Converting DualSPHysics to solve strictly Incompressible SPH

Alex Chow, Benedict D. Rogers, Peter K. Stansby, Steven Lind

School of Mechanical, Aeronautical and Civil Engineering
University of Manchester
UK

PhD research funded by the EPSRC and UoM Research Impact Fund

Outline of Presentation

- Motivations for research / PhD project aims
- WCSPH vs ISPH
- Why DualSPHysics?
- ISPH on the GPU in DualSPHysics implementation challenges
- Methodology
  - Which DualSPHysics files to change?
  - ISPH projection step
  - Inserting new functions (for solving the Pressure Poisson Equation)
  - Implementing open-source linear solver libraries
  - Boundary conditions
- Research challenges
- Results and Simulations
- Conclusions & Future Developments

Motivations and project aims

Renewable energy is required!

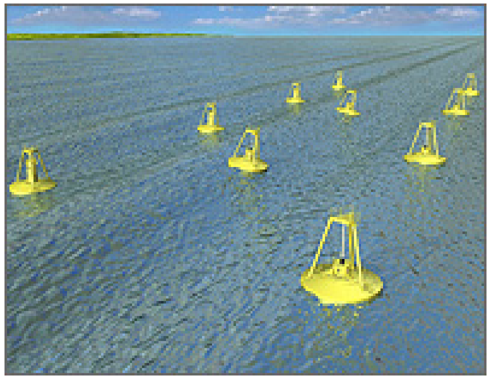
- Recent trends show an increase in offshore development for renewable energies (EWEA, 2016)
- Offshore environments are harsh and difficult to design efficiently and cost effectively

Project aims:

- To create a computational model for solving **incompressible free-surface flows**
- For modelling breaking wave-structure impacts

The European Wind Energy Association (EWEA), *Offshore statistics*. (2016) doi: http://www.ewea.org/statistics/offshore/

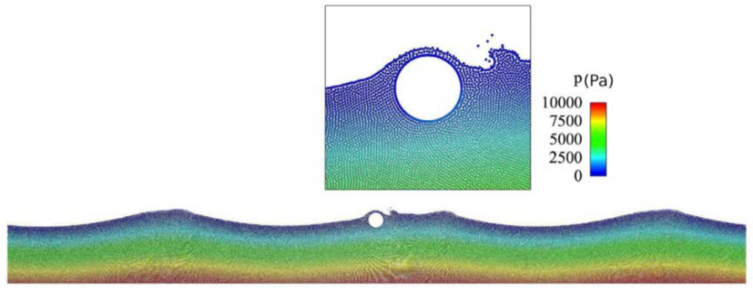
Wind Turbines


Wave Energy Devices

Tidal Stream Turbines

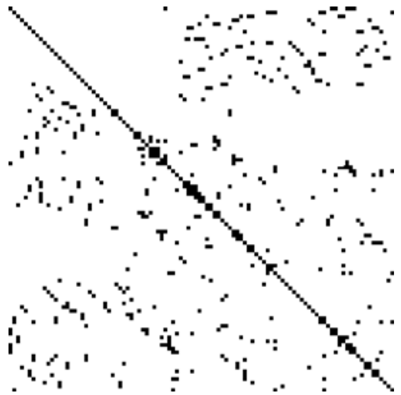
Motivations and project aims

- Incompressible SPH (ISPH) is ideal to model free-surface flows with a smooth, noise-free pressure field.



Wave impact on a circular cylinder (Skillen et al., 2013)

- However, **ISPH has slow computational times.**
    - Due to the need for the **solution of a pressure Poisson equation matrix**, Ax=B



Pressure Poisson equation matrix

- Use of **Graphics Processing Unit (GPU)** to speed things up.

- Hasn't been investigated properly before!

A. Skillen, S. Lind, P. K. Stansby, and B. D. Rogers, *Incompressible smoothed particle hydrodynamics (SPH) with reduced temporal noise and generalised Fickian smoothing applied to body-water slam and efficient wave-body interaction.* Comput. Methods Appl. M. (2013) 265:163-173.

WCSPH vs ISPH

- **Traditional (Weakly-Compressible) SPH uses an artificial equation of state to link density and pressure**, allowing density to vary by 1%.
- **ISPH enforces incompressibility and solves pressure through a pressure Poisson equation (PPE)** in the form of a sparse matrix.

$$p = B\left[\left(\frac{\rho}{\rho_{ref}}\right)^{\gamma} - 1\right]$$

The WCSPH artificial equation of state

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

The ISPH pressure Poisson equation is solved using a system matrix in the form Ax=B

- The **ISPH PPE is the most computationally expensive** part of the algorithm, in terms of **both memory and time.**

ISPH Pressure Poisson Equation

For a 1 million particle simulation, the PPE matrix has (when h/dx=1.3) :

- Approximately 20 million non-zero elements in 2D
- Approximately 100 million non-zero elements in 3D

To be solved every timestep

For this project, the application of a breaking wave–structure impacts will require **several million particles** in a simulation

Why use DualSPHysics?

- DualSPHysics is a WCSPH code BUT there are still **common functions and variables between ISPH and WCSPH** that can be reused: **neighbourlist and particle-reordering, kernel calculations etc.**
- Already highly optimised.
- Implementing in CPU then transferring to GPU is easier than straight to GPU.
- DualSPHysics has a high range of functionality: **floating objects, wavemaker, DEM coupling, multi-phase etc.** Not the biggest concern during implementation but relatively easy to include in the future.

ISPH on the GPU in DualSPHysics implementation challenges

ISPH on the GPU:

- **Memory expensive** method for **memory limited hardware**.
- **Higher resolutions** = Higher matrix condition numbers
  = **Longer PPE solve times**
- **Solving Poisson equations for particle methods on GPUs** is a relatively **new area for research**

Implementation into DualSPHysics:

- Large piece of code - difficult to fully understand the inner workings
  - Challenges in identifying variables to keep/modify/extract
  - Maintaining consistency in the code (parameter values, precision etc.), especially between CPU and GPU.
  - Challenges in adhering to existing DualSPHysics code when adding new code.

