

Incompressible SPH (ISPH) on the GPU

A thesis is submitted to the University of Manchester for the degree
of Doctor of Philosophy in the Faculty of Science and Engineering

2018

Alex D. Chow

School of Mechanical , Aerospace, and Civil Engineering

Blank page

Contents

List of Published Works	9
List of Figures	10
List of Tables	17
List of Symbols	18
List of Abbreviations	21
Abstract	24
Declaration	25
Copyright Statement	26
Acknowledgements	27
1 Introduction	28
1.1 Background and motivation	28
1.2 Smoothed particle hydrodynamics (SPH) and incompressible SPH (ISPH)	29
1.3 Parallel programming and the graphics processing unit (GPU)	30
1.4 Aim and objectives	31
1.5 Thesis outline	32
2 Literature Review	34
2.1 Introduction	34
2.2 The violent incompressible free-surface flow	34

2.3	Applications and the need for computational fluid dynamics (CFD) . . .	36
2.4	Options for CFD	38
2.4.1	Mesh-based methods vs meshless methods	38
2.5	Smoothed particle hydrodynamics (SPH)	45
2.5.1	Weakly-compressible SPH (WCSPH)	46
2.5.2	Incompressible SPH (ISPH)	53
2.6	Hardware acceleration and parallel programming for SPH	63
2.6.1	HPC with message passing interface (MPI) and open multi-processing (OpenMP)	64
2.6.2	The graphics processing unit (GPU)	66
2.6.3	Open-source software for the GPU	71
2.7	Challenges of ISPH on the GPU	72
2.8	Conclusions	74
3	SPH Methodology	76
3.1	Introduction	76
3.2	SPH fundamentals	76
3.2.1	Interpolation and the SPH kernel	76
3.2.2	Kernel gradients	79
3.2.3	Discretisation error	81
3.3	ISPH methodology	83
3.3.1	A projection-based ISPH algorithm with shifting	83
3.3.2	The PPE matrix	86
3.3.3	Laplacian operator	88
3.3.4	Free-surface identification	88
3.3.5	Particle shifting	89
3.3.6	Time step constraint	92
3.4	An ISPH boundary condition suitable for GPUs	92
3.4.1	Mirror particles	93
3.4.2	Fixed dummy particles	94
3.4.3	Adapting the Marrone et al. boundary condition for ISPH . . .	95

3.5	Conclusions	99
4	Implementing ISPH on the GPU	101
4.1	Introduction	101
4.2	The graphics processing unit (GPU)	101
4.2.1	GPU architecture	102
4.2.2	The GPU memory model	105
4.2.3	GPU limitations	108
4.2.4	The CUDA parallel programming framework	110
4.3	Addressing the challenges of ISPH on the GPU	113
4.4	WCSPH DualSPHysics v4.0 and elements requiring conversion to ISPH	114
4.4.1	Software package structure	115
4.4.2	Neighbour list and particle sorting	117
4.4.3	Extending the DualSPHysics boundary condition	119
4.4.4	Time stepping scheme in DualSPHysics	120
4.5	The pressure projection implementation	121
4.5.1	Stage 1 - Intermediate step	123
4.5.2	Stage 2 - Setup and solve PPE matrix	123
4.5.3	Stage 3 - Corrector step	130
4.5.4	Stage 4 - Particle shifting	130
4.6	Solving the PPE matrix on the GPU	131
4.6.1	Matrix preconditioning	132
4.6.2	Direct methods	133
4.6.3	Iterative methods and Krylov subspace methods	134
4.6.4	Open-source linear algebra libraries and ViennaCL	139
4.6.5	The Jacobi Preconditioner	140
4.6.6	Algebraic multigrid as a Preconditioner	141
4.6.7	Modifying the ViennaCL MIS(2)AMG preconditioner for ISPH	143
4.7	Addressing the GPU memory limitations	154

4.8	Conclusions	156
5	Validation Tests	157
5.1	Introduction	157
5.2	Technical specifications of hardware and software	157
5.3	Impulsively started plate	158
5.4	2-D incompressible flow around a moving square in a rectangular box	161
5.4.1	$Re=100$	162
5.4.2	$Re=1$ million	164
5.5	Dambreak	165
5.5.1	2-D validation	166
5.5.2	3-D validation	168
5.6	Performance Analysis	170
5.6.1	Maximum particles for GPU RAM	170
5.6.2	Dambreak runtime comparisons	171
5.6.3	Preconditioner performance for ISPH	177
5.6.4	Profiling	180
5.6.5	Solver initialisation	185
5.7	Conclusions	188
6	Application of ISPH to a Numerical Wave Basin	192
6.1	Introduction	192
6.2	A numerical wave basin	194
6.2.1	Extensions to the methodology	194
6.2.2	Numerical wave tank geometry	203
6.2.3	Focused wave generation	204
6.3	Results comparisons and analysis	206
6.3.1	Validation of free-surface elevation vs linear theory	206
6.3.2	Experimental comparisons	208

6.3.3	Investigating hydrostatic and non-hydrostatic pressure during peak loading	214
6.3.4	Computation time and memory	221
6.4	Conclusions	223
7	Conclusions and Recommendations	226
7.1	General conclusions	226
7.2	Detailed conclusions	228
7.2.1	Implementing ISPH on the GPU	228
7.2.2	Performance	231
7.2.3	Numerical wave basin	232
7.3	Recommendations for future research	234
7.3.1	Algorithmic developments	234
7.3.2	Linear solvers and preconditioners for ISPH	235
7.3.3	Improving the physical model	236
7.3.4	Multi-GPU implementation	237
7.3.5	Suggestions of future applications	237
	References	274
A	Kernel gradient normalisation matrix inverse	275
B	Derivation of the pressure Poisson equation (PPE)	276
B.1	A Poisson equation for a continuous domain	276
B.2	The PPE with divergence-free velocity field in the projection method	278
C	Effect of boundary conditions on the Lagrangian PPE matrix	280
D	Paraview python scripts	282
D.1	Free-surface elevation at a focal point	283
D.2	Extracting cylinder data	285
D.3	Fluid particles around cylinder	289

Word count: 72,350

List of Published Works

Parts of this research project have been published, as follows:

In peer-reviewed journals:

- Chow, A. D., Rogers, B. D., Lind, S. J. and Stansby, P. K., Incompressible SPH (ISPH) with fast Poisson solver on a GPU, *Comput. Phys. Commun.* 226 (2018) 81-103.

In specialist conferences:

- Chow, A. D., Rogers, B. D., Lind, S. J. and Stansby, P. K., A fast incompressible SPH solver for free-surface flows on the GPU, in: *Proceedings of the 12th International SPHERIC Workshop 2017*.
- Chow, A. D., Rogers, B. D., Lind, S. J. and Stansby, P. K., Investigating breaking-wave forces on a vertical cylinder with a 3-D incompressible SPH (ISPH) numerical wave basin accelerated on a GPU, in: *Proceedings of the 13th International SPHERIC Workshop 2018*.

