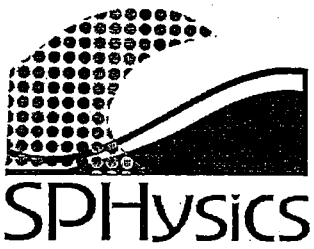
High-Performance Computing applied to Particle Hydrodynamics (SPH)



Benedict D. Rogers

School of Mechanical, Aerospace & Civil Engineering, University of Manchester

@SPH_Manchester

MANCHESTER 1824 · UNIVERSITÀ DEGLI STUDI DI PARMA · TÉCNICO LISBOA · UNIVERSITEIT GENT · Universidade Vigo

SPH Course, University of Parma, 19-21 February 2019

Overview

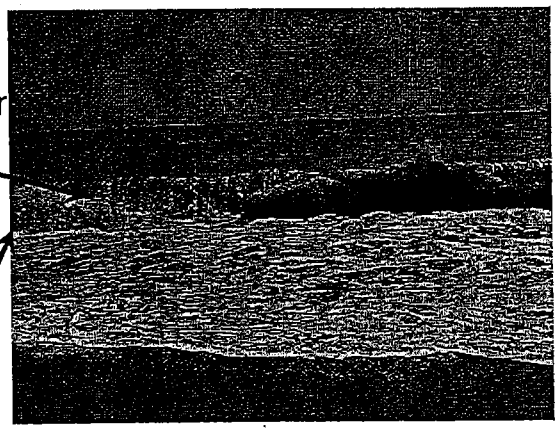
- Motivation, why SPH, why hardware acceleration is needed
- Basics of SPH algorithms: Neighbour Lists, Particle Interactions, System Update
- Options for hardware acceleration
- MPI, Domain Decomposition for SPH, Load Balancing and Optimisations
- The use of Graphics Processing Units (GPUs)
- Multi-Phase implementations
- Outlook

SPHysics

My Original Motivation for SPH

- **Free-surface flows** are rarely singly connected, e.g. beaches & wave energy devices

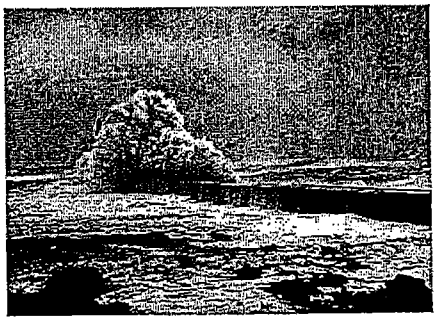
Breaking waves on beaches



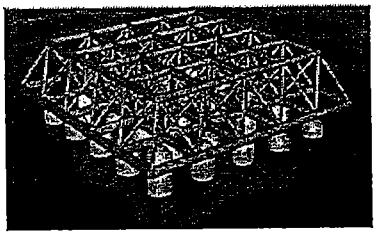
(Photo courtesy of F. Raichlen)

Very complex Multi-phase Multiscale problems

Overtopping:



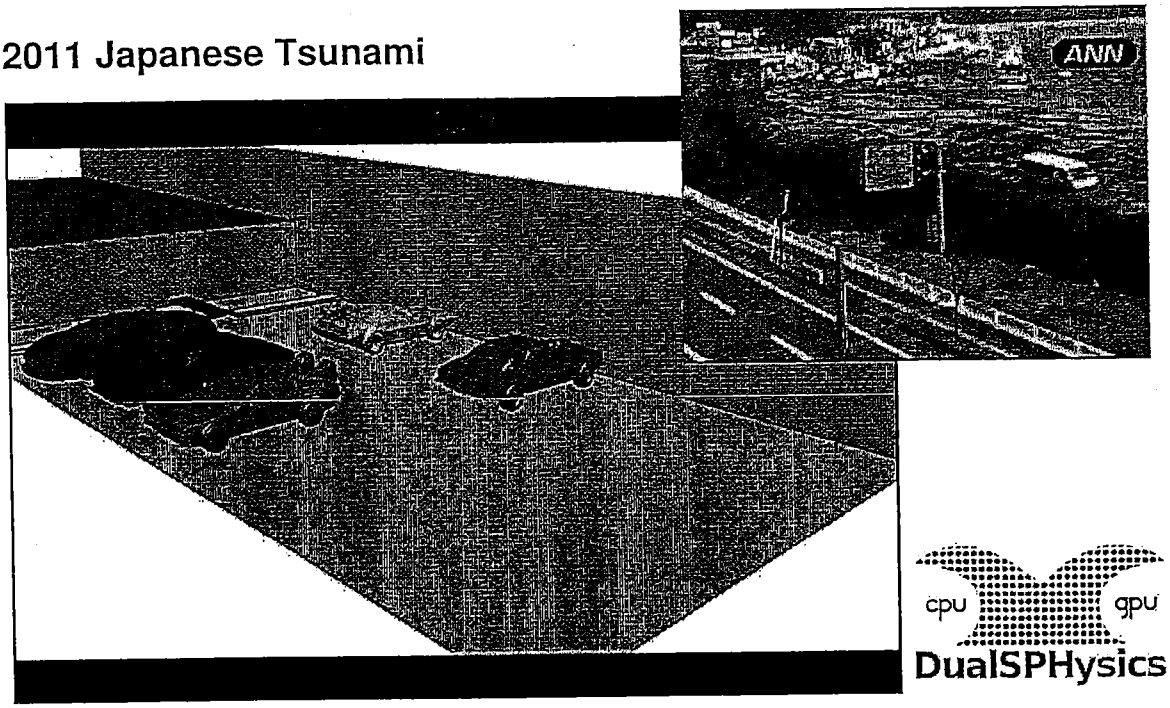
Wave Energy Devices: Manchester Bobber



SPH: Has some distinct advantages in simulating these situations

Classical SPH Formulation Example

2011 Japanese Tsunami



cpu gpu
DualSPHysics

Crespo et al. (2012)

Contents

1. Why does SPH require specialised computing?
2. The 3 main steps of an SPH simulation
3. Massive parallelisation with MPI
4. Novel computing: GPUs

MANCHESTER 1824

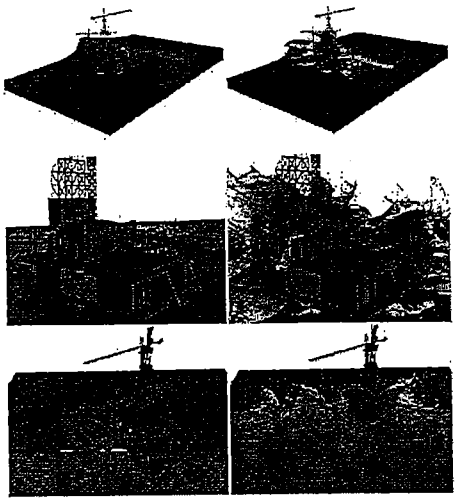
Contents

1. Why does SPH require specialised computing?
2. The 3 main steps of an SPH simulation
3. Massive parallelisation with MPI
4. Novel computing: GPUs

MANCHESTER 1824

Motivation for SPH simulations with large number of particles

- Real engineering problems are 3-D, very large and/or act on multiple scales.



e.g. 1 billion particles of Jose on 64 GPUs with Weakly Compressible SPH

Dominguez et al. (2013) Computer Physics Comm.

Very large particle numbers and high resolutions are required for high accuracy = **100+ million particles**

- An SPH Code should be able to scale on PetaFlops HPC platforms, with consideration of software development for Exascale.

3-D applications require millions of particles

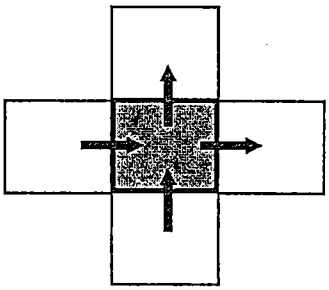
SPH *was* prohibitively expensive computationally

MANCHESTER 1824

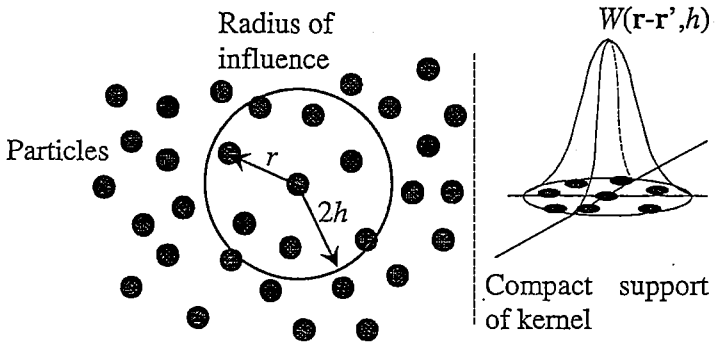
SPH simulations require acceleration, Why?

- The main problem is due to the interpolation procedure itself
- In Finite Volume, Finite Element & Finite Difference Schemes, the stencil around any cell usually contains only a small number of neighbouring cellsl e.g. **In 2-D FVM only 4 neighbouring cells**

Finite Volume (FVM) Stencil



SPH Stencil



- In **2-D**, each particle typically interacts with **20-50 particles**
- In **3-D**, each particle typically interacts with **100-400 particles**

Contents

1. Why does SPH require specialised computing?
2. The 3 main steps of an SPH simulation
3. Massive parallelisation with MPI
4. Novel computing: GPUs

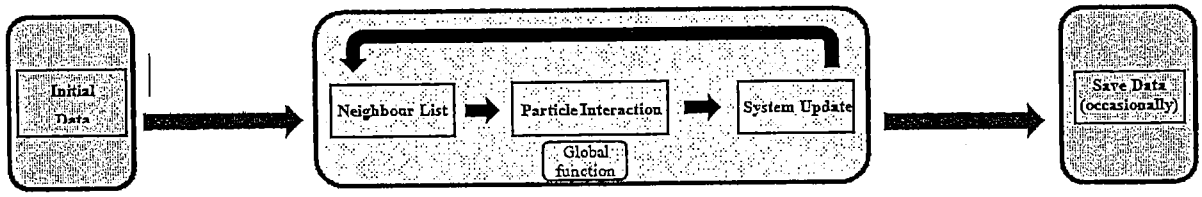
MANCHESTER 1824

Let's remind ourselves of:

the 3 main steps of any SPH code

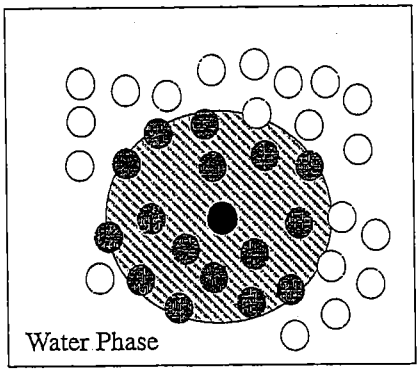
MANCHESTER 1824

Overview of an SPH code structure



We have 3 main steps:

1. Neighbour list (NL) creation
2. Particle Interaction (PI)
3. System Update (SU)



3 Main Steps of an SPH code

Step 1: Nearest neighbour searching

MANCHESTER 1824

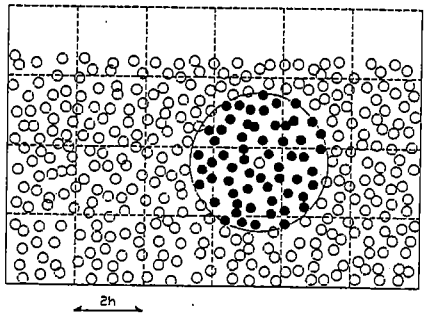
NEAREST NEIGHBOUR SEARCHING

In order to perform the summations required in SPH, we need to know which particles are near to each other

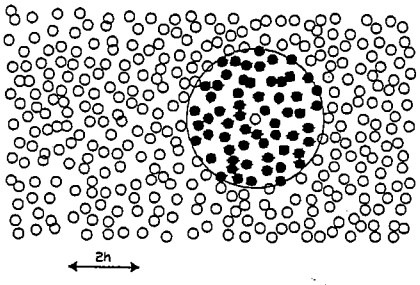
We cannot perform a summation over all the particles as this would take a prohibitive amount of time → hence the use of the compact support of the kernels

There are two general options for dividing up the computational domain efficiently:

(i) Divide up the domain into $2h$ boxes



(ii) Construct a Verlet list which is updated occasionally



NEAREST NEIGHBOUR SEARCHING

(i) Divide up the domain into 2*h* boxes

We split our domain into a set number of 2*h* boxes in each co-ordinate direction: *ncx, ncy, ncz*

We find in which 2*h* box a particle resides

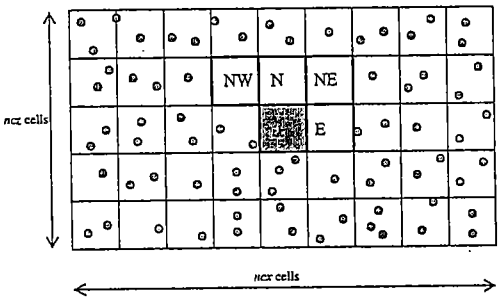
Pseudo-code:

$$\text{int}\left(\frac{x_i - x_0}{2h}\right)+1 \Rightarrow \text{icell}$$

$$\text{int}\left(\frac{y_i - y_0}{2h}\right)+1 \Rightarrow \text{jcell}$$

We make use of kernel symmetry (gradient asymmetry) to avoid repeating particle interactions

We sweep over the grid looking at particle interactions **between neighbour cells** where $r_{ij} < 2h$ (see SPHysics guide)



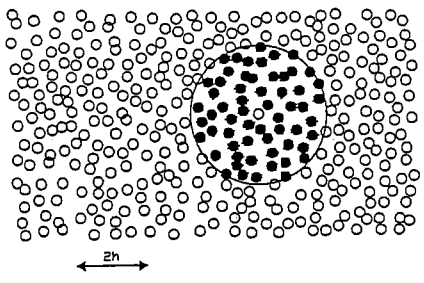
NEAREST NEIGHBOUR SEARCHING

(ii) Construct a Verlet list which is updated occasionally

We search our domain to create a list of neighbours for each particle that exist within 2*h* box or similar

Create a linked list and perform **sorting** algorithms to order particles for efficient sequential access

For more information see Dominguez et al. (2010)



3 Main Steps of an SPH code

Step 2: FORCE COMPUTATION

MANCHESTER 1824

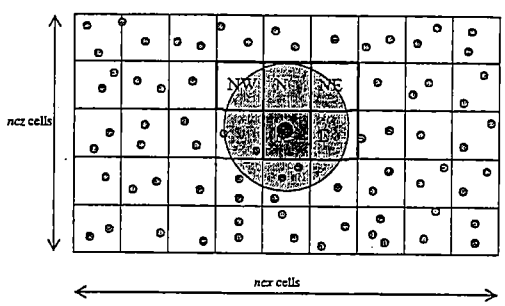
FORCE COMPUTATION

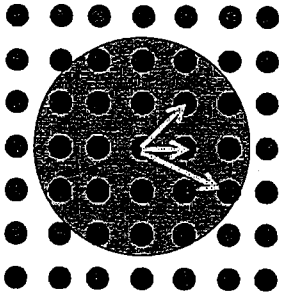
This computes the acceleration terms in the governing equations and is the MOST TIME CONSUMING PART

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \sum_j m_j \left(\mathbf{v}_i - \mathbf{v}_j\right) \cdot \nabla_i W_{ij}$$

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla_i W_{ij} + \sum_j m_j \frac{4\upsilon_o}{\rho_i + \rho_j} \frac{\mathbf{r}_{ij} \cdot \nabla_i W_{ij}}{r_{ij}^2 + 0.01h^2} \left(\mathbf{v}_i - \mathbf{v}_j\right) + \mathbf{F}_i$$

We sweep through our $2h$ cells calculating our interactions for all the neighbours





$$\sum f_j \nabla_i W_{ij} = f_1 \nabla_i W_{i1} + f_2 \nabla_i W_{i2} + f_3 \nabla_i W_{i3} + \ldots$$

3 Main Steps of an SPH code

Step 3: SYSTEM UPDATE: Time-stepping or Time integration

MANCHESTER 1824

SPH Time Integration

Let's see how we can march the simulation through time.