Methodology – Files to change

	CPU	GPU
Arrangement of computation	JSphCpuSingle.cpp JSphCpuSingle.h	JSphGpuSingle.cpp JSphGpuSingle.h
Executing computation	JSphCpu.cpp JSphCpu.h	JSphGpu.ker.cu JSphGpu_ker.h
Adding new variables	JSphCpu.h	JSphGpu.h
Adding new memory	JSphCpu.cpp	JSphGpu.cpp
Adding new parameters	JSph.cpp JSph.h Types.h	JSph.cpp JSph.h Types.h

Methodology – Implementing the ISPH projection step

$$\mathbf{r}_i^* = \mathbf{r}_i^n + \Delta t \mathbf{u}_i^*$$

Initial advection of particles

$$\mathbf{u}_i^* = \mathbf{u}_i^n + \Delta t \sum_j V_j \frac{2\nu \mathbf{r}_{ij}^* . \nabla_i W_{ij}}{\mathbf{r}_{ij}^{*2} + \eta^2} \mathbf{u}_{ij}^n$$

Find intermediate velocity from viscous forces

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

Find pressure from the pressure Poisson equation (PPE)

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^* - \Delta t \left(\frac{\nabla p_i^{n+1}}{\rho} + \mathbf{F}_i^n \right)$$

Correct the velocity with pressure and external forces

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \Delta t \left(\frac{\mathbf{u}_i^{n+1} + \mathbf{u}_i^n}{2} \right)$$

Correct the position with the new velocity

$$\delta \mathbf{r_s} = -D \nabla C$$

Shift particles to improve distribution

Methodology – Implementing the ISPH projection step

$$\mathbf{r}_i^* = \mathbf{r}_i^n + \Delta t \mathbf{u}_i^*$$

$$\mathbf{u}_i^* = \mathbf{u}_i^n + \Delta t \sum_j V_j \frac{2\nu \mathbf{r}_{ij}^* . \nabla_i W_{ij}}{\mathbf{r}_{ij}^{*2} + \eta^2} \mathbf{u}_{ij}^n$$

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

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^* - \Delta t \left(\frac{\nabla p_i^{n+1}}{\rho} + \mathbf{F}_i^n \right)$$

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \Delta t \left(\frac{\mathbf{u}_i^{n+1} + \mathbf{u}_i^n}{2} \right)$$

$$\delta \mathbf{r}_s = -D \nabla C$$

DualSPHysics Symplectic step

Predictor

$\frac{d\rho}{dt} \quad \frac{d\mathbf{u}_i}{dt}$	Interaction_Forces()
$\delta \mathbf{r}_s = -D \nabla C$	RunShifting()
(Particle update)	ComputeSymplecticPre()

Corrector

$\frac{d\rho}{dt} \quad \frac{d\mathbf{u}_i}{dt}$	Interaction_Forces()
$\delta \mathbf{r}_s = -D \nabla C$	RunShifting()
(Particle update)	ComputeSymplecticCorr()

Methodology – Implementing the ISPH projection step

Predictor

$$\boxed{\mathbf{r}_i^* = \mathbf{r}_i^n + \Delta t \mathbf{u}_i^*} \longrightarrow \text{Init\_Advection()}$$

$$\frac{d\rho}{dt} \quad \frac{d\mathbf{u}_i}{dt}$$

$$\boxed{\mathbf{u}_i^* = \mathbf{u}_i^n + \boxed{\Delta t \sum_j V_j \frac{2\nu \mathbf{r}_{ij}^*.\nabla_i W_{ij}}{\mathbf{r}_{ij}^{*2} + \eta^2} \mathbf{u}_{ij}^n}} \longrightarrow \text{Interaction\_Forces(ForcesPre)}$$

~~$\delta \mathbf{r}_s = -D\nabla C$ RunShifting()~~

(Particle update) → ComputeSymplecticPre()

$$\boxed{\nabla.\left(\frac{1}{\rho}\nabla p^{n+1}\right)_i = \frac{1}{\Delta t}\nabla.\mathbf{u}_i^*} \longrightarrow \text{SolvePPE()}$$

Corrector

$$\frac{d\rho}{dt} \quad \frac{d\mathbf{u}_i}{dt}$$

$$\boxed{\mathbf{u}_i^{n+1} = \mathbf{u}_i^* - \boxed{\Delta t\left(\frac{\nabla p_i^{n+1}}{\rho} + \mathbf{F}_i^n\right)}} \longrightarrow \text{Interaction\_Forces(ForcesCorr)}$$

~~$\delta \mathbf{r}_s = -D\nabla C$ RunShifting()~~

$$\boxed{\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \Delta t\left(\frac{\mathbf{u}_i^{n+1} + \mathbf{u}_i^n}{2}\right)}$$

(Particle update) → ComputeSymplecticCorr()

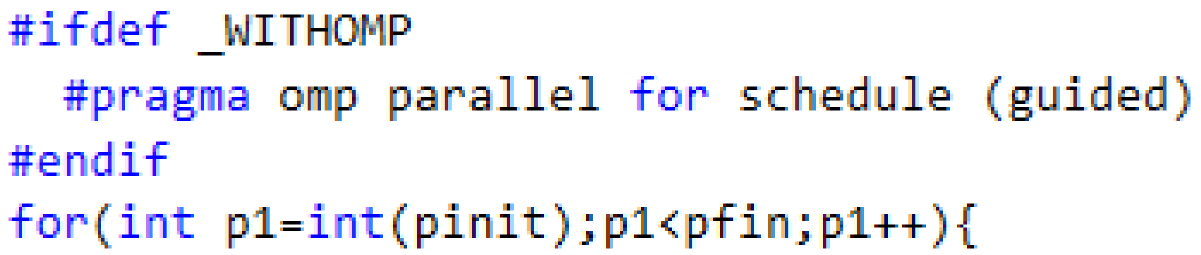
$$\boxed{\delta \mathbf{r}_s = -D\nabla C} \longrightarrow \text{RunShifting()}$$

Methodology – Inserting new computation functions for the PPE setup

- Use existing code as a template and insert necessary equations within

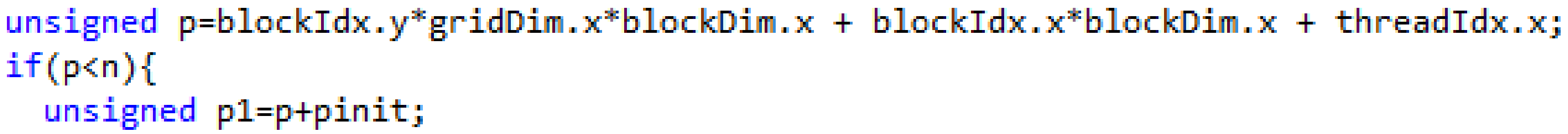
For Example:

<u>CPU – Begin loop through particles</u>



```
#ifdef _WITHOMP
  #pragma omp parallel for schedule (guided)
#endif
for(int p1=int(pinit);p1<pfin;p1++){
```

<u>GPU – Begin loop through particles</u>



```
unsigned p=blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
if(p<n){
  unsigned p1=p+pinit;
```

Other code that can be reused:

- Calling the neighbour list for each particle
- Calculating the particle interactions for each

Methodology – Implementing a sparse linear solver library for solving the PPE

- Currently using an **open-source sparse matrix solver library** for the CPU and GPU to setup a **preconditioner for the matrix and solve the PPE system.**

- Using an appropriate open-source library to solve the matrix is good:
  - Highly optimised.
  - Saves coding time.
  - Provides different options for preconditioners and linear solvers.