List of Figures

2.1	The free-surface interface between water and air.	35
2.2	A violent breaking-wave [22]	36
3.1	The SPH kernel radius of influence (highlighted in yellow).	78
3.2	PPE matrix element notation for ISPH. Example for a 5 particle system.	87
3.3	The truncation of the kernel for a particle, i , near a solid boundary. .	93
3.4	A fluid particle (blue), i , at a distance, dr , perpendicular to the boundary line generates the mirror particle (orange), i_m	94
3.5	An example of fixed dummy particles for a stationary boundary ($\mathbf{u} =$ 0). Fixed dummy particles (grey) are uniformly distributed and pos- sess the same velocity and pressure as their corresponding normal direction wall particle (green).	94
3.6	A fixed dummy particle, i , is mirrored perpendicular to the boundary line to create a unique interpolation point (UIP), (Ii)	96
3.7	The boundary mirroring process to find the positions of each bound- ary particle's UIP.	100
4.1	A basic model for the interaction between the CPU and GPU.	102
4.2	An example of an SM microarchitecture. This particular model is the GP104 SM from the GTX 1080 [289]. “LD/ST” stands for load/store unit. “SFU” stands for special function unit.	104
4.3	The GPU memory model. Figures are representative of a range of modern Nvidia GPU architectures.	108
4.4	GPU memory access times.	108
4.5	The CUDA programming model [236].	111

4.6	Flow diagram the DualSPHyics software package [15].	116
4.7	The cell-linked list grid in 2D.	118
4.8	The order of particle groups in DualSPHyics.	118
4.9	Flow diagram of the key repeating steps in the DualSPHyics Predictor-Corrector time step on a GPU: neighbour list, force computation, system update.	121
4.10	Flow diagram of the ISPH projection step on the GPU implemented into DualSPHyics. Highlighted are the 4 distinct particle sweeps in the simulation. CSR denotes Compressed Sparse Row storage format for matrices.	122
4.11	An example of the CSR matrix arrays <code>column</code> and <code>aValues</code> for a fluid particle, 17, a boundary particle, 8, and a free-surface particle, 18.	127
4.12	An example coefficient matrix and the corresponding Jacobi preconditioner matrix. Blank elements indicate a value of zero.	141
4.13	AMG matrix levels	142
4.14	Obtaining AMG matrix level 1 from level 0, in the context of ISPH. .	145
4.15	Sequential aggregation coarsening strategy example.	148
4.16	The interpolation matrix corresponding to the example in Fig. 4.15. Blank entries indicate a zero entry.	149
4.17	A system of particles with identified 2 coarse particles, which have the same neighbours as each other.	152
4.18	Simple 1-D meshed method example for the propagation of coarse aggregate indices resulting from the parallel MIS(2)AMG algorithm .	153
5.1	The initial geometry of the impulsively started plate test case. The centre line (dashed) indicates a symmetry between the top and bottom half of the setup.	159

5.2	Impulsively started plate convergence plot for both CPU (crossed marker) and GPU (square marker) codes; the relative L_2 error norm of the free surface elevation for $u_p = 0.2$ m/s, $d = 0.5$ m at $t = 0.6$ s. The solid trendline shows a convergence rate of 1.18 and the dashed lines above and below represent first and second order convergence respectively.	160
5.3	Impulsively started plate ($dp = 0.00625$ m) close-up of fluid particle (blue) positions near the plate and free surface. Particles coloured in red are included within the calculation of L_2 (Eq. (5.2)). The solid line shows the theoretical free surface according to Eq. (5.1).	160
5.4	Impulsively started plate pressure plot for $u_p = 0.2$ m/s, $h = 0.5$ m at $t = 0.6$ s. Here, the initial particle spacing is 0.00625 m. Units in Pa.	161
5.5	The setup for the 6 th SPHERIC benchmark test case: 2-D incompressible flow around a moving square in a rectangular box.	162
5.6	2-D incompressible flow around a moving square in a rectangular box velocity magnitude field comparison between Incompressible-DualSPHysics and a finite difference reference solution [18]. Units in m/s.	163
5.7	Runtime comparisons for computing the first 50 time steps of the 2-D incompressible flow around a moving square in a rectangular box test case. A Jacobi preconditioner is used for all tests.	164
5.8	2-D incompressible flow around a moving square in a rectangular box, $Re = 1$ million. Particle are coloured according to ID number.	165
5.9	Dambreak test geometry. A replication of the experiment of Koshizuka and Oka [19], but with extended tank walls.	166
5.10	2-D dambreak using 680,000 fluid particles. Pressure is plotted in Pa.	167
5.11	Comparison of the dambreak toe with experimental data of Koshizuka and Oka [19].	168

5.12	3-D dambreak simulation, $t = 0.3$ s. The geometry is the same as that in Fig. 5.9 and extended in the y -direction by 0.2 m. $dp = 0.002$ m. Pressure is plotted in Pa.	169
5.13	Comparison of the quintic spline and Wendland kernel for the position of the dambreak toe in 3D with experimental data of Koshizuka and Oka [19].	170
5.14	Runtime comparisons for the first 10 time steps of the 2-D dambreak case (as in Section 5.5.1) with the Jacobi and MIS(2)AMG preconditioners.	172
5.15	As in Fig. 5.14 showing the MIS(2)AMG plots only.	172
5.16	2-D dambreak GPU speed ups	174
5.17	Runtime comparisons for the first 10 time steps of the 3-D dambreak case (as in Section 5.5.2) with the Jacobi and MIS(2)AMG preconditioners.	175
5.18	As in Fig. 5.17 showing the Jacobi plots only, and an additional plot for runs using the quintic spline.	175
5.19	3-D dambreak GPU speed ups	176
5.20	Comparison of the number of solver iterations taken per time step during the 2-D dambreak simulation between the Jacobi and MIS(2)AMG preconditioners. Here, $dp = 10^{-3}$, resulting in 42,632 fluid particles.	178
5.21	A snapshot of the 2D dambreak simulation from afar and a close-up of the area encircled. These images show the fragmentation that occur in the flow leading to more than one linear system within the computational domain. The snapshot correlates to time step 3500 in Fig. 5.20. Multiple independent systems of fluid are more evident with higher resolutions.	178
5.22	Comparison of the cumulative time spent solving the matrix for the simulation as in Fig. 5.20.	179

