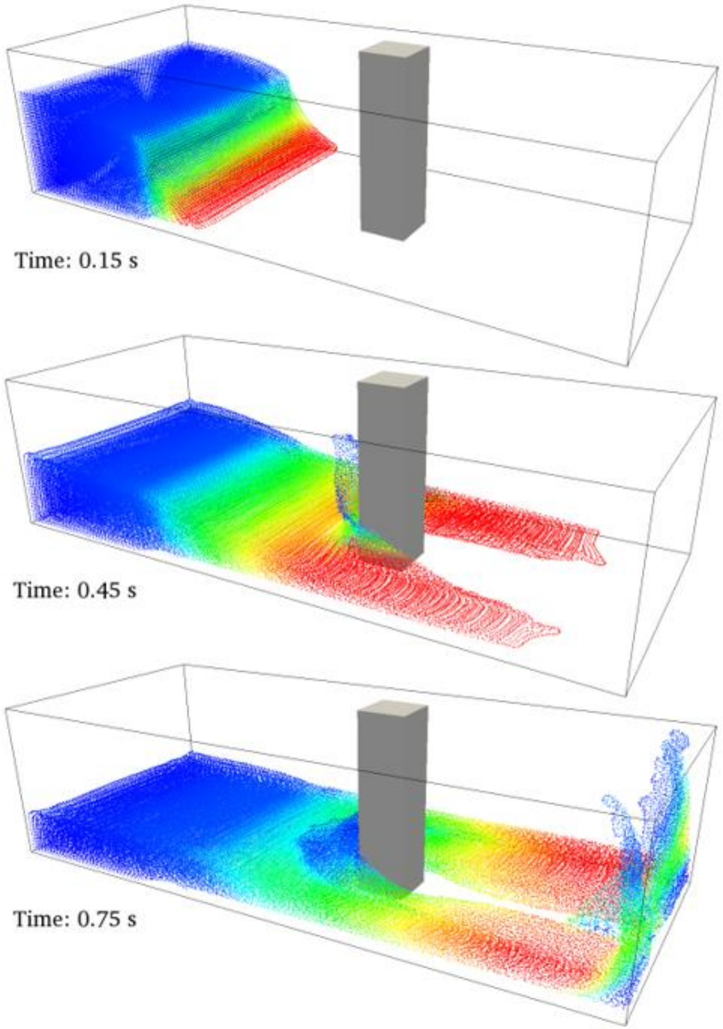
- The simulation of **real problems** requires a high resolution which implies simulating **millions of particles**.

⬇

The **time required** to simulate a few seconds is **too large**. One second of physical time can take several days of calculation.

1.2. Why is SPH too slow?

The SPH method is very expensive in terms of computing time.



For example, a simulation of this dam break

300,000 particles

✚

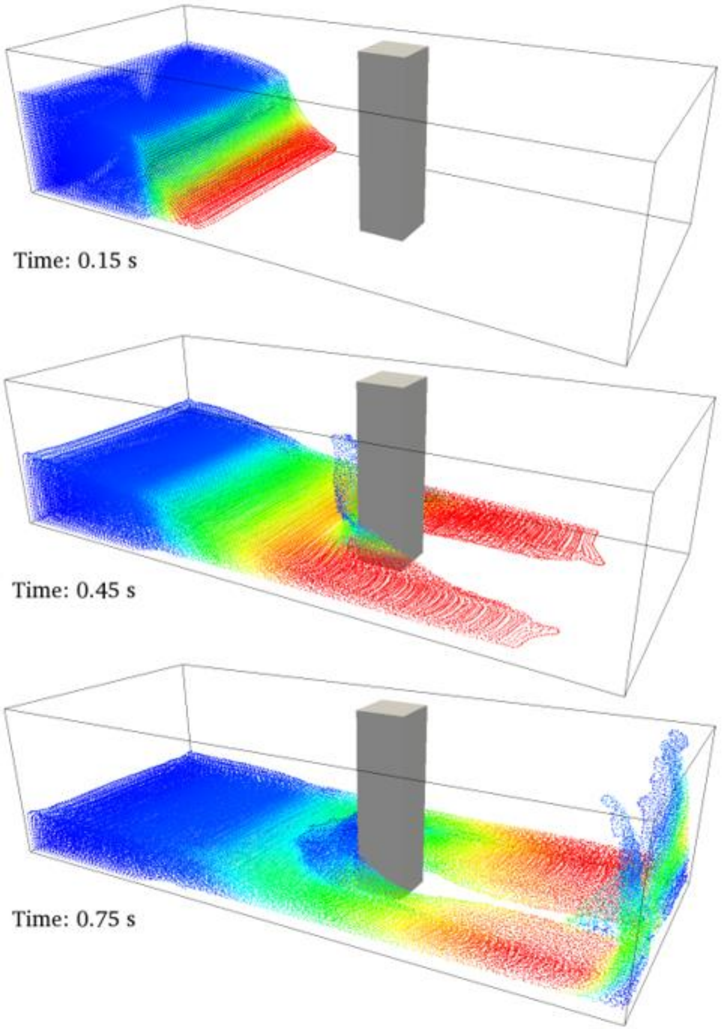
1.5 s (*physical time*)

➡

Takes more than
15 hours
(*execution time*)

1.2. Why is SPH too slow?

The SPH method is very expensive in terms of computing time.



For example, a simulation of this dam break

300,000 particles

+

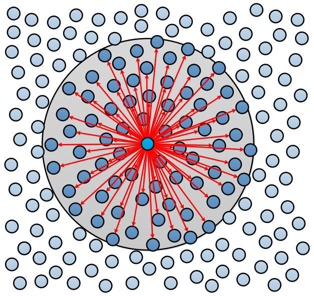
1.5 s (*physical time*)

➡

Takes more than
15 hours
(*execution time*)

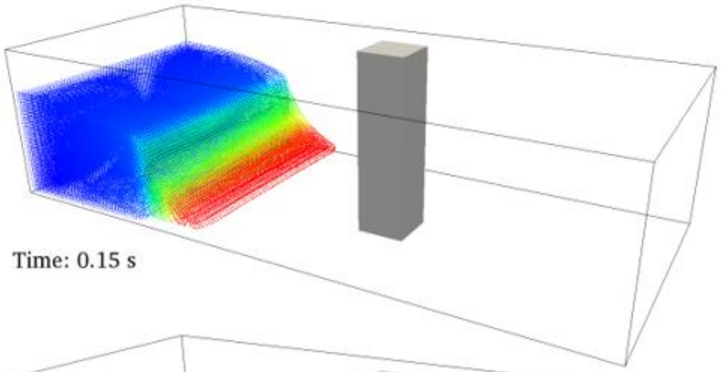
because:

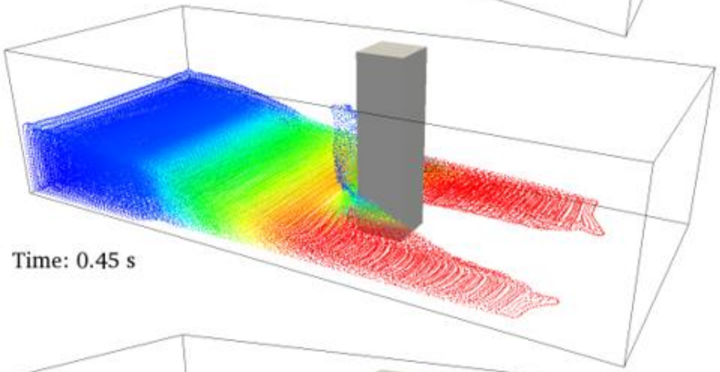
- Each particle interacts with **more than 250 neighbours**.

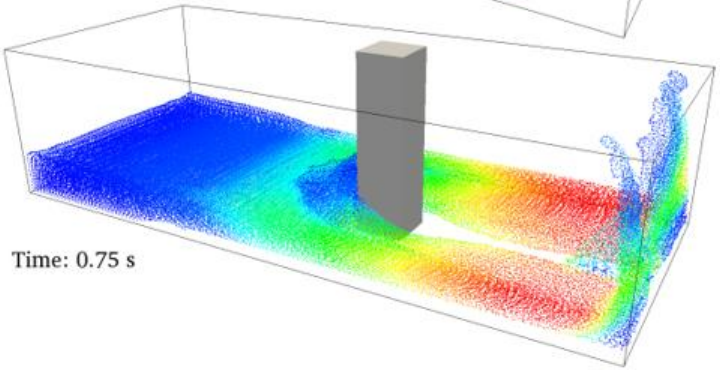


1.2. Why is SPH too slow?

The SPH method is very expensive in terms of computing time.







For example, a simulation of this dam break

300,000 particles

+

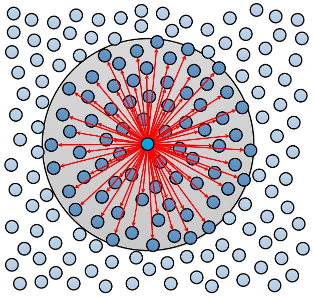
1.5 s (*physical time*)

➡

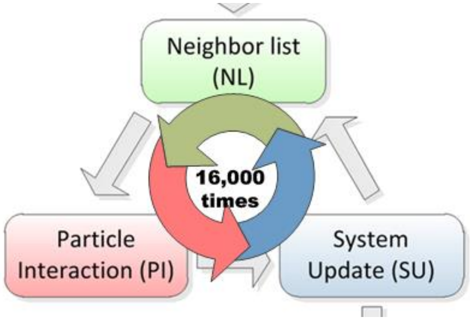
Takes more than
15 hours
(*execution time*)

because:

- Each particle interacts with **more than 250 neighbours**.



- **$\Delta t=10^{-5}-10^{-4}$ so more than 16,000 steps** are needed to simulate 1.5 s of physical time.



1.2. Why is SPH too slow?

Drawbacks of SPH:

- SPH presents a **high computational cost** that increases when increasing the number of particles.

✚

- The simulation of **real problems** requires a high resolution which implies simulating **millions of particles**.

⬇

The **time required** to simulate a few seconds is **too large**. One second of physical time can take several days of calculation.

It is necessary to use HPC techniques to reduce these computation times.

1.3. High Performance Computing (HPC)

HPC includes multiple techniques of parallel computing and distributed computing that allow you to execute several operations simultaneously.

The main techniques used to accelerate SPH are:

- **OpenMP** (Open Multi-Processing)
  - Model of parallel programming for systems of shared memory.
  - Portable and flexible programming interface using directives.
  - Its implementation does not involve major changes in the code.
  - The improvement is limited by the number of cores.

Multi-core processor

OPENMP IS THE BEST OPTION TO OPTIMIZE THE PERFORMANCE OF THE MULTIPLE CORES OF THE CURRENT CPUS.

1.3. High Performance Computing (HPC)

HPC includes multiple techniques of parallel computing and distributed computing that allow you to execute several operations simultaneously.

The main techniques used to accelerate SPH are:

- **MPI** (Message Passing Interface)
  - Message-passing library specification for systems of distributed memory: parallel computers and clusters.
  - Several processes are communicated by calling routines to send and receive messages.
  - The use of MPI is typically combined with OpenMP in clusters by using a hybrid communication model.
  - Very expensive for a small research group.

MPI cluster

MPI IS THE BEST OPTION TO COMBINE THE RESOURCES OF MULTIPLE MACHINES CONNECTED VIA NETWORK.

1.3. High Performance Computing (HPC)

HPC includes multiple techniques of parallel computing and distributed computing that allow you to execute several operations simultaneously.

The main techniques used to accelerate SPH are:

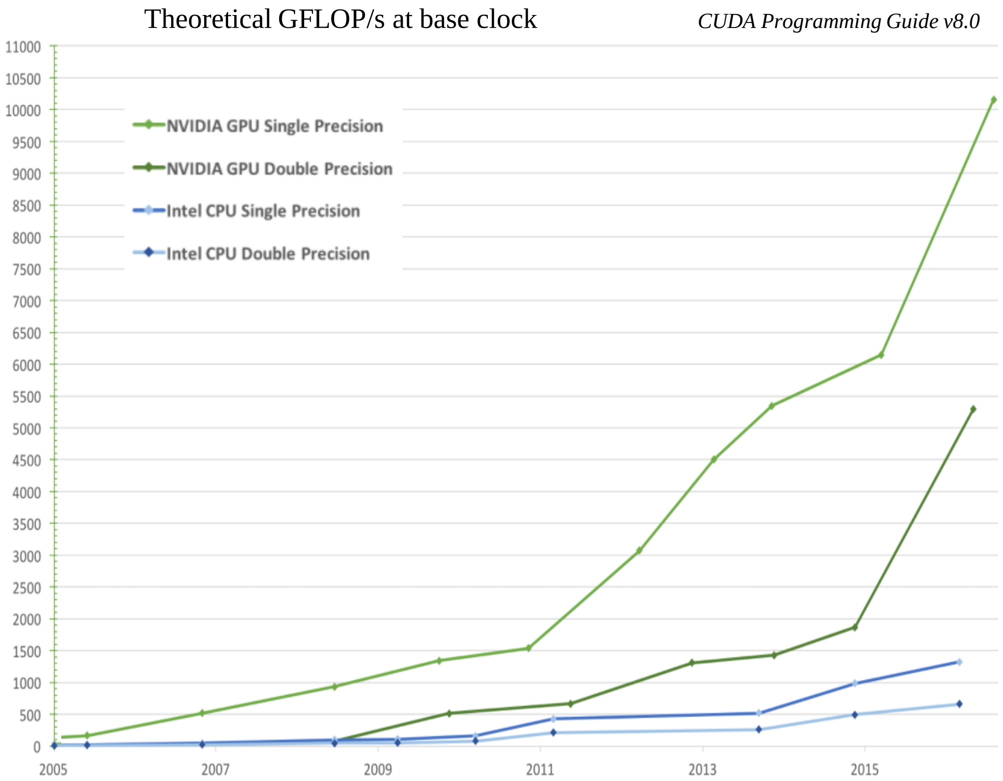
- **GPGPU** (General-Purpose Computing on Graphics Processing Units)
  - It involves the study and use of parallel computing ability of a GPU to perform general purpose programs.
  - New general purpose programming languages and APIs (such as Brook and **CUDA**) provide an easier access to the computing power of GPUs.
  - New implementation of the algorithms used in CPU is necessary for an efficient use in GPU.

GPU

1.3. High Performance Computing (HPC)

Graphics Processing Units (GPUs)

- powerful parallel processors
- designed for graphics rendering
- their computing power has increased much faster than CPUs.



Advantages: GPUs provide a high calculation power with very low cost and without expensive infrastructures.

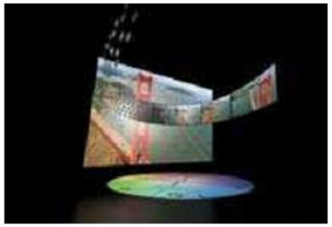
Drawbacks: An efficient and full use of the capabilities of the GPUs is not straightforward.

1.3. High Performance Computing (HPC)

Why GPUs?

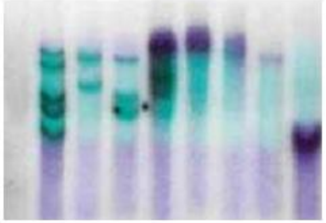
GPUs are an accessible tool to accelerate SPH,
all numerical methods in CFD and any computational method

5X
Digital Content Creation
Adobe

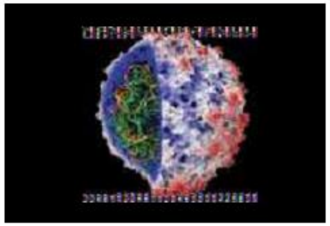


18X
Video Transcoding
Elemental Technologies

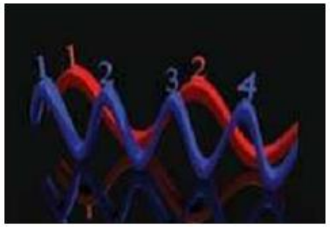
20X
3D Ultrasound
TechniScan



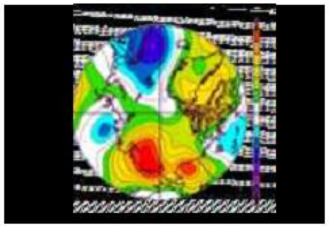
30X
Gene Sequencing
U of Maryland



36X
Molecular Dynamics
U of Illinois, Urbana-Champaign



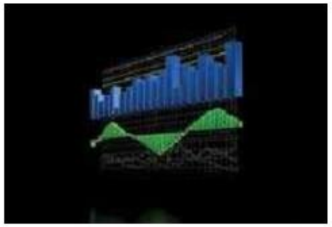
50X
MATLAB Computing
AccelerEyes



80X
Weather Modeling
Tokyo Institute of Technology

100X
Astrophysics
RIKEN

146X
Medical Imaging
U of Utah



149X
Financial Simulation
Oxford University

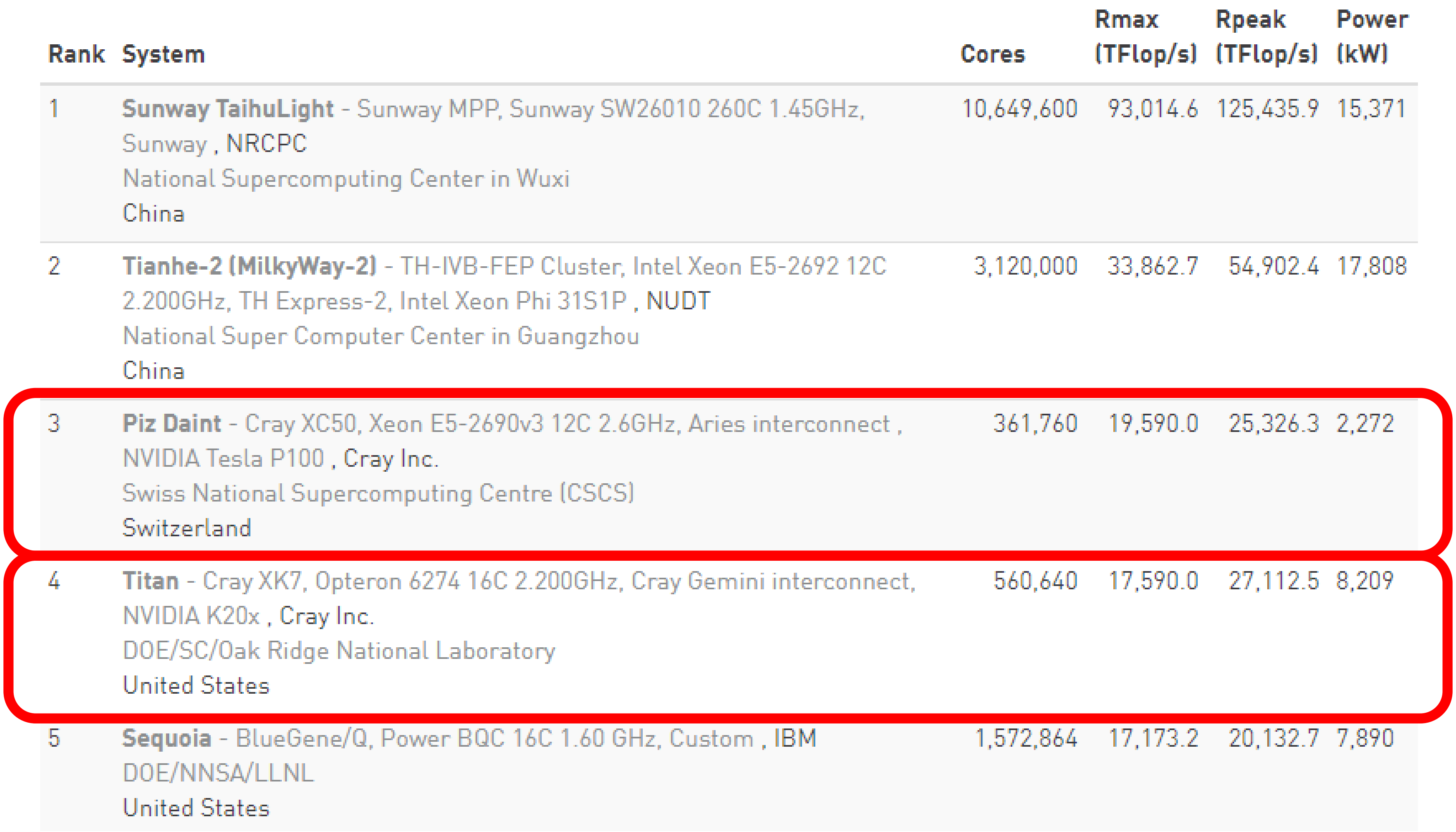
http://www.nvidia.com

1.3. High Performance Computing (HPC)

Why GPUs?