Methodology – Implementing a sparse linear solver library for solving the PPE

- This project is currently using the ViennaCL Library (Rupp, 2010)
  - OpenMP and CUDA availablility

- To include in DualSPHysics, insert the library file path into the properties:
  - **CPU:** C/C++ ->Additional Include Directories
  - **GPU:** JSphGpu_ker.cu
    ->Properties->Custom Build Tool->Command Line

- Insert necessary "#include" library files into where needed, JSphCpu.cpp / JSphGpu_ker.cu etc.
  - **IF using for CPU and GPU, encase "#include" files with "#ifndef _WITHGPU" "#endif" to avoid CPU/GPU conflicts**

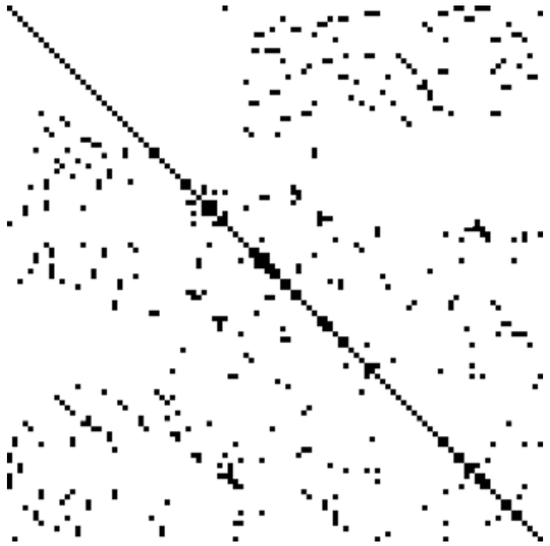
K. Rupp, J. Weinbub, and F. Rudolf. Automatic performance Optimization in ViennaCL for GPUs. In *POOSC '10: Proceedings of the 9th Workshop on parallel/High-Performance Object-Oriented Scientific Computing,* pages 1-6, New York, NY, USA, 2010. ACM.

Methodology – Solving the PPE

- The ISPH PPE is a **non-symmetric sparse matrix**.

- The matrix represents a system in the form, Ax=B.

- Common solvers in ISPH literature used are GMRES and **BiCGSTAB**.



- **High Resolutions =** large condition numbers **= Long solve times**

Methodology – PPE preconditioning

Reducing the solution time: Preconditioning

- Preconditioning lowers the condition number of system so it is faster to solve. For a preconditioner, applied to matrix A :
  - $Ax=b \rightarrow PAP^TPx=b$

- **Preconditioning** the matrix system is **necessary for convergence** of a solution at **high resolutions**.

- In ISPH, the Jacobi preconditioner is typically used.

- Jacobi preconditioning is simple and memory effective.

But…

Methodology – PPE preconditioning

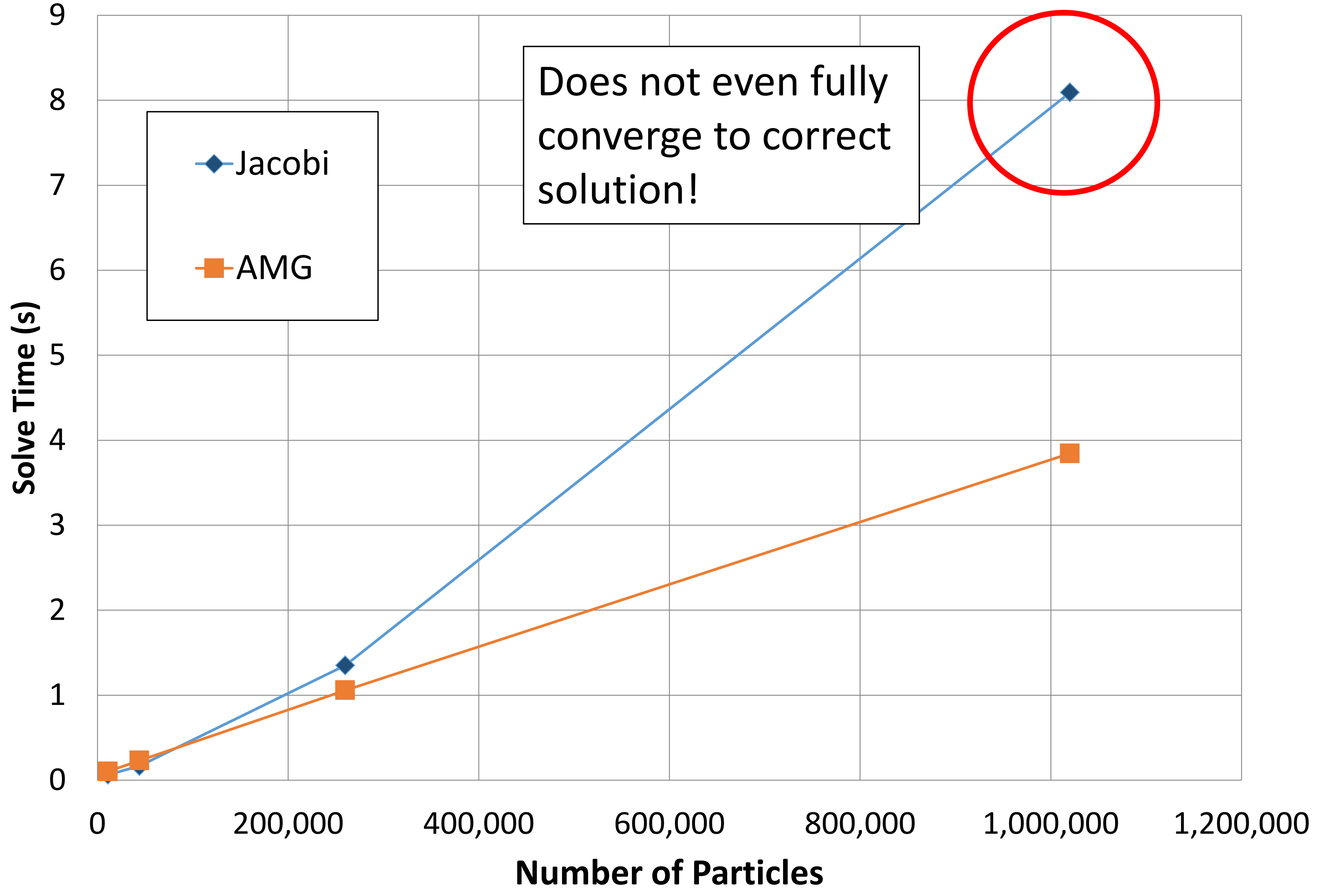
- Guo et al. (2013) has shown the Jacobi does not scale well for high resolutions and may not even converge to a solution.
- Instead he uses an Algebraic Multigrid preconditioner.