There is an enormous body of work on time stepping. We have a huge choice of schemes (some more sophisticated than others). As our particles (computation or nodal points) are moving we must use at least a scheme that is **2nd-order accurate in time.**

To begin with we rewrite all our governing equations in the following simplified form:

$$\frac{d\mathbf{v}_a}{dt} = \mathbf{F}_a \qquad \text{Momentum}$$

$$\frac{d\rho_a}{dt} = D_a \qquad \text{Continuity}$$

$$\frac{d\mathbf{r}_a}{dt} = \mathbf{v}_a \qquad \text{Position}$$

$$\frac{de_a}{dt} = E_a \qquad \text{Energy}$$

SPH Time Integration

2nd-order accurate Predictor-Corrector

Predictor step: This scheme predicts the evolution in time as

$$\mathbf{v}_a^{n+1/2} = \mathbf{v}_a^n + \frac{\Delta t}{2}\mathbf{F}_a^n \qquad \rho_a^{n+1/2} = \rho_a^n + \frac{\Delta t}{2}D_a^n$$

$$\mathbf{r}_a^{n+1/2} = \mathbf{r}_a^n + \frac{\Delta t}{2}\mathbf{v}_a^n \qquad e_a^{n+1/2} = e_a^n + \frac{\Delta t}{2}E_a^n$$

Corrector step: These values are then corrected using forces at half step

$$\mathbf{v}_a^{n+1/2} = \mathbf{v}_a^n + \frac{\Delta t}{2}\mathbf{F}_a^{n+1/2} \qquad \rho_a^{n+1/2} = \rho_a^n + \frac{\Delta t}{2}D_a^{n+1/2}$$

$$\mathbf{r}_a^{n+1/2} = \mathbf{r}_a^n + \frac{\Delta t}{2}\mathbf{v}_a^{n+1/2} \qquad e_a^{n+1/2} = e_a^n + \frac{\Delta t}{2}E_a^{n+1/2}$$

Finally, the values are calculated at the end of the time step

$$\mathbf{v}_a^{n+1} = 2\mathbf{v}_a^{n+1/2} - \mathbf{v}_a^n \qquad \rho_a^{n+1} = 2\rho_a^{n+1/2} - \rho_a^n$$

$$\mathbf{r}_a^{n+1} = 2\mathbf{r}_a^{n+1/2} - \mathbf{r}_a^n \qquad e_a^{n+1} = 2e_a^{n+1/2} - e_a^n$$

Contents

1. Why does SPH require specialised computing?
2. The 3 main steps of an SPH simulation
3. Massive parallelisation with MPI
4. Novel computing: GPUs

MANCHESTER 1824

Let's look at how the 2 main hardware acceleration options work

First, the CHOICE of HARDWARE determines the options of parallelisation technique ...

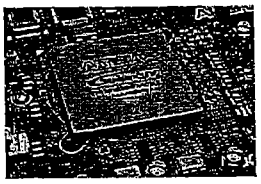
MANCHESTER 1824

Hardware Acceleration: the options

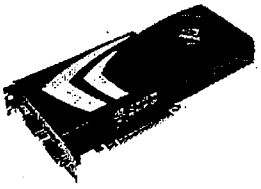
(i) Using parallel (**supercomputer**) machines with lots of cores (individual CPUs) and divide the work over them

Qu: What's the difference between a parallel machine and a supercomputer?

(ii) **FPGAs** – Field Programmable Gate Arrays: well used in astrophysics simulations, but expensive and not portable



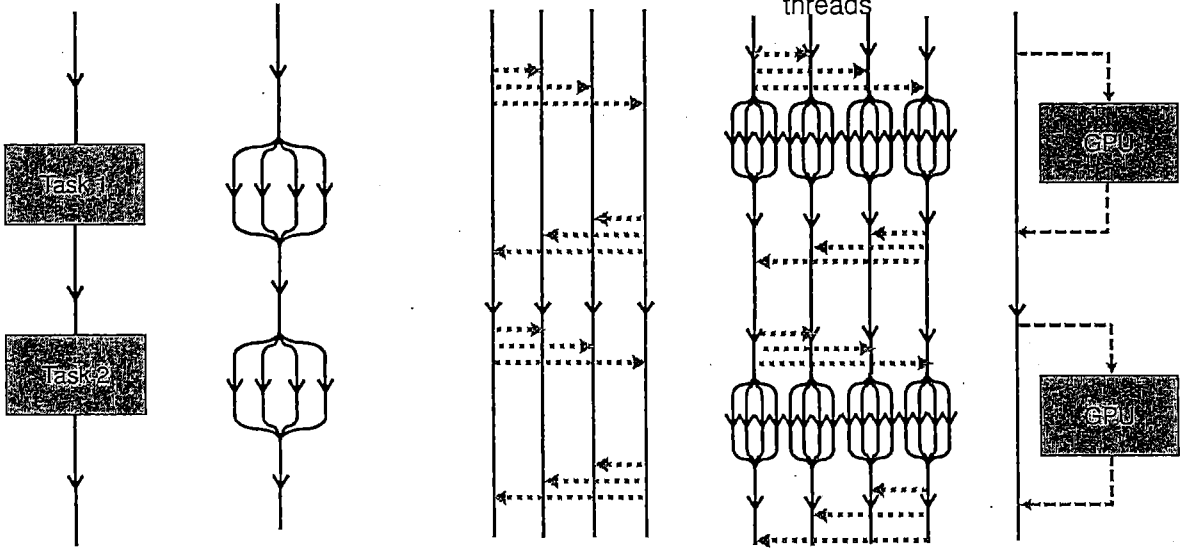
(iii) **GPUs** – Graphics Processing Units: the **hot topic** of scientific computing



Simulation Acceleration: the options

Depending on hardware 5 basic **PROGRAMMING** options exist:

Single Thread	Multiple Cores using OpenMP (shared memory)	Multiple Processes on Multiple Nodes using MPI (distributed memory)	MPI+OpenMP	CPU+GPU
1 thread	4 threads	4 cores	4 processes x 4 threads	1 thread + GPU



(cpu+mpi) dd / tasks

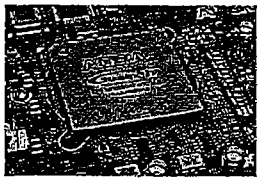
Hardware Acceleration: the options

(i) Using parallel (**supercomputer**) machines with lots of cores (individual CPUs) and divide the work over them

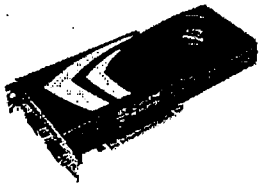
Qu: What's the difference between a parallel machine and a supercomputer?

→ interconnect speed = super

(ii) **FPGAs** – Field Programmable Gate Arrays: well used in astrophysics simulations, but expensive and not portable



(iii) **GPUs** – Graphics Processing Units: the **hot topic** of scientific computing



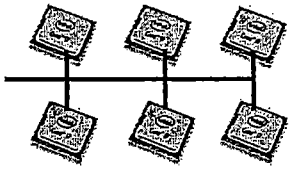
Hardware Acceleration for SPH

Option 1: Parallel computing using Distributed Memory with MPI

Parallel Algorithm Development

How to speed up a computation? 1 processor is not enough!

So, use lots of processors!



Spread the computational load between many processors

This requires communication

In the 1990s, the Message Passing Interface (MPI) emerged as the standard approach to achieve this communication:

Data is transferred between different processors using MPI commands:

e.g. to send data MPI_SEND
to receive data MPI_RECV

Parallel Algorithm Development

There are 3 key steps for massive parallelisation:

1. **Domain Decomposition** to distribute the load of the computation among different processors

2. **Load Balancing** so processors are not left doing nothing

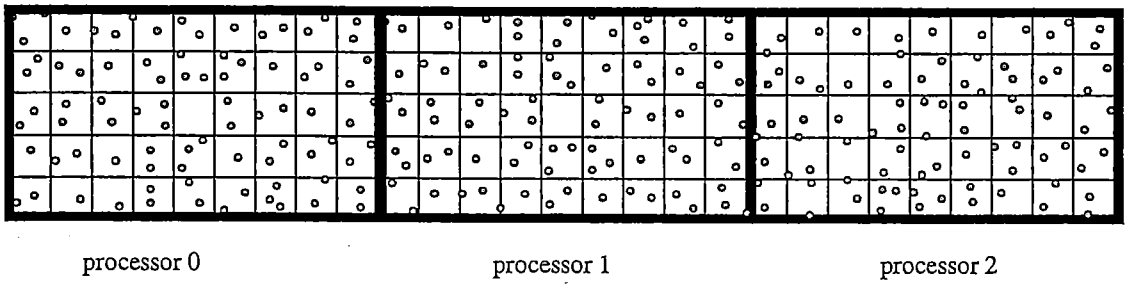
3. **Optimisation:** Reducing cache access time

Key Step 1: Domain Decomposition

MANCHESTER 1824

Parallel Algorithm Development

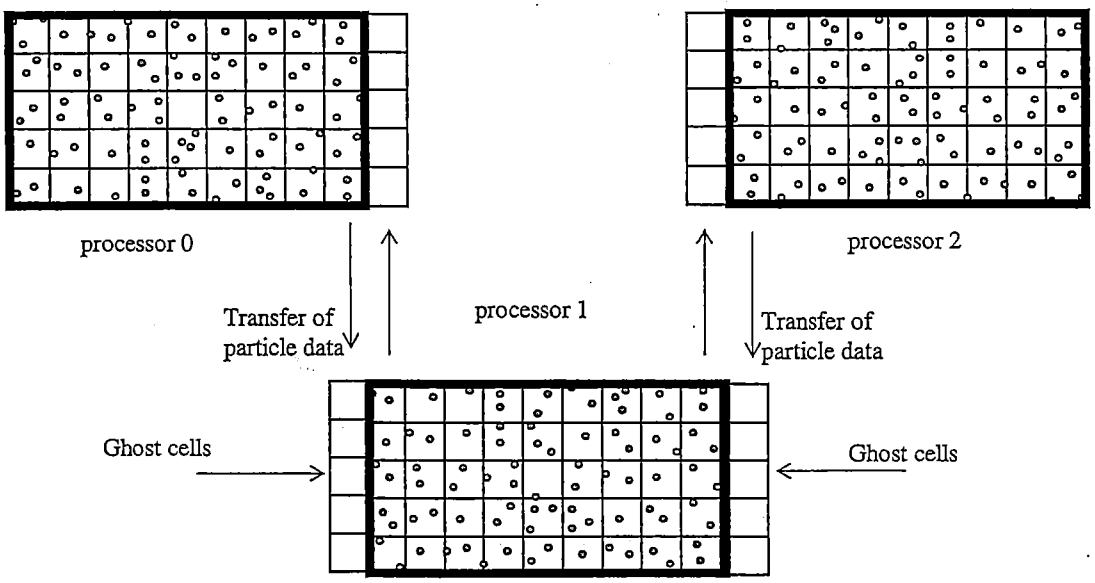
Connectivity of the particles is achieved using the common technique of splitting the domain into $2h$ square boxes



Subdivision of the domain across processors

Parallel Algorithm Development

Communication is then necessary



Data is exchanged between each processor using ghost cells

This is ideal for simple rectangular domains that are completely full

But it's very limited for SPH!

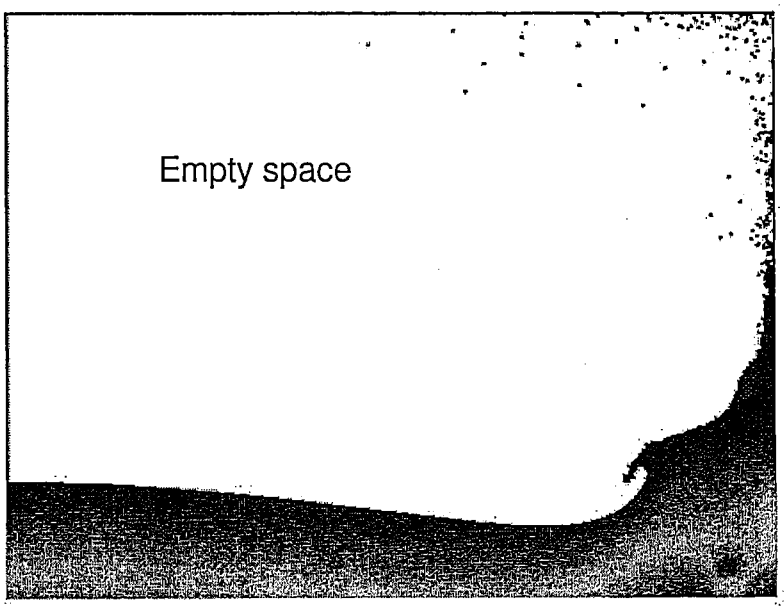
More sophisticated Domain Decomposition

Guo et al. (2018) CPC

MANCHESTER 1824

Why is a more sophisticated domain decomposition needed?

- Many of our applications will have highly fragmented fluid phases



How do we use our computing resources efficiently where particles are located?

- Octree
- ORB
- Space Filling Curves

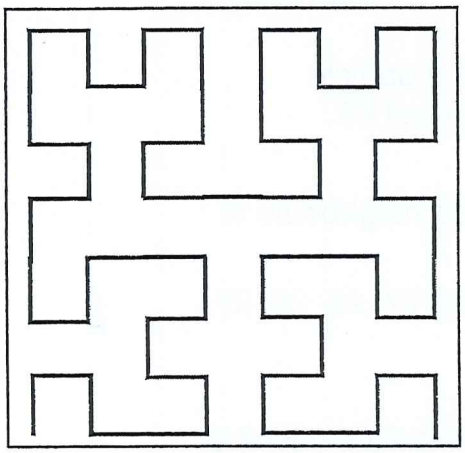
MANCHESTER 1824

Generating Space Filling Curve

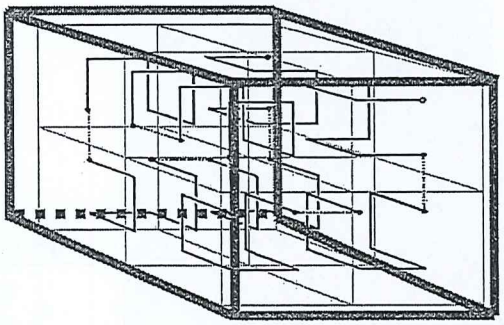
How do you map a space efficiently?

SPACE FILLING CURVES: Hilbert Space Filling Curve

2-D

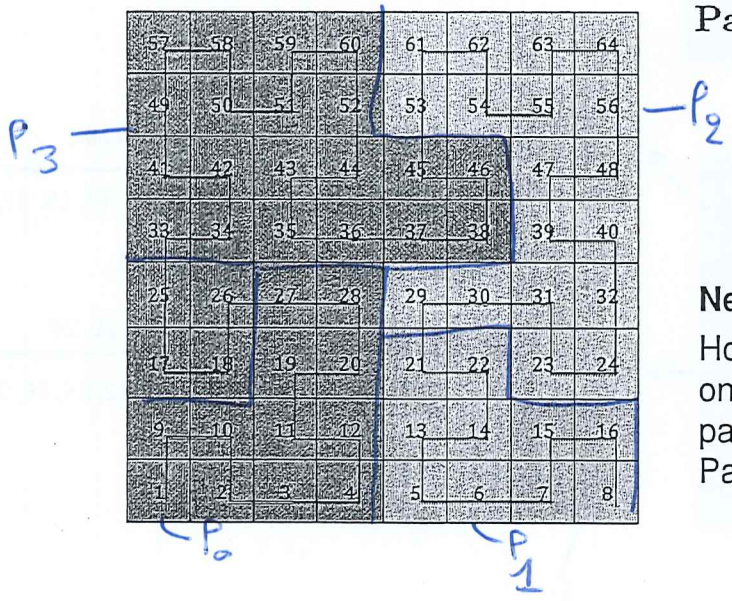


3-D



How to use a Space Filling Curve for Domain Decomposition

1. Cover the domain using the Space Filling Curve
2. Use a Domain Decomposition algorithm (e.g. Zoltan) to split the domain into subdomains



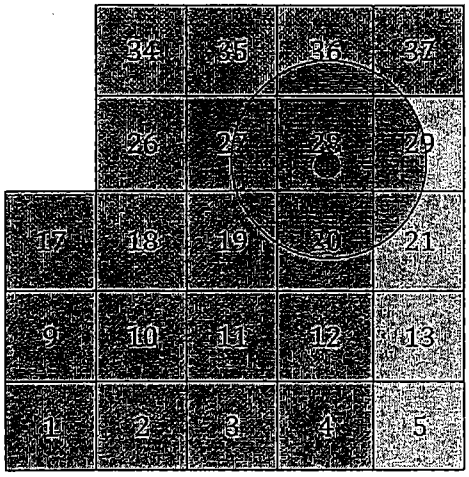
Partitions: P_0=blue, P_1=green, P_2=yellow, P_3=pink

Next Question:

How do particles located on P0 interact with particles on other Partitions?

Halo Creation

- Consider Particles located in Partition P_0. They need the particle information in the cells located on other partitions



Partitions: P_0=blue,
P_1=green
P_2=yellow
P_3=pink

Consider a particle located in cell 28.