5.23	Percentage of time spent on the 4 stages of the algorithm out of the ISPH pressure projection time step for different devices. The first 10 time steps of the of the dambreak case (Section 5.5), in 2D and 3D, are tested using approximately 1 million fluid particles and the Jacobi preconditioner. Stage 2 is split up into two parts, Stage 2a and Stage 2b, for the population and solving of the PPE matrix respectively.	180
5.24	Dambreak runtimes using solver initialisation (SI) and comparing to results obtained from Figs 5.14 and 5.17 without the use of SI (No SI).	187
5.25	Runtime reductions, as a percentage, achieved by the use of solver initialisation.	188
5.26	New GPU-CPU speed ups with use of solver initialisation (SI) and comparing to results obtained from Figs 5.14 and 5.26b without the use of SI (No SI).	189
5.27	Percentage of time spent on the 4 stages of the ISPH pressure projection step algorithm on the GTX 1070 GPU using solver initialisation (SI) compared to results obtained from Fig. 5.23 without use of solver initialisation (No SI).	190
6.1	Improving the boundary condition by exclusion of boundary particles above the free-surface line for each time step.	198
6.2	Non-dimensional absolute error of pressure from fluid particles directly adjacent to the boundary on one side of the tank in a 2-D still water case. $dp = 0.005$ m, still water depth $d = 0.5$ m. The theoretical pressure at the bed is 4900 Pa.	200
6.3	Plan view of different arrangements of particles for representation of a cylindrical column. Red indicates a column particle, blue indicates a boundary particle on the tank bottom.	201
6.4	Implementation details for the radial arrangement of the column. Plan views are used for clarity of the particle distributions.	201
6.5	Total horizontal force on a column (with different particle arrangements) subject to regular wave loading.	202

6.6	Plan view of the numerical wave basin geometry	203
6.7	Spatial convergence of the ISPH model with linear theory for the free-surface elevation of a JONSWAP wave group. $A_N = 0.05$ m. The RHS of the plot shows enlarged peak elevations.	207
6.8	Convergence rate of the L_2 error norm of the peak free-surface elevation at the focal point for the simulation in Fig. 6.7. $A_N = 0.05$ m, $d = 0.505$ m.	208
6.9	Free-surface elevation of the various focused wave groups at the position of the cylinder-front.	210
6.10	Post-breaking event of focused wave group F14 at $t = 12.0$ s.	211
6.11	Simulation snapshots of breaking-wave case F15.	212
6.12	θ^c around the cylinder.	212
6.13	Total horizontal force on the cylinder subject to the various focused wave groups.	213
6.14	Evaluating the free-surface elevation around the cylinder where subscript <i>col</i> represents a unique set within the cylinder.	216
6.15	The method for determining the free-surface elevation around the cylinder is validated by mapping nearby fluid particles (coloured by pressure value) over the cylinder-surface particles (coloured in black) with $P \neq 0$	217
6.16	Incompressible-DualSPHysics results for the total, hydrostatic, and non-hydrostatic horizontal force on the cylinder subject to the various focused wave groups.	218
6.17	Centreline vertical planes through the centre of the cylinder and fluid domain in the x - z plane for each case at the times, $t =$ (a) 9.7, (b) 9.6, (c) 11.95, and (d) 11.9 s, corresponding to peak total force experienced by the cylinder for F3, F16, F14, and F15 respectively. The wave direction is from left to right.	219

6.18	Free-surface elevation around the column for all cases at their respective times of maximum total horizontal force experienced. $t = 9.7, 9.6, 11.95,$ and 11.9 s for cases F3, F16, F14, and F15 respectively. . .	220
6.19	Normalised cylinder particle pressures relative to the mean still-water-level for each case at times of peak total force experienced. Particles with zero pressure are omitted.	221
6.20	Fluid velocities of free-surface particles within a distance h of the cylinder surface for all cases at instances of peak horizontal force experienced (as in Fig. 6.19). Normal and tangential components consider x - and y -velocity components, u and v	222
C.1	An example 1-D case of uniformly distributed fluid particles (blue) and fixed dummy particles (grey) with unique interpolation points. Arrows labelled “UIP” point to the location of a UIP belonging to the boundary particle indicated by the subscript. The associated matrix is shown where non-zero elements are shaded in grey. Elements within the red box indicate fluid-fluid particle interaction values.	281

List of Tables

3.1	Normalisation factor (α_D) values	79
3.2	Experiences of different boundary conditions regarding the satisfaction of desirable SPH boundary condition criteria.	96
5.1	GPU properties. GFLOPS stand for floating-point operations per second on the order of giga (10^9).	158
5.2	Approximate maximum total number of particles (in millions) that can be computed for a various GPU RAM size. 2-D simulations use the quintic spline, 3-D simulations use the Wendland kernel.	171
5.3	GPU speed ups for each stage of algorithm with approximately 1 million particles in 2D and 3D. Results are obtained from the same data used Fig. 5.23. CPU (1) and CPU (16) refer to CPU single and 16-threaded respectively.	182
5.4	Theoretical and achieved occupancy and branch divergence percentages, and single and double precision computational throughput for the CUDA kernels (parallel GPU functions) executed for each particle sweep in the algorithm. Results are taken from the first time step of the 3-D dambreak simulation (see Section 5.5.2) and performed on the GTX 1070 GPU.	185
6.1	Focused wave groups.	208
6.2	Incompressible-DualSPHysics computation times	222
6.3	Number of particles for simulations in Section 6.3.2	223

List of Symbols

The following list of symbols are used throughout this thesis. Any other notation is introduced when used locally.

$(Ii)j$	Interaction between UIP Ii and particle j	(x_f, y_f)	Focal point coordinates
*	Intermediate value	(x_{fs}, y_{fs})	Point in space to measure free-surface elevation
α_D	Kernel normalisation factor	$[\mathbf{A}]$	PPE matrix
α_m	PPE system smoothing coefficient for near free-surface regions	$[\mathbf{M}]$	Preconditioning matrix
α_{shift}	Constant to control shifting normal to free surface	$[\mathbf{P}]$	Interpolation matrix
β	Equilibrium term for controlling shifting normal to free surface	\mathbf{b}	PPE source term vector
		\mathbf{f}	Acceleration due to external forces
		$\mathbf{i}, \mathbf{j}, \mathbf{k}$	Cartesian unit direction vectors
Δt	Time step size	$\mathbf{n}, \mathbf{s}, \mathbf{s}_b$	Normal, tangential, and bi-tangential direction vectors
$\delta \mathbf{r}_s$	Particle shifting distance		
η	Free-surface elevation	\mathbf{r}	Position vector with Cartesian components
η_s	A constant to prevent singularities	(x, y, z)	

\mathbf{u}	Velocity vector with components (u, v, w)	dp	Original particle spacing
\mathbf{x}	PPE solution vector	f_p	Spectral peak frequency
$\nabla_i W_{ij}$	Normalised kernel gradient	g	Acceleration due to gravity
$\nabla_i \omega_{ij}$	Kernel gradient	H	Wave height
ν	Kinematic viscosity	h	Smoothing length
$\omega(r, h)$	Smoothing kernel	i	Interpolation particle
ω^{MLS}	MLS smoothed kernel	Ii	UIP belonging to particle i
ω_{ij}	Smoothing kernel	ij	Interaction between particles i and j
ϕ	A quantity	j	Neighbouring particle
ρ	Density	L	Wavelength
θ_i^c	Angular position of particle i around cylinder	L_2	L_2 error norm
A_N	Peak amplitude	m	Mass
A_{ij}	Element of matrix $[\mathbf{A}]$ between particle i and j	n	Time step number
$b_{m,i}$	Element of PPE source term vector \mathbf{b} for particle i	$N_{n,max}$	Maximum number of particle neighbours in a uniformly distributed Cartesian arrangement
C	Concentration	Nnz	Number of non-zero entries in PPE matrix
C_{CFL}	CFL condition number	Nnz_{max}	Maximum number of non-zero entries in PPE matrix
d	Still water depth	np	Total number of particles

npb	Number of boundary particles	Re	Reynolds number
		T	Wave period
$npbok$	Number of boundary particles near fluid region	t	Time
npf	Number of fluid particles	u_p	Piston velocity
npm	Number of particles included in PPE matrix	u_{max}	Maximum velocity
		V	Volume
P	Pressure	$x_{m,i}$	Element of PPE solution vector \mathbf{x} for particle i
r_c	Cylindrical column radius		