Solution:

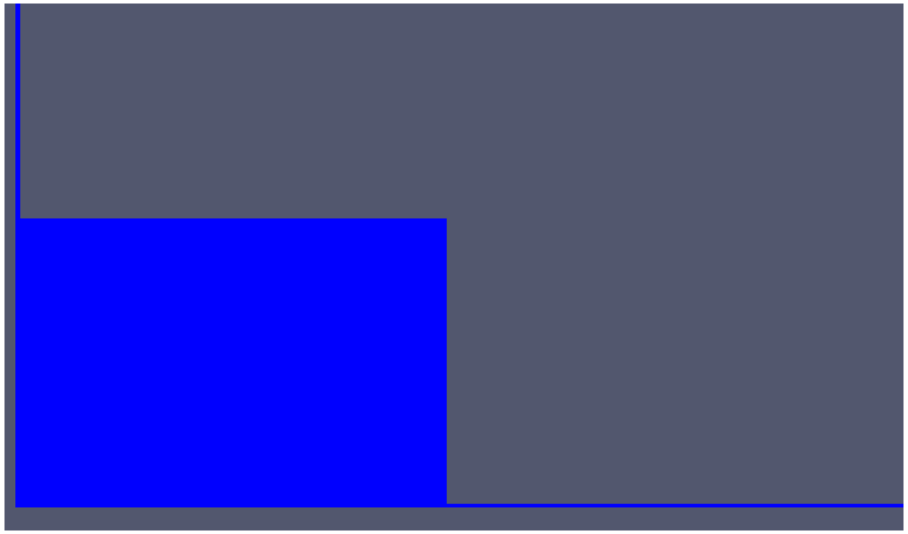
- **The Algebraic Multigrid (AMG) preconditioner** scales much better at higher resolutions and provides a very quick solution time.

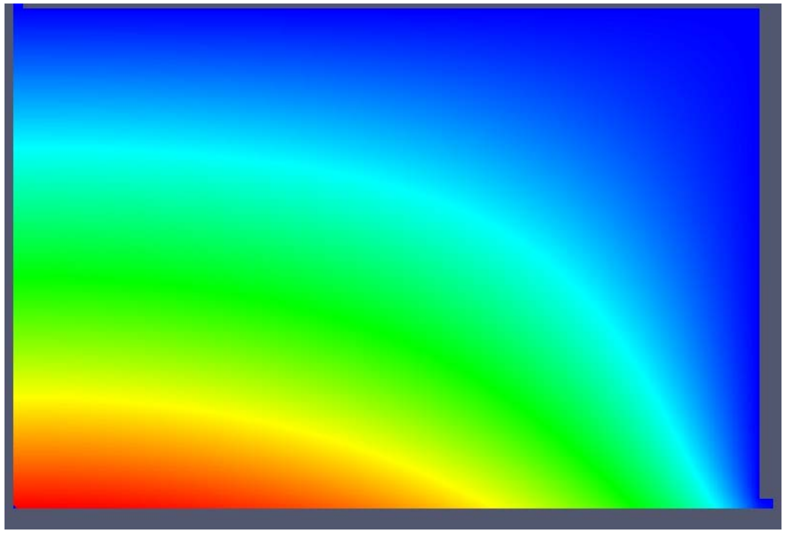
X. Guo, S. J. Lind, B. D. Rogers, P. K. Stansby, and M. Ashworth, *Efficient Massive Parallelisation for Incompressible Smoothed Particle Hydrodynamics with 10^8 Particles,* Proceedings of the 8th international SPHERIC workshop (2013)