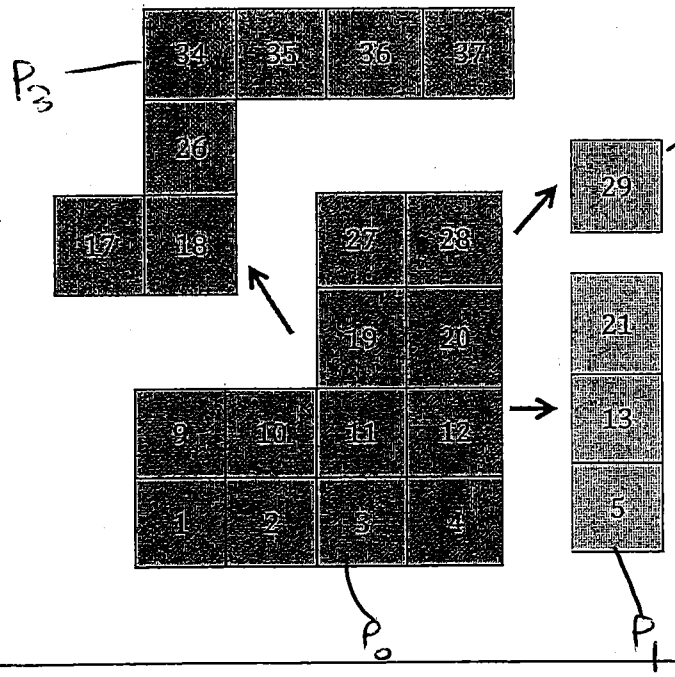
It will have neighbours in cells:

19, 20, 21, 27, 28, 29, 35, 36, 37

Hence we create a **HALO**

Halo Creation: MPI_SEND

- Zoltan for Domain Decomposition & halo
- Hilbert Space Filling Curve



Partitions: P_0=blue,
P_1=green
P_2=yellow
P_3=pink

Halo Send and Receive Plan for cells:

P0:SEND(to send to P3): 9, 10, 11, 19, 27, 28

P0:SEND(to send to P2): 20, 28

P0:SEND(to send to P1): 4, 12, 20, 28

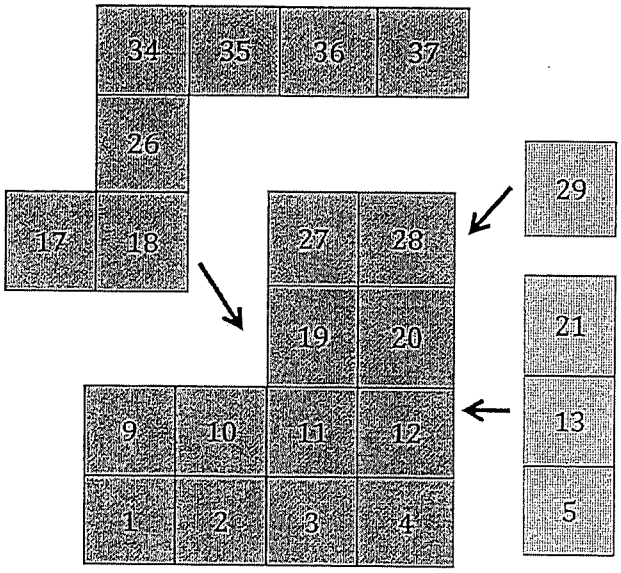
P0:RECV(from P3): 17, 18, 26, 32, 35, 36, 37

P0:RECV(from P2): 29

P0:RECV(from P1): 5, 13, 21

Halo Creation: MPI_RECV

- Zoltan for Domain Decomposition & halo
- Hilbert Space Filling Curve



Partitions: P_0=blue,
P_1=green
P_2=yellow
P_3=pink

Halo Send and Receive Plan for cells:

P0:SEND(to send to P3): 9, 10, 11, 19, 27, 28

P0:SEND(to send to P2): 20, 28

P0:SEND(to send to P1): 4, 12, 20, 28

P0:RECV(from P3): 17, 18, 26, 32, 35, 36, 37

P0:RECV(from P2): 29

P0:RECV(from P1): 5, 13, 21

Key Step 2: Load Balancing

MANCHESTER 1824

Load Balancing

There are 2 key principles of load balancing for any parallel code:

- Make Each Partition do the same amount of work
- Reduce Communication

This requires

- Load Balancing Strategy
- Sophisticated Communication (overlapping communication, etc.)

The aim is to give **SCALABILITY** over 1000s of cores

Load Balancing: Zoltan

ZOLTAN is a library for massively parallel systems:
http://www.cs.sandia.gov/Zoltan/

Each cell, i, has a weight is computed as: $wgts(i) = \frac{NP(i)}{NP_{total}}$

Where NP = number of particles

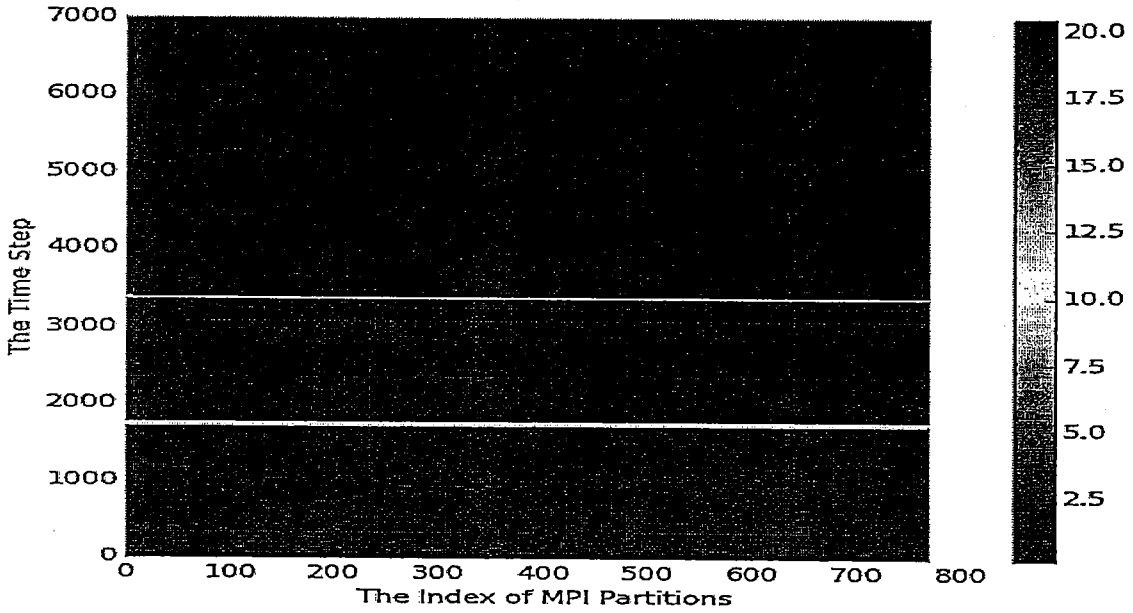
Load on each processor = sum of cell weights = $load = \Sigma_{i=1}^{N_k} wgts(i)$

Imbalance for each processor = $imbalance = \frac{load_{max}}{load_{average}} = \frac{max(load)}{NP_{total}/K}$

If Imbalance > 20% tolerance, Use ZOLTAN to partition the cells along the HSFC = Hilbert Space Filling Curve
(See Guo et al. 2018 CPC).

Load Balancing: Heat Maps

The easiest way to check that your code is well balanced is using HEAT MAPS:

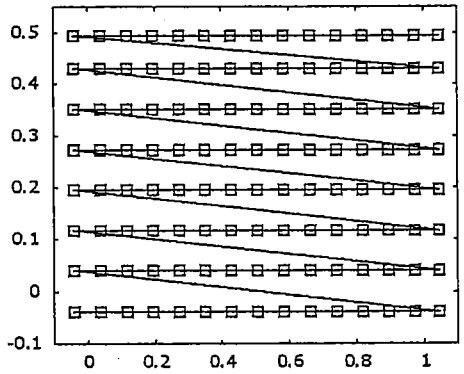


Key Step 3: Reducing Cache Access Time

MANCHESTER 1824

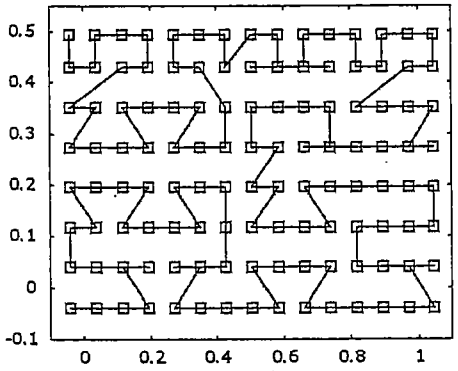
Particles-Cells reordering

The *inherited* cell linked list of particles would sweep through the domain in a sequential order:



The natural order of cells

To improve *cache* performance and avoid overchecking of neighbours use Hilbert Space Filling Curve [HSFC] to reorder the list of particles for neighbours only:



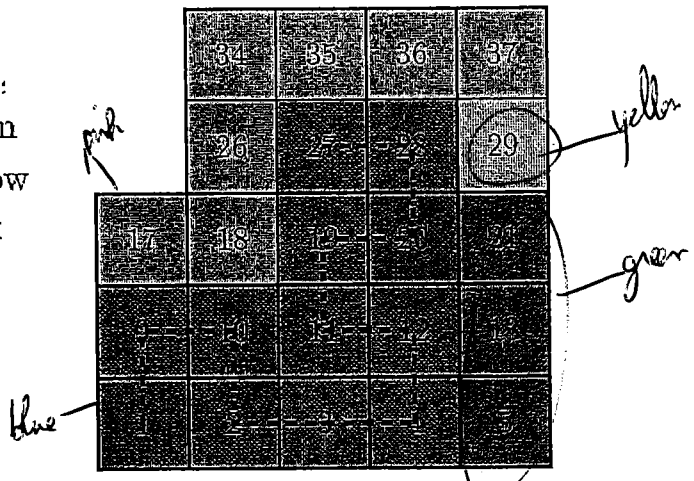
The HSFC order of cells

(see Ogen et al. 2010)

Reordering Cells on a Partition

- The order of the particles is important to maximise the time it takes to retrieve the values of particle properties from memory (CACHE ACCESS)

Partitions: P_0=blue,
P_1=green
P_2=yellow
P_3=pink



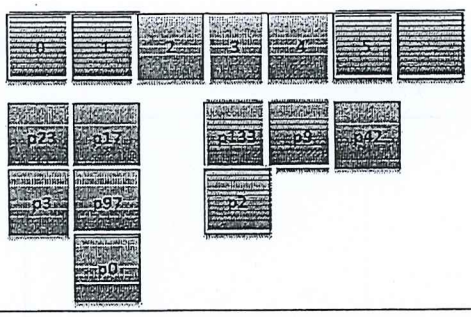
Cell's Natural Order: 1 2 3 4 9 10 11 12 19 20 27 28

Cells HSFC Order: 1 9 10 2 3 4 12 11 19 20 28 27

Cell Linked List: preconditioned dynamic vector and linked list

Particle arrangement within the cells

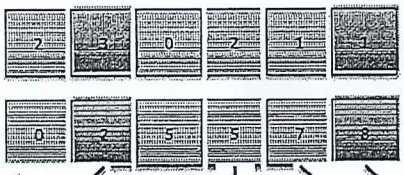
Grid Cell:



Apply pre-conditioning

Count:

Offset:



New list of particles stored for faster access

Storage:

Dominguez *et al.* (2011)

Reduces Memory Footprint: crucial for using massive HPC systems

General Performance Considerations for SPH Parallelization

We don't just want a highly parallelised SPH scheme, we also need to solve the Pressure Poisson Equation over 10-30,000 cores!

The challenges can be summarised as:

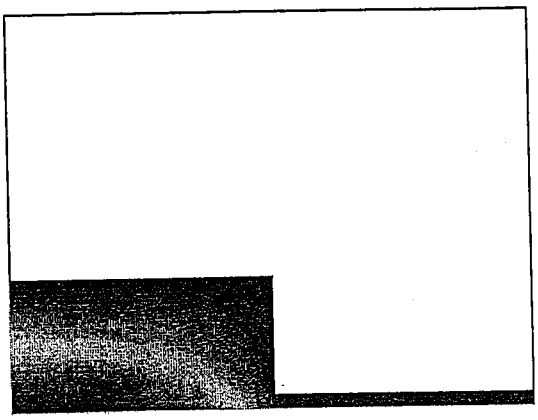
- Efficient neighbour list searching
- Depends upon a **good assignment** of particles to processors;
  - Grouping **physically close particles** within a single processor
  - Reduces inter-processor communication
  - Re-partitioning of the particles to maintain geometric locality of objects within processors
- Low load balancing costs for non-uniformly distributed particles arising from solving a complex, highly nonlinear and distorted flow and adaptive SPH.

+ Strong & Weak Scaling

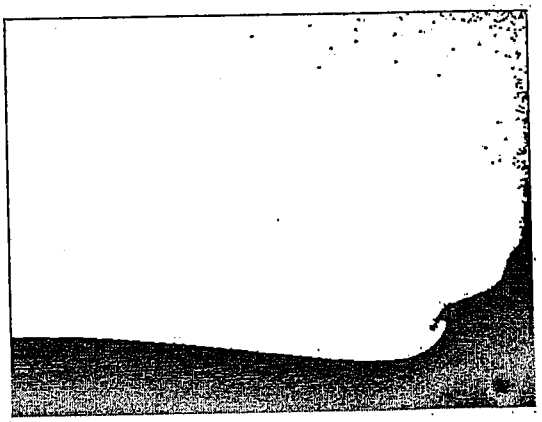
Demonstration Case: 2-D Wet-Bed Dam Break

What does this look like when we run it for a wet-bed dam break?

An example of a violent nonlinear flow requiring highly adaptive domain decomposition across empty space:



$t = 1$



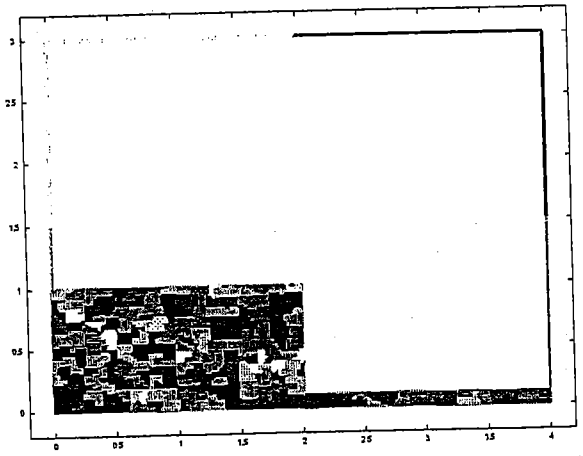
$t = 5720\Delta t$

Colours denote pressure

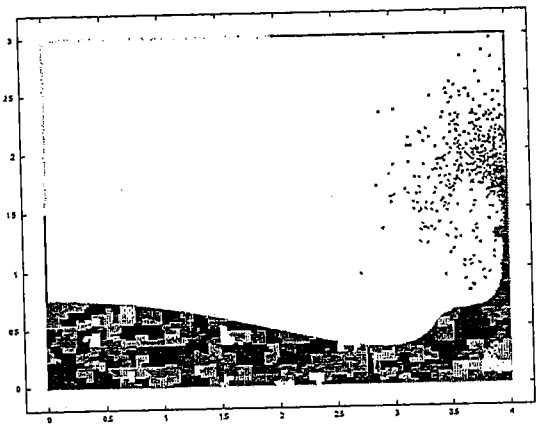
Dynamic Domain Decomposition with HSFC

What does this look like when we run it for a wet-bed dam break?

An example of a violent nonlinear flow requiring highly adaptive domain decomposition across empty space:



256 partitions at $t = 1$



256 partitions at $t = 5720\Delta t$

Colours denote partitions

What about Incompressible SPH (ISPH)?

MANCHESTER 1824

Incompressible SPH method

- ISPH_DF – A Divergence-free Projection method in SPH

$$\mathbf{r}_i^* = \mathbf{r}_i^n + \delta t\, \mathbf{u}_i^n$$ Particles are advected to $\mathbf{r}_i^*$

$$\mathbf{u}_i^* = \mathbf{u}_i^n + \left(\nu \nabla^2 \mathbf{u}_i^n + \mathbf{F}_i^n\right)\delta t$$ Calculate $\mathbf{u}_i^*$ without pressure gradient.

$$\nabla \cdot \left(\frac{1}{\rho}\nabla p^{n+1}\right)_i = \frac{1}{\delta t}\nabla \cdot \mathbf{u}_i^*$$ Solve pressure Poisson equation to get pressure P_i^{n+1}.

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^* - \frac{\delta t}{\rho}\nabla p_i^{n+1}$$ Calculate velocity $\mathbf{u}_i^{n+1}$.

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \delta t\left(\frac{\mathbf{u}_i^{n+1} + \mathbf{u}_i^n}{2}\right)$$ Particles' positions are centered in time.

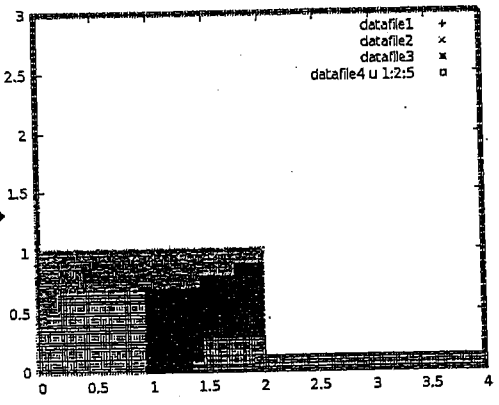
$$\delta \mathbf{r}_s = -\mathcal{D}\left(\frac{\partial C}{\partial s}\mathbf{s} + \alpha\left(\frac{\partial C}{\partial n} - \beta\right)\mathbf{n}\right)$$ Shift particles to maintain stability.

*ISPH (Cummins & Rudman 1999)

Parallel ISPH Solver features

In addition to the ISPH formulation

- ISPH utilizes **Zoltan** for spatial **Domain Decomposition** and **Dynamic Load Balancing** for evenly distributed use of computational resources.



- To solve Poisson Pressure Equation

$$\underbrace{\nabla \cdot \left(\frac{1}{\rho} \nabla p^{n+1} \right)_i = \frac{1}{\delta t} \nabla \cdot \mathbf{u}_i^*}$$

$$\rightarrow \quad -\frac{1}{\rho} \sum_j V_j (p_j - p_i) \nabla W_{ij}$$

ISPH use **PETSc** (Portable, Extensible Toolkit for Scientific Computation).