List of Abbreviations

ALE	Arbitrary Lagrange-Euler
ALU	Algorithmic logic unit
AMG	Algebraic multigrid
API	Application programming interface
BEM	Boundary element method
Bi-CGSTAB	Bi-conjugate gradient stabilized method
CFD	Computational fluid dynamics
CFL	Courant-Friedrichs-Lewy
CG	C for graphics
CG-S	Conjugate-gradient squared method
CPU	Central processing unit
CSR	Compressed row storage
CUDA	Compute Unified Device Architecture
DEM	Diffuse element method
DPD	Dissipative particle dynamics
ELI-SPH	Eulerian-Lagrangian incompressible smoothed particle hydrodynamics
ELMMC	Eulerian-Lagrangian marker and microcell method
EoS	Equation of State
FDM	Finite difference method
FEM	Finite element method
FK	Froude-Krylov
FLOPS	Floating-point operations per second
FLOPS	Floating-point operation

FPM	Finite pointset method
FPU	Floating-point unit
FVM	Finite volume method
FVPM	Finite volume particle method
GB	Gigabytes
GMRES	Generalized minimal residual method
GPGPU	general-purpose graphics processing unit
GPU	Graphics processing unit
HPC	High performance computing
ISPH	Incompressible smoothed particle hydrodynamics
KB	Kilobytes
LD/ST	Load/store unit
LGA	Lattice-gas automata
LRPIM	Local radial point interpolation method
MAC	Marker-and-cell
MB	Megabytes
MD	Molecular dynamics
MINRES	Minimal residual method
MIS(2)AMG	Maximum independent set (2) algebraic multigrid
MLPG	Meshless local Petrov-Galerkin method
MLS	Meshless local Petrov-Galerkin method based on Rankine source
MLS	Moving least squares
MPI	Message passing interface
MPS	Moving particle semi-implicit method
OpenCL	Open Computing Language
OpenGL	Open Graphics Library
OpenMP	Open multi-processing
OPS	Optimised particle shifting
PCI	Peripheral Component Interconnect

PIC	Particle-in-cell
PPE	Pressure Poisson equation
QALE-FEM	Quasi-arbitrary Lagrange-Euler finite element method
RAM	Random-access memory
RANS	Reynolds-averaged Navier-Stokes
SIMD	Single instruction, multiple data
SIMT	Single instruction, multiple threads
SM	Streaming multiprocessor
SPH	Smoothed particle hydrodynamics
SPHERIC	Smoothed particle hydrodynamics European Research Interest Community
UIP	Unique interpolation point
VOF	Volume-of-fluid
VTK	Visualisation Toolkit
WCSPH	Weakly-compressible smoothed particle hydrodynamics
XML	EXtensible Markup Language

Abstract

Alex Chow
Doctor of Philosophy
The University of Manchester
August 2018

Incompressible SPH (ISPH) on the GPU

Incompressible free-surface flows involving highly complex and violent phenomena are of great importance to the engineering industry. Applications such as breaking-wave impacts, fluid-structure interaction, and sloshing tanks demand an accurate and noise-free pressure field, and require large-scale simulations involving millions of computation points. This thesis addresses the need with the novel use of a graphics processing unit (GPU) to accelerate the incompressible smoothed particle hydrodynamics (ISPH) method for highly non-linear and violent free-surface flows using millions of particles in three dimensions.

Compared to other simulation techniques, ISPH is robust in predicting a highly accurate pressure field, through the solution of a pressure Poisson equation (PPE), whilst capturing the complex behaviour of violent free-surface flows. However, for large-scale engineering applications the solution of extremely large PPE matrix systems on a GPU presents multiple challenges: constructing a PPE matrix every time step on the GPU for moving particles, overcoming the GPU memory limitations, establishing a robust and accurate ISPH solid boundary condition suitable for parallel processing on the GPU, and exploiting fast linear algebra GPU libraries.

A new GPU-accelerated ISPH algorithm is presented by converting the highly optimised weakly-compressible SPH (WCSPH) code DualSPHysics and combining it with the open-source ViennaCL linear algebra library for fast solutions of the ISPH PPE. The challenges are addressed with new methodologies: a parallel GPU algorithm for population of the PPE matrix, mixed precision storage and computation, and extension of an existing WCSPH boundary treatment for ISPH. Taking advantage of a GPU-based algebraic multigrid preconditioner for solving the PPE matrix required modification for ISPH's Lagrangian particle system.

The new GPU-accelerated ISPH solver, Incompressible-DualSPHysics, is validated through a variety of demanding test cases and shown to achieve speed ups of up to 25.3 times and 8.1 times compared to single and 16-threaded CPU computations respectively. The influence of free-surface fragmentation on the PPE matrix solution time with different preconditioners is also investigated. A profiling study shows the new code to concentrate the GPU's processing power on solving the PPE.

Finally, a real-engineering 3-D application of breaking focused-wave impacting a surface-piercing cylindrical column is simulated with ISPH for the first time. Extensions to the numerical model are presented to enhance the accuracy of simulating wave-structure impact. Simulations involving over 5 million particles show agreement with experimental data. The runtimes are similar to volume-of-fluid and particle-in-cell solvers running on 8 and 80 processors respectively. The 3-D model enables post-processing analysis of the wave mechanics around the cylinder.

This study provides a substantial step for ISPH. Incompressible-DualSPHysics achieves resolutions previously too impractical for a single device allowing for the simulation of many industrial free-surface hydrodynamic applications.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

- I. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
- II. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1998 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.
- III. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
- IV. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy¹, in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations² and in The University's policy on presentation of Theses.

¹<http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420>

²<http://www.library.manchester.ac.uk/about/regulations/>

Acknowledgements

The 3-4 years of my Ph.D. studies to produce this thesis is amongst my most proudest accomplishments. However, it would not have been such a rewarding and enjoyable experience without a number of people, to whom this section is dedicated towards.

Firstly, I would like to thank my three supervisors Dr Benedict D. Rogers, Prof. Peter K. Stansby, and Dr Steven J. Lind for their invaluable guidance and support. I could not have asked for better mentors to work with and I am very grateful for all they have taught me. I would also like to thank the SPH group at the University of Manchester for the laughs, discussions, fruitful comments, and sharing of frustrations. This goes out especially to Dr Georgios Fourtakas, Dr Athanasios Mokos, Aaron English and Annelie Baines.