TOP500 LIST – JUNE 2017

https://www.top500.org/lists/2017/06/



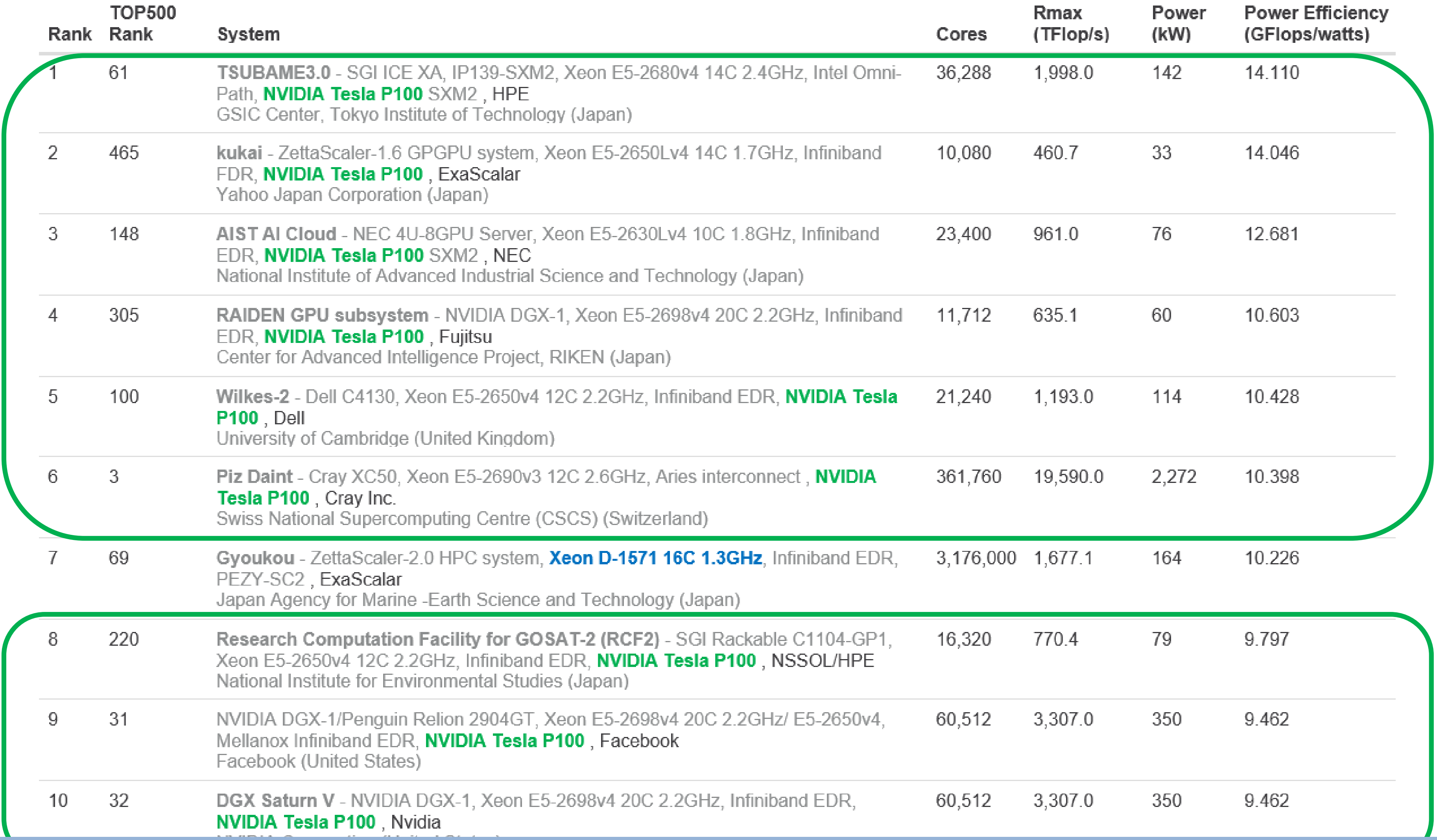
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890

1.3. High Performance Computing (HPC)

Why GPUs?

GREEN500 LIST – JUNE 2016

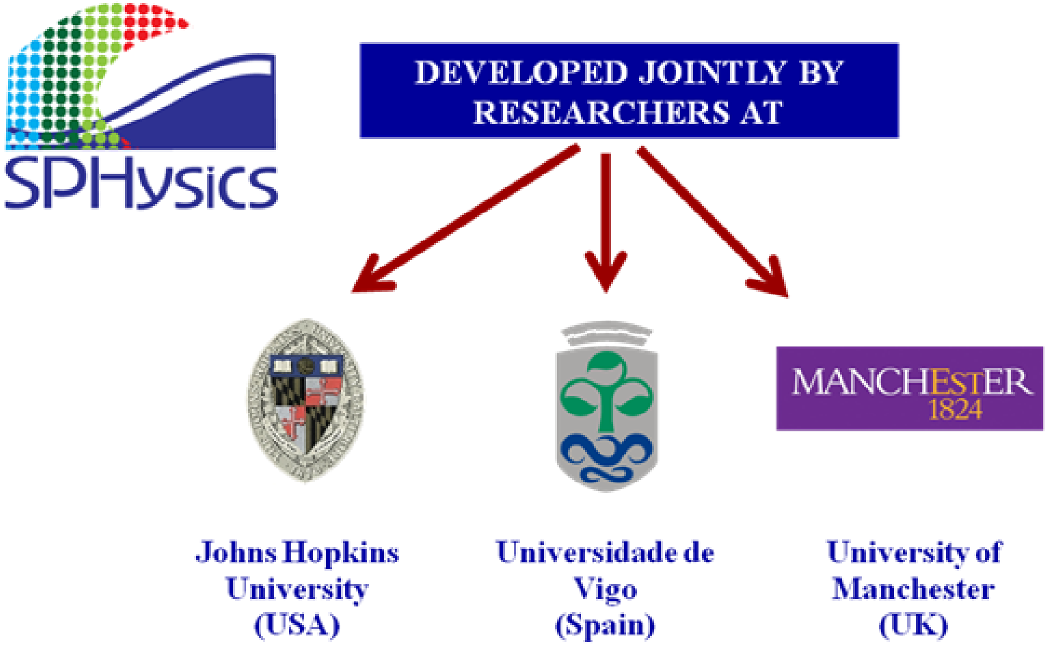
https://www.top500.org/green500/lists/2017/06/



Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	61	**TSUBAME3.0** - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, **NVIDIA Tesla P100** SXM2 , HPE GSIC Center, Tokyo Institute of Technology (Japan)	36,288	1,998.0	142	14.110
2	465	**kukai** - ZettaScaler-1.6 GPGPU system, Xeon E5-2650Lv4 14C 1.7GHz, Infiniband FDR, **NVIDIA Tesla P100** , ExaScalar Yahoo Japan Corporation (Japan)	10,080	460.7	33	14.046
3	148	**AIST AI Cloud** - NEC 4U-8GPU Server, Xeon E5-2630Lv4 10C 1.8GHz, Infiniband EDR, **NVIDIA Tesla P100** SXM2 , NEC National Institute of Advanced Industrial Science and Technology (Japan)	23,400	961.0	76	12.681
4	305	**RAIDEN GPU subsystem** - NVIDIA DGX-1, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, **NVIDIA Tesla P100** , Fujitsu Center for Advanced Intelligence Project, RIKEN (Japan)	11,712	635.1	60	10.603
5	100	**Wilkes-2** - Dell C4130, Xeon E5-2650v4 12C 2.2GHz, Infiniband EDR, **NVIDIA Tesla P100** , Dell University of Cambridge (United Kingdom)	21,240	1,193.0	114	10.428
6	3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , **NVIDIA Tesla P100** , Cray Inc. Swiss National Supercomputing Centre (CSCS) (Switzerland)	361,760	19,590.0	2,272	10.398
7	69	**Gyoukou** - ZettaScaler-2.0 HPC system, **Xeon D-1571 16C 1.3GHz**, Infiniband EDR, PEZY-SC2 , ExaScalar Japan Agency for Marine -Earth Science and Technology (Japan)	3,176,000	1,677.1	164	10.226
8	220	**Research Computation Facility for GOSAT-2 (RCF2)** - SGI Rackable C1104-GP1, Xeon E5-2650v4 12C 2.2GHz, Infiniband EDR, **NVIDIA Tesla P100** , NSSOL/HPE National Institute for Environmental Studies (Japan)	16,320	770.4	79	9.797
9	31	NVIDIA DGX-1/Penguin Relion 2904GT, Xeon E5-2698v4 20C 2.2GHz/ E5-2650v4, Mellanox Infiniband EDR, **NVIDIA Tesla P100** , Facebook Facebook (United States)	60,512	3,307.0	350	9.462
10	32	**DGX Saturn V** - NVIDIA DGX-1, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, **NVIDIA Tesla P100** , Nvidia	60,512	3,307.0	350	9.462

1.4. DualSPHysics project

The DualSPHysics code was created starting from SPHysics.



DEVELOPED JOINTLY BY RESEARCHERS AT

Johns Hopkins University (USA)

Universidade de Vigo (Spain)

University of Manchester (UK)

SPHysics is a numerical model SPH developed for the study of free-surface problems.

It is a code written in Fortran90 with numerous options (different kernels, several boundary conditions,…), which had already demonstrated high accuracy in several validations with experimental results… but it is too slow to apply to large domains.

1.4. DualSPHysics project

First version in late 2009.

It includes **two implementations**:

- **CPU**: C++ and OpenMP.
- **GPU**: CUDA.

Both options optimized for the best performance of each architecture.

Why two implementations?

This code can be used on machines with GPU and without GPU.

It allows us to make a fair and realistic comparison between CPU and GPU.

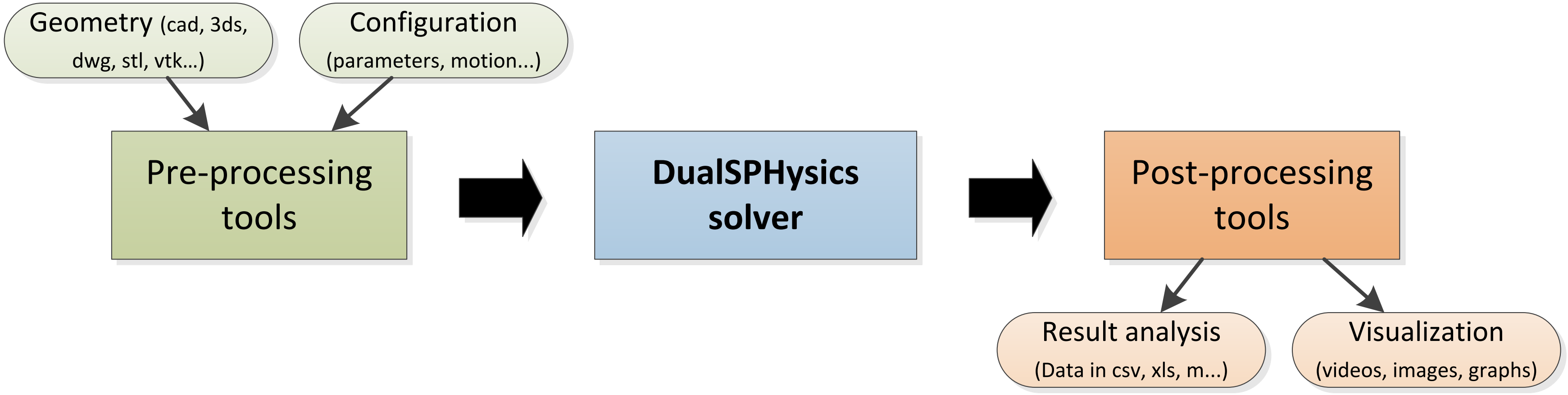
Some algorithms are complex and it is easy to make errors difficult to detect. So they are implemented twice and we can compare results.

It is easier to understand the code in CUDA when you can see the same code in C++.

Drawback: It is necessary to implement and to maintain two different codes.

1.4. DualSPHysics project

DSPH project includes:


Geometry (cad, 3ds, dwg, stl, vtk...)
Configuration (parameters, motion...)
Pre-processing tools
DualSPHysics solver
Post-processing tools
Result analysis (Data in csv, xls, m...)
Visualization (videos, images, graphs)

Pre-processing tools:

- Converts geometry into particles.
- Provides configuration for simulation.

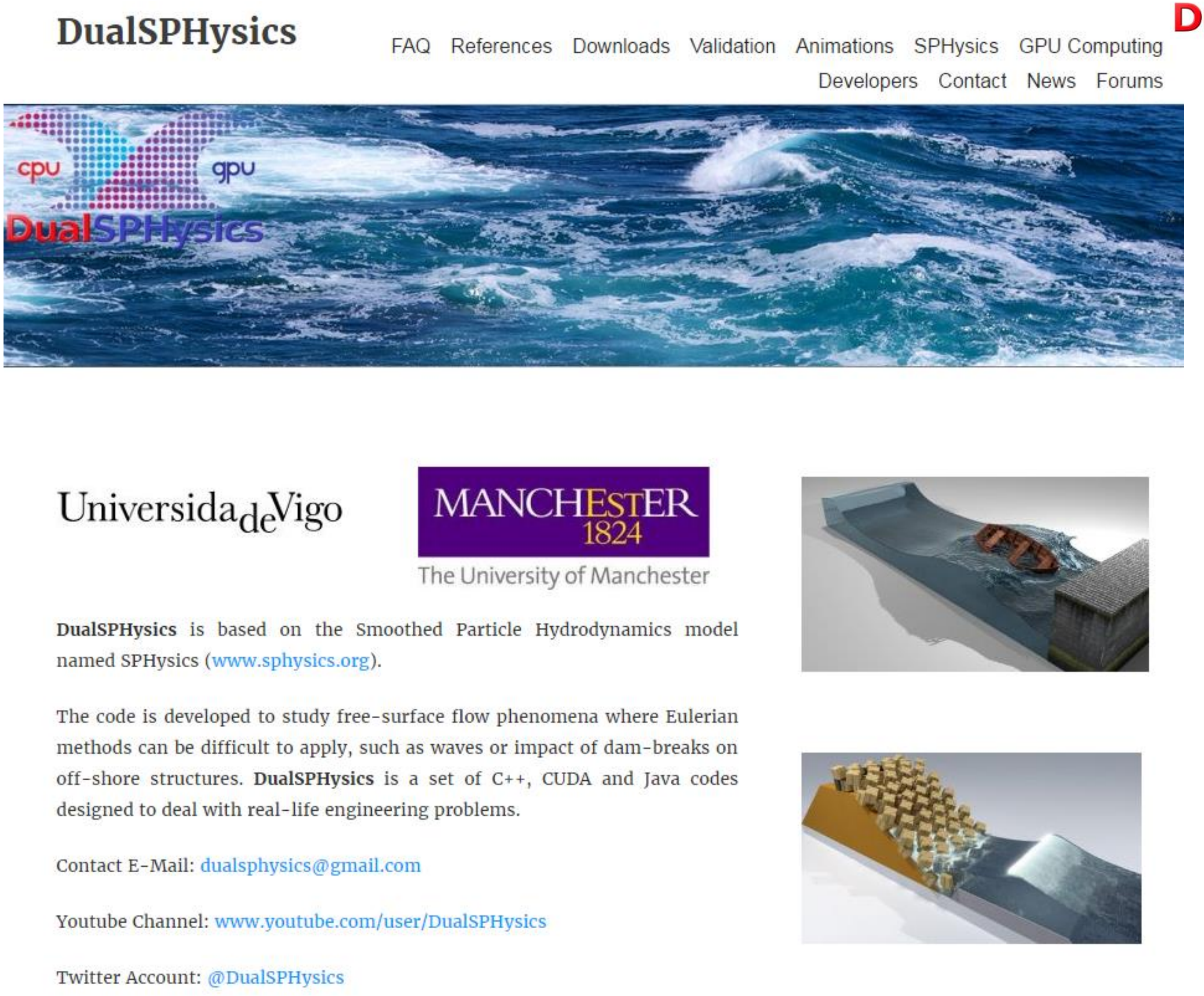
DualSPHysics solver:

- Runs simulation using SPH particles.
- Obtains data simulation for time intervals.

Post-processing tools:

- Calculates magnitudes using particle data.
- Generates images and videos starting form SPH particles.

1.4. DualSPHysics project



DualSPHysics is based on the Smoothed Particle Hydrodynamics model named SPHysics (www.sphysics.org).

The code is developed to study free-surface flow phenomena where Eulerian methods can be difficult to apply, such as waves or impact of dam-breaks on off-shore structures. **DualSPHysics** is a set of C++, CUDA and Java codes designed to deal with real-life engineering problems.