GPU Still Water Preconditioner Total Solve Times

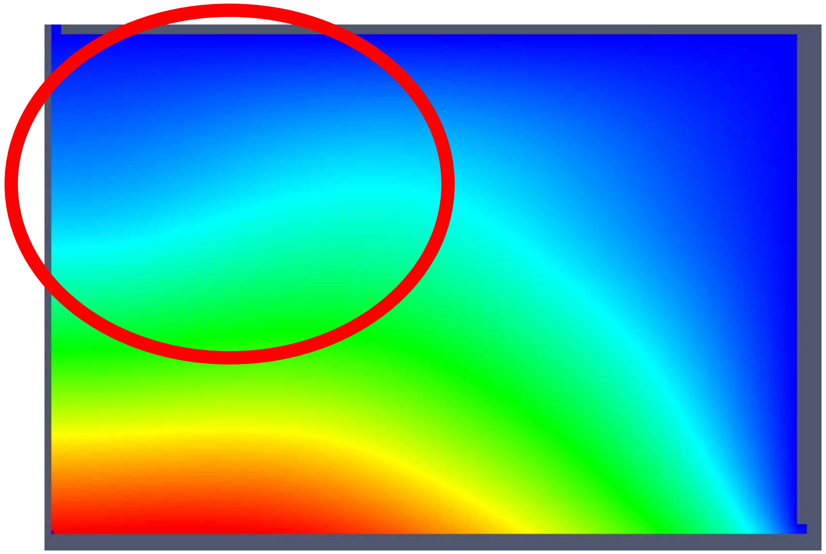


Methodology – PPE preconditioning

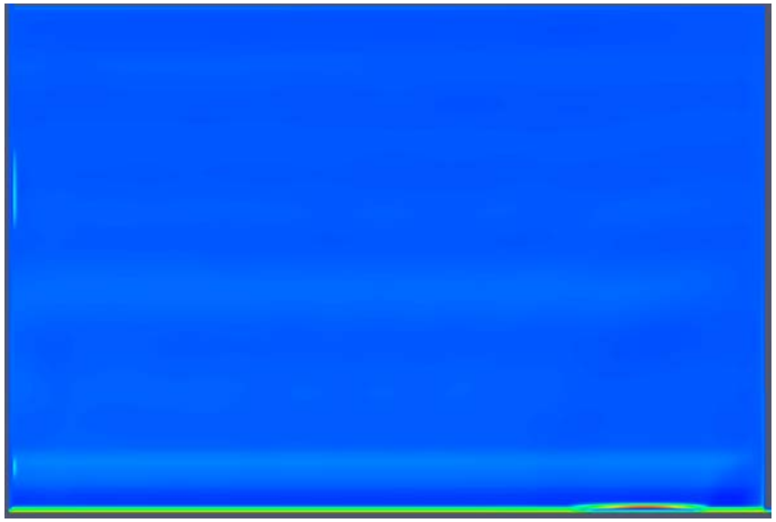




AMG: 1,577,091 particles



Jacobi: 270,891 particles



Jacobi: 1,577,091 particles

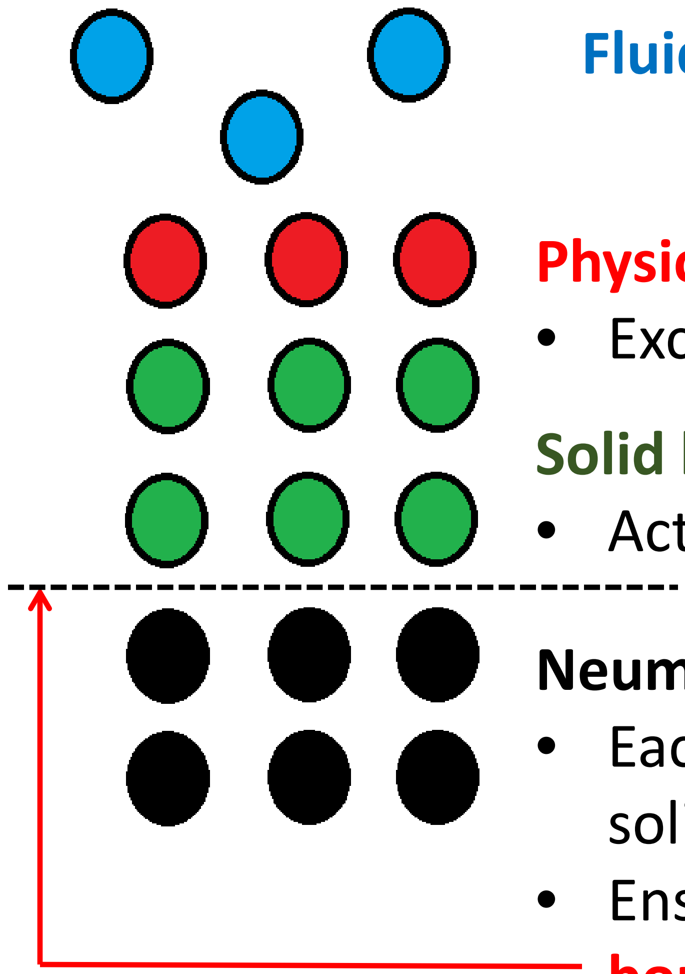
Methodology – Boundary conditions

- Taking advantage of the existing fixed dummy particle generation in GenCase and DualSPHysics.

- Dummy particles are:
  - Simple for parallel implementation
  - Can deal with complex geometries

- However, still need to be modified for the ISPH to include Neumann boundary conditions for the PPE matrix.

Methodology – Boundary conditions



Fluid Domain

Physical boundary particles

- Excluded from boundary velocity interpolation

Solid boundary particles

- Act as normal fixed particles

Neumann boundary particles

- Each Neumann particle pressure = closest solid boundary particle pressure + dp/dn
- Ensures exact “mirroring” across **Neumann boundary line**

Research Challenges

- Getting the ISPH formulation right! No ISPH literature using a combination of:
    - Wendland Kernel
    - Dummy boundary particles
    - Kinematic viscosity as low as $10^{-6}m^2/s$
- Implementing linear solver libraries – some more demanding than others, mixed level of documentation
    - Invested 6 months into finding out why the parallel MIS2-AMG preconditioner for the GPU failed to consistently work with ISPH, which has now been corrected.
- CPU code to GPU code not always straightforward – some differences in implementation
- Making sure CPU and GPU code give the same/similar results – mixed precision issues (maintaining consistency)

Research Challenges

Researching ISPH on the GPU has opened up many other new and exciting possible avenues of research:

- The on going development of ISPH in general
- ISPH for higher resolutions and the challenges associated with it
- The **solution of the ISPH PPE** at **high resolutions**:
  - **In particular, on the GPU**
  - **Preconditioning** Poisson equation matrices for **particle methods**.
    - **In particular, on the GPU**