I would like to thank the Engineering and Physical Sciences Research Council (grants EP/L504877/1 and EP/M506436/1) and the Manchester Research Impact Fund for funding this research project, else it would not have been.

A special mention goes to Dr Karl Rupp who I would like to thank for our helpful conversations regarding the ViennaCL library, which has played a major part to the success of this research.

I would like to give a big thanks to my parents, Jane and Gordon, my brother, Robert, and his wife, Carmen, for their continual support throughout my academic career and all the fun family meals.

I would like to stress to my friends (too many to mention) that I did not forget them during my pursuit of this doctorate. I am forever grateful for their understanding. It was always wonderful to see them all when I could.

Finally, I express my deepest gratitude and appreciation to my wonderful partner Pei Ting Chung. Thank you for your patience, encouragement, support, and the distractions from the stressful times. Whether you were in Malaysia, Singapore or Manchester, you always kept me smiling.

Chapter 1

Introduction

1.1 Background and motivation

Violent hydrodynamic free-surface flows are of great importance to the engineering industry, and there is a requirement to address these for engineering design where there is currently a lack of appropriate tools. Applications for the civil, mechanical, coastal and marine industries include breaking-wave impacts, sloshing tanks, green-water overtopping, and open-channel flows. Understanding these flow behaviours and their impact forces with structures is critical to successful engineering design.

Engineers can make use of physical experiments and/or numerical modelling and analytical approaches to determine impact forces and structural response. Whilst experiments provide real-world data from an actual flow, there are often undesirable issues such as high costs, lack of resources, and time allocation. Therefore, experimental design is often too impractical, or even impossible, to undertake. Moreover, capturing or reproducing certain complex flow phenomena may become challenging when performing laboratory investigations at reduced scales.

There is however, a wealth of experimental flow data and analytical solutions to help validate numerical models used in scientific simulations, which aim to predict flow behaviour according to the governing equations. Simulations can be less intensive in terms of cost and resources, and dependent on the application, the actual scale of the problem can be simulated. Moreover, they can be repeated many times with varying parameters for little extra effort, and they allow one to visualise and

analyse flow details often too impractical to obtain from an experiment.

The computation of complex phenomena by direct solution of the governing equations is usually impossible, thus simplified models have often been used in the past. Computational fluid dynamics (CFD) is the study of fluid flows by numerical simulation methods. The field of CFD has established many methods to solve the governing equations of fluid motion, known as the Navier-Stokes equations, for a wide range of problems including free-surface flow applications. Conventionally, these methods have used a computational grid or mesh to perform the simulation. Techniques such as the finite difference method (FDM), the finite element method (FEM), and the finite volume method (FVM) have all found widespread application in industry and academic research covering a vast array of hydrodynamic problems. Software packages using these mesh-based methods include OpenFOAM [1], ANSYS Fluent [2], and Star-CCM+ [3]. However, they have limitations and problems when applied to violent free-surface flows involving extreme flow deformations and discontinuities due to the presence of a computational grid in the numerical method.

Therefore, this research develops a new numerical model and engineering tool for simulating hydrodynamic applications involving violent free-surface flows, where accurate computation of the pressure field is required. The novel use of a graphics processing unit (GPU) for hardware acceleration is chosen to enhance practicality of simulations for large-scale 3-D engineering applications.

Recently, techniques with no computational grid or mesh have appeared and are starting to challenge the dominance of FDM, FEM and FVM. One such meshless approach is smoothed particle hydrodynamics (SPH), a novel method well-suited for computing violent free-surface flows.

1.2 Smoothed particle hydrodynamics (SPH) and incompressible SPH (ISPH)

SPH was originally created for the field of astrophysics by Gingold and Monaghan [4] and Lucy [5] in 1977. The method represents the domain as a set of computa-

tional interpolation points more commonly referred to as particles. As a Lagrangian method, each particle is able to move freely within the domain and possesses physical quantities, such as pressure and velocity, of the representative material mass. The independence from a computational mesh allows SPH to simulate highly non-linear and complex phenomena.

Since Monaghan [6] applied SPH to free-surface flows in 1994, the method has had many successes modelling such flows within engineering applications including breaking waves [7], sloshing tanks [8], body-water slam [9], and water column converters [10]. For hydrodynamic applications, SPH has two main formulations for describing a fluid. Weakly-compressible SPH (WCSPH) is the original form of SPH and uses an artificial equation of state that allows density to vary within about 1% of a reference value. Traditionally, WCSPH exhibits spurious fluctuations in the pressure field and subsequent instabilities. However, over the years WCSPH has been much improved with noise being significantly reduced (see Section 2.5 later). The second formulation, incompressible SPH (ISPH), enforces incompressibility of the fluid by keeping the density constant and solving pressure by means of a pressure Poisson equation (PPE) requiring solution of a sparse matrix. ISPH has been shown to produce highly accurate and near noise-free pressure fields [11]. ISPH is therefore an attractive simulation tool for applications including wave impact and fluid-structure interaction, and is the focus of the research objective in this thesis. However, ISPH is more computationally demanding compared to other numerical methods and WCSPH. Therefore, parallelism and hardware acceleration is required for ISPH to make it practical and attractive for real 3-D engineering applications.

1.3 Parallel programming and the graphics processing unit (GPU)

The simulation of 3-D engineering applications may take a large amount of computational time depending on the complexity of the flow, domain size, and required physical time to be simulated. For example, a case such as breaking waves is particu-

larly demanding, requiring millions of particles in a large computational domain and the simulation of several seconds of physical time where the time step is on the order of 10^{-4} to 10^{-6} seconds. Furthermore, although no mesh generation is required, Lagrangian methods such as ISPH are particularly time consuming due to the need to compute up to several hundred neighbouring particle-particle interactions for each individual particle at every time step.

Parallel programming and hardware acceleration allows code to be executed on thousands of processors simultaneously and has been utilised within scientific computing to significantly reduce the computational time of complex large-scale simulations. Originating from the gaming industry, graphics processing unit (GPU) hardware acceleration has recently emerged as a relatively cheap, and energy-efficient high performance computing (HPC) tool compared to that of conventional HPC central processing unit (CPU) clusters. Parallel programming is more complex than conventional serial coding and much of the current research in the field focuses upon optimising parallel algorithms for different hardware types. However, the emergence of parallel programming paradigms such as Compute Unified Device Architecture (CUDA) [12] and Open Computing Language (OpenCL) [13] greatly reduces the effort required to program such hardware.

Lagrangian methods and N-body simulations, such as SPH, are well suited for parallel processing, thus such methods have advanced rapidly with new computational hardware such as GPUs. With a simpler formulation, implementing WCSPH on a GPU is now common practice, however the field of ISPH still requires such a code.