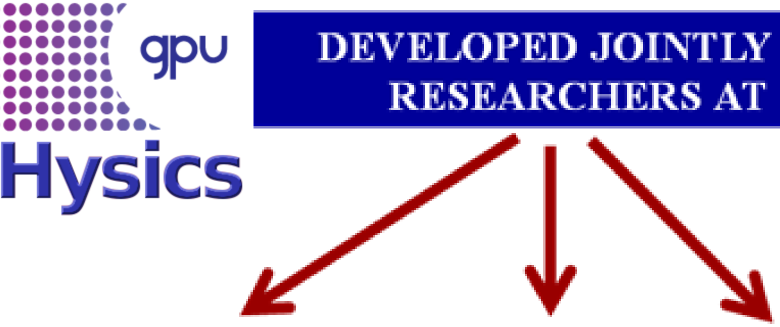
Contact E-Mail: dualsphysics@gmail.com

Youtube Channel: www.youtube.com/user/DualSPHysics

Twitter Account: @DualSPHysics

www.dual.sphysics.org

1.4. DualSPHysics project

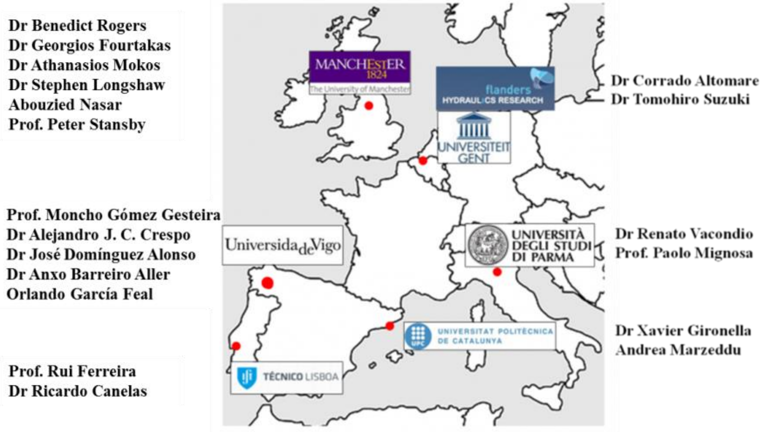


DEVELOPED JOINTLY BY RESEARCHERS AT

Universidade de Vigo (Spain)

University of Manchester (UK)

and many contributors



Dr Benedict Rogers
Dr Georgios Fourtakas
Dr Athanasios Mokos
Dr Stephen Longshaw
Abouzied Nasar
Prof. Peter Stansby

Prof. Moncho Gómez Gesteira
Dr Alejandro J. C. Crespo
Dr José Domínguez Alonso
Dr Anxo Barreiro Aller
Orlando García Feal

Prof. Rui Ferreira
Dr Ricardo Canelas

Dr Corrado Altomare
Dr Tomohiro Suzuki

Dr Renato Vacondio
Prof. Paolo Mignosa

Dr Xavier Gironella
Andrea Marzeddu

MANCHESTER 1824
The University of Manchester
flanders HYDRAULICS RESEARCH
UNIVERSITEIT GENT
UniversidadeVigo
UNIVERSITÀ DEGLI STUDI DI PARMA
UNIVERSITAT POLITÈCNICA DE CATALUNYA
TÉCNICO LISBOA

www.dual.sphysics.org

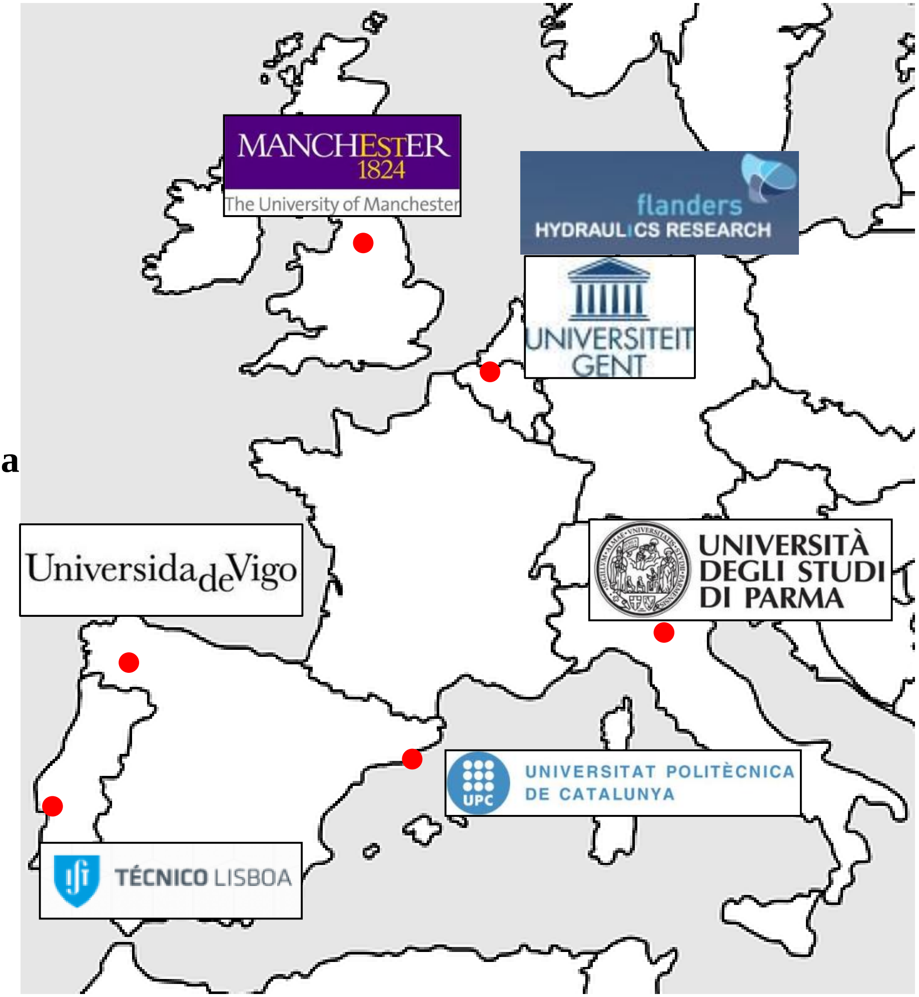
1.4. DualSPHysics project

People working on DualSPHysics project:

Dr Benedict D. Rogers
Dr Athanasios Mokos
Dr Georgios Fourtakas
Prof. Peter Stansby
Alex Chow
Annelie Baines

Prof. Moncho Gómez Gesteira
Dr Alejandro J.C. Crespo
Dr Jose M. Domínguez
Dr José González-Cao
Orlando G. Feal
Andrés Vieira

Prof. Rui Ferreira
Dr Ricardo Canelas

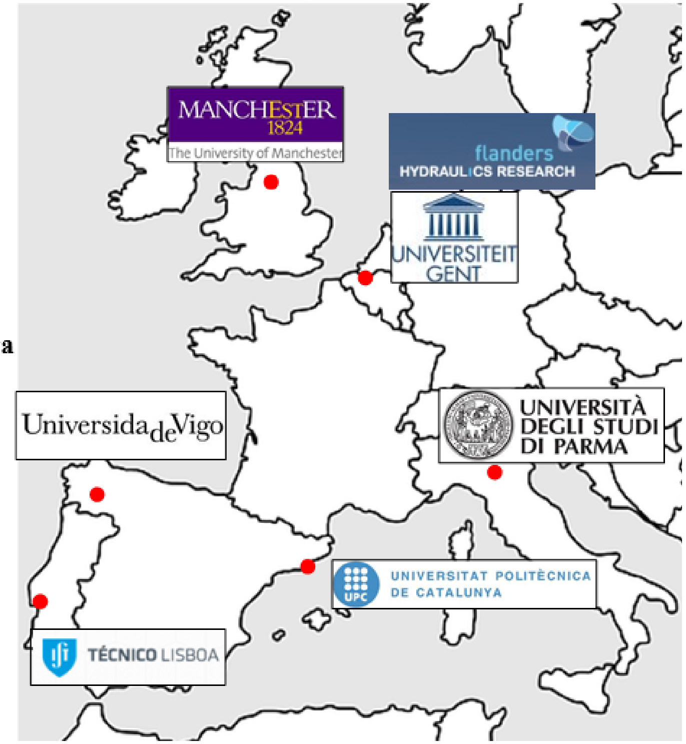

MANCHESTER 1824
The University of Manchester
flanders HYDRAULICS RESEARCH
UNIVERSITEIT GENT
UniversidadeVigo
UNIVERSITÀ DEGLI STUDI DI PARMA
UNIVERSITAT POLITÈCNICA DE CATALUNYA
UPC
TÉCNICO LISBOA

Dr Corrado Altomare
Dr Tomohiro Suzuki

Dr Renato Vacondio
Prof. Paolo Mignosa

Dr Xavier Gironella
Andrea Marzeddu

1.4. DualSPHysics project


MANCHESTER 1824
The University of Manchester
flanders HYDRAULICS RESEARCH
UNIVERSITEIT GENT
UNIVERSITÀ DEGLI STUDI DI PARMA
UniversidadeVigo
UNIVERSITAT POLITÈCNICA DE CATALUNYA
UPC
TÉCNICO LISBOA

Dr Benedict D. Rogers
Dr Athanasios Mokos
Dr Georgios Fourtakas
Prof. Peter Stansby
Alex Chow
Annelie Baines

Prof. Moncho Gómez Gesteira
Dr Alejandro J.C. Crespo
Dr Jose M. Domínguez
Dr José González-Cao
Orlando G. Feal
Andrés Vieira

Prof. Rui Ferreira
Dr Ricardo Canelas

Dr Corrado Altomare
Dr Tomohiro Suzuki

Dr Renato Vacondio
Prof. Paolo Mignosa

Dr Xavier Gironella
Andrea Marzeddu

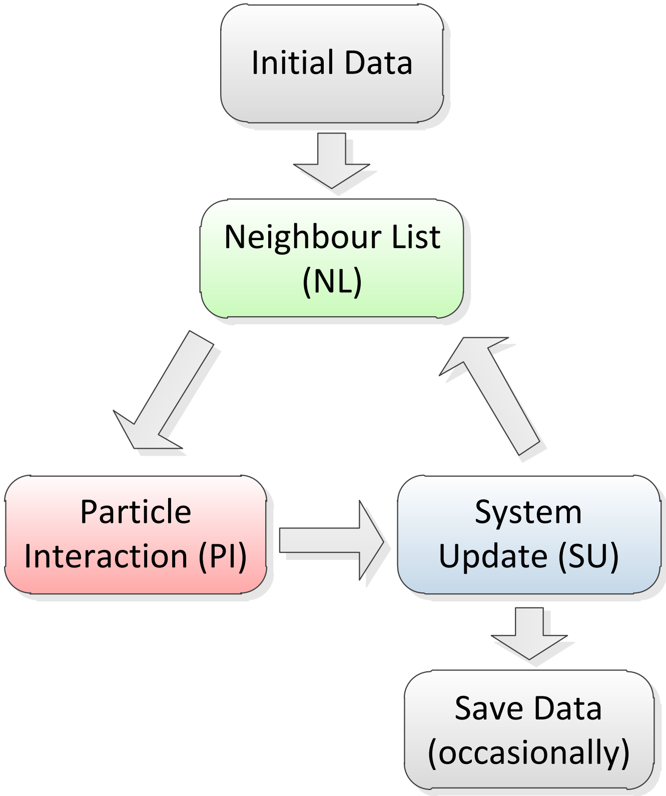
It has been downloaded and used by researchers but also by companies: *NASA JSC, BAE Systems, Volkswagen AG, McLaren Racing Ltd, Forum NOKIA, NVIDIA, AECOM, HDR Engineering, ABPmer, DLR, Maine Marine Composites, CFD-NUMERICS, BMT Group, Oak Ridge National Laboratory, Rainpower Norway, American Wave Machines,, National Renewable Energy Laboratory in U.S.A., Atria Power Corporation Ltd., Global Hydro Energy, Carnegie Wave Energy Ltd, etc.*

Outline

1. Introduction
   - 1.1. Smooth Particle Hydrodynamics
   - 1.2. Why is SPH too slow?
   - 1.3. High Performance Computing (HPC)
   - 1.4. DualSPHysics project
2. **DualSPHysics implementation**
   - **2.1. Implementation in three steps**
   - **2.2. Neighbour list approaches**
3. CPU acceleration
4. GPU acceleration
   - 4.1. Parallelization problems in SPH
   - 4.2. GPU optimisations
5. Multi-GPU acceleration
   - 5.1. Dynamic load balancing
   - 5.2. Latest optimisations in Multi-GPU
   - 5.3. Large simulations
   - 5.4. Future improvements
6. New Multi-GPU approach

2.1. Implementation in three steps

For the implementation of SPH, the code is organised in **3 main steps** that are repeated each time step till the end of the simulation.

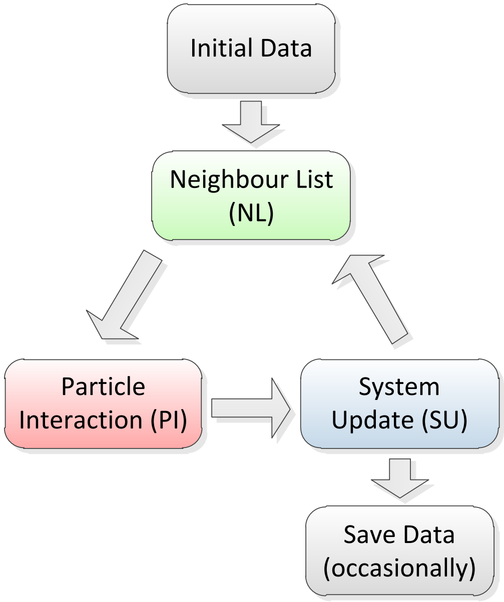


Neighbour list (NL):
Particles are grouped in cells and reordered to optimise the next step.

Particle interactions (PI):
Forces between particles are computed, solving momentum and continuity equations.
This step takes **more than 95%** of execution time.

System update (SU):
Starting from the values of computed forces, the magnitudes of the particles are updated for the next instant of the simulation.

2.2. Neighbour list approaches



Particle Interaction (PI) consumes more than 95% of the execution time. However, its implementation and performance depends greatly on the Neighbour List (NL).

NL step creates the neighbour list to optimise the search for neighbours during particle interaction.

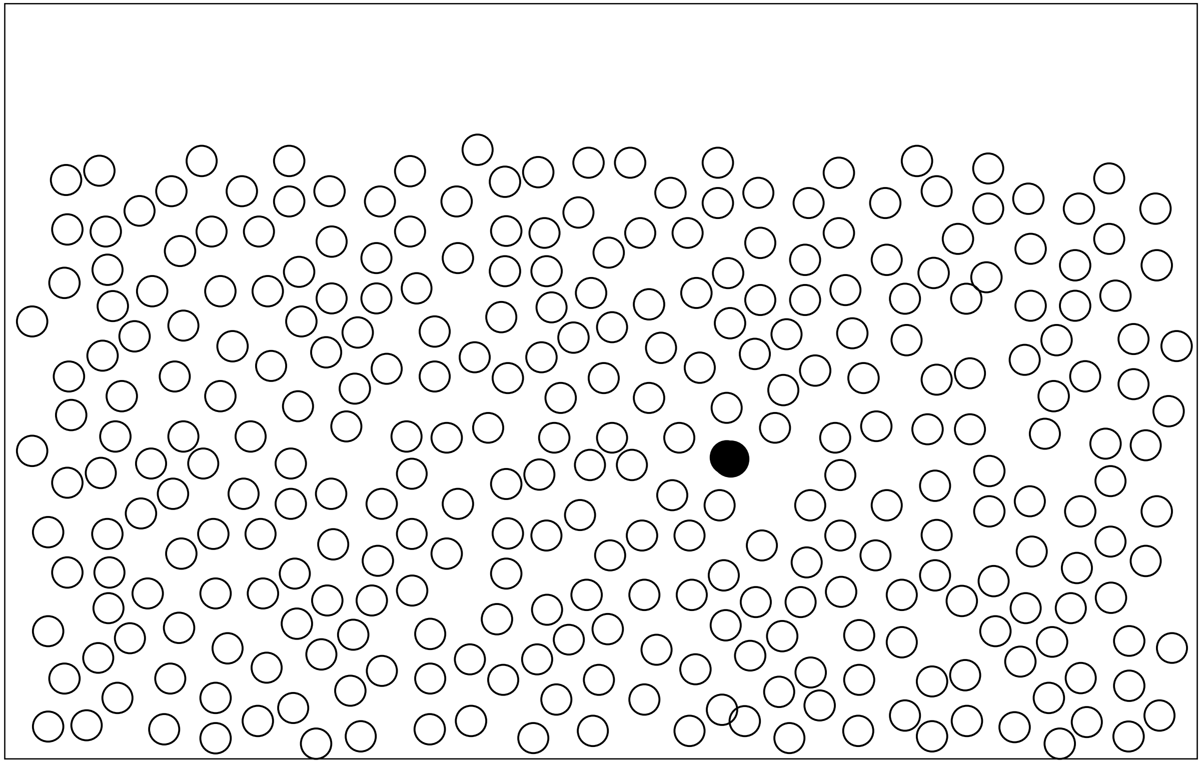
Two approaches were studied:

- **Cell-linked list (CLL)**
- **Verlet list (VL)**
  - **Classical Verlet List (VL_C)**
  - **Improved Verlet List (VL_X)**

2.2. Neighbour list approaches

Cell-linked List (CLL)

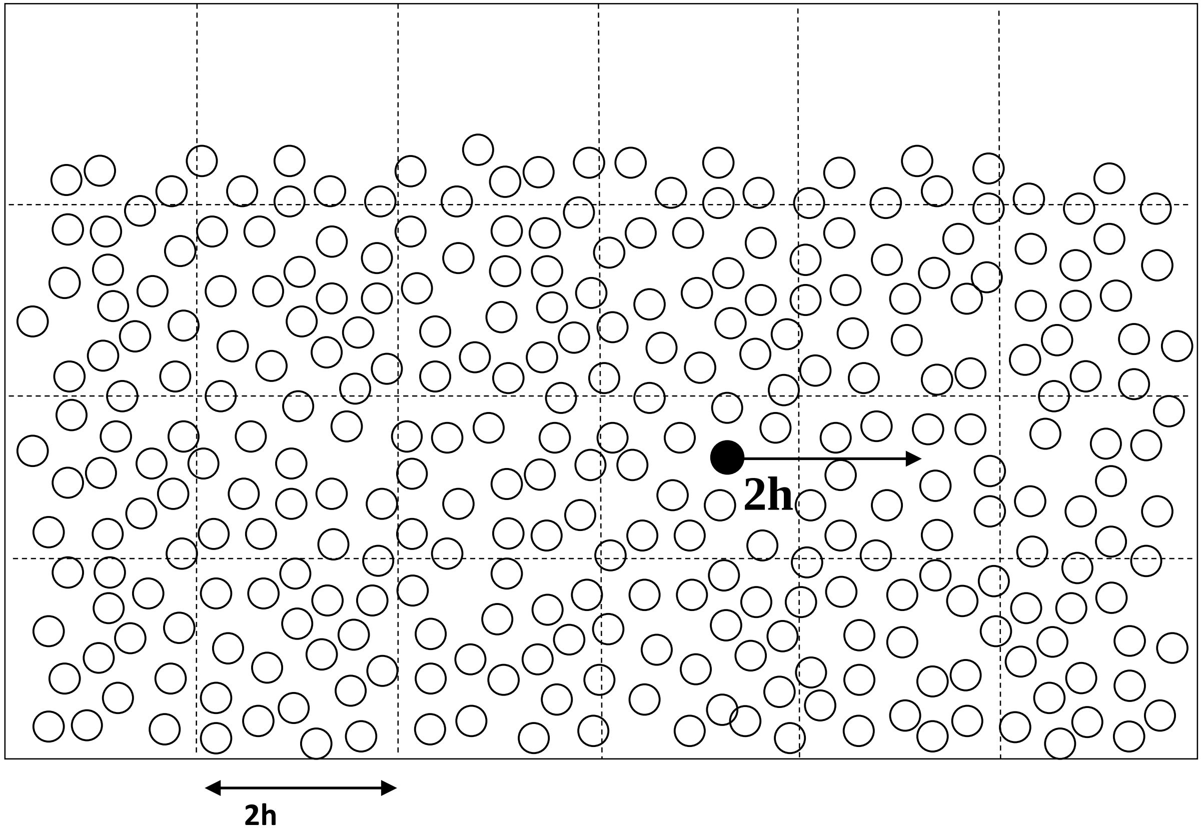
- The computational domain is divided in cells of side *2h* (cut-off limit).
- Particles are stored according to the cell they belong to.
- So each particle only looks for its potential neighbours in the adjacent cells.