Despite the on going challenges, the **rewards are high**

Results

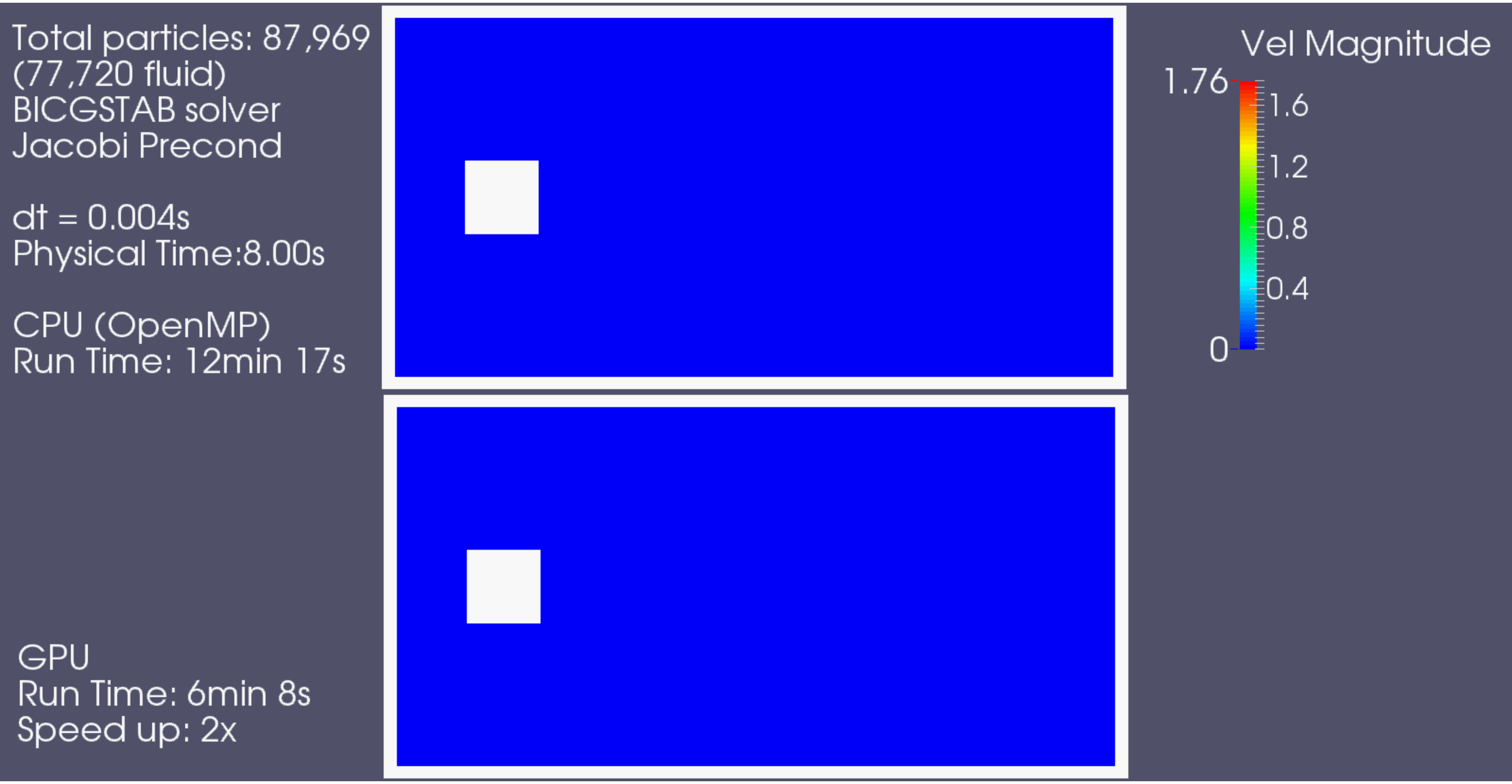
CPU
Intel i-7 4790 processor
Clockspeed: 3.6 GHz
Cores: 4
Threads: 8
OpenMP enabled

GPU
Nvidia GeForce GTX 980
Clockspeed: 1.126 GHz
CUDA Cores: 2048

- Wendland kernel: h=1.8dp

ViennaCL Library:
- Linear solver: BiCGStab, tolerance = 10^{-5}

SPHERIC Benchmark Test Case 6 Moving Box, Re=150



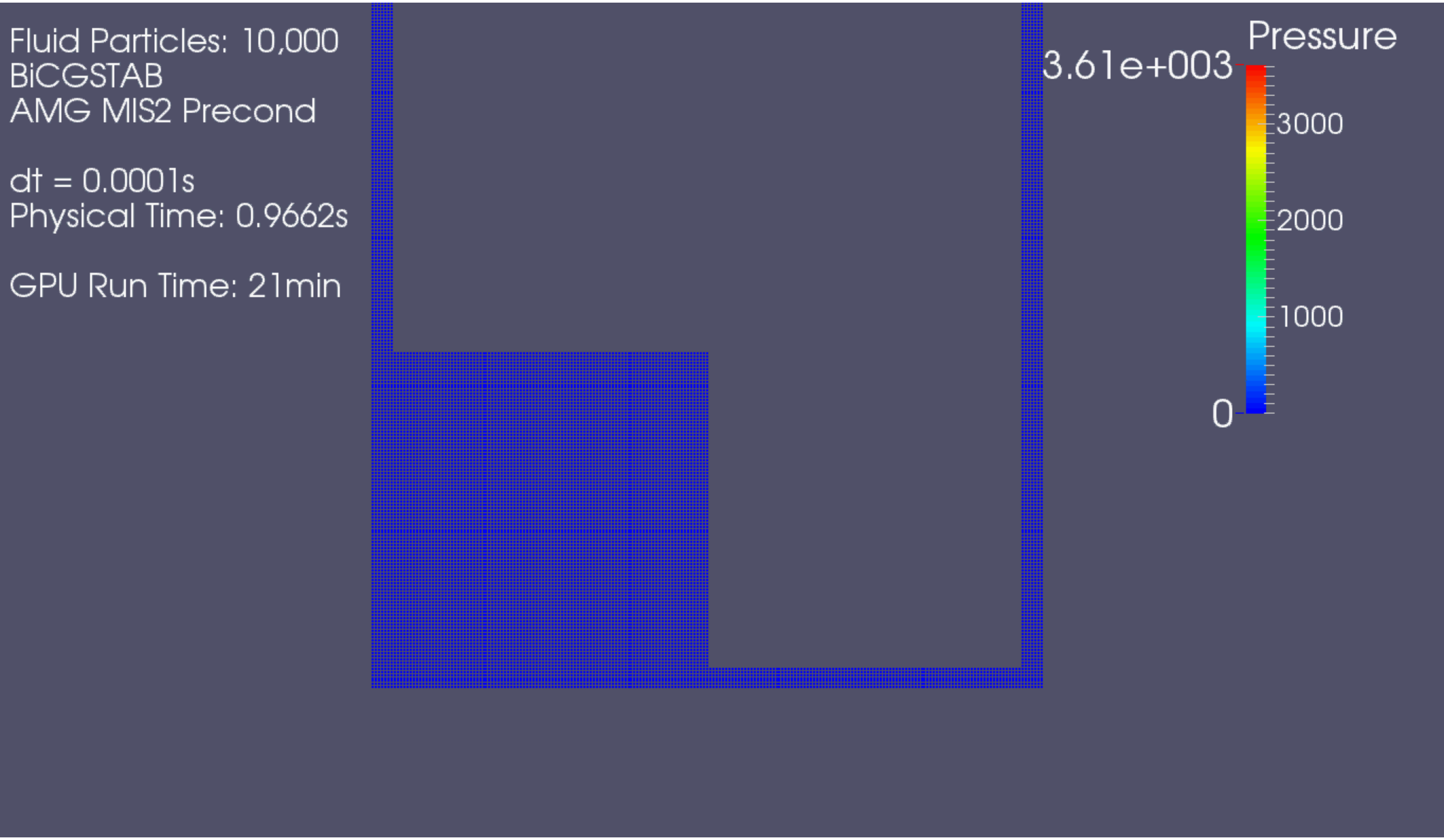
Total particles: 87,969
(77,720 fluid)
BICGSTAB solver
Jacobi Precond