1.4 Aim and objectives

The aim of this thesis is to provide a numerical solver for incompressible and violent free-surface flows with application to real 3-D engineering problems by development of an open-source GPU-accelerated ISPH solver. The use of parallel programming and GPU hardware acceleration will reduce simulation times of the computationally expensive ISPH method. The solver created within this project will be a powerful

engineering tool with relatively cheap hardware requirements, accessible to both research groups and small companies.

To achieve this aim, the research objectives are:

- Establish a parallel programming algorithm for ISPH and its implementation on the GPU.
- Validate the accuracy and robustness of the resultant numerical solver with 2-D and 3-D test cases for confined and free-surface flows.
- Benchmark the performance analysis of the GPU-accelerated ISPH code against single and multi-threaded CPU implementations.
- Simulate a real engineering problem of a focused breaking-wave impact on a surface-piercing cylinder involving violent free-surface flow and compare to physical experimental results.

1.5 Thesis outline

Immediately following this chapter will be a detailed justification of this study including a review of published literature mainly regarding advances in SPH. Current hardware acceleration options are also presented in more detail to establish the motivation for using the GPU to accelerate ISPH. This is followed by identifying the challenges of implementing ISPH on the GPU.

Chapter 3 presents the fundamentals of the SPH method and the ISPH model used for this study. An assessment of SPH boundary conditions appropriate for the GPU is undertaken before implementation of the Marrone et al. [14] boundary condition for ISPH on the GPU is proposed.

In Chapter 4, the novel implementation of ISPH on the GPU is described in detail, which includes:

- The use of the open-source WCSPH solver software, DualSPHysics [15], and its conversion to an ISPH code for the GPU.

- Implementing the ISPH pressure projection step onto the GPU, with special attention to the parallel population of the ISPH PPE matrix.
- The use of open-source linear algebra libraries for the GPU and in particular, the ViennaCL linear algebra library [16]. A modification to the ViennaCL library that allows the maximum independent set (2) algebraic multigrid (MIS(2)AMG) preconditioner to be used with ISPH is also presented.
- The methods implemented to address the GPU memory limitations.

Chapter 5 presents a number of test cases for validation of the new GPU-accelerated ISPH code, Incompressible-DualSPHysics. The impulsively started plate case is a demanding test that shows the order of convergence of the methodology compared with the analytical solution of Peregrine [17]. The stability and robustness of the code is demonstrated with the case of 2-D incompressible flow around a moving square in a rectangular box [18] a benchmark test case of the SPH European Research Interest Community (SPHERIC). Validation for a free-surface flow is made by modelling the dambreak of Koshizuka and Oka [19]. The performance of the new code is analysed for the dambreak case, in 2D and 3D, and compared to a single and multi-threaded CPU version of the code.

The model for a numerical wave basin is presented in Chapter 6 where extensions to the numerical methodology in Chapter 3 are made for the application of wave propagation and impact. Focused wave groups, of non-breaking and breaking nature, are simulated and comparisons of free-surface elevation and forcing on a surface-piercing cylinder are made against the experimental data of Zang et al. [20]. An investigation of non-hydrostatic and hydrostatic pressures during peak loading events is also made with post-processing analysis.

The final Chapter draws conclusions from the study, summarising the accomplishments and providing recommendations for future study to further improve upon the research.

Chapter 2

Literature Review

2.1 Introduction

This chapter presents a critical examination of current literature regarding CFD and identifies the shortcomings within the field with respect to violent hydrodynamic engineering applications. The review establishes the need for ISPH on the GPU. A comparative overview between mesh-based and meshless methods is made, followed by an investigation of SPH and its two main formulations, WCSPH and ISPH. Current hardware acceleration technologies and their role within SPH is also included, with a particular focus on the evolution of GPU implementations. To help reduce the complexity of implementing an efficient ISPH solver on the GPU, several open-source GPU software are identified. Finally, the key challenges of ISPH on the GPU, which this thesis will address, are established.

Prior to the main literature review however, the definition of a “violent incompressible free-surface flow” is established in the following section.

2.2 The violent incompressible free-surface flow

The definition of a violent incompressible free-surface flow can be established incrementally by each explaining the “free-surface”, “incompressible”, and “violent” parts of the flow characteristics one after the other:

- **The free-surface flow:** A flow where there is a free surface present. The

surface of a fluid which is subject to zero parallel stresses is said to be a free surface. This can also be defined mathematically by the “free-surface condition” where $DP/Dt = 0$, such that P is pressure and t is time. A common example is that of the interface between water and air as depicted in Fig. 2.2. These boundary conditions defining the free surface can be elaborated further as detailed by Dean and Dalrymple [21].

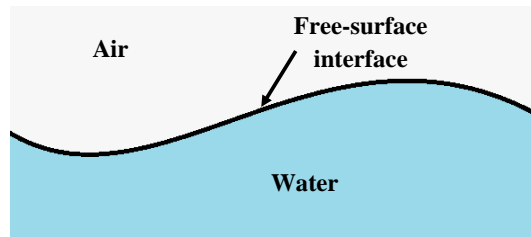


Fig. 2.1: The free-surface interface between water and air.

- **The incompressible free-surface flow:** A free-surface flow where the fluid is described as incompressible, that is the density of the fluid does not change. In fluid dynamics, an incompressible fluid is described with the incompressible Navier-Stokes equations, namely the conservation of mass equation:

$$\nabla \cdot \mathbf{u} = 0, \quad (2.1)$$

and the conservation of momentum equation:

$$\frac{d\mathbf{u}}{dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{u} + \mathbf{f}, \quad (2.2)$$

where \mathbf{u} is velocity, t is time, ρ is density, P is pressure, ν is kinematic viscosity, and \mathbf{f} is the acceleration due to any external forces in the system such as gravity. Both equations are expressed in the Lagrangian form.

The divergence-free velocity condition (Eq. (2.1)) arises as a consequence of the incompressibility condition, $d\rho/dt = 0$. The conservation of momentum equation (Eq. (2.2)) describes the fluid acceleration due to the force from a pressure gradient, the internal viscous stress forces, and any constant force fields (gravity in this case). In reality, there are no known completely incom-

pressible fluids. However, there are many fluids, such as water, which can be considered to incompressible due to the speed of sound in the medium.

- **The violent incompressible free-surface flow:** An incompressible free-surface flow is considered to be violent when internal viscous stress forces of the fluid are large enough to overcome the external forces (this is seen from Eq. (2.2)). For example, in a breaking-wave, the internal stresses of the fluid overcome internal body forces such as gravity, resulting in a separation of the fluid's surface, also referred to as fragmentation, as seen in Fig. 2.2.



Fig. 2.2: A violent breaking-wave [22]

However, in particularly violent cases, such as breaking-wave impacts, aeration can occur at the free surface and the Mach number will change, causing significant compressibility effects.

2.3 Applications and the need for computational fluid dynamics (CFD)

In hydrodynamics, there are many violent free-surface flows of concern to engineers during the design process. The following list includes such engineering applications, which are also recognised as “real world” problems where the flows are in uncontrolled conditions:

- **Waves and wave-structure impacts:** The behaviour of oceanic waves are of great importance to the design of offshore and coastal structures, which can also extend to near-coastal regions inland that are susceptible to tsunamis.