2.2. Neighbour list approaches

Cell-linked List (CLL)

- The computational domain is divided in cells of side *2h* (cut-off limit).
- Particles are stored according to the cell they belong to.
- So each particle only looks for its potential neighbours in the adjacent cells.

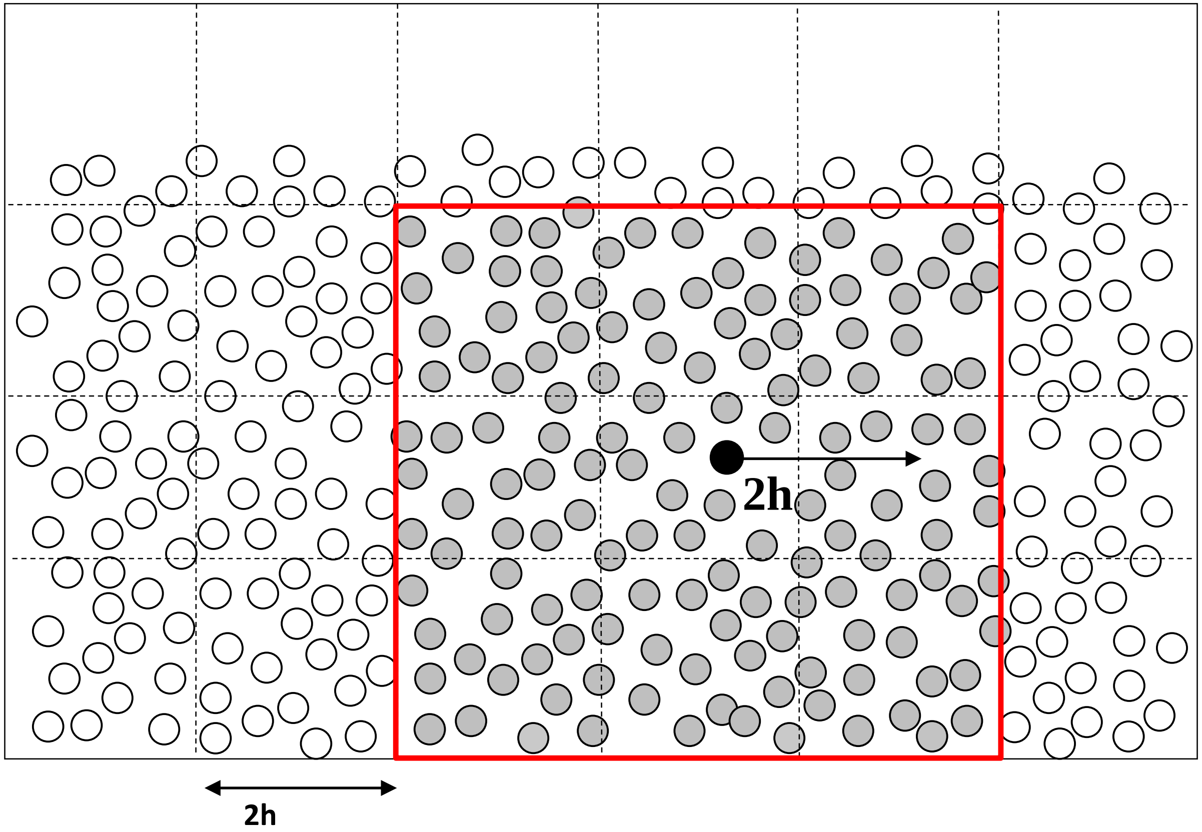


In this example:

2.2. Neighbour list approaches

Cell-linked List (CLL)

- The computational domain is divided in cells of side *2h* (cut-off limit).
- Particles are stored according to the cell they belong to.
- So each particle only looks for its potential neighbours in the adjacent cells.



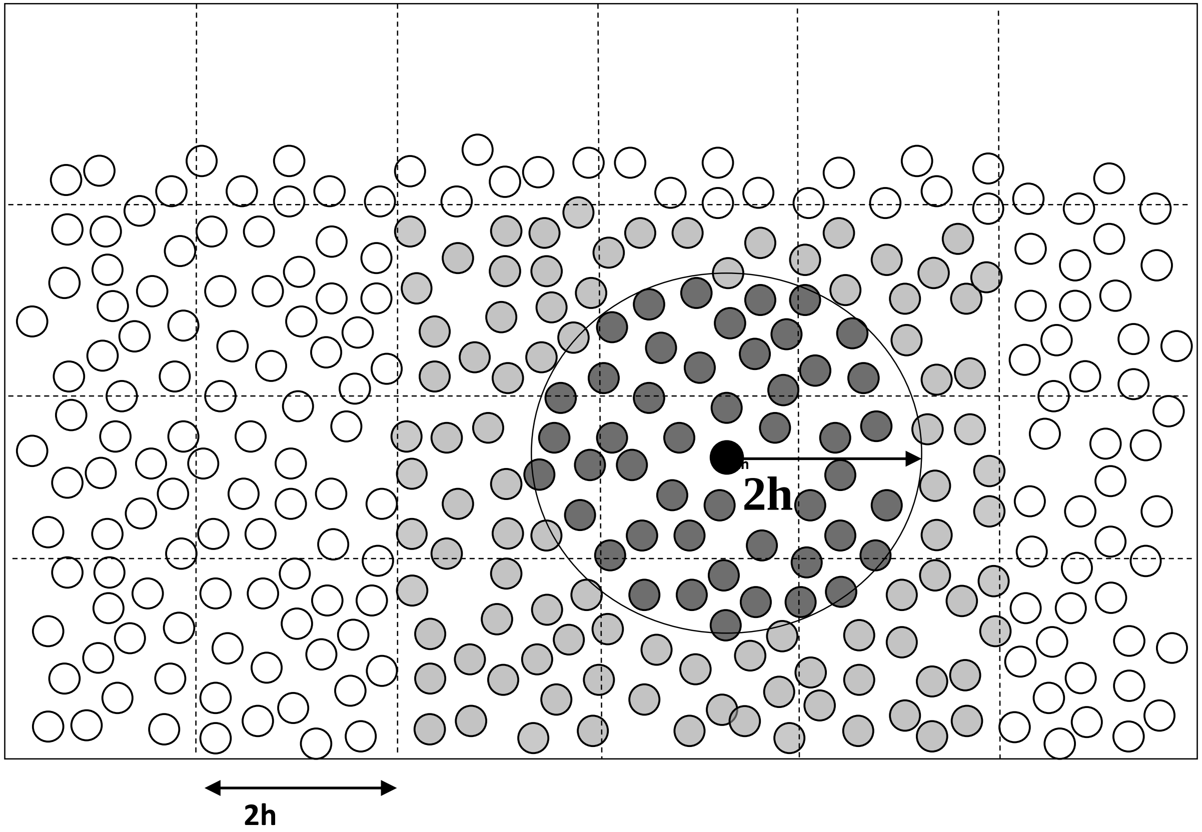
In this example:

141 Potential neighbours (gray particles)

2.2. Neighbour list approaches

Cell-linked List (CLL)

- The computational domain is divided in cells of side *2h* (cut-off limit).
- Particles are stored according to the cell they belong to.
- So each particle only looks for its potential neighbours in the adjacent cells.



In this example:

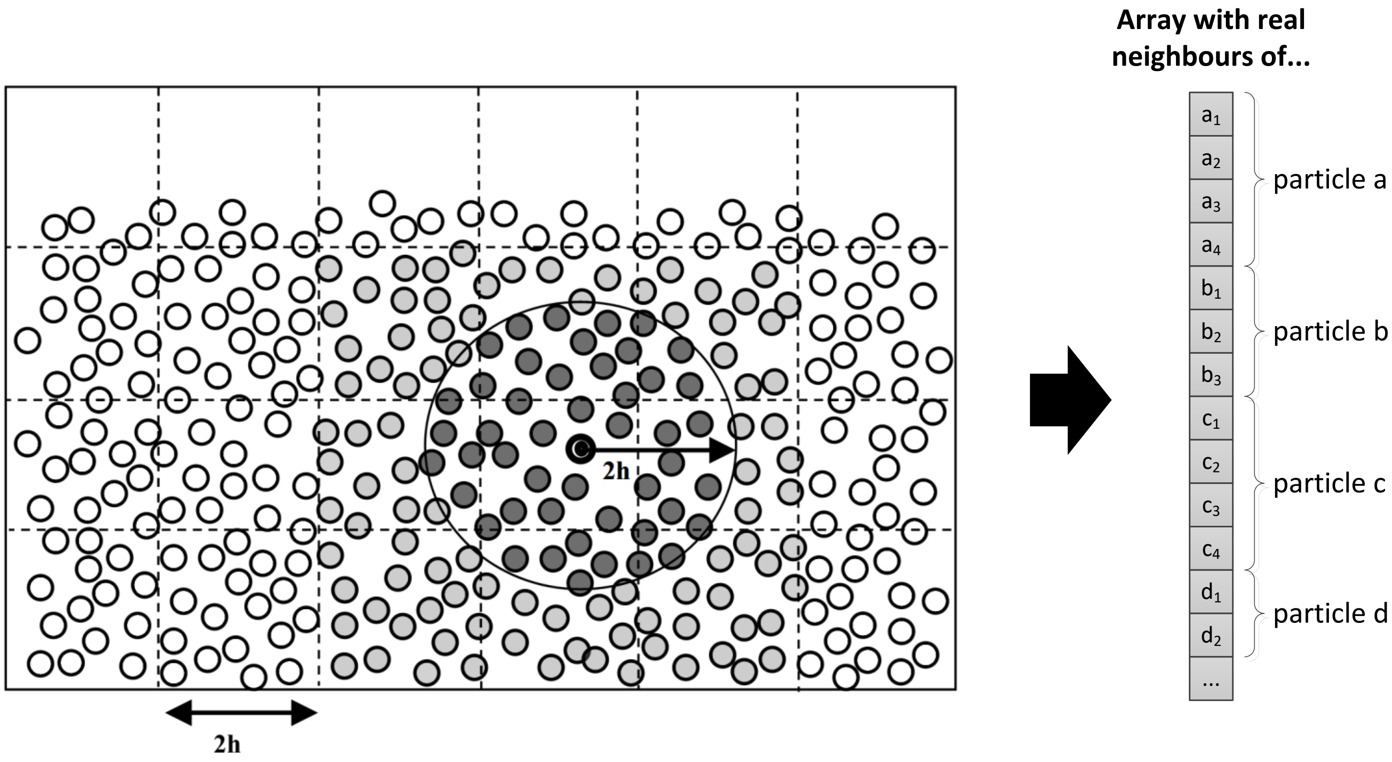
141 Potential neighbours
(gray particles)

47 real neighbours
(dark gray particles)

2.2. Neighbour list approaches

Verlet List

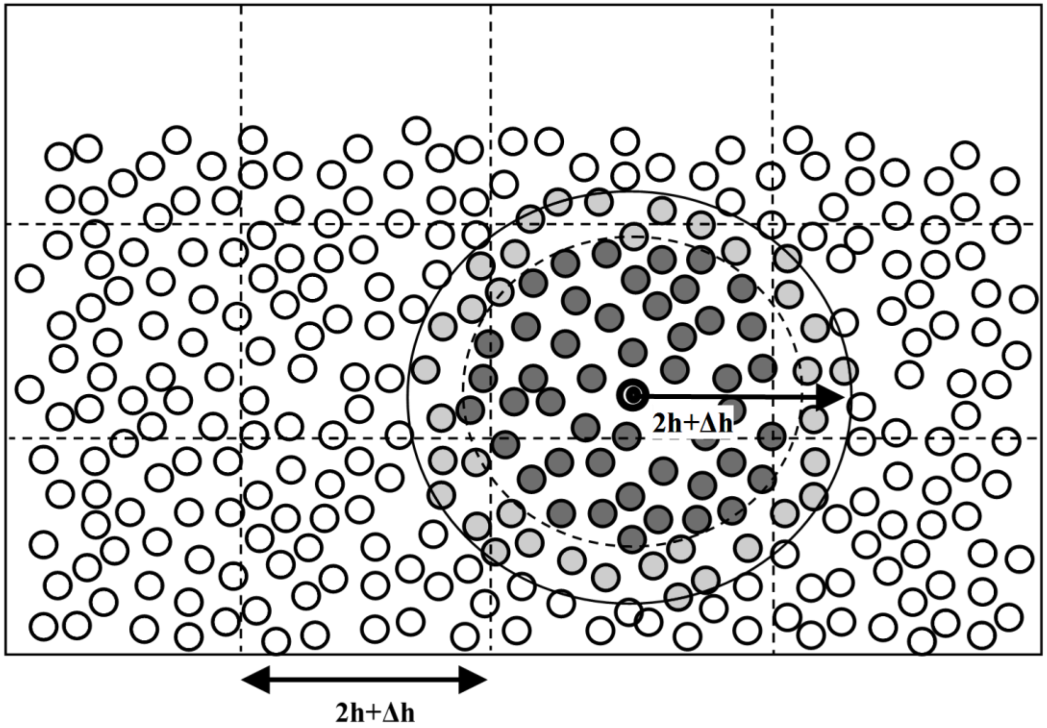
- *The computational domain is divided in cells of side 2h (cut-off limit).*
- *Particles are stored according to the cell they belong to.*
- *So each particle only looks for its potential neighbours in the adjacent cells.*
- **Array of real neighbours is created for each particle.**



2.2. Neighbour list approaches

Improved Verlet List (VL_X)

- Δh is calculated in the same way as in VL_C but the number of steps the list is kept (X instead of C) is only tentative.
- The constant v=1 (instead of 1.2) is used because no extra distance is necessary.
- **The same list can be used for several time steps.**



$$\Delta h = v(2 \cdot V_{max} \cdot C \cdot dt)$$

V_{max}**:** maximum velocity

C**:** time steps that list is fixed

dt**:** physical time for one time step

v**:** constant to remove inaccuracies in calculations

2.2. Neighbour list approaches

The best Neighbour List approach is…

	Computational runtime	Memory requirements
CLL (Cell-linked list)	**fast**	**minimum**
VL$_X$ (improved Verlet list)	the fastest (only 6% faster than CLL)	very heavy (30 times more than CLL)
VL$_C$ (classical Verlet list)	the slowest	the most inefficient

DualSPHysics is designed to simulate large number of particles. So that, **Cell-linked list is the best option** to be implemented since it provides the **best balance between the performance and the memory usage**.

Domínguez JM, Crespo AJC, Gómez-Gesteira M, Marongiu JC. 2011. Neighbour lists in Smoothed Particle Hydrodynamics. **International Journal For Numerical Methods in Fluids**, 67(12): 2026-2042. doi:10.1002/fld.2481.

Outline

1. Introduction
    1.1. Smooth Particle Hydrodynamics
    1.2. Why is SPH too slow?
    1.3. High Performance Computing (HPC)
    1.4. DualSPHysics project
2. DualSPHysics implementation
    2.1. Implementation in three steps
    2.2. Neighbour list approaches
3. **CPU acceleration**
4. GPU acceleration
    4.1. Parallelization problems in SPH
    4.2. GPU optimisations
5. Multi-GPU acceleration
    5.1. Dynamic load balancing
    5.2. Latest optimisations in Multi-GPU
    5.3. Large simulations
    5.4. Future improvements
6. New Multi-GPU approach

3. CPU acceleration

Previous ideas:

SPH is a Lagrangian model so particles are moving during simulation.

Each time step NL sorts particles (data arrays) to improve the memory access in PI stage since the access pattern is more regular and efficient.

Another advantage is the ease to identify the particles that belongs to a cell by using a range since the first particle of each cell is known.

Four optimizations have been applied to DualSPHysics:

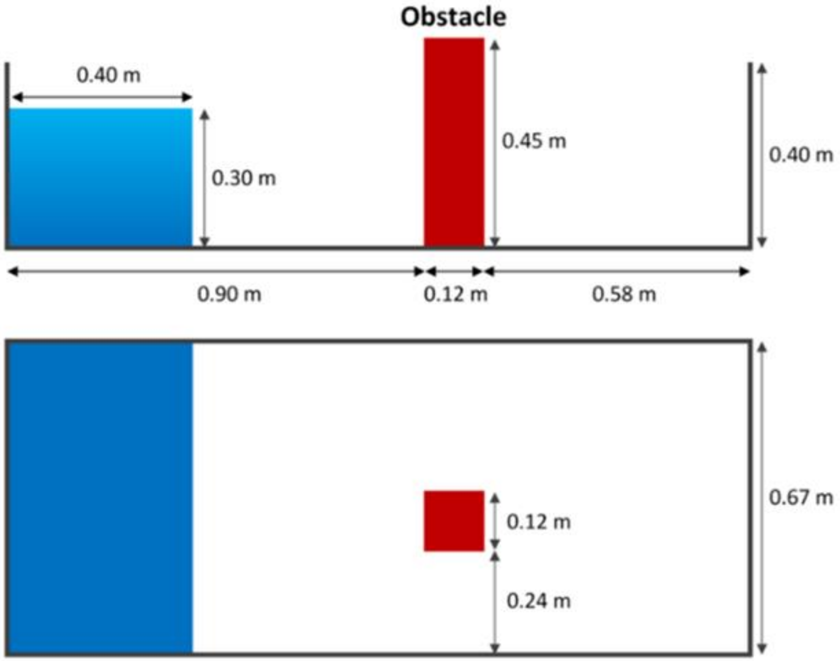
- Applying symmetry to particle-particle interaction.
- Splitting the domain into smaller cells.
- Using SSE instructions.
- Multi-core implementation using OpenMP.