- Fortran 90, MPI.

General Performance Considerations for ISPH Parallelization

We don't just want a highly parallelised SPH scheme, we also need to solve the Pressure Poisson Equation over 10-30,000 cores!

The challenges can be summarised as:

- Efficient neighbour list searching
- Depends upon a **good assignment** of particles to processors;
  - Grouping **physically close particles** within a single processor
  - Reduces inter-processor communication
  - Re-partitioning of the particles to maintain geometric locality of objects within processors
- Low load balancing costs for non-uniformly distributed particles arising from solving a complex, highly nonlinear and distorted flow and adaptive SPH.

- Efficient and scalable pressure Poisson solver

Solving Pressure Poisson Equation(PPE) with PETSc

$$\nabla \cdot \left(\frac{1}{\rho} \nabla p^{n+1} \right)_i = \frac{1}{\delta t} \nabla \cdot \mathbf{u}_i^*$$

After cell & particle reordering .... Let's solve the Pressure Poisson Equation = matrix eq

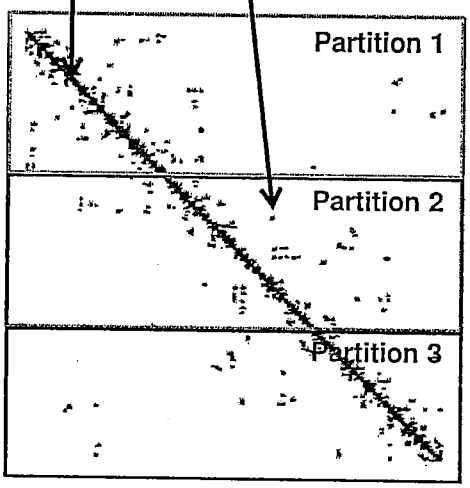
$$\mathbf{AX} = \mathbf{b} \quad \text{------} \quad a_{ii} P_i + \sum \alpha_i a_{ij} P_j = \alpha_i b_i$$

ISPH uses PETSc for the sparse linear solver.

- the non-zeros of each row in the matrix represent each particle's neighbour particles
- The PETSc algorithm partitions the matrix by continuous rows, which requires a renumbering of particles so that insertions of values to global matrix become local operations without incurring extra communications

$$\mathbf{AX} = \mathbf{b} \Leftrightarrow \mathbf{PAPTPX} = \mathbf{Pb}$$

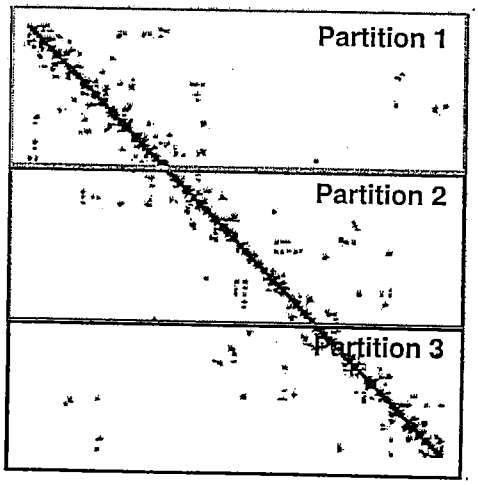
- benefit of reducing the bandwidth of coefficients matrix



Sparse Pattern of PPE Matrix

Problem with convergence of the PETSc matrix solver:

- Many solvers can be slow to converge
- Previously used **Jacobi(preconditioner)** +Bi-CGSTAB for single core but this is prohibitively slow.
- With a large number of particles (100+ million) there can be **no convergence** at all.
- For example, with **Jacobi+BiCGSTAB** won't converge, often requires 10,000+ iterations to converge every Δt.



Sparse Pattern of PPE Matrix

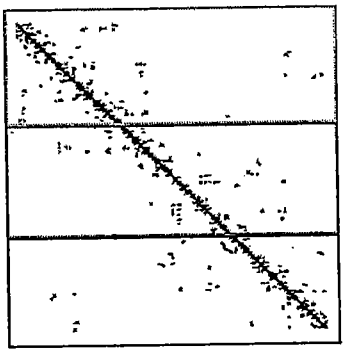
Key Question for ISPH for 100+ million particles over 10,000 cores:

How to address convergence for 100+ million particles for sparse PPE matrix? Need good/efficient preconditioner

SUPERFAST preconditioning for convergence of the PETSc matrix solver:

SOLUTION?

How to address converge for 100+ million particles?



Sparse Pattern of PPE Matrix

Multigrid preconditioner:

HYPRE BoomerAMG

BoomerAMG has 2 phases:

1. setup(**PCSETUP**)

selection of coarse grids

creation of the interpolation operators

the representation of the fine grid matrix operator on each coarse grid

2. solve(**PCAPPLY**)

matrix-vector multiply

+ the smoothing operator

ISPH PPE Solver Performance Analysis: strong scaling

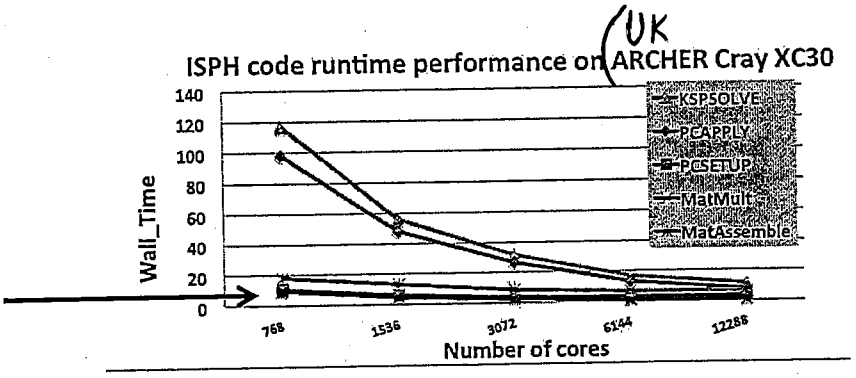
Strong scaling = fixed number 100million particles but vary number of cores

But with more cores → more communication and this normally hurts your computation runtime ☹ a bit challenging

Question 1: What is the pre-conditioning's cost in PETSc for ISPH PPE?

Wall clock time for pre-conditioning operations (PCSETUP) is very small for all # cores,

PCAPPLY(smoother and SpMV) has the same cost with solver phase (KSPSOLVE).

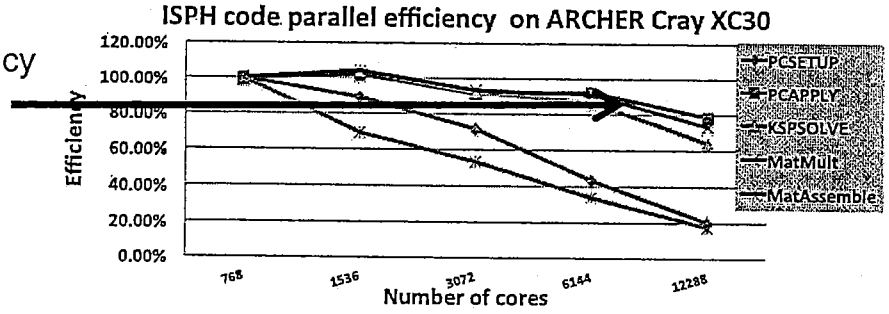


ISPH PPE Solver Performance Analysis: strong scaling

Strong scaling = fixed number 100million particles but vary number of cores

Question 2: How does multigrid preconditioner improve ISPH performance?

Over 80% efficiency with10,000 cores



Number of iterations for convergence changed from

10,000+ → 12 iterations per timestep!!

Answer: quite promising for 100+million particles

... but may not be enough for ExaSCale

Some Key Results from MASSIVELY PARALLEL SPH CODES

For SPH:

40 million – EDF using Spartacus on a Blue Gene
120 million – EPFL using a Blue Gene
200 million – EC using SPH-flow in France
1 billion – UK Astrophysics Consortium
1 billion – DualSPHysics Consortium
1 billion – SWIFT astrophysics
1.2 billion – Karlsruhe, Germany

Impressive, BUT these simulations needed massive HPC supercomputers. This raises important questions:

- What are the limitations?
- Are these accessible to all potential users/developers/researchers of SPH?

Limits of an SPH Parallel Codes

- What is the **limit** of what we can simulate on supercomputing clusters?
- Another way of asking that question is how many particles can we realistically use in a simulation before we get hit by prohibitive communication and just the sheer number of processors that we need to support **millions-billions** of particles
- Thankfully, there is a new solution!!!! .....

Contents

1. Why does SPH require specialised computing?
2. The 3 main steps of an SPH simulation
3. Massive parallelisation with MPI
4. Novel computing: GPUs

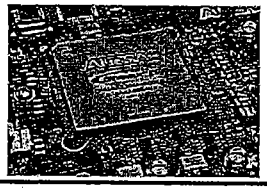
MANCHESTER 1824

Hardware Acceleration: the options

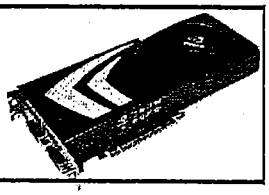
(i) Using parallel (**supercomputer**) machines with lots of cores (individual CPUs) and divide the work over them

Qu: What's the difference between a parallel machine and a supercomputer?

(ii) **FPGAs** – Field Programmable Gate Arrays: well used in astrophysics simulations, but expensive and not portable



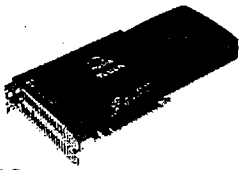
(iii) **GPUs** – Graphics Processing Units: the **hot topic** of scientific computing



Hardware Acceleration for SPH

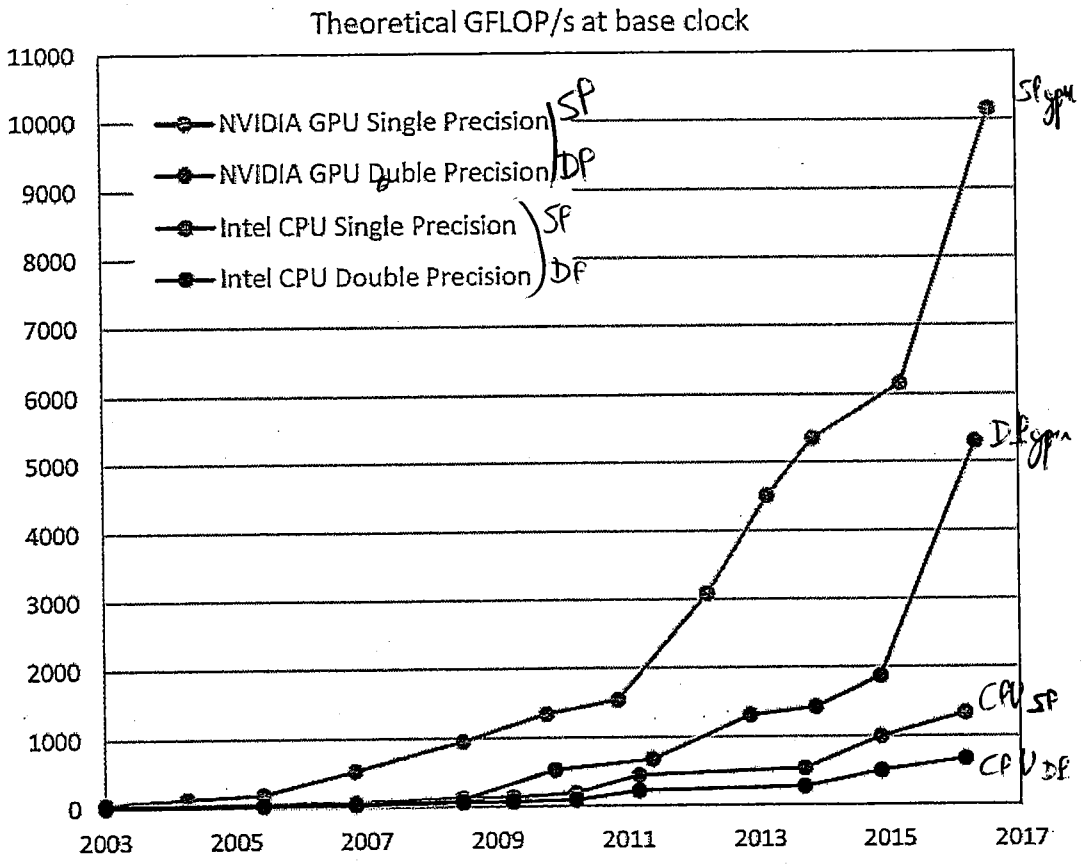
Option 2: GPUs

What is a GPU?



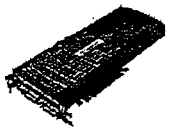
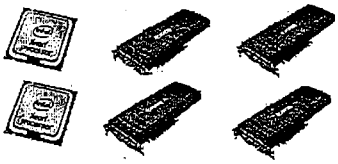
- **GPU = Graphics Processing Unit**
  - They are used in all your computers for the graphics
- **GPUs developed independently of the CPU**
  (CPU=Central Processing Unit used for normal computation)
  - This means the architecture is very different
  - In order to play Games in real time, they process a lot of information very quickly
  - Hence, they consist of many **parallel threads**
- **nVidia GPUs have an dedicated language - CUDA**
  - CUDA = Compute Unified Device Architecture
  - Make sure you buy a a **CUDA-enabled** GPU!!!
  - Programming is far more difficult as it is a lower level language than C/C++ & Fortran
  - You must know how to use the architecture
- **nVidia** produce specific GPUs for **Scientific Computing: FERMI** cards

SPHysics



Proposals to accelerate SPH

How to accelerate your SPH code with less than ....

£250	Intel® Core ™ i7 940 at 2.93GHz with 4 cores 	**Multi-core OpenMP** Speedup: 4x
£350	GTX 680 at 1GHz with 1536 cores 	**GPU CUDA** Speedup: 60-90x
£15,000	2 Intel Xeon E5620 + 4 GTX680 	**Multi-GPU CUDA + MPI** Speedup: 100-500??

TOP SUPERCOMPUTERS IN THE WORLD Nov 2018

http://www.top500.org

R_{max} and R_{peak} values are in TFlops. For more details about other fields, check the TOP500 description.

TOP 10 Sites for November 2018

1° Sunway Taihulight (China) 93.0 petaflops/s (consumption: 15371 KW) (CPUs)
3° Titan (USA) 17.5 petaflops/s (consumption: 8209 KW) (with GPUs)

Rank	Site	System	Cores	[TFlop/s]	[TFlop/s]	[kW]

Energy Efficient GPU co-processors are now a key component in HPC

Part of computing Emerging Technology 2019 Conference:

http://emit.tech

	High Performance Computing Japan	M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path Fujitsu

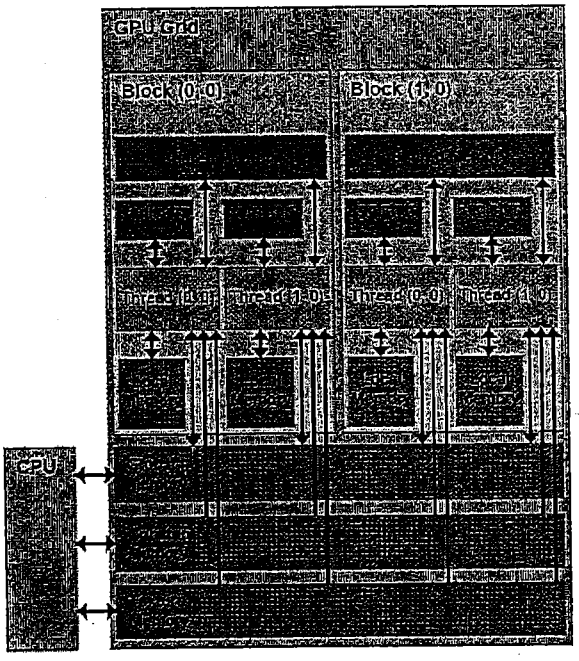
09-11/04/2019 UK

(Hoffer)

Factors affecting Performance Issues on GPUs

- Several memory spaces with different access modes and performance (shared memory, global memory, texture memory, ..)
- Branch divergence (avoid if statements)
- Non-balanced workload (not all streaming multi-processors finish simultaneously)
- Streaming Multi-Processor Occupancy
- No coalescent memory access (this is how data is grouped together on the GPU, 32, 64 or 128 bytes)

Memory Hierarchy



Inside each SM, there are **REGISTERS** which are fast and hold data but are small in number

The GPU generally consists of Streaming Multiprocessors (SMs) with access to different types of memory:

Global Memory: largest & slowest

Constant Memory: Read-only resides in Global Memory

Texture Memory: in Global memory optimised for 2-D spatial data locality

Local Memory: used only when registers have all been used

Shared Memory: is **FAST** as it is located in each Streaming Multiprocessor (SM)

So, programming a GPU is not easy but also not so difficult

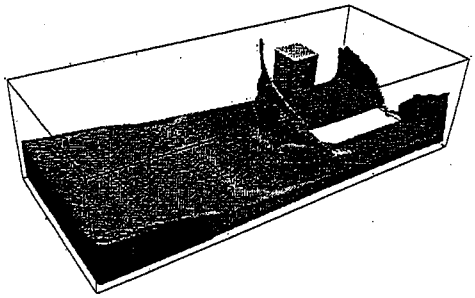
There are now several GPU codes ...