dt = 0.004s
Physical Time:8.00s

CPU (OpenMP)
Run Time: 12min 17s

GPU
Run Time: 6min 8s
Speed up: 2x

Dambreak



Fluid Particles: 10,000
BiCGSTAB
AMG MIS2 Precond

dt = 0.0001s
Physical Time: 0.9662s

GPU Run Time: 21min

Conclusions and future work

<u>Project aim:</u> to develop a solver for incompressible free-surface flows capable of modelling breaking wave-structure impacts

- To achieve the project aim, a novel method of implementing ISPH on the GPU is being used, DualSPHysics is the vehicle used to create the model.

- The main idea for converting DualSPHysics is to use as much as possible of what is already available.
  - This is also applies for the use of an open-source linear solver library for the PPE.

- Current work already shows a significant improvement in speed and resolution capability from current ISPH literature.

- Lots of new research can be made with an accelerated ISPH.

Conclusions and future work

Future work:

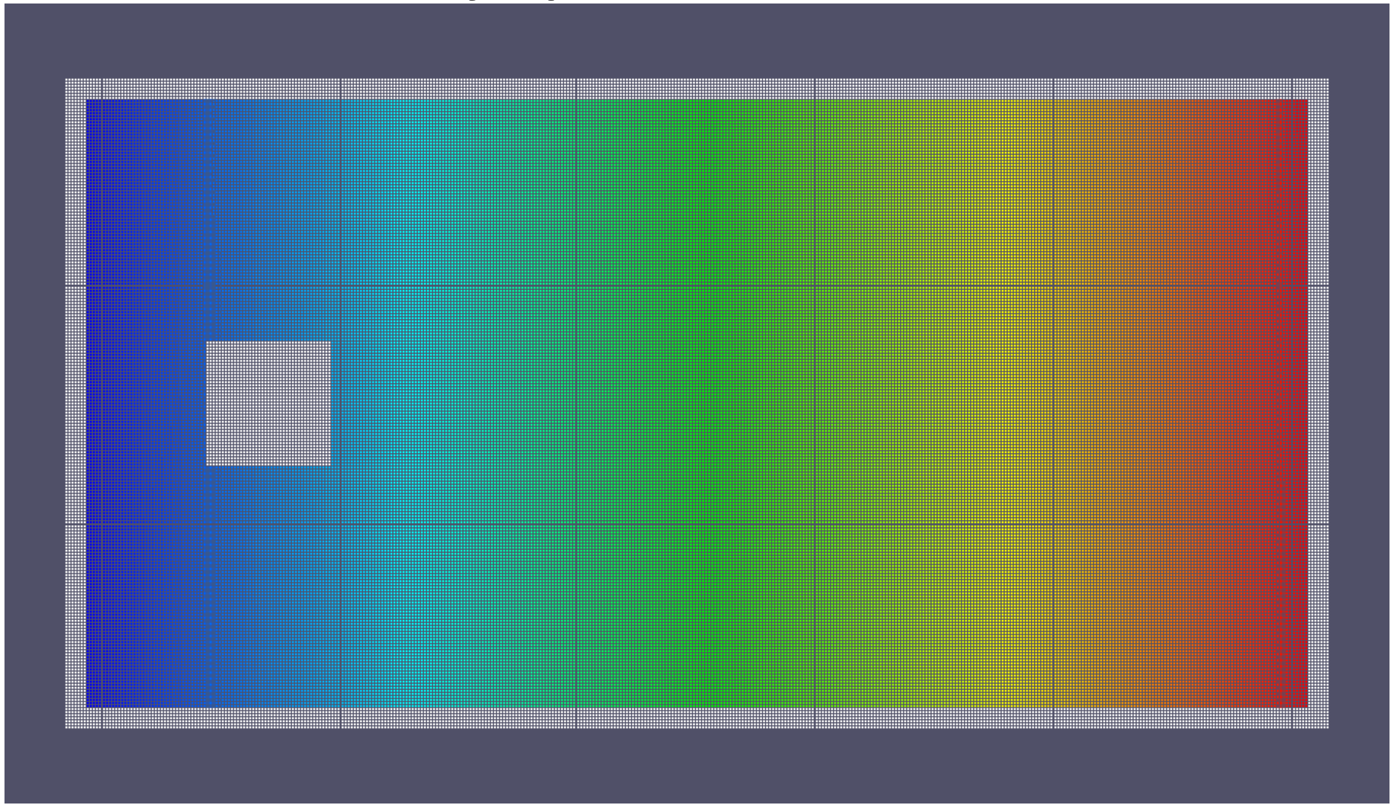
- Fix bugs
    - Currently on about the “10th last bug”…

- The model needs to be extended to 3D.

- More rigorous analysis and accuracy testing needs to be done to fully validate the model.

Thank you

Any questions?



Acknowledgements

- All of the DualSPHysics team
    - In particular: Dr Georgios Fourtakas, Dr Athansios Mokos, Dr Jose Dominguez, and Dr Stephen Longshaw
- The developer of the ViennaCL library Karli Rupp

Methodology – Inserting new computation functions for the PPE setup

CPU – Call p1 neighbours

```
int cxini,cxfin,yini,yfin,zini,zfin;
GetInteractionCells(dcell[p1],hdiv,nc,cellzero,cxini,cxfin,yini,yfin,zini,zfin);


for(int z=zini;z<zfin;z++){
  const int zmod=(nc.w)*z+cellinitial;
  for(int y=yini;y<yfin;y++){
    int ymod=zmod+nc.x*y;
    const unsigned pini=beginendcell[cxini+ymod];
    const unsigned pfin=beginendcell[cxfin+ymod];
```