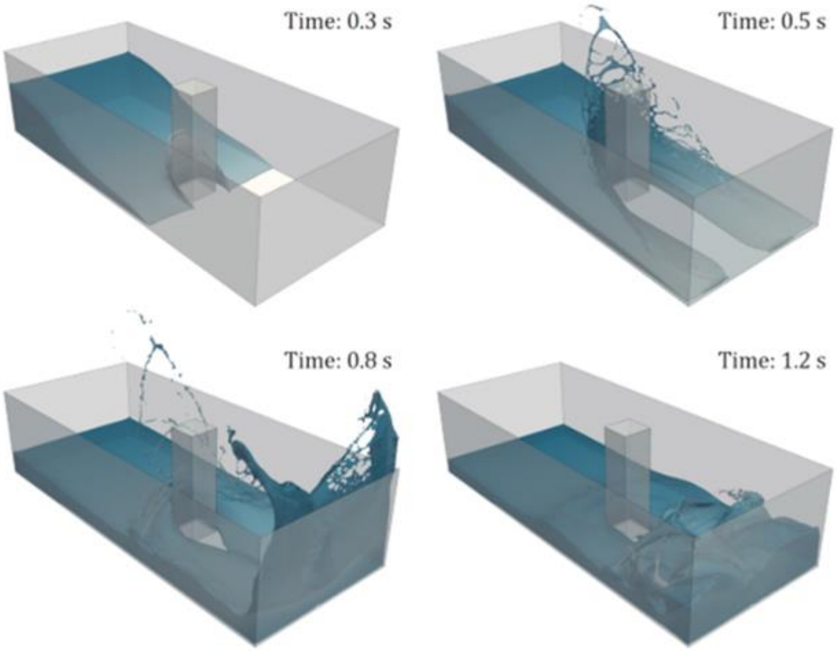
Domínguez JM, Crespo AJC and Gómez-Gesteira M. 2013. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. **Computer Physics Communications**, 184(3): 617-627

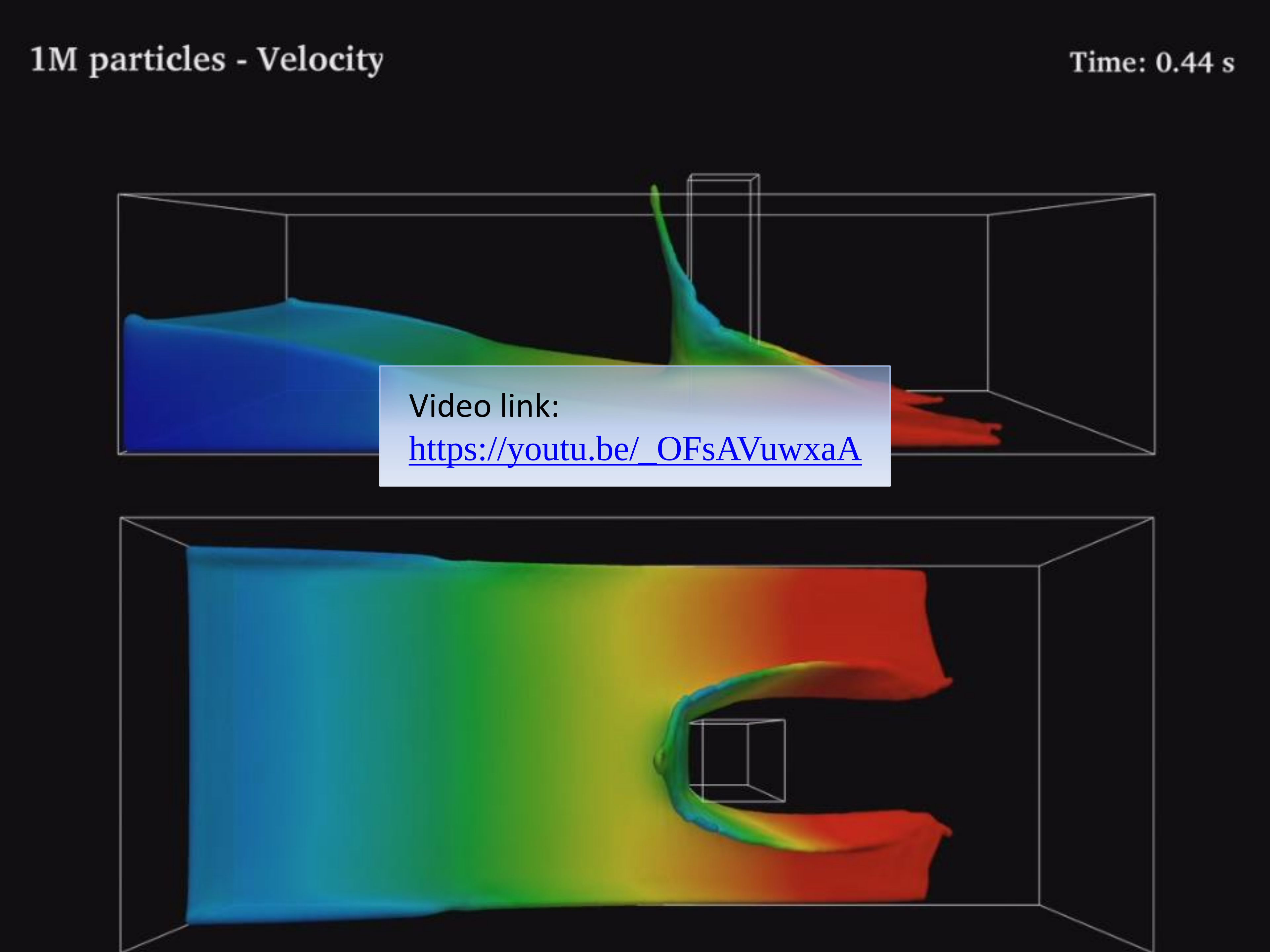
3. CPU acceleration

Testcase for results

- **Dam break flow impacting on a structure** (experiment of Yeh and Petroff at the University of Washington).
- Physical time of simulation is **1.5 seconds**.



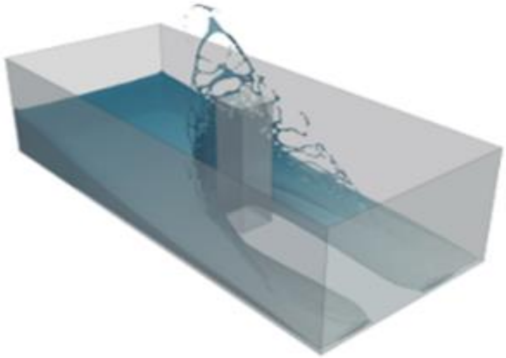




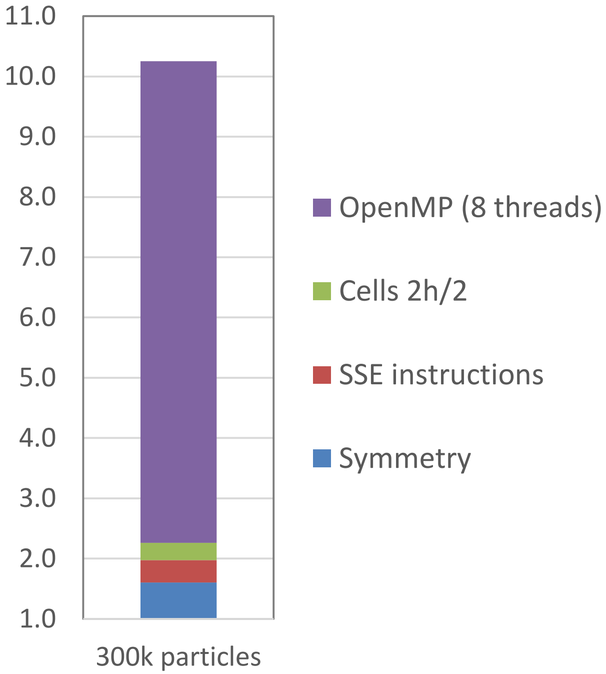
Video link:
https://youtu.be/_OFsAVuwxaA

3. CPU acceleration

Speedup for 300k particles applying all optimizations



Speedup 10.25x



Hardware and configuration for results

- **Hardware:** Intel® Core ™ i7 940 at 2.93 GHz (4 physical cores, 8 logical cores with Hyper-threading), with 6 GB of DDR3 RAM memory at 1333 MHz.
- **Operating system:** Ubuntu 10.10 64-bit.
- **Compiler:** GCC 4.4.5 (compiling with the option –O3).

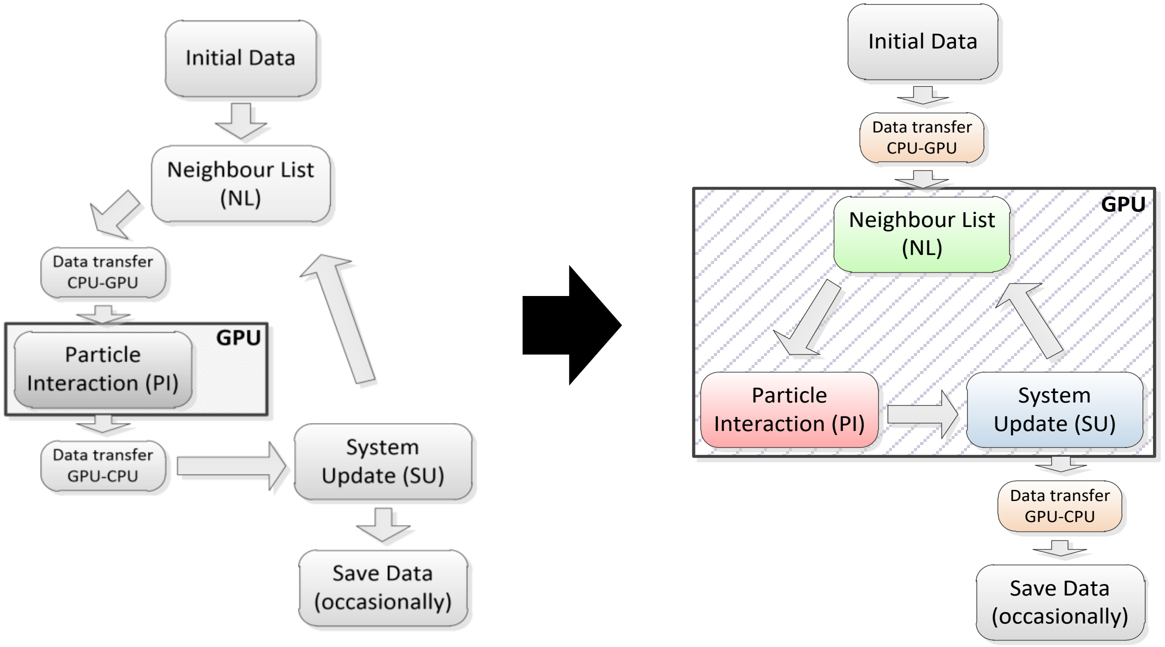
Outline

1. Introduction
    1.1. Smooth Particle Hydrodynamics
    1.2. Why is SPH too slow?
    1.3. High Performance Computing (HPC)
    1.4. DualSPHysics project
2. DualSPHysics implementation
    2.1. Implementation in three steps
    2.2. Neighbour list approaches
3. CPU acceleration
4. **GPU acceleration**
    4.1. Parallelization problems in SPH
    4.2. GPU optimisations
5. Multi-GPU acceleration
    5.1. Dynamic load balancing
    5.2. Latest optimisations in Multi-GPU
    5.3. Large simulations
    5.4. Future improvements
6. New Multi-GPU approach
7. DualSPHysics applications

4. GPU acceleration

Full GPU implementation

- DualSPHysics was implemented using the CUDA programming language to run SPH method on Nvidia GPUs.
- **GPU is used in all steps** (Neighbour List, Particle Interaction and System Update).
- This approach is the most efficient since:
  - All particle data is kept in GPU memory and the **transfers CPU-GPU are removed**.
  - **Neighbour List and System Update are parallelized**, obtaining a speedup also in this part of the code.



4. GPU acceleration

GPU implementation

DualSPHysics was implemented using the CUDA programming language to run SPH method on Nvidia GPUs.

Important: An efficient and full use of the capabilities of the GPUs is not straightforward. It is necessary to know and to take into account the details of the GPU architecture and the CUDA programming model.

Differences regarding the CPU implementation:

- Each GPU thread calculates the interaction between a target particle and its neighbours.
- The symmetry is not used in particle interaction because it cannot be applied efficiently on GPU.

4.1. Parallelization problems in SPH

Problems in Particle Interaction step

These problems appears since each thread has to interact with different neighbours.

- **Code divergence:**

  GPU threads are grouped into sets of 32 (*warps*) which execute the same operation simultaneously. When there are different operations in one warp these operations are executed sequentially, giving rise to a significant loss of efficiency.

- **No coalescent memory accesses:**

  The global memory of the GPU is accessed in blocks of 32, 64 or 128 bytes, so the number of accesses to satisfy a warp depends on how grouped data are. In SPH a regular memory access is not possible because the particles are moved each time step.

- **No balanced workload:**

  Warps are executed in *blocks* of threads. To execute a block some resources are assigned and they will not be available for other blocks till the end of the execution. In SPH the number of interactions is different for each particle so one thread can be under execution, keeping the assigned resources, while the rest of threads have finished.

4.2. GPU optimisations

Five optimizations have been applied to DualSPHysics to avoid or minimize the problems previously described.

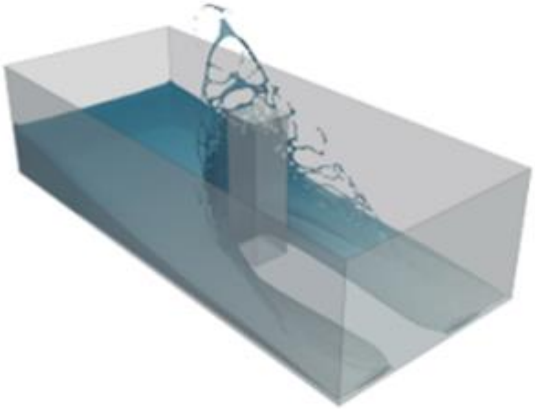
- Maximizing the occupancy of GPU.
- Reducing global memory accesses.
- Simplifying the neighbour search.
- Adding a more specific CUDA function of interaction.
- Division of the domain into smaller cells.

Domínguez JM, Crespo AJC and Gómez-Gesteira M. 2013. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. **Computer Physics Communications**, 184(3): 617-627

4.2. GPU optimisations

Testcase for results

- **Dam break** flow impacting on a structure.
- Simulating **1.5 seconds** of physical time.

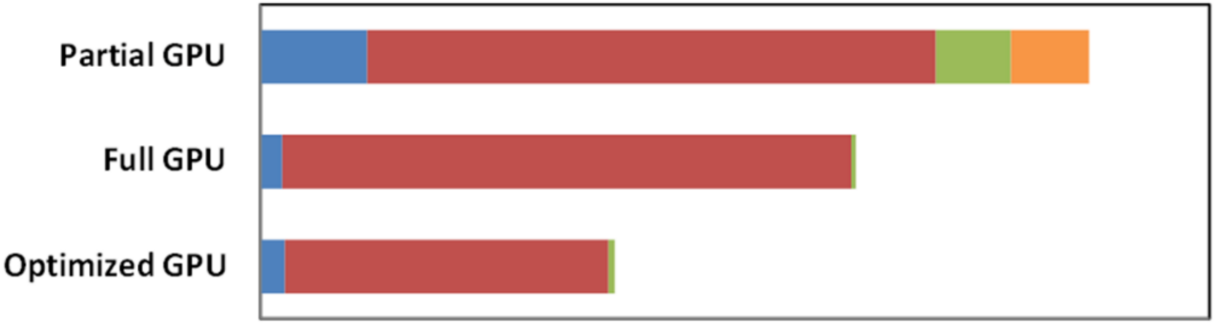


Hardware for results

	Number of cores	Processor clock	Memory space	Compute capability
Intel Xeon X5500	1-8	2.67 GHz		
Tesla 1060	240	1.30 GHz	4 GB	1.3
GTX 480	480	1.40 GHz	1.5 GB	2.0
GTX 680	1536	1.14 GHz	2 GB	3.0
Tesla K20	2496	0.71 GHz	5 GB	3.5
GTX Titan	2688	0.88 GHz	6 GB	3.5

4.2. GPU optimisations

Computational runtimes (in seconds) using GTX 480 for different GPU implementations (partial, full and optimized) when simulating 500,000 particles.



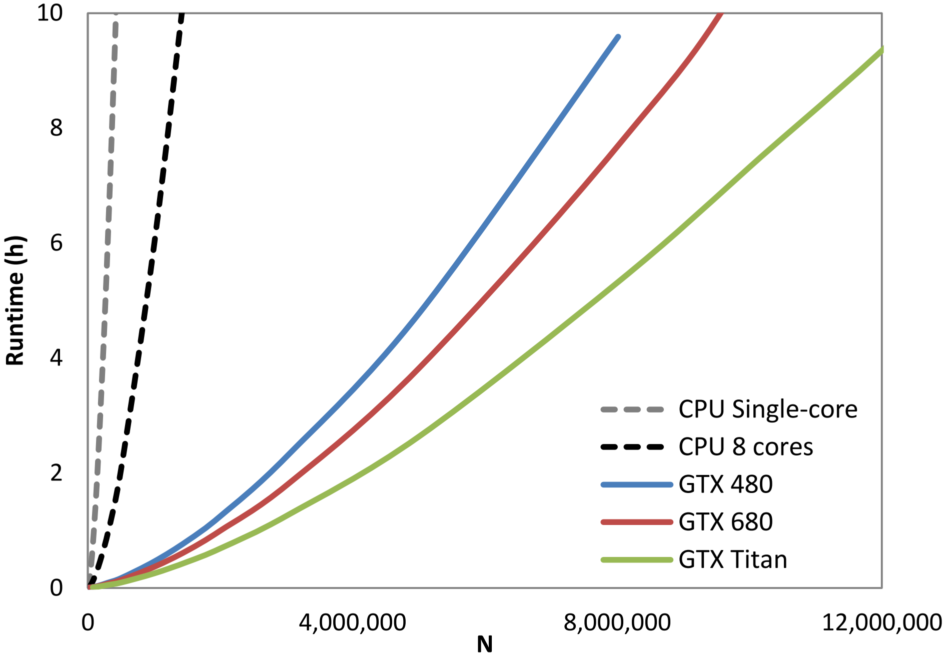
	Optimized GPU	Full GPU	Partial GPU
🟦 NL	64.89	58.30	281.62
🟥 PI	852.46	1498.50	1496.78
🟩 SU	16.78	11.44	198.92
🟧 Data transfer			204.97

Full GPU is **1.26x** faster than Partial GPU.

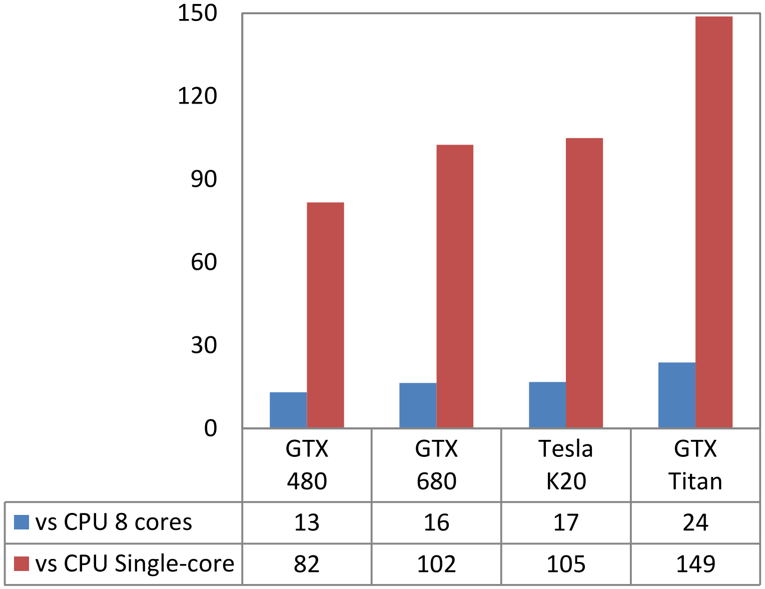
Optimized GPU is **2.12x** faster than Partial GPU.

4.2. GPU optimisations

Runtime for CPU and different GPU cards.



Speedups of GPU against CPU simulating 1 million particles.