In coastal areas, sea walls and breakwaters aim to protect infrastructure from incoming waves [23]. Further offshore, the environment is highly unpredictable and aggressive. Oncoming waves from multiple directions pose a risk to ships and structures such as oil platforms and wind turbines [24]. Furthermore, the movement of ships and response of floating objects can generate waves themselves, increasing the complexity of the surrounding flow [21]. In addition to complex, violent, and highly fragmented flow behaviour, high impact forces, wave overtopping, and sediment scour are all possibilities in the presence of waves.

- **Sloshing problems:** Sloshing refers to the movement of liquid within a container and characterises a highly non-linear free-surface with a complex range of motion types [25]. Dynamic loads caused by the motion of the liquid can have a significant effect on the structure in which it is contained, and in some cases the environment outside of the structure such as waves propagating from the hull of a ship [26]. The phenomenon is applicable to containers such as automotive vehicles including petroleum transport trucks, anti-roll vessels in ships, and tall structures subjected to earthquakes such as water towers or buildings with tuned liquid dampers.
- **Open-channel flows:** Effective and durable design of water infrastructure systems is vital for society and the economy [27]. Investigation of flows over weirs, in rivers, spillways, outflows, and other waterworks, can aid with design efficiency and determine whether the flow has any detrimental effects such as cavitation damage [28], which arises from large local pressures created from a collapsing bubble in highly aerated flow. The numerical investigation of phenomenon such as the hydraulic jump gives insight to the complexity of the flow which hosts considerable pressure fluctuations and fluid-air mixing [29].

When considering such flows in engineering design, the role of CFD is complementary, or even tantamount, to physical experiments. Although experiments provide results of real flow, there are many cases in which they are impossible, too

impractical, and time consuming. The scale of the aforementioned problems are such that prototypes are often too large and expensive to test at full scale and reducing the dimensions restricts the observation of any complex flow phenomena. Some problems require facilities which simply do not exist [30,31].

CFD is required to fill in the gaps where experiments are limited and reduce the uncertainty of particular engineering design problems. Furthermore, numerical simulations offer the capability to repeat experiments with different designs or parameter calibrations without significant additional resources. With the correct mathematical models, CFD has the ability to approximate with high fidelity any desired flow at the actual scale of the problem. The use of CFD is now common practice in the modern industry, a necessity for engineering design (of such complex problems) to be an efficient process.

2.4 Options for CFD

The field of CFD offers a wide range of numerical methods and approximations for many different fluid phenomena. For the applications of this thesis, the numerical solver must be able to compute highly non-linear incompressible free-surface flows of a violent nature characterised by fragmentations and discontinuities.

2.4.1 Mesh-based methods vs meshless methods

There are two ways to describe mathematically a flow field in CFD, Eulerian, and Lagrangian. In the Eulerian frame, one measures fluid variables and rate of change over a fixed location in space. Any fluid quantity, ϕ , is expressed as a function of a fixed position, \mathbf{r} , and time, t , i.e. $\phi = \phi(\mathbf{r}, t)$. On the other hand, the Lagrangian approach follows identifiable pieces of matter (material elements) moving with the flow. Quantities in this case are expressed as functions of the position of a fluid mass centre point, \mathbf{r} , with an initial position, \mathbf{r}_0 , and time, i.e. $\phi = \phi(\mathbf{r}(t, \mathbf{r}_0), t)$.

Conventional meshed-based applications follow an Eulerian description of the flow. The mesh, or computational grid, comes from the discretisation of the compu-

tational domain as multiple control volumes, and allows for the efficient computation of governing equations with the approximation of derivatives and other mathematical operations. Finite volume methods (FVM), finite difference methods (FDM) and finite element methods (FEM) all rely upon a computational grid and are widely used within the engineering industry for most applications [32].

In the context of simulating free surfaces, there exists several mesh-based methods involving the advection of particles on the underlying mesh, with a vast number of variations. Here, a brief general overview of the more popular methods is made:

- **Particle-in-cell (PIC):** The particle-in-cell (PIC) method [33, 34] employs particles which are able to move with the velocity field, and a fixed background Eulerian mesh. Particle data is projected onto the background mesh to solve the pressure field whilst enforcing the incompressibility condition. Pressures at particle positions are then found via interpolation of the mesh points. The method was purposefully designed for hydrodynamic problems of large distortions and discontinuities. The original formulation suffered from excessive numerical diffusion due to the back and forth interpolation between the particles and background mesh [35], although this has since improved greatly with development [36–38].
- **Marker-and-cell (MAC):** The marker-and-cell (MAC) method [39, 40], defines fluid-containing cells in the computational grid with massless “marker particles”. The presence of a free surface in a cell is identified if the cell contains marker particles and the adjacent cells are without any such particles. Marker particles move from one cell to another at each time step according to an interpolation of the background velocity field. The method has since evolved, improving the resolution of the free-surface interface and including definitions of the surface curvature and normals [41]. However, it has been criticised for its heavy computational memory requirements compared to other meshed methods.
- **Volume-of-fluid (VOF) and Level set method :** The volume-of-fluid

(VOF) [42] method eliminates the need for storing particle data and instead represents the amount of fluid inside a cell with a fractional volume value. VOF is now perhaps the most popular of the mesh based methods for free-surface flows due to its simplicity. However, with the use of volume fractions, the method originally saw difficulty in defining correct fluid surface curvatures, which the VOF community has invested great effort to develop [43–46]. Another popular method, the Level set method [47], defines the surfaces with smoothed signed distance functions that can be solved by high-order schemes for the advection equation. The LS method though produces higher numerical error than VOF, especially near interfaces with extreme deformations and separation. Coupled Level set / VOF methods [48–50] have since emerged to give the high numerical accuracy of VOF, and the well-defined interface curvatures of the LS method providing a smoothness of discontinuous physical quantities [50]. Unfortunately though, such methods still does not obey exact conservation of mass around the free-surface interface [51].

Some meshed-based methods use an arbitrary Lagrange-Euler (ALE) approach [52, 53], which allow the vertex points of the mesh to move according to Lagrangian mechanics or any other prescribed mesh velocity. Although such methods can match the free surface better than the purely Eulerian techniques, the efficient handling of excessive mesh deformation remains to be a problem in some cases [54].

In addition to the more traditional aforementioned CFD methods, the Lattice Boltzmann method [55, 56] has made a notable emergence as a promising tool for the application of fluid dynamics problems. In the method, a lattice pattern is used to define grid nodes and the governing Lattice Boltzmann equation is used to determine the transport of particle distribution functions (and therefore particles) across the lattice. A local equilibrium distribution function is used to recover flow equations. The method was derived from the lattice-gas automata (LGA) [57] method, and originally used to model problems of microscopic and mesoscopic nature but has since been applied towards a diverse range of macroscopic free-surface flow applications [58–62]. The solution of the Lattice Boltzmann equation compared to

the Navier-Stokes equations possesses several advantages. There is no non-linear convection term, so the formulation is relatively simple, this includes an equation of state for pressure when solving incompressible flow. The simplicity of the method also makes it highly parallelisable [63] and there is relative ease for including complex geometries [64]. Areas still in development for the method however include the accurate enforcement of boundary conditions, limitations regarding the Mach number, and memory consumption [65].