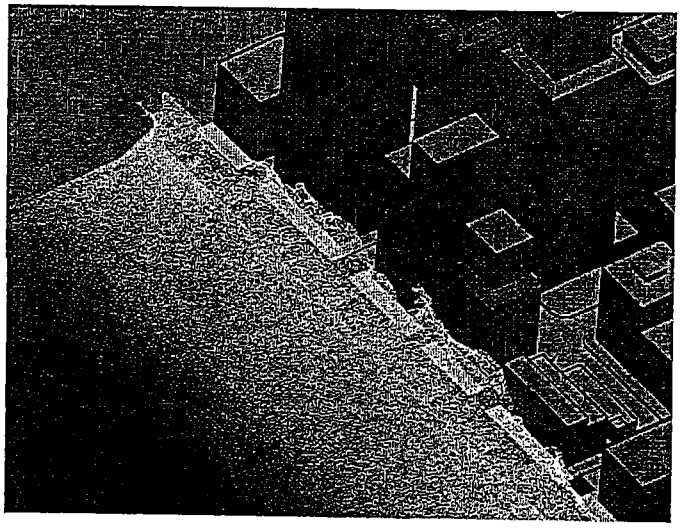
MANCHESTER 1824

DualSPHysics, new GPU computing on SPH models

cpu gpu

DualSPHysics





A.J.C. Crespo, J.M. Dominguez, A. Barreiro and M. Gómez-Gesteira
EPHYSLAB, Universidade de Vigo, SPAIN

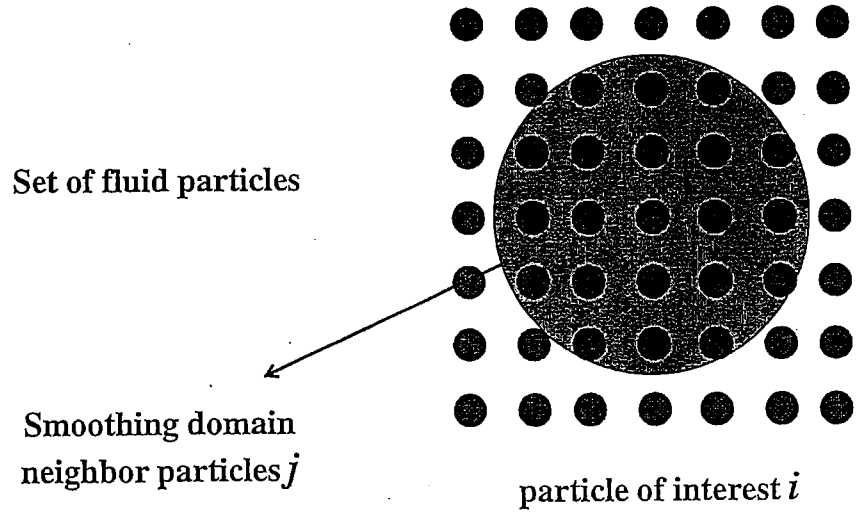
B. D. Rogers and D. Valdez-Balderas
MACE, The University of Manchester, U.K.

Universidade Vigo

MANCHESTER 1824

SPH Momentum Equation

$$\left(\frac{du}{dt}\right)_i = -\sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right) \nabla_i W_{ij} + g$$

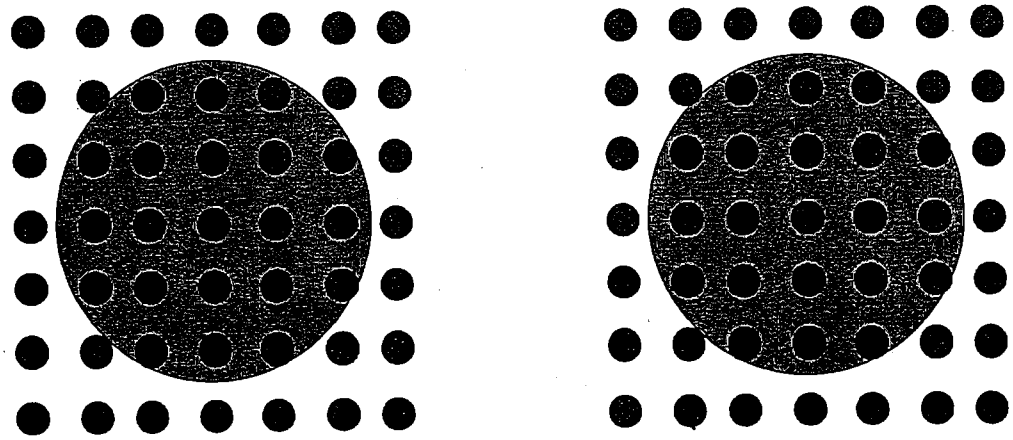


SPH Momentum Equation

SUMMATION SOLUTION

CPU / Fortran

GPU / CUDA

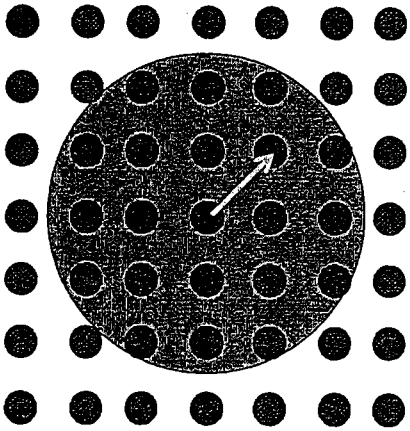


$i \rightarrow j$ $\boxed{a_i}$ $\left(\frac{du}{dt}\right)_i = -\sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right) \nabla_i W_{ij} + g$

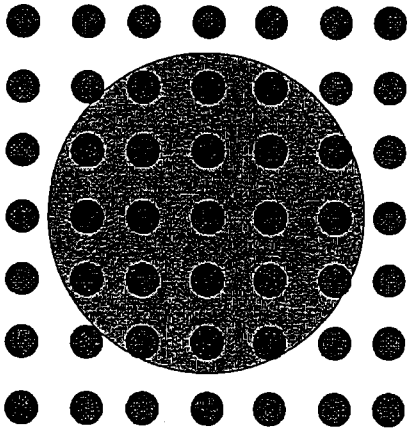
SPH Momentum Equation

SUMMATION SOLUTION

CPU / Fortran



GPU / CUDA



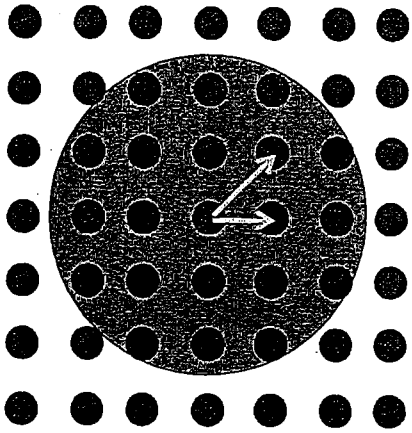
$i \rightarrow j = 1$ $\boxed{a_{ij=1}}$

$$a_i = a_{i1} + \ldots$$

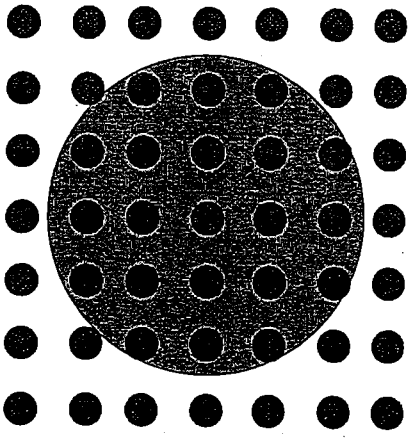
SPH Momentum Equation

SUMMATION SOLUTION

CPU / Fortran



GPU / CUDA



$i \rightarrow j = 1$

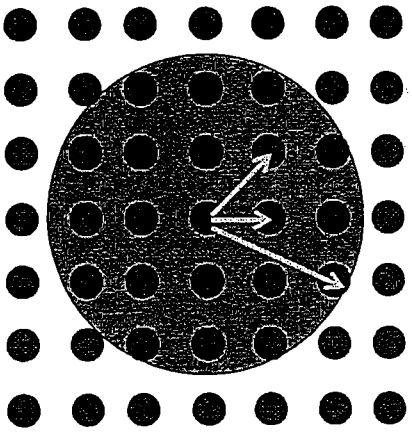
$i \rightarrow j = 2$ $\boxed{a_{ij=2}}$

$$a_i = a_{i1} + a_{i2} + \ldots$$

SPH Momentum Equation

SUMMATION SOLUTION

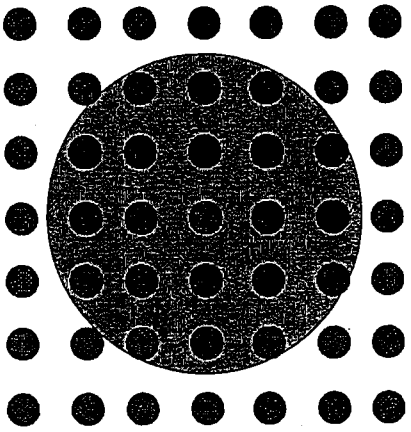
CPU / Fortran



$i \rightarrow j = 1$
$i \rightarrow j = 2$
$i \rightarrow j = 3$ $\boxed{a_{ij=3}}$

$$\boxed{a_i = a_{i1} + a_{i2} + a_{i3} + \ldots}$$

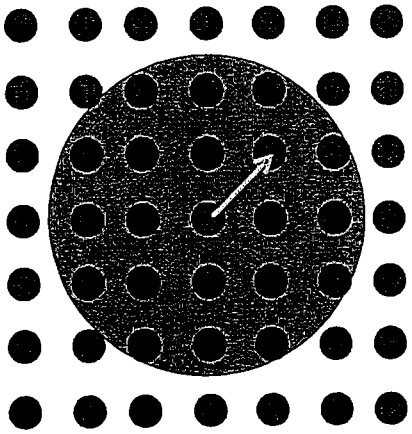
GPU / CUDA



SPH Momentum Equation

SUMMATION SOLUTION

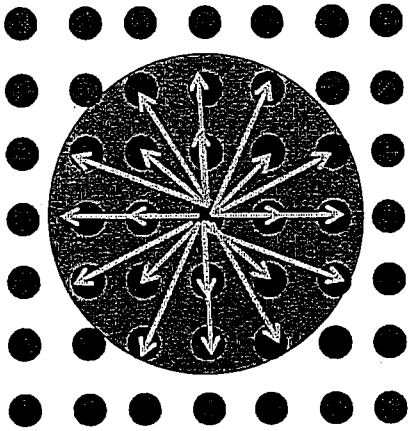
CPU / Fortran



$i \rightarrow j = 1$ $\boxed{a_{ij=1}}$

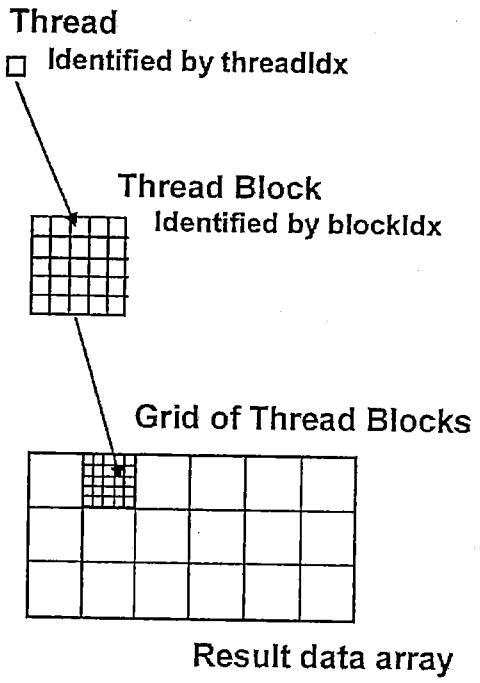
$$\boxed{a_i = a_{i1} + \ldots}$$

GPU / CUDA



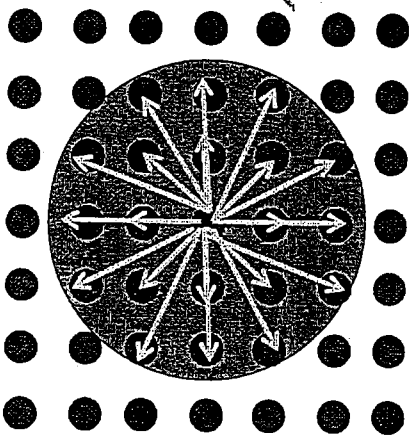
$i \rightarrow j$ $\boxed{a_{ij}}$

$$\boxed{a_i = \sum a_{ij}}$$



SUMMATION SOLUTION

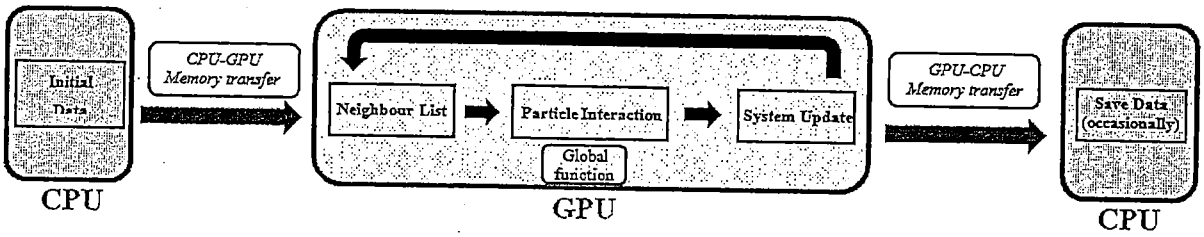
GPU / CUDA



Speedups of 20-100

$$\left(\frac{du}{dt}\right)_i = -\sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right) \nabla_i W_{ij} + g \qquad i \rightarrow j$$

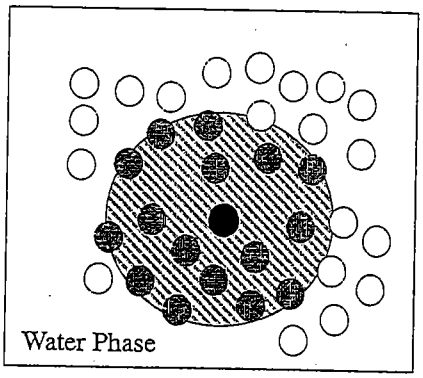
Overview of GPU code structure



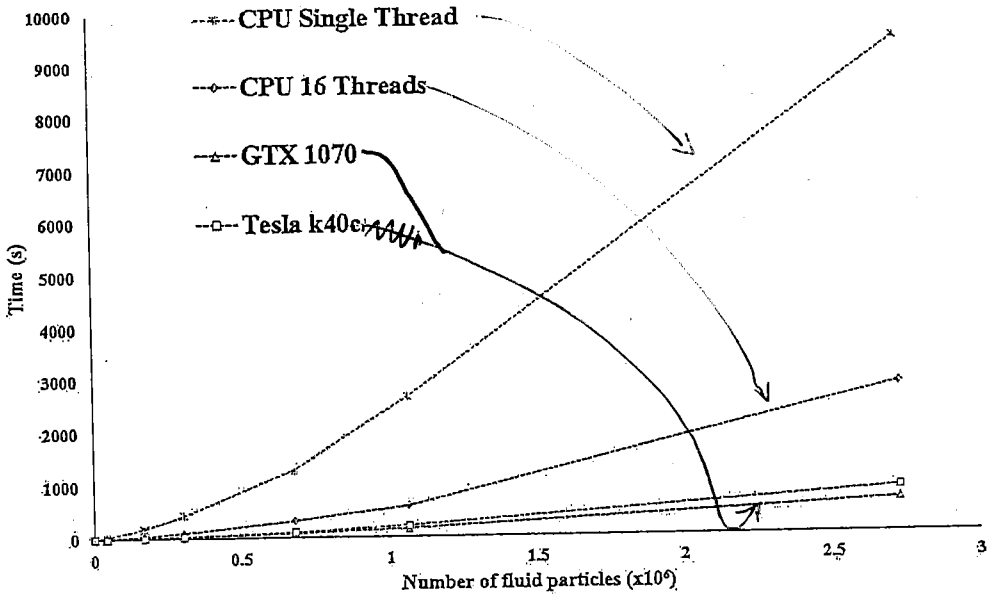
We have 3 main steps:

1. Neighbour list (NL) creation
2. Particle Interaction (PI)
3. System Update (SU)

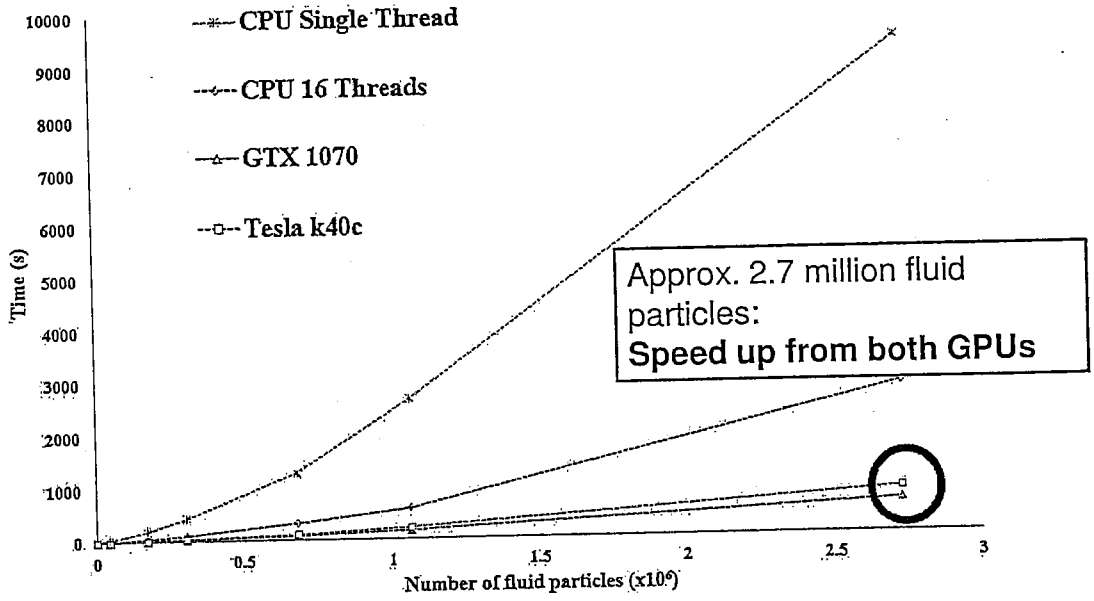
All within 1 single global function



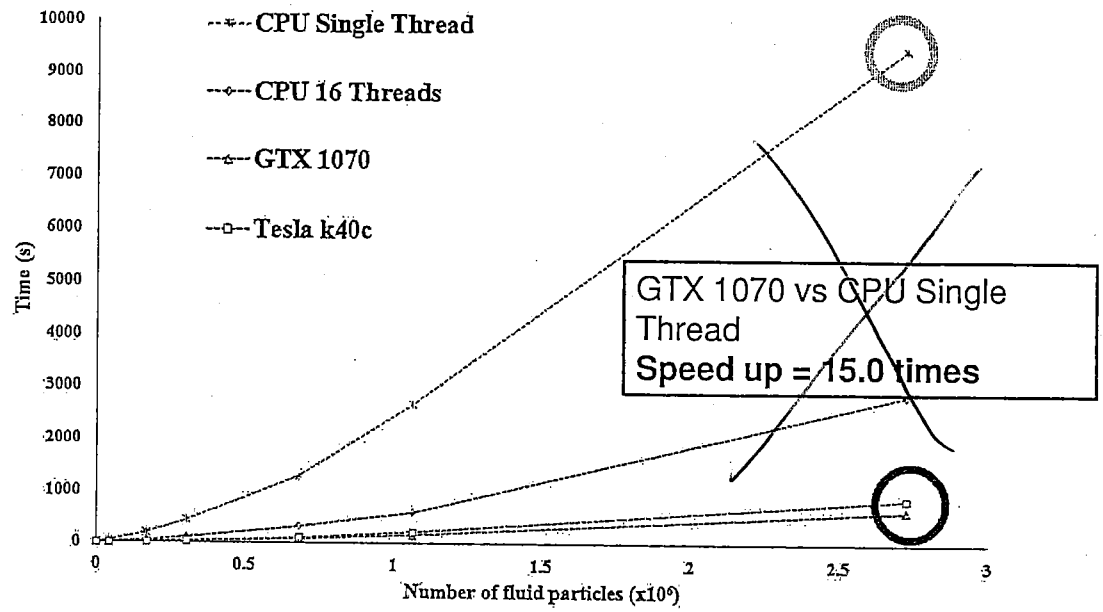
2D Dambreak runtime comparisons



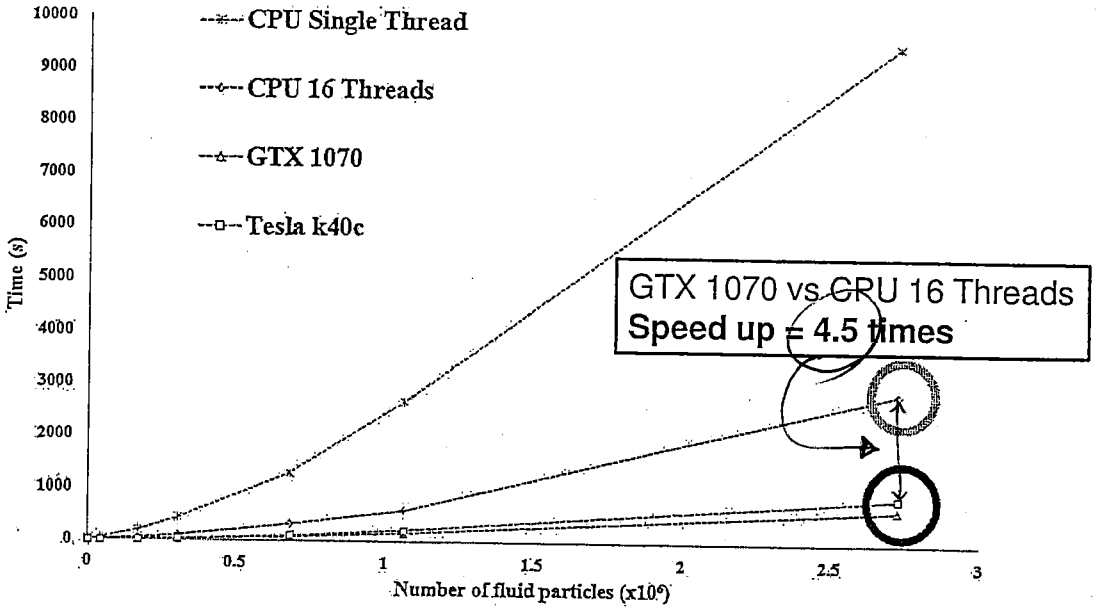
ISPH on a GPU (Chow et al. 2018)



ISPH on a GPU (Chow et al. 2018)



ISPH on a GPU (Chow et al. 2018)



ISPH on a GPU (Chow et al. 2018)

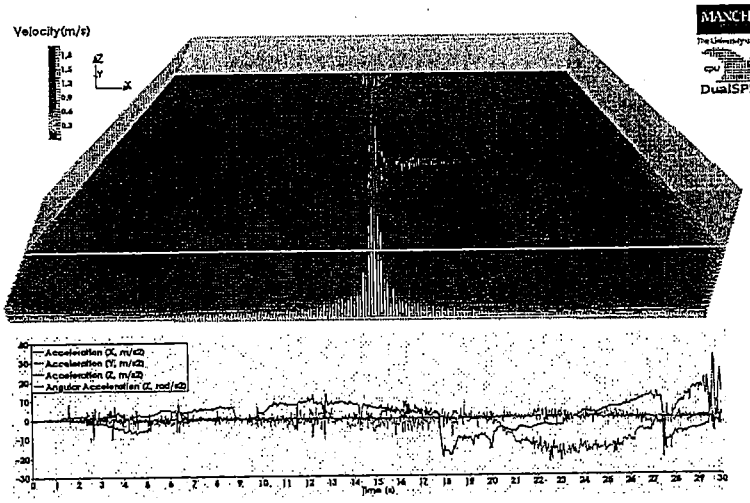
Fuel-tank sloshing with Leading Motorsport Company (F1)

Real engineering problems are now accessible

Only allowed to show highly simplified geometry (NDA)

Accelerations are up to 5*g* ≃ fighter jet

Comparisons with in-tank footage were close.



- Lost race because engine stopped (even with Fuel) ⇒ because fuel was far from pump in the tank!

cpu gpu DualSPHysics

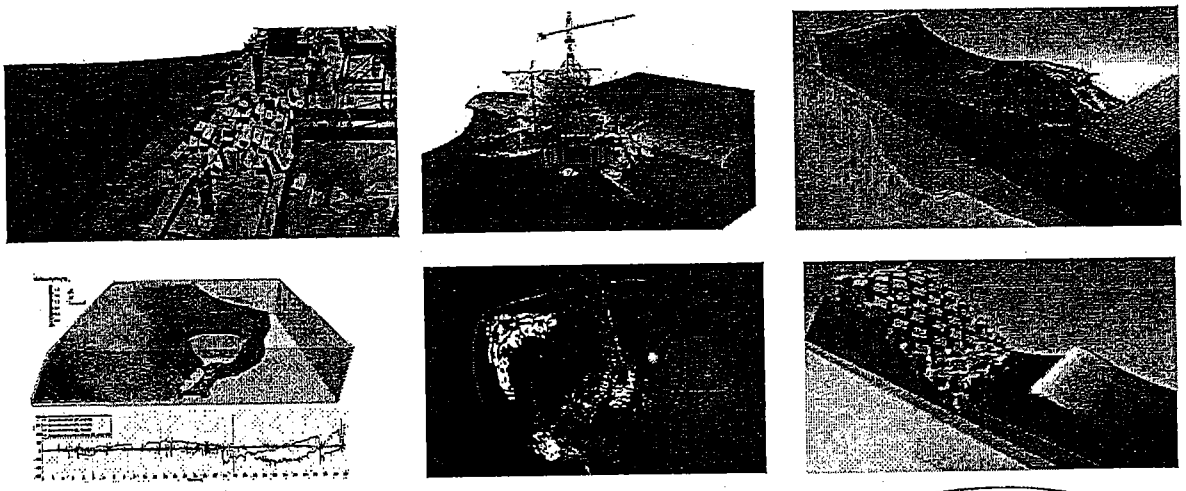
Longshaw & Rogers (2015), Advances Engineering Software

Funded by Knowledge Transfer Account (KTA), *now the IAA*

DualSPHysics Project:

runs on multi-core CPU or GPU

cpu gpu DualSPHysics

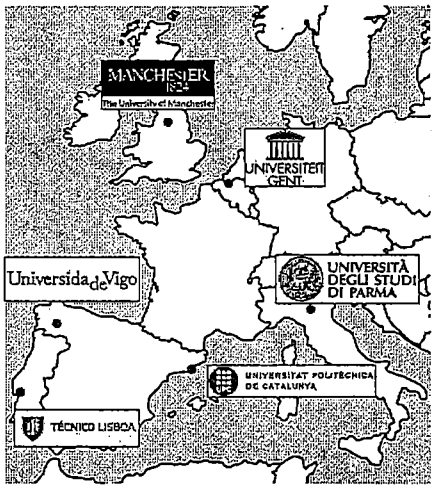


2nd DualSPHysics Users Workshop
University of Manchester · 13-14 September 2016

5th Users Workshop
March 2020, Barcelona, Spain

DualSPHysics Project:

- University of Manchester
- University of Vigo (Spain)
- University of Parma (Italy)
- University of Lisbon (Portugal)
- University of Ghent (Belgium)



Websites

- Free open-source **SPHysics** code:
  http://www.sphysics.org
  http://www.dual.sphysics.org

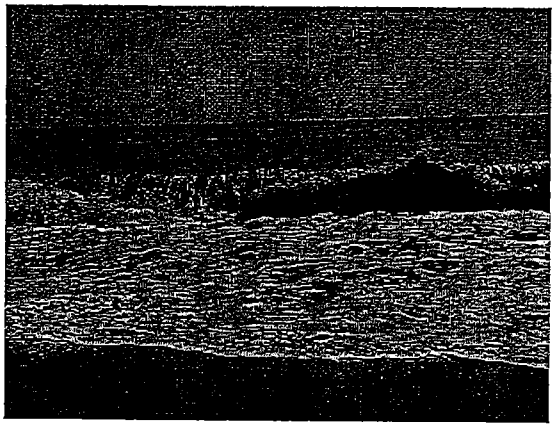
cpu gpu
DualSPHysics

SPHysics

Downloaded 15,000+ times: Open-source plug & play SPH code for free-surface flow

Okay, BUT

Everything there was **mono-phase**



(Photo courtesy of F. Raichlen)

MANCHESTER
1824

SPH free-surface Applications

Multi-Phase Modelling: WATER & AIR

Mokos *et al.* (2015, 2017) on GPUs

MANCHESTER 1824

Challenges of a multi-phase GPU code

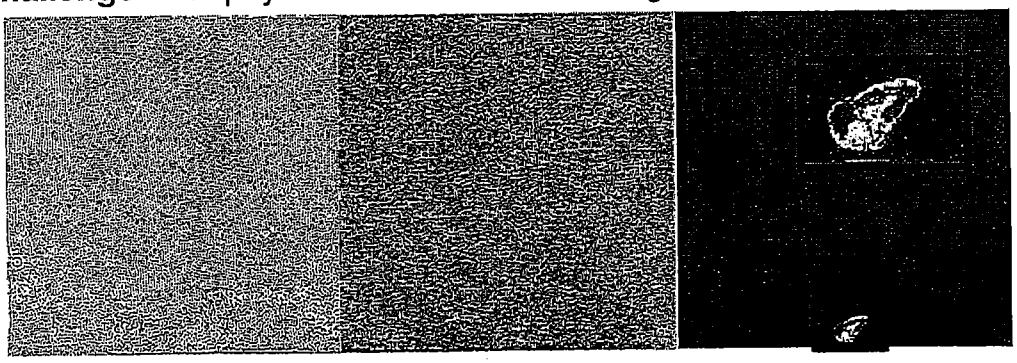
- **Challenge 1**: Interaction of a gas and a liquid phase
  - Large density ratio (~1000)
  - Large pressure gradients in the interface
  - Treatment of the gas phase

Use appropriate formulation (Yildiz' talk)

- **Challenge 2**: Computational treatment
  - SPH is computationally expensive
  - Increased number of particles due to the second phase

Use a GPU

- **Challenge 3**: Unphysical Voids Created at Higher Resolutions



Challenges of a multi-phase GPU code

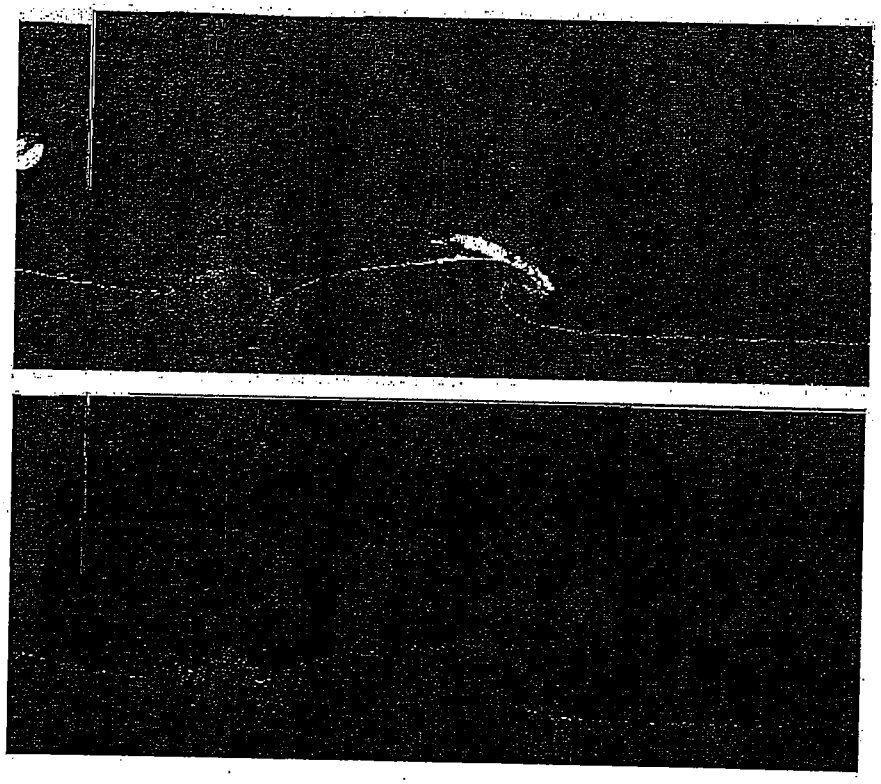
Issue: Voids appear only in high resolutions

Solution: Fickian-based approach by Lind et al. (2012)

- Numerical treatment based on Fick's law: $\delta \mathbf{r}_s = -D\nabla C_i$
- Shifting dependent on particle concentration: $\nabla C_i = \sum_j C_{ij} \frac{m_j}{\rho_j} \nabla W_{ij}$
- Diffusion based on particle velocity (Skillen et al. 2013): $D = -A_s h \left\| u \right\|_i \Delta t$
- Free-surface correction term: $\delta \mathbf{r}_s = -D\left(\frac{\partial C_i}{\partial s}\mathbf{s} + \frac{\partial C_i}{\partial b}\mathbf{b} + \alpha\left(\frac{\partial C_i}{\partial n} - \beta \right)\mathbf{n} \right)$
  - Used only for the liquid phase

Wet Dam Break

- **Original Result**
- **Particle Shifting**



Demands of a multi-phase GPU code

Distinguish particles belonging to different phases

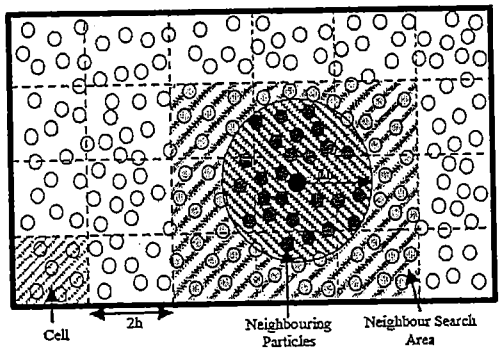
- Load different initial data for each phase
- ID-system recognising phase of each particle (use of *mk*values)

Optimise the multi-phase model

- Different equations used for each phase

Maintain the existing structure of DualSPHysics

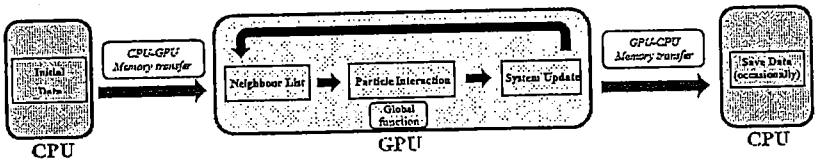
- Integration with other capabilities of the code, such as motion
- Maintain the efficient cell-linked-list structure

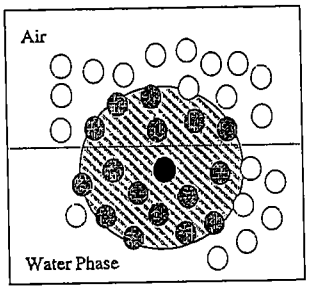


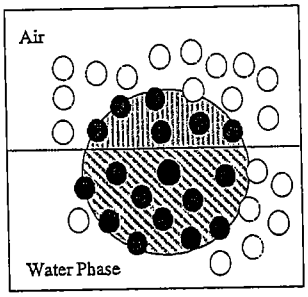
- Figure 1: Example of the Cell-linked List using 2h×2h 2h Cell Neighbour Search Area 2h Neighbouring Particles

Demands of a multi-phase GPU code

Could we use the original code structure?