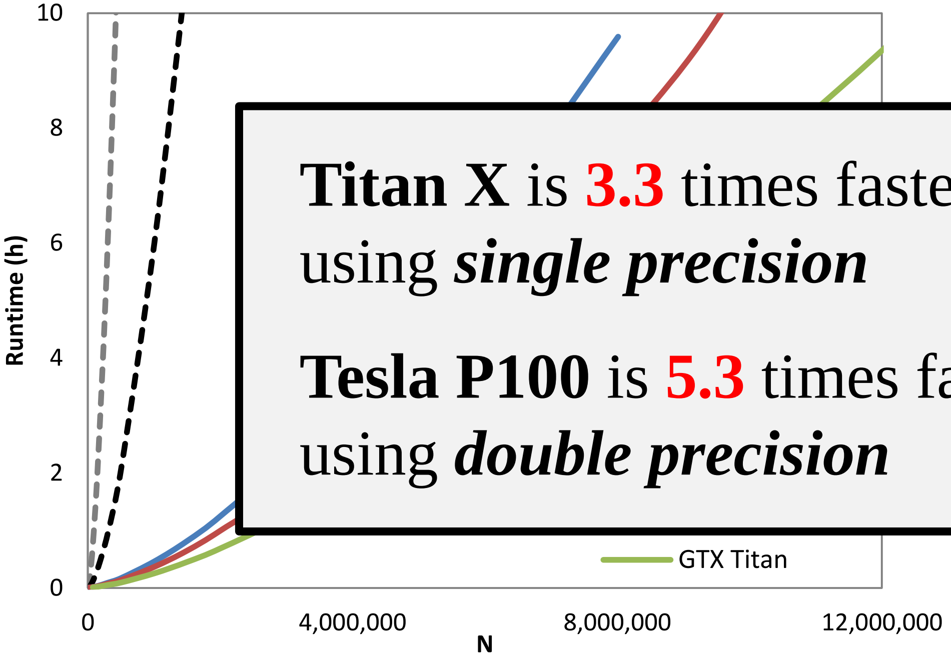
	GTX 480	GTX 680	Tesla K20	GTX Titan
🟦 vs CPU 8 cores	13	16	17	24
🟥 vs CPU Single-core	82	102	105	149

After optimising the performance of DualSPHysics on CPU and GPU...

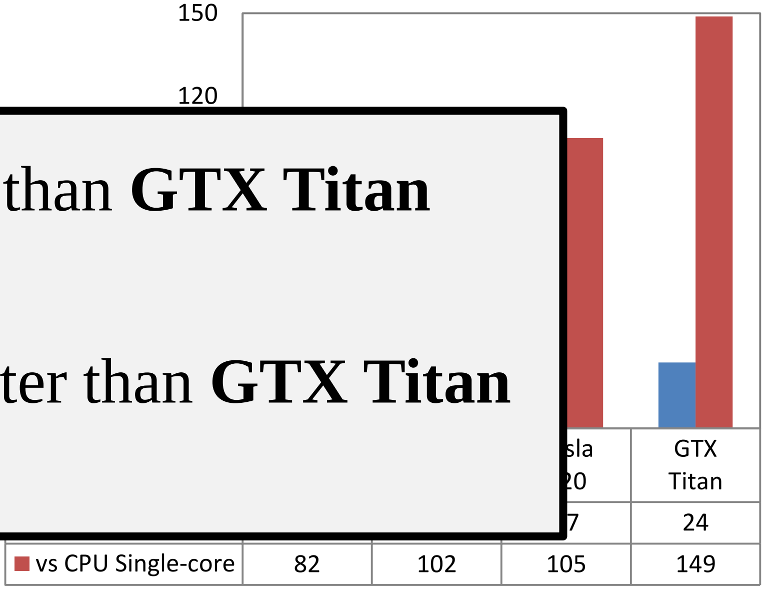
The most powerful GPU (**GTX Titan**) is **149 times faster** than CPU (single core execution) and **24 times faster** than the CPU using all 8 cores.

4.2. GPU optimisations

Runtime for CPU and different GPU cards.



Speedups of GPU against CPU simulating 1 million particles.



Titan X is 3.3 times faster than **GTX Titan** using ***single precision***

Tesla P100 is 5.3 times faster than **GTX Titan** using ***double precision***

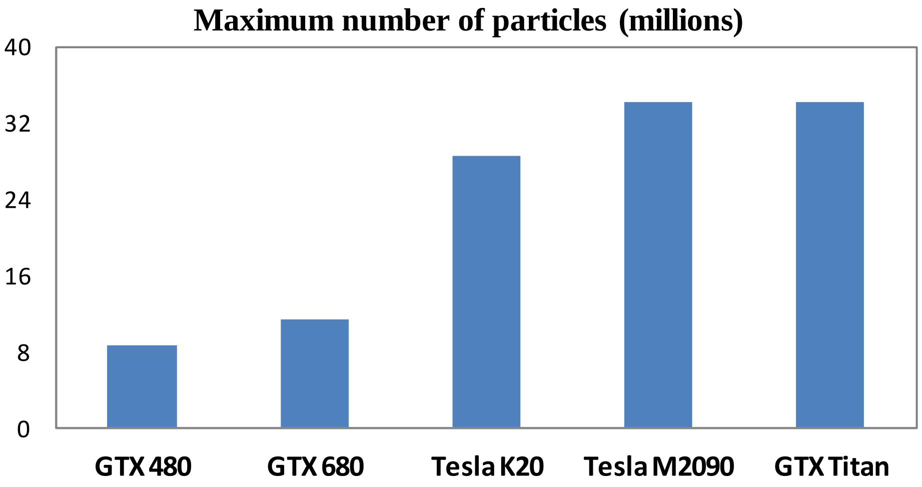
After optimising the performance of DualSPHysics on CPU and GPU...

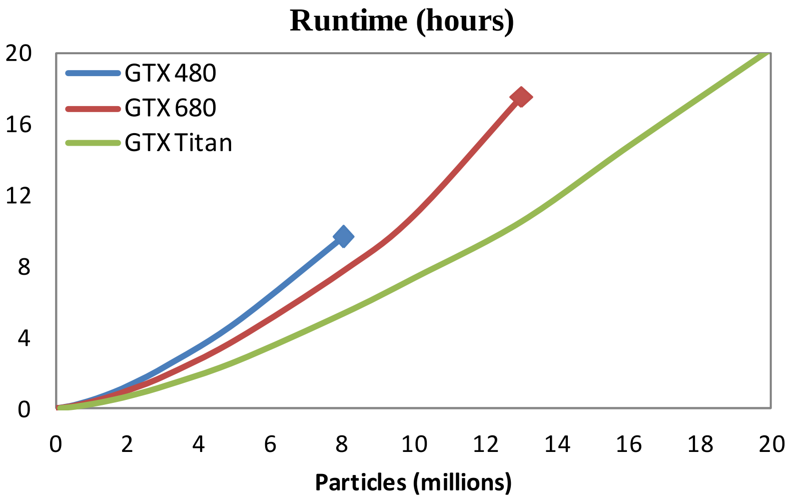
The most powerful GPU (**GTX Titan**) is **149 times faster** than CPU (single core execution) and **24 times faster** than the CPU using all 8 cores.

4.2. GPU optimisations

The simulation of **real cases implies huge domains with a high resolution**, which implies simulating tens or hundreds of million particles.

The use of one GPU presents important **limitations**:
- Maximum number of particles depends on the memory size of GPU.
- Time of execution increases rapidly with the number of particles.

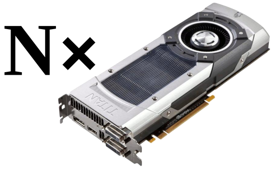




Outline

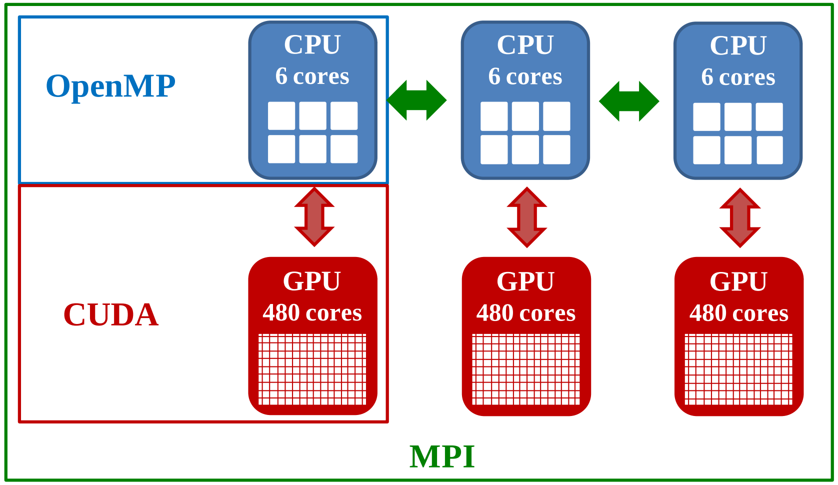
1. Introduction
   1.1. Smooth Particle Hydrodynamics
   1.2. Why is SPH too slow?
   1.3. High Performance Computing (HPC)
   1.4. DualSPHysics project
2. DualSPHysics implementation
   2.1. Implementation in three steps
   2.2. Neighbour list approaches
3. CPU acceleration
4. GPU acceleration
   4.1. Parallelization problems in SPH
   4.2. GPU optimisations
5. **Multi-GPU acceleration**
   5.1. Dynamic load balancing
   5.2. Latest optimisations in Multi-GPU
   5.3. Large simulations
   5.4. Future improvements
6. New Multi-GPU approach

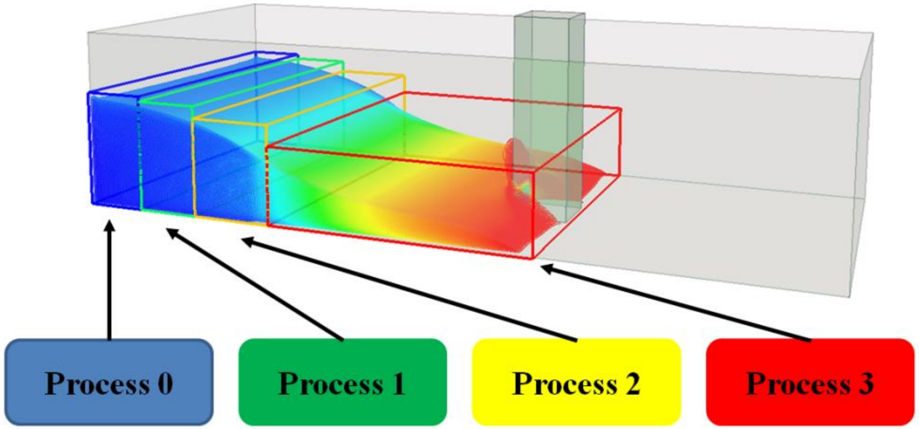
5. Multi-GPU implementation



MPI is used to combine resources of multiple machines connected via network.

The **physical domain** of the simulation **is divided among the different MPI processes**. Each process only needs to assign resources to manage a subset of the total amount of particles for each subdomain.





5. Multi-GPU implementation

The use of MPI implies an **overcost**:

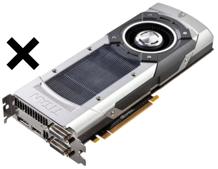
- **Communication**: Time dedicated to the interchange of data between processes.
- **Synchronization**: All processes must wait for the slowest one.

Solutions:

- **Overlapping** between force computation and communications: while data is transferred between processes, each process can compute the force interactions among its own particles. In the case of GPU, the CPU-GPU transfers can also be overlapped with computation using *streams* and *pinned memory*.
- **Load balancing**. A dynamic load balancing is applied to minimise the difference between the execution times of each process.

5. Multi-GPU implementation

N×



Dynamic load balancing

Due to the nature Lagrangian of the SPH method, is necessary to balance the load throughout the simulation.

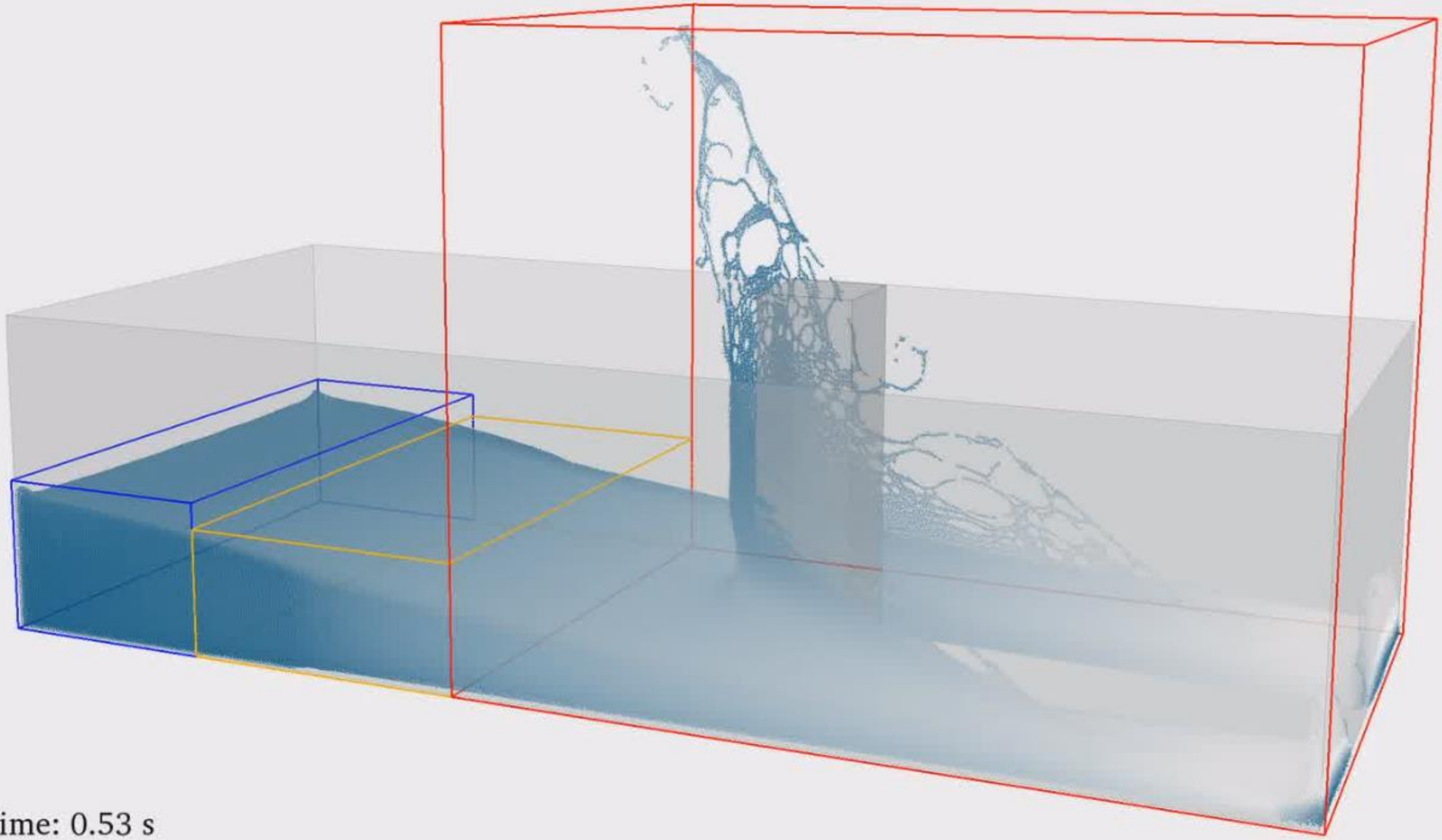
FIRST approach according to the **number of fluid particles**

The number of particles must be redistributed after some time steps to get the workload balanced among the processes and minimise the synchronisation time.

SECOND approach according to the **required computation time** of each device

Enables the adaptation of the code to the features of a heterogeneous cluster achieving a better performance.

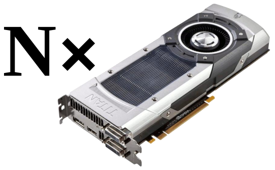
GPUs: 3 x GTX480
MPI: Dynamic Balancing-Np
Particles: 6 Millions
Steps: 42,624
Runtime: 2.6 hours



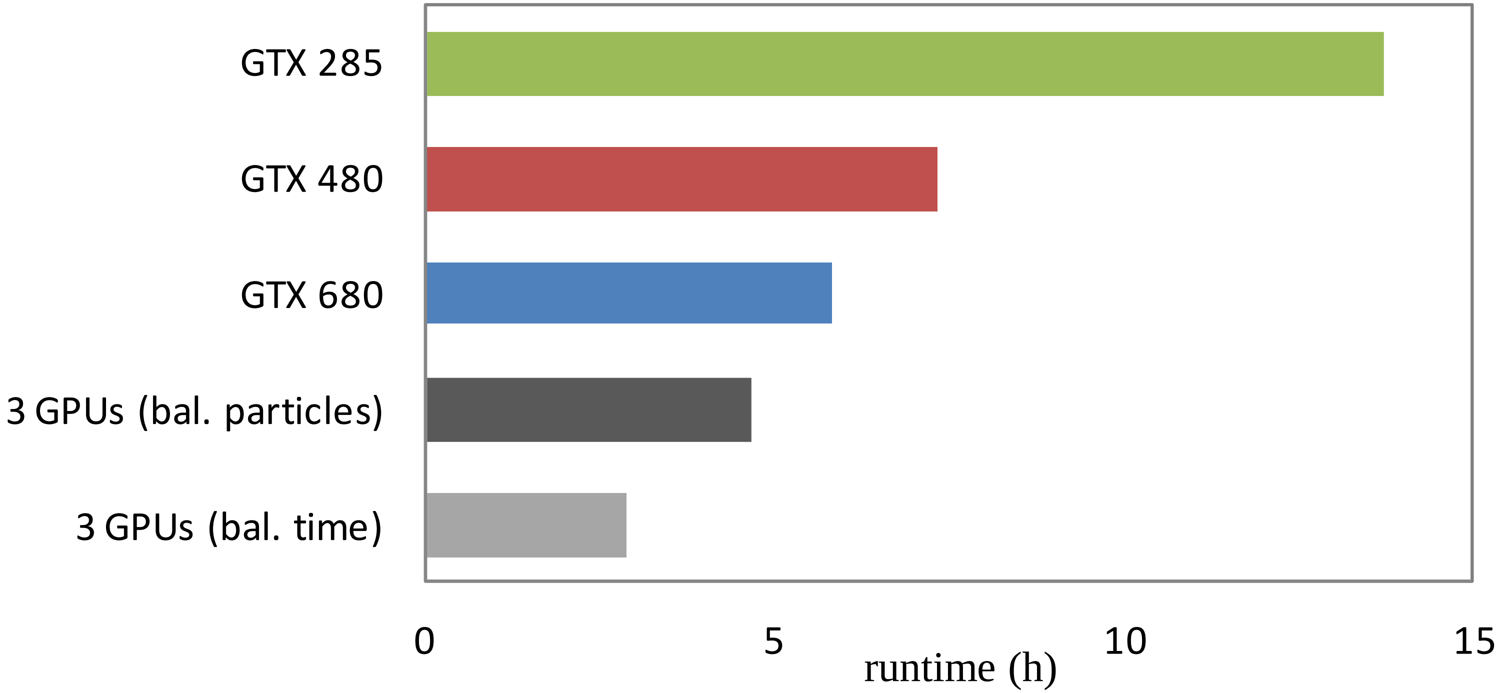
Time: 0.53 s

5.1. Dynamic load balancing

N×

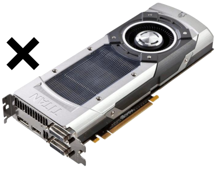


Results using one GPU and several GPUs with dynamic load balancing



Hardware: GTX 680 GTX 480 GTX 285

5.1. Dynamic load balancing

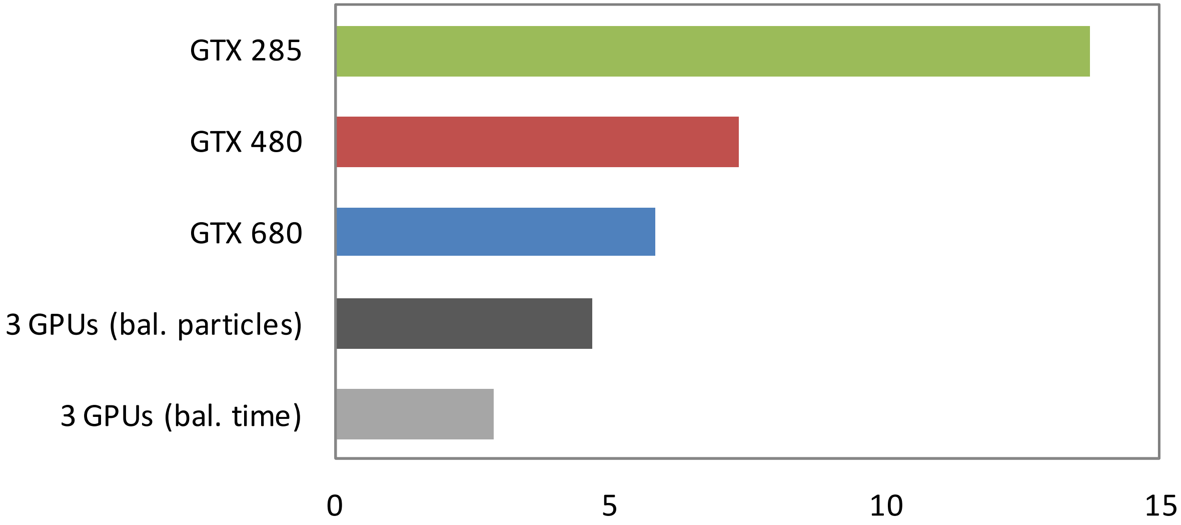
N× 

Results using one GPU and several GPUs with dynamic load balancing

- Using the fastest GPU (GTX 680) **5.8 hours**
- Using three different GPUs

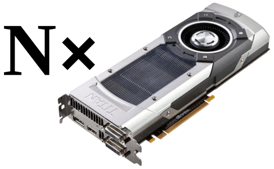
  According to the number of fluid particles **4.6 hours**
  According to the required computation time **2.8 hours**

The second approach is 1.7x faster than first approach and 2.1x faster than one GPU.