Exclude 'cellinitial' for boundary neighbours

CPU – Call p2 neighbours

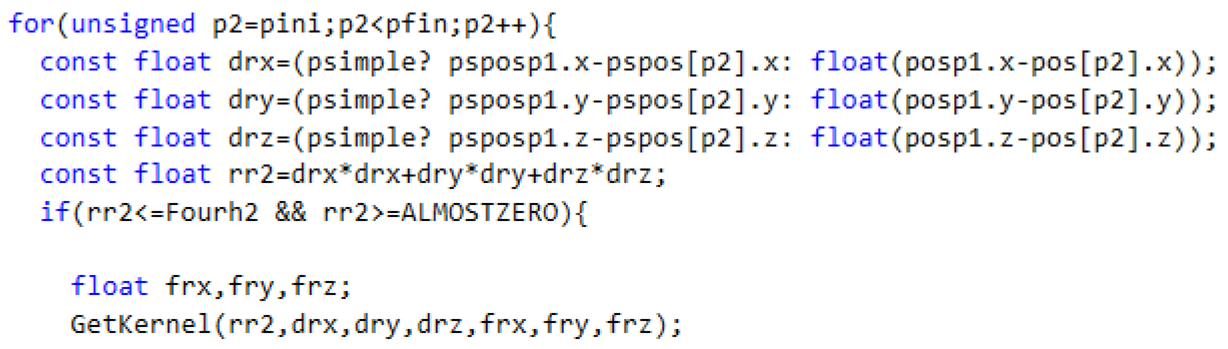
```
for(int z=zini;z<zfin;z++){
  int zmod=(nc.w)*z+cellfluid;
  for(int y=yini;y<yfin;y++){
    int ymod=zmod+nc.x*y;
    unsigned pini,pfin=0;
    for(int x=cxini;x<cxfin;x++){
      int2 cbeg=begincell[x+ymod];
      if(cbeg.y){
        if(!pfin)pini=cbeg.x;
        pfin=cbeg.y;
      }
    }
    if(pfin){
```

Exclude 'cellinitial' for boundary neighbours

//Insert __DEVICE__ function for particle interaction equation here

Methodology – Inserting new computation functions (for the PPE setup)

<u>CPU – p1 interact with p2</u>

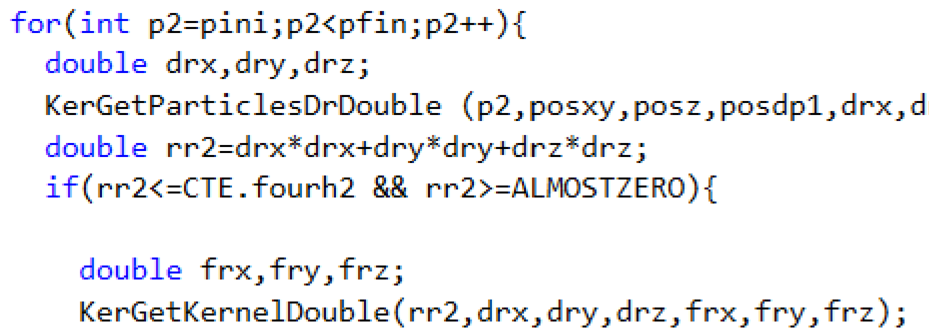


```
for(unsigned p2=pini;p2<pfin;p2++){
  const float drx=(psimple? psposp1.x-pspos[p2].x: float(posp1.x-pos[p2].x));
  const float dry=(psimple? psposp1.y-pspos[p2].y: float(posp1.y-pos[p2].y));
  const float drz=(psimple? psposp1.z-pspos[p2].z: float(posp1.z-pos[p2].z));
  const float rr2=drx*drx+dry*dry+drz*drz;
  if(rr2<=Fourh2 && rr2>=ALMOSTZERO){

    float frx,fry,frz;
    GetKernel(rr2,drx,dry,drz,frx,fry,frz);
```

//Insert equations here

<u>GPU – p1 interact with p2</u>

//Call device function for particle interaction here {



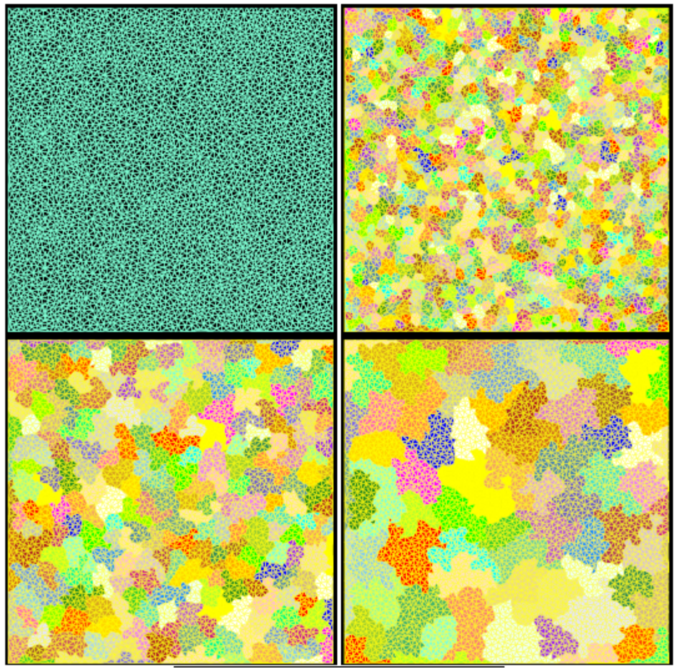
```
for(int p2=pini;p2<pfin;p2++){
  double drx,dry,drz;
  KerGetParticlesDrDouble (p2,posxy,posz,posdp1,drx,dry,drz);
  double rr2=drx*drx+dry*dry+drz*drz;
  if(rr2<=CTE.fourh2 && rr2>=ALMOSTZERO){

    double frx,fry,frz;
    KerGetKernelDouble(rr2,drx,dry,drz,frx,fry,frz);
```

//Insert equations here

Algebraic Multigrid (AMG)

A quick Algebraic Multigrid (AMG) overview:

- The preconditioner breaks the matrix down into different "levels" of fineness, the original matrix is the "finest level"
- The different levels will reduce errors of high and low frequencies in the result
- This produces a much faster convergence rate compared to other preconditioners
- This method is also highly scalable unlike the Jacobi



The AMG creates different levels of a matrix (Sumant et al., 2009)

P. S. Sumant, A. C. Cangellaris, and N. R. Aluru. A node-based agglomeration AMG solver for linear elasticity in thin bodies. Coummunications in Numerical Methods in Engineering, 25:219-236, 2009.