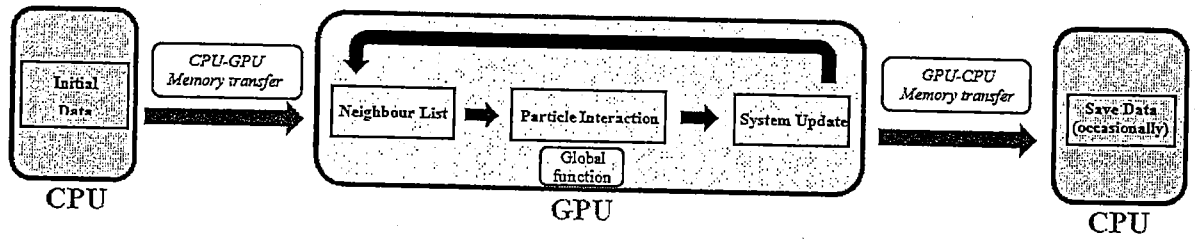


Well, in effect we have 4 choices:

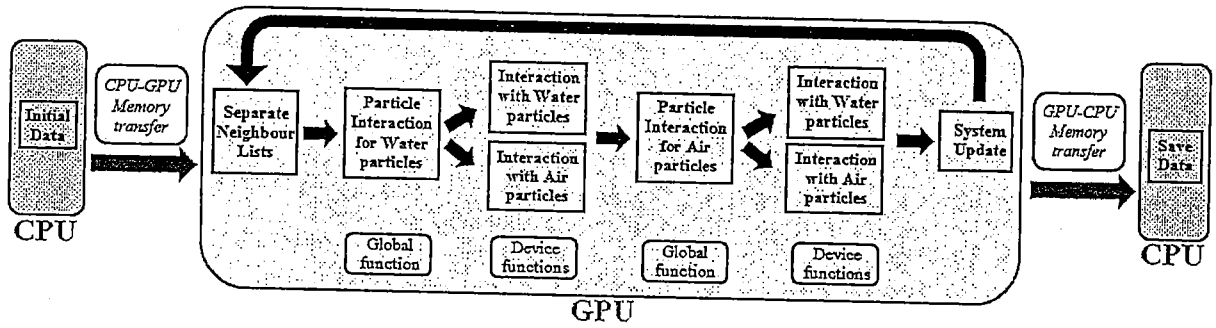
1. Use *if*-statements to distinguish each phase
2. Use binary multipliers (0 or 1 for each phase)
3. Separate neighbour lists + separate global & device functions
4. Separate neighbour lists + Intermediate CPU-GPU function (avoids repeated CPU function calls)

Multi-phase GPU code: separate neighbour lists 2

Original Mono-Phase structure:

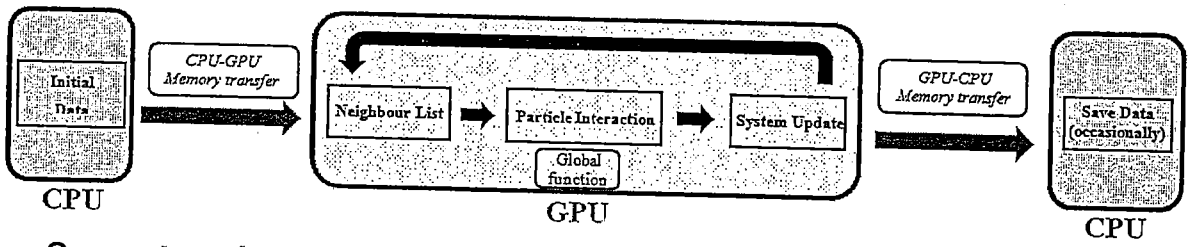


Separate neighbour lists + separate global & device functions):

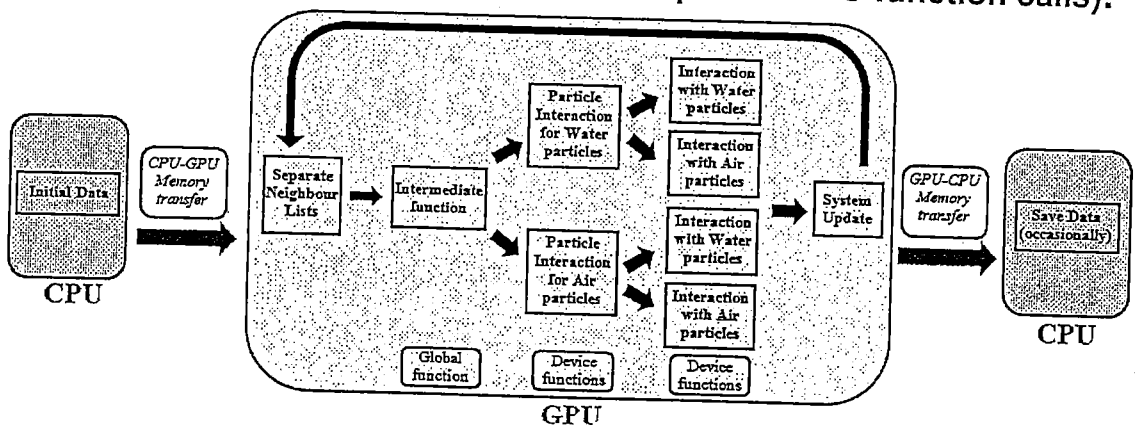


Multi-phase GPU code: separate neighbour lists 2

Original Mono-Phase structure:

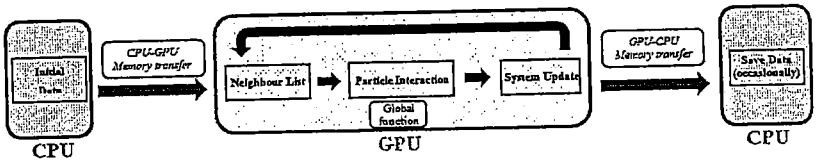


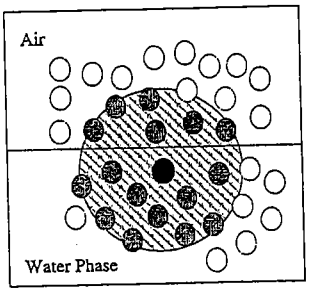
Separate neighbour lists + Intermediate CPU-GPU function
(avoids repeated CPU function calls):

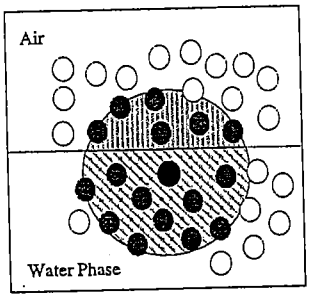


Demands of a multi-phase GPU code

Could we use the original code structure?





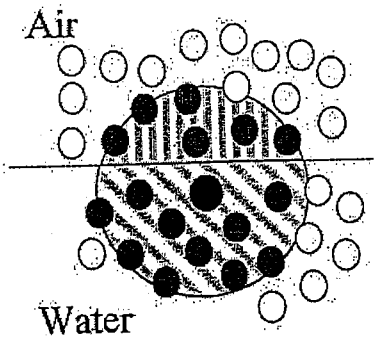


Well, in effect have 4 choices:

1. Is *if*-statements to distinguish each phase
2. Use binary multipliers (0 or 1 for each phase)
3. Separate neighbour lists + separate global & device functions
4. Separate neighbour lists + Intermediate CPU-GPU function (avoids repeated CPU function calls)

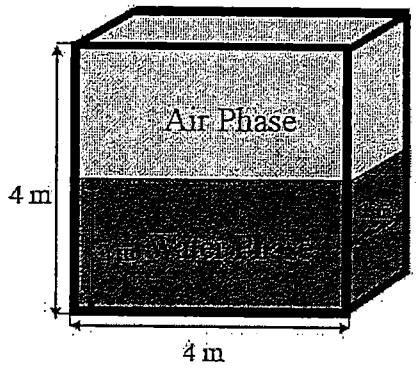
Optimisation of SPH on GPUs

- Calculating inter-particle forces is the most demanding part of SPH
- Research has shown the best practices for optimising:
  - Eliminate conditional *if* statements
  - Reduction of logical operations
  - Minimise CPU-GPU interaction
  - Minimise memory (local and global) transfers
  - Balance computational load on the GPU
  - Separate particle and neighbour lists for each phase is beneficial for large particle numbers

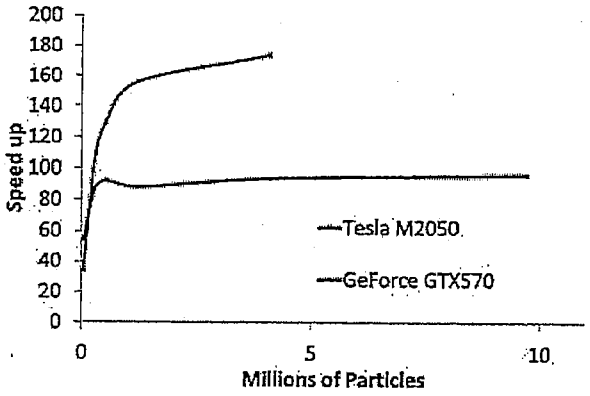


(Mokos et al. 2015)

Runtime Results – 3D

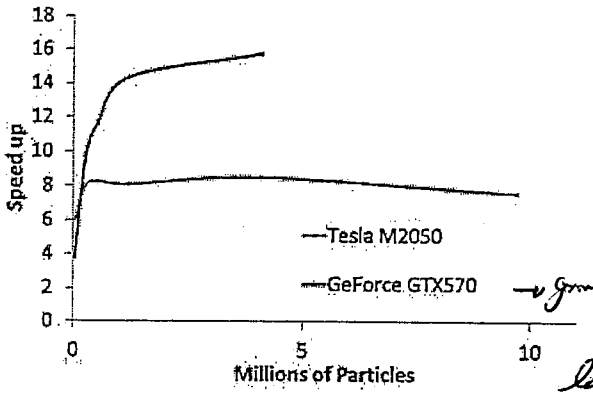


Single Thread



- Results for version 2
- Speed-up is GPU-dependent
- Speedup up to 170 compared to single CPU core

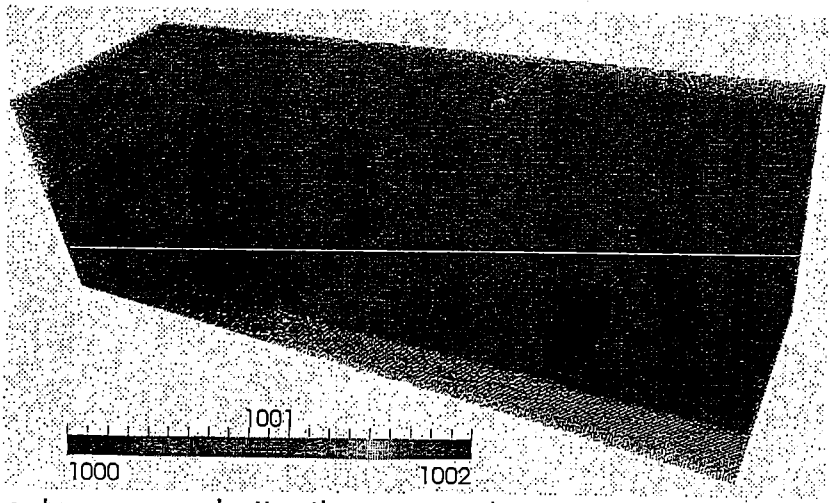
OpenMP



- Speedup up to 16 compared to an 8-thread OpenMP computation

3-D Obstacle Impact: Air & Water (WCSPH)

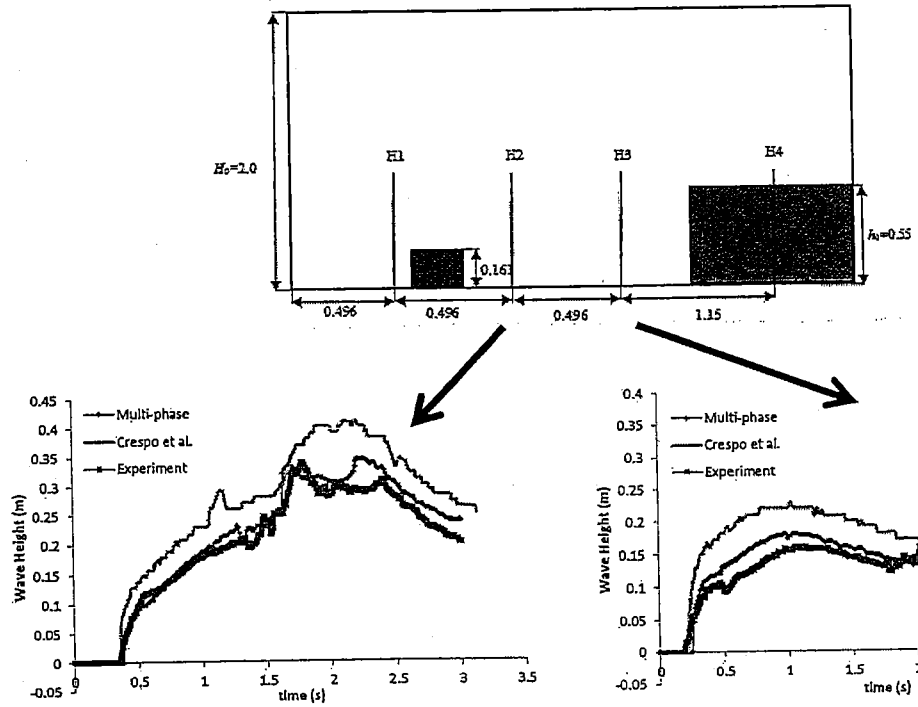
- Each phase is modelled as weakly compressible.
- Simplest SPH multi-phase formulation chosen for ease of GPU implementation



- Water Density
- Agreement for impact pressures better than monophase
- Simulation runtime for 5 million particles :140 hours for 3s!!!
- (we will come back to this later)

Most are AIR particles

3-D Obstacle Impact: Air & Water (WCSPH)



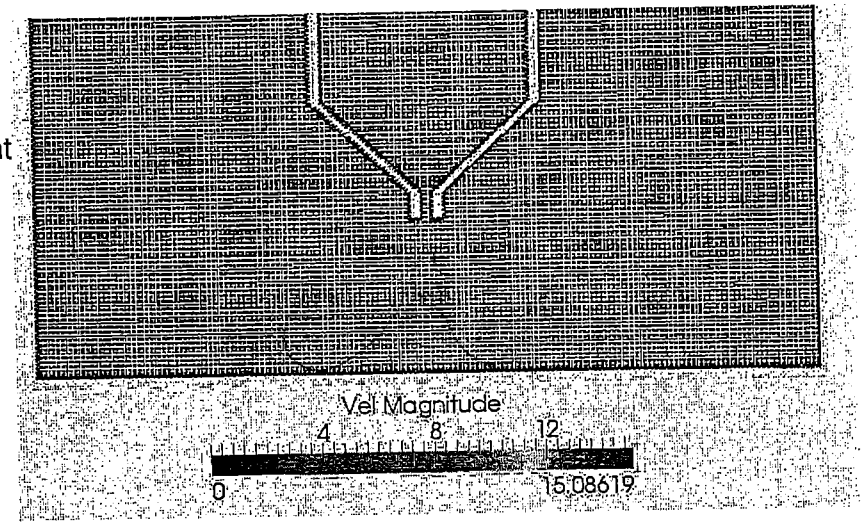
- Agreement for impact pressures better than monophase

Nuclear Applications: mixing

- goal - mix/agitate the waste so that it won't explode!