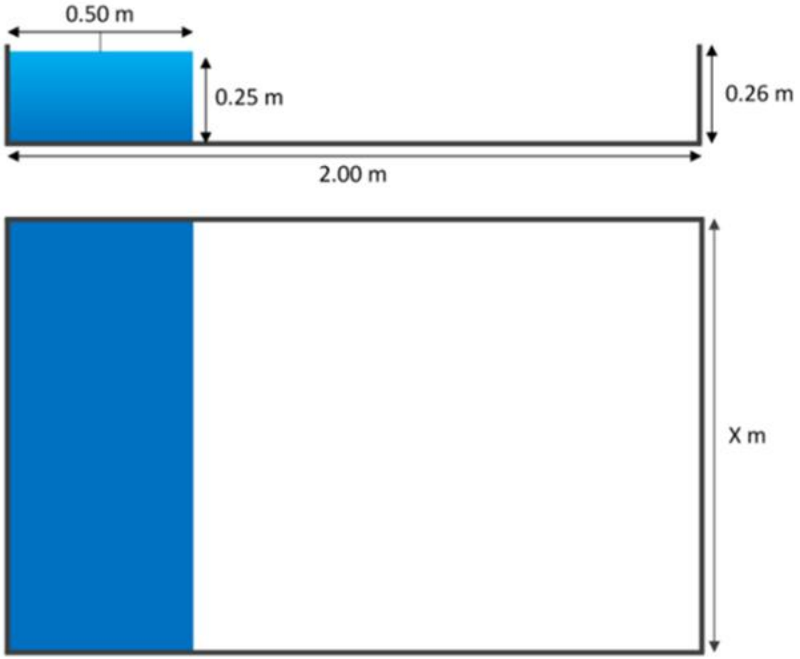
5.2. Latest optimisations in Multi-GPU

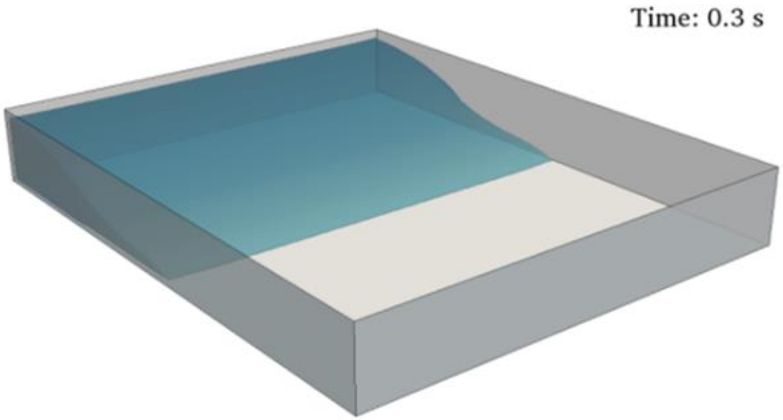
N×



Testcase for results

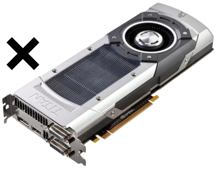
- **Dam break flow**.
- Physical time of simulation is **0.6 seconds**.
- The number of used particles varies from 1M to 1,024M particles.





5.2. Latest optimisations in Multi-GPU

N×



Results of efficiency

The simulations were carried out in the **Barcelona Supercomputing Center** BSC-CNS (Spain). This system is built with **256 GPUs Tesla M2090**.

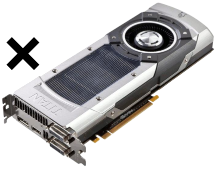
All the results presented here were obtained single precision and Error-correcting code memory (ECC) disabled.

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Activity at BARCELONA SUPERCOMPUTING CENTER:
"Massively parallel Smoothed Particle Hydrodynamics scheme using GPU clusters"

5.2. Latest optimisations in Multi-GPU

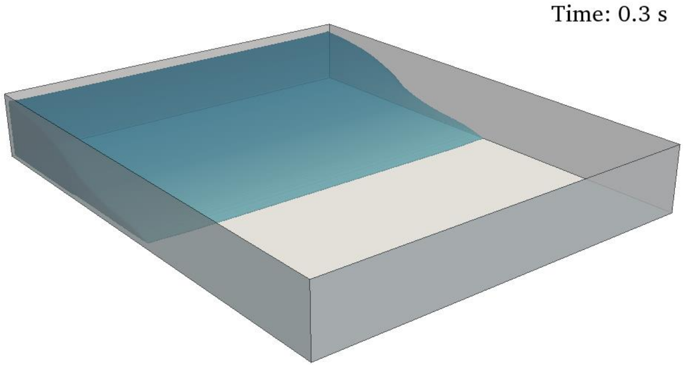
N×

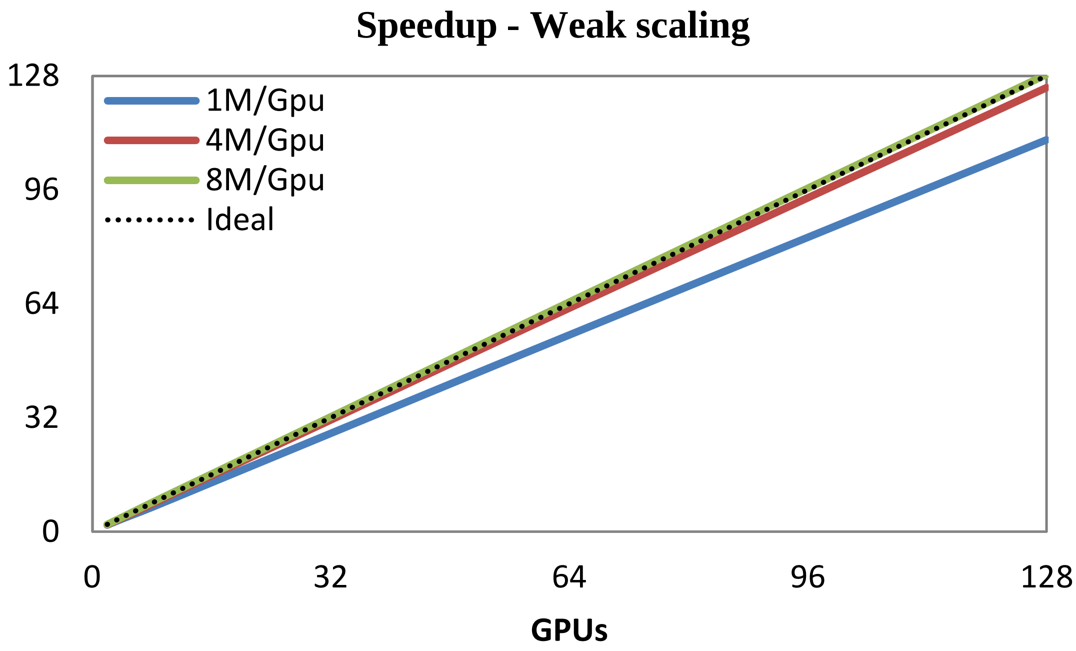


Efficiency close to 100% simulating 4M/GPU with 128 GPUs Tesla M2090 of BSC.

This is possible because the time dedicated to tasks exclusive of the multi-GPU executions (communication between processes, CPU-GPU transfers and load balancing) is minimum.

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación



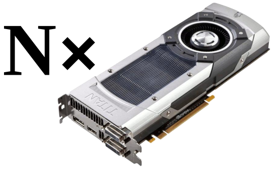


$$S(N)=\frac{T(N_{ref})\cdot N}{T(N)\cdot N_{ref}}$$

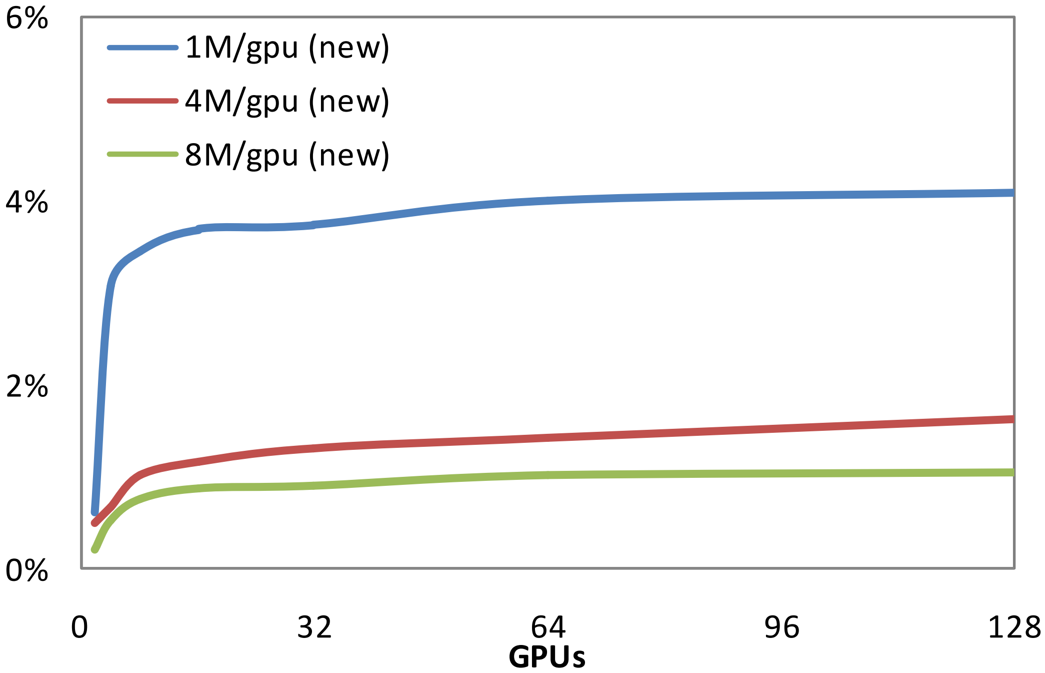
$$E(N)=\frac{S(N)}{N}$$

5.2. Latest optimisations in Multi-GPU

N×



Percentage of time dedicated to tasks exclusive of the multi-GPU executions (including the latest improvements).

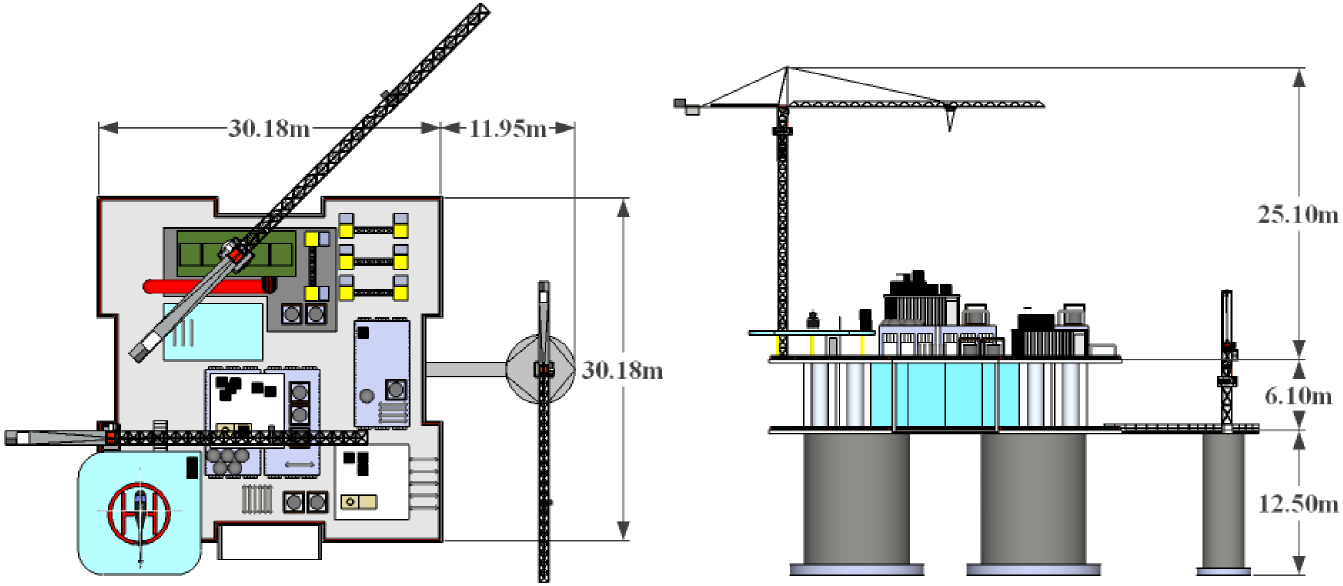


5.3. Large simulations

64×

Simulation of 1 billion SPH particles

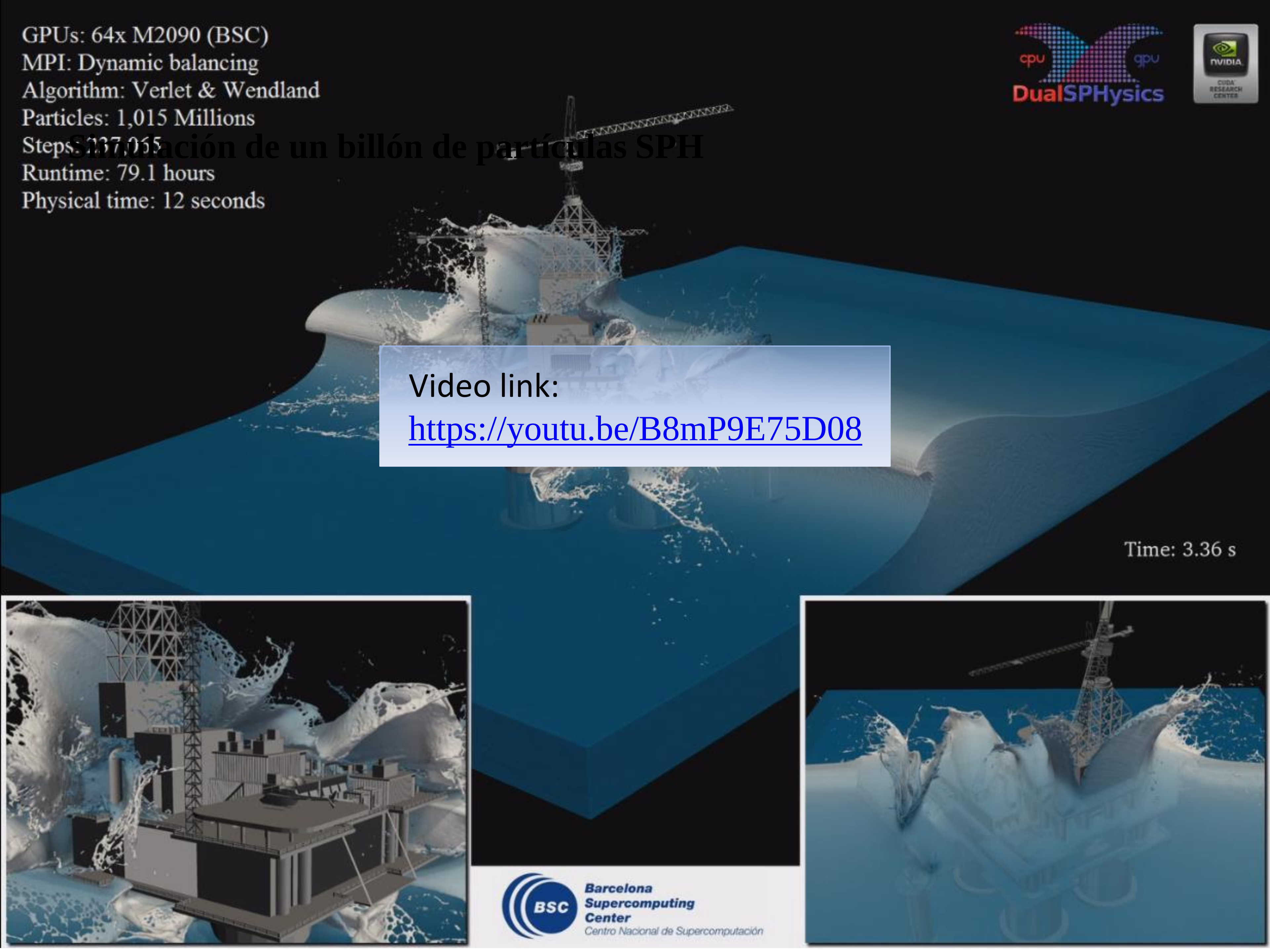
Large wave interaction with oil rig using 10^9 particles



dp= 6 cm, h= 9 cm
np = 1,015,896,172 particles
nf = 1,004,375,142 fluid particles
physical time= 12 sec
# of steps = 237,065 steps
runtime = 79.1 hours

using 64 GPUs Tesla M2090 of the BSC-CNS

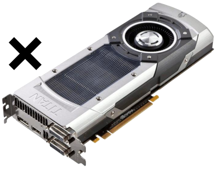
Simulación de un billón de partículas SPH


GPUs: 64x M2090 (BSC)
MPI: Dynamic balancing
Algorithm: Verlet & Wendland
Particles: 1,015 Millions
Steps: 237,065
Runtime: 79.1 hours
Physical time: 12 seconds
cpu
gpu
DualSPHysics
NVIDIA
CUDA RESEARCH CENTER
Time: 3.36 s
BSC
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Video link:
https://youtu.be/B8mP9E75D08

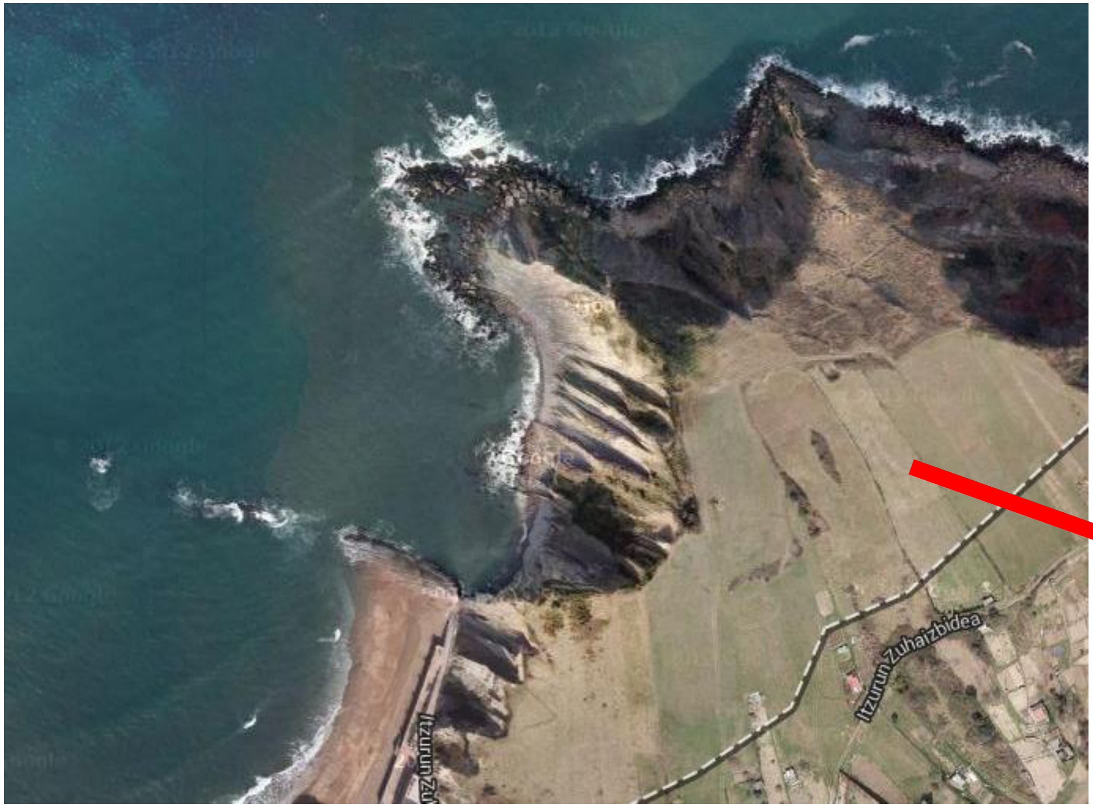
5.3. Large simulations

32×



Simulation of a real case

Using 3D geometry of the beach Itzurun in Zumaia-Guipúzcoa (Spain) in Google Earth



32 x M2090 (BSC)

Particles: **265 Millions**

Physical time: **60 seconds**

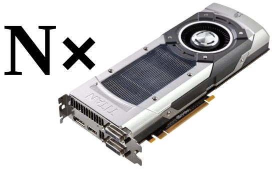
Steps: 218,211

Runtime: **246.3 hours**

France
Chemillé
Poitiers
La Rochelle
Île de Sein
Limoges
Bay of Biscay
Bordeaux
Agen
Toulouse
Montpell
Perpignan
Andorra
A Coruña
Carballo
Santiago de Compostela
Lugo
Gijón/Xixón
Oviedo
Llanes
Santander
Bilbao
A
Donostia
León
Vitoria-Gasteiz
Pontevedra
Ourense
Vigo
Burgos
Valladolid
Zaragoza
Lleida
Garrotxa
Sabadell
Reus
Barcelona
Braga
Porto
Salamanca
Ávila
Madrid
Palancares y Tierra Muerta
Coimbra
Portugal
España (Spain)
Albacete
Valencia
Balearic Sea
Palma
Ciutadella

Video links:
https://youtu.be/nDKlrRA_hEA
https://youtu.be/kWS6-0Z_jIo

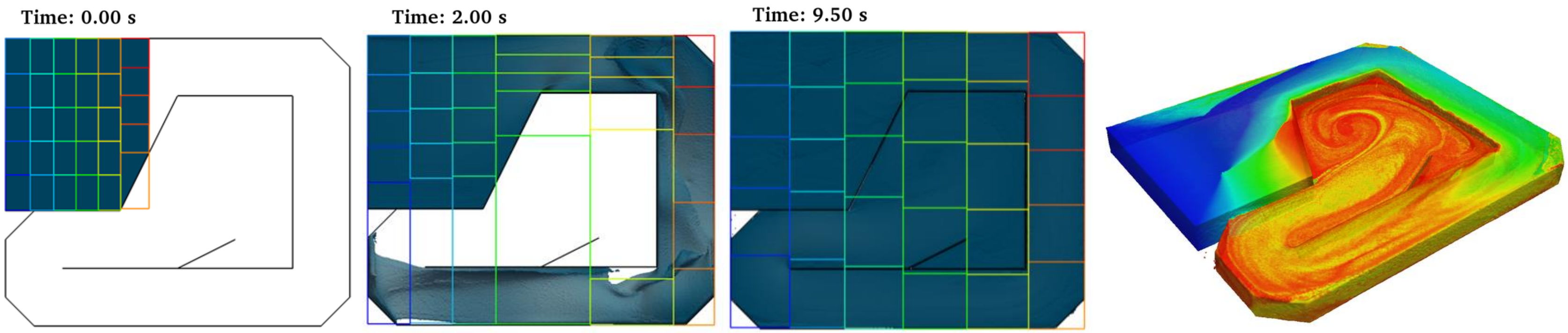
5.4. Future improvements



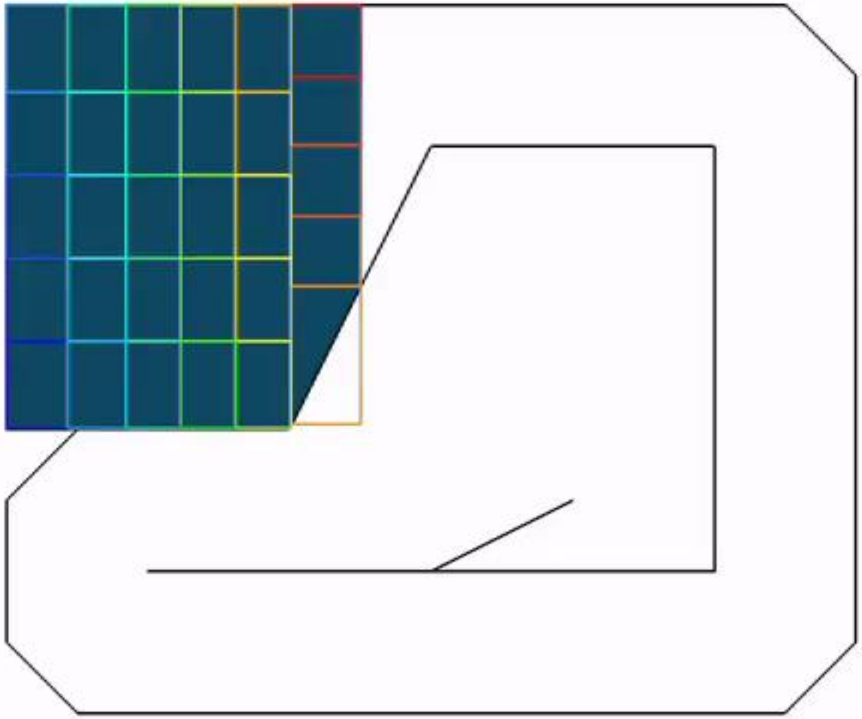
Decomposition in 2D and 3D

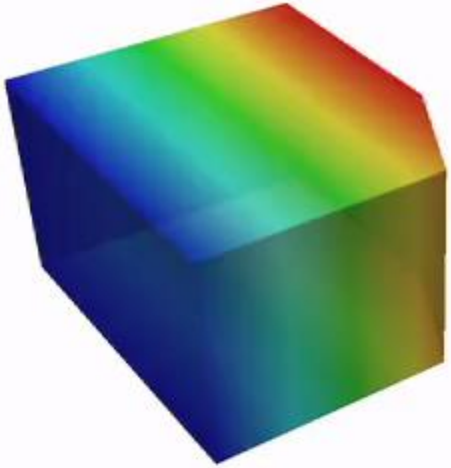
- Now only 1D but 2D and 3D will be implemented in the future.
- 1D approach is correct when the domain of simulation is very narrow but this approach is not well adapted to other domains.
- A 2D and 3D decomposition is necessary for a better distribution of the work load when using hundreds of GPUs.

Example of the 2D decomposition we are working on



2D decomposition





Time: 0.00 s

Outline

1. Introduction
    1.1. Smooth Particle Hydrodynamics
    1.2. Why is SPH too slow?
    1.3. High Performance Computing (HPC)
    1.4. DualSPHysics project
2. DualSPHysics implementation
    2.1. Implementation in three steps
    2.2. Neighbour list approaches
3. CPU acceleration
4. GPU acceleration
    4.1. Parallelization problems in SPH
    4.2. GPU optimisations
5. Multi-GPU acceleration
    5.1. Dynamic load balancing
    5.2. Latest optimisations in Multi-GPU
    5.3. Large simulations
    5.4. Future improvements
6. **New Multi-GPU approach**

6. New Multi-GPU approach

Consumers can now easily purchase desktop machines or a single compute node with 4-8 GPUs for only a few thousand euros.

New Multi-GPU code optimized for Multi-GPU machines

CUDA (and OpenMP), not MPI

Only for several GPUs in the same machine
Typical clusters have 2, 4 or 8 GPUs

Simulations with 100-200M particles
120M using 4x GTX Titan (6GB)

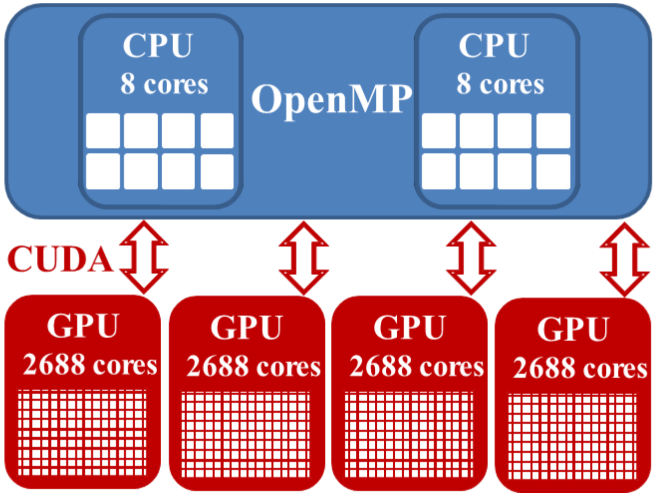
6. New Multi-GPU approach

Consumers can now easily purchase desktop machines or a single compute node with 4-8 GPUs for only a few thousand euros.

For example,

at Universidade de Vigo

2 x CPU:
Intel Xeon E5-2640 v2
8 cores at 2GHz
4 x GPU:
Nvidia Titan
2688 cores at 837Mhz
6GB GDDR5



6. New Multi-GPU approach

Consumers can now easily purchase desktop machines or a single compute node with 4-8 GPUs for only a few thousand euros.

⬇

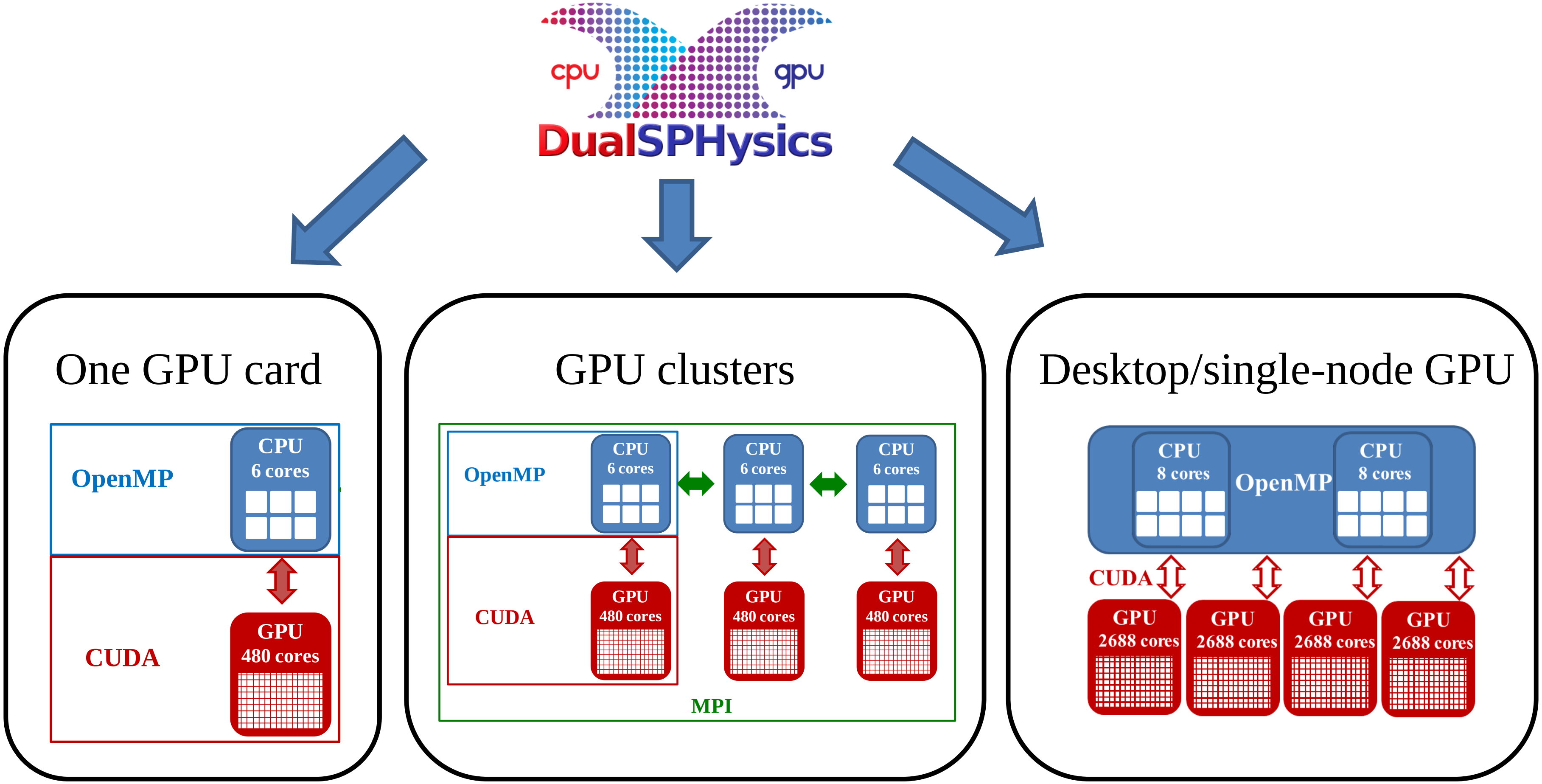
New Multi-GPU code optimized for Multi-GPU machines

CUDA (and OpenMP), not MPI

Only for several GPUs in the same machine
Typical clusters have 2, 4 or 8 GPUs

Simulations with 100-200M particles
120M using 4x GTX Titan (6GB)

6. New Multi-GPU approach



cpu gpu DualSPHysics

One GPU card

OpenMP

CPU 6 cores

CUDA

GPU 480 cores

GPU clusters

OpenMP

CPU 6 cores

CPU 6 cores

CPU 6 cores

CUDA

GPU 480 cores

GPU 480 cores

GPU 480 cores

MPI

Desktop/single-node GPU

CPU 8 cores

OpenMP

CPU 8 cores

CUDA

GPU 2688 cores

GPU 2688 cores

GPU 2688 cores

GPU 2688 cores

2011-today

Release of
open-source code

since 2012

Simulations on
Supercomputing
Centers (BSC)

2018

To be released as
open-source code

6. New Multi-GPU approach

New Multi-GPU code optimized for Multi-GPU machines

Advantages:

- More portable and easy to use in Linux and Windows
- Simpler code and easier to modify
- More efficient communication. MPI overhead was removed.
- Not special pre-processing and post-processing tools
- Updated code. It will include all capabilities in the last version of DualSPHysics

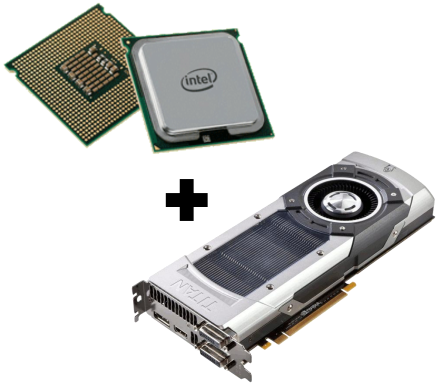
Drawbacks :

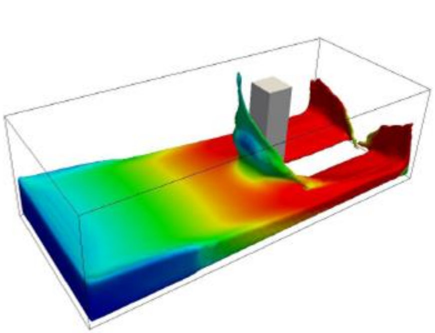
- Limited number of GPUs (2-8 GPUs)
- Limited size of simulation. Not 10^9 particles
- Does not work in distributed systems

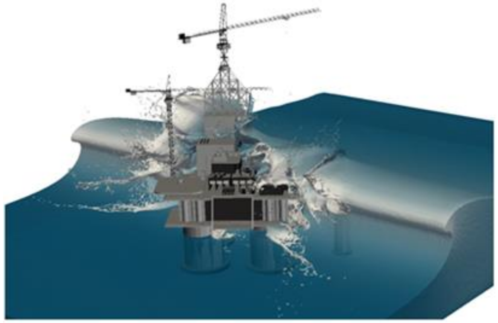
Video link:
https://youtu.be/EvSDFRfJToQ

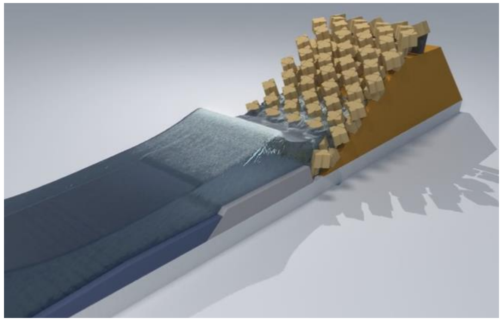
HPC for SPH methods: multicore GPU and multiGPU

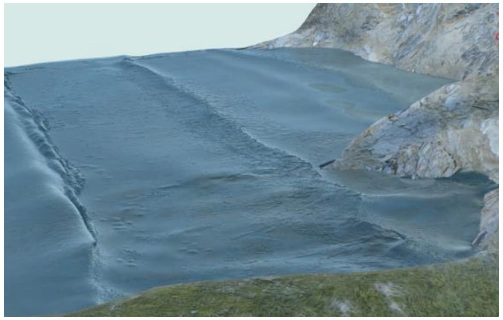
cpu gpu
DualSPHysics











Universidade Vigo

MANCHESTER
1824
The University of Manchester

José Domínguez, A. Mokos, B.D. Rogers,
A.J.C. Crespo and M. Gómez-Gesteira