Submerged jet impinging on sediment

- Configuration
  - $\rho_s = 1.54\ \rho_w$
  - $\mu_s^{cr} = 5x10^3\ \mu_w$
  - Cohesive sediment
  - 60 000 particles



DualSPHysics

Fourtakas & Rogers (2016), Advances Water Resources

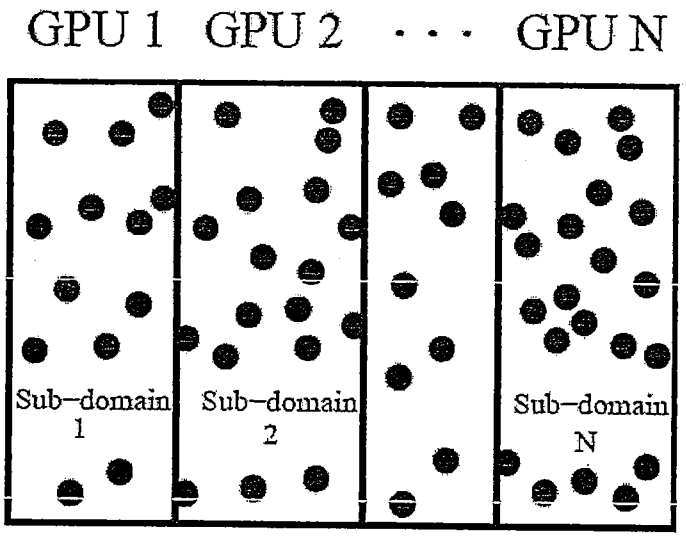
NATIONAL NUCLEAR LABORATORY

Multi-GPU for SPH

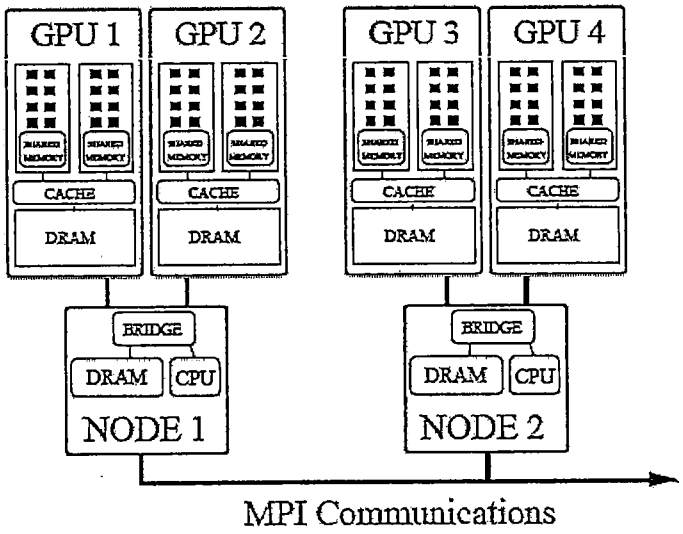
Baldez-Valderas *et al.* (2013)
Dominguez *et al.* (2013)

MANCHESTER
1824

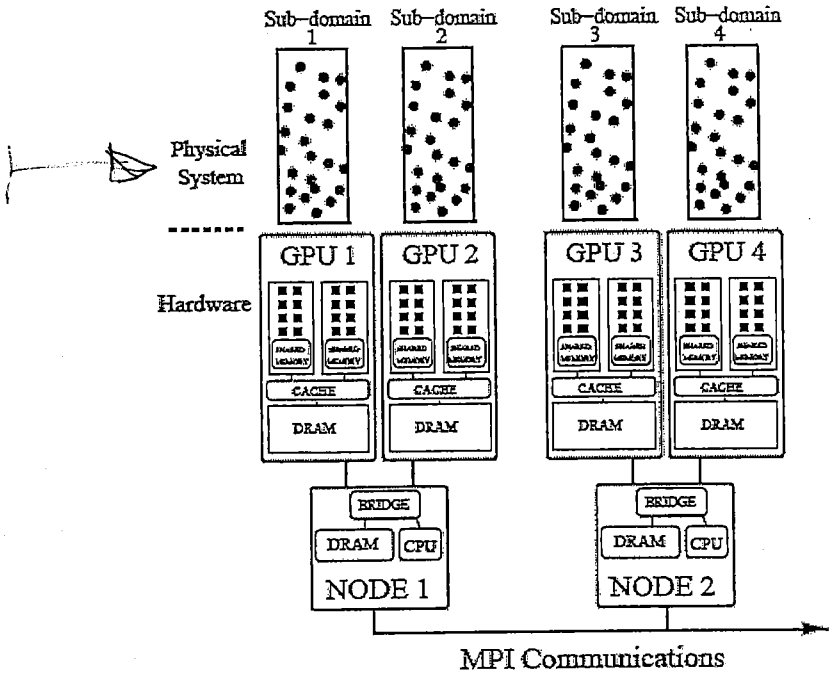
General strategy: spatial decomposition



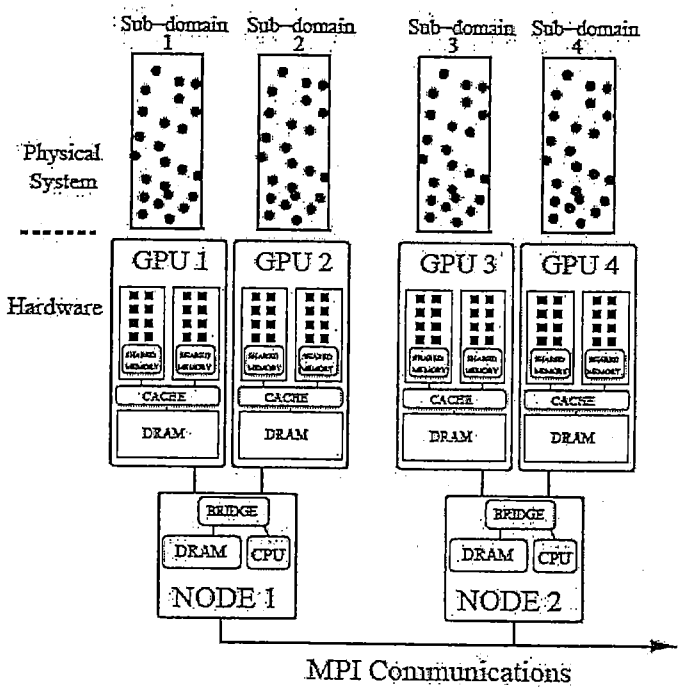
General strategy: a multi-GPU system



General strategy: a multi-GPU system



General strategy: a multi-GPU system



Additional tasks:

- Particle migration
- Determination of halo

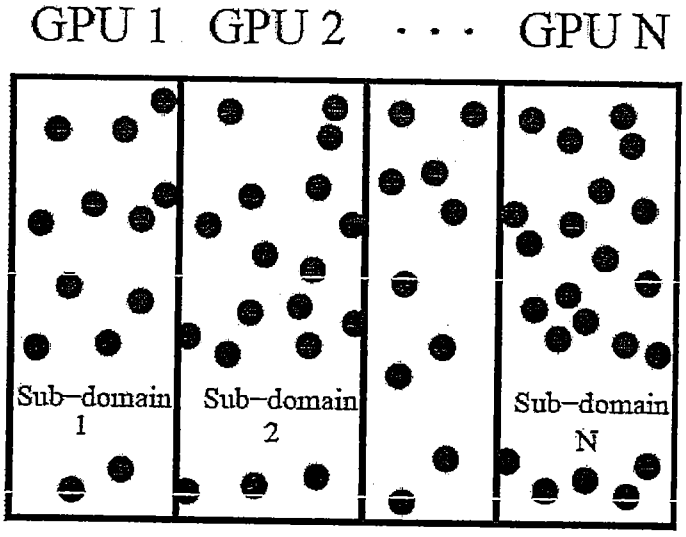
Overhead caused by

- Data preparation
- GPU-CPU data transfers
- Inter-CPU communications

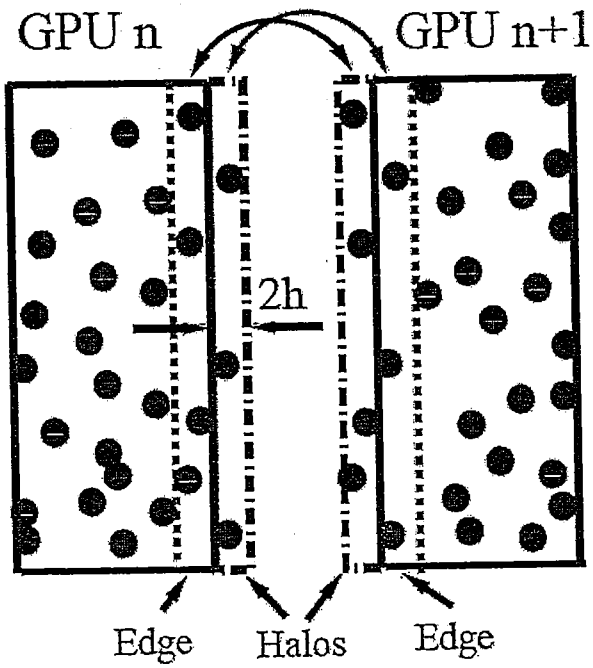
Question:

- Where are the bottlenecks: CPU-CPU or GPU-CPU communications, data preparation?

General strategy: spatial decomposition



General strategy: halo building

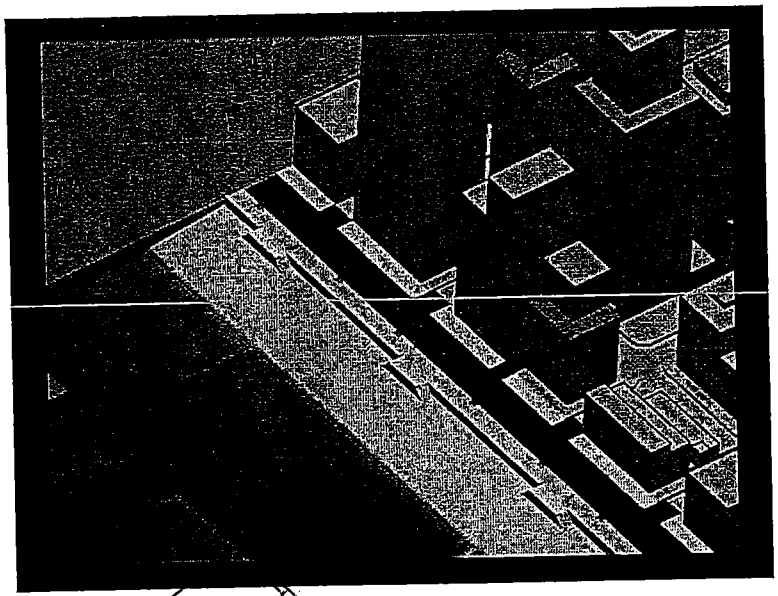


MANCHESTER 1824

The University of Manchester

Multi-GPU DualSPHysics

- Dual-SPHysics (cpu & gpu) – SPEEDUPS of **400+** for a Desktop machine = Supercomputer of 1000's cores
- **Valdez-Balderas et al. (2013)**



cpu gpu

DualSPHysics

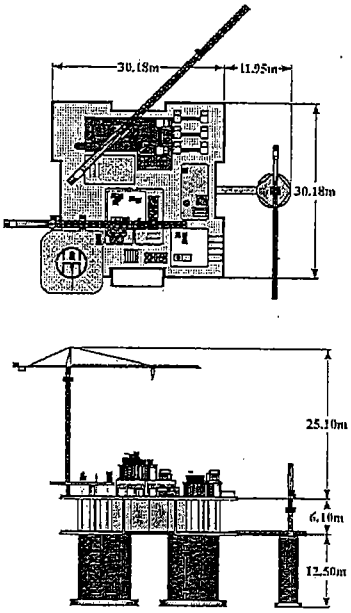
4 GPUs	2.1 hours
1 GPUs	5.7 hours
1 CPU (4 core)	3.5 days
1 CPU (1 core)	2 weeks

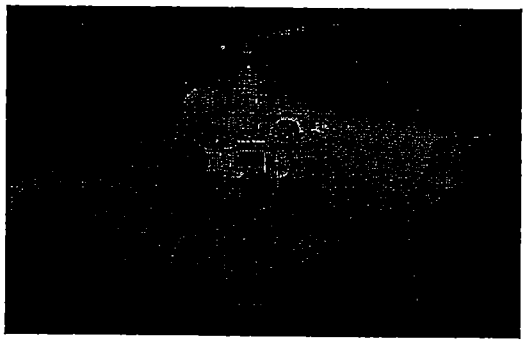
5.3 million particles.

GPU0 GPU1 GPU2 GPU3

SPH Application with MultiGPU DualSPHysics

Large wave interaction with oil rig using 1 BILLION (10^9) particles





Click here for Movie 1

Dominguez et al. (2013) Computer Physics Comm.

We have post-processing tools efficient enough to visualise the simulation of more than 1000M

dp = 6 cm
np = 1,015,896,172 particles
nf = 1,004,375,142 fluid particles
physical time = 12 sec
# of steps = 237,342 steps
runtime = 91.9 hours

using 64 GPUs Tesla 2090 of the BSC

Conclusions & Outlook

- SPH is a costly method which needs hardware acceleration
- Like many computational methods, various options exist
- Massive parallelization is possible but really requires sophisticated techniques
- More attractive is to use efficient computational devices
- → Graphics Processing Units (GPUs)
- Complex physics with Multi-Phase requires careful work

SPHysics

Parallel Shopping with MPI

- SPH has excessively long runtimes: weeks and months
- Uses MPI formalism (MPI = Message Passing Interface)

- *à Paris*:

Thank you

Acknowledgments

- U-Man: Peter Stansby, Steve Lind, George Fourtakas, Abouzied Nasar
- U-Vigo: Alex Crespo, Jose Dominguez, Moncho Gomez-Gesteira
- U-Parma: Renato Vacondio FHR: Corrado Altomare

Websites

- Free open-source
- **SPHysics** codes:

http://www.sphysics.org

http://www.dual.sphysics.org

cpu gpu DualSPHysics SPHysics

SPH@Manchester https://sph-manchester.weebly.com

International SPH organisation: @SPH_Manchester

SPH research and engineering international community = SPHERIC

http://spheric-sph.org

REFERENCES

Chow, A.D., Rogers, B.D., Stansby, P.K., Lind, S.J., 2018, Incompressible SPH (ISPH) with fast Poisson solver on a GPU, *Computer Physics Communications*, 26, 81-103, doi: 10.1016/j.cpc.2018.01.005.

Crespo AJC, Domínguez JM, Rogers BD, Gómez-Gesteira M, Longshaw S, Canelas R, Vacondio R, Barreiro A, García-Feal O. 2015. DualSPHysics: open-source parallel CFD solver on Smoothed Particle Hydrodynamics (SPH). *Computer Physics Communications*, 187: 204-216. doi: 10.1016/j.cpc.2014.10.004.

Domínguez, J.M., Crespo, A.J.C., Gómez-Gesteira, M., Marongiu, J.C., 2010, Neighbour lists in Smoothed Particle Hydrodynamics. *International Journal for Numerical Methods in Fluids*. doi: 10.1002/fld.2481.

Domínguez, J.M., Crespo, A.J.C., Valdez-Balderas, D., Rogers, B.D. and Gómez-Gesteira M. 2013. New multi-GPU implementation for Smoothed Particle Hydrodynamics on heterogeneous clusters. *Computer Physics Communications*, 184: 1848-1860. doi:10.1016/j.cpc.2013.03.008.

Fourtakas G., Rogers B.D. 2016. Modelling multi-phase liquid-sediment scour and resuspension induced by rapid flows using Smoothed Particle Hydrodynamics (SPH) accelerated with a graphics processing unit (GPU). *Advances in Water Resources*, 92: 186-99. doi:10.1016/j.advwatres.2016.04.009.

REFERENCES

Guo, X., Rogers, B.D., Lind, S.J., Stansby, P.K., 2018, New massively parallel scheme for Incompressible Smoothed Particle Hydrodynamics (ISPH) for highly nonlinear and distorted flow, *Computer Physics Communications*, 233, 16-28, doi: 10.1016/j.cpc.2018.06.006

Longshaw, S.M., Rogers, B.D. 2015. Automotive Fuel Cell Sloshing Under Temporally and Spatially Varying High Acceleration Using GPU Based Smoothed Particle Hydrodynamics (SPH). *Advances in Engineering Software*, 83: 31–44. doi: 10.1016/j.advengsoft.2015.01.008.

Mokos A, Rogers B.D., Stansby P.K., Domínguez J.M. 2015. Multi-phase SPH modelling of violent hydrodynamics on GPUs. *Computer Physics Communications*, 196: 304-316. doi: 10.1016/j.cpc.2015.06.020.

Mokos, A., Rogers, B.D., Stansby, P.K. 2016. A multi-phase particle shifting algorithm for SPH simulations of violent hydrodynamics with a large number of particles. *Journal of Hydraulic Research*, 55 (2), 143-162. doi: 10.1080/00221686.2016.1212944.

Valdez-Balderas D., Domínguez, J.M., Rogers, B.D., Crespo, A.J.C. 2013. Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters. *Journal of Parallel and Distributed Computing*, 73(11): 1483-1493. doi:10.1016/j.jpdc.2012.07.010.