Compared to meshless methods, meshed methods (based on FEM, FVM, and FDM) have been so successful due to their efficiency and accuracy for a wide range of applications. Higher order approximation schemes are available and physics such as incompressibility and turbulence can be implemented more rigorously. But despite the numerous methods and successful commercial software packages (e.g. OpenFOAM [1, 66], ANSYS Fluent [2], STAR-CCM+ [3]) in CFD, the mesh-based techniques still show disadvantages when it comes to modelling free-surface flows, and more specifically those of a violent nature. Mass conservation at, and large transient deformations of, the free surface are typical problems. Although one can reduce the effects of such issues with adaptive remeshing and refinement, this solution can be a highly expensive procedure, and there is the complexity of maintaining accuracy as information is passed between successive meshes [67]. Moreover, the methods struggle to deal with extreme changes in topology, describing break-up of the fluid, which are present in many violent flows.

Meshless methods, on the other hand, are able to deal with such problems. They follow a Lagrangian description of the flow field and usually require no explicit treatment of the free surface. The domain is discretised as a set of computational points which each contain physical properties of the local flow field [68]. Derivatives and quantities are found through interpolation and points are free to move according to governing equations. Due to the Lagrangian nature of meshless methods, they are ideal for problems involving large deformations, discontinuities and highly non-linear complex phenomena, which are all present in violent free-surface flows [67]. A major drawback of these methods though, is their large computational expense

which originally hindered their development. As explained later in Section 2.6.1 the advancement of computing power over the years has enabled meshless particle methods to gain traction and develop towards more practical applications. Presented in the following is a brief overview of several meshless methods:

- **Meshless local Petrov-Galerkin method (MLPG):** The meshless local Petrov-Galerkin method (MLPG) was first proposed by Atluri and Zhu [69] and based on the local symmetric weak form and moving least squares (MLS) approximation. Integrations are made within local domains of various regular shapes such as spheres, rectangles, and ellipsoids etc. However, the MLS approximations create problems in boundary condition implementation and it is, for certain problems, computationally expensive relative to other methods [70]. The applications of MLPG are mainly in solid mechanics, but the method has been exercised in hydrodynamics such as within an extension of the method by Ma [71] to simulate water waves; and the interaction of violent waves with elastic structures using the MLPG method based on Rankine source (MLPG-R) by Sriram and Ma [72]. Developments made by Najafi et al. [73] allowed for Reynolds numbers up to and including 10,000, whereas earlier formulations restricted the method to Reynolds numbers of no more than 400 due to instabilities [74].
- **Local radial point interpolation method (LRPIM):** Liu and Gu [75] developed the local radial point interpolation method (LRPIM), which randomly distributes an N number of nodes into a domain and evaluates a function $\phi(\mathbf{r})$ using radial basis functions. Compared to MLPG, LRPIM has a lower computational cost, due to its simple interpolation [75], and better capability for dealing with boundary conditions [76]. LRPIM was originally created for the analysis of free vibration in 2-D solids but has since been used for modelling the dissipation process of excess pore water pressure [77] and a number of natural convection problems in incompressible flows [76].
- **Diffuse element method (DEM):** The diffuse element method (DEM) by

Nayroles et al. [78] is an expansion on the principles of the FEM. FEM interpolants are replaced with a weighted least squares fitting evaluated over a local domain of nodes. This approximation is then used in the Galerkin method to create the discrete functions. This makes for a more accurate computation of gradients compared to the FEM model. Kronguaz and Belytschko [79] noticed the original formulation constructs derivatives that, although consistent, are not integrable, imposing difficulties when modelling more complex fluid flows [80]. However, Kronguaz and Belytschko [81] proposed an extension of the method to solve the issue and also increase the rate of convergence.

- **Dissipative particle dynamics (DPD):** First formulated by Hoogerbrugge and Koelman [82], dissipative particle dynamics (DPD) aimed to be a combination of an improvement upon two methods: molecular-dynamics (MD) [83], and lattice-gas automata (LGA) [57]. The first application of the method was shown to be computationally cheaper than MD, and much more flexible than LGA when simulating microscopic hydrodynamic phenomena. On the other hand, the model is isothermal, meaning a temperature gradient cannot be created as the method follows Brownian dynamics and does not conserve energy [84].
- **Vortex methods:** Vortex methods, first proposed by Chorin [85], solve the Navier-Stokes equations with the vorticity variable instead of the velocity field. The approach works on the basis that contours moving relative to a diffusion velocity conserve velocity circulation. The technique considers viscosity and density to remain constant and mainly deals with viscous, incompressible fluids. Relatively small dependence on the Reynolds number and infinite region capabilities are a couple of the attractive qualities of the method [85]. However, in Chorin's method, single point vortices are created and accuracy is lost very quickly if the delta function is of the same relative magnitude to the original vortex spacing [86]. This can give rise to singularities, and so "vortex blobs" of finite width are commonly used [86]. Investigated applications mainly revolve around situations where there is high vorticity and/or turbu-

lence. These include the analysis of bluff body/cylinder flows and wall-vortex interactions by Cottet et al. [87], and the vortex rope phenomena for unsteady flows [88].

- **Smoothed particle hydrodynamics (SPH):** Smoothed particle hydrodynamics (SPH) splits the domain into a discrete set of interpolation points, more commonly known as particles, where each one possesses its own set of physical properties such as position, mass, and velocity, etc. The formulation of, and equations used in SPH are much simpler relative to other CFD methods [68]. SPH was first introduced by Gingold and Monaghan [4] and Lucy [5] for the investigation of astrophysical phenomena, but is now popular within fluid mechanics for its effortless ability to model complex free-surface flows including applications where large deformations and discontinuities may occur. However, there are difficulties in wall treatment, and the time step is required to be much smaller than other techniques which makes the method computationally expensive [6]. SPH has also been combined with DPD to create the smoothed DPD method [89] for mesoscopic-scale simulations where thermal fluctuations in the fluid are important.
- **Finite pointset method (FPM):** Originally introduced as “General Smoothed Particle Hydrodynamics” by Kuhnert [90], the finite pointset method (FPM) is a derivative of SPH, except that instead of interpolating values through symmetric kernels, the FPM approximates by the MLS method. Similar to SPH, particles move with velocity and possess their individual properties of the domain with the exception of mass. This method has been found to successfully demonstrate free-surface flows and surface tension problems [91], and also two-phase flow separation [92]. The FPM is advantageous over SPH in terms of dealing with boundary conditions, although some problems can arise with the Neumann boundary. The method can be a time consuming process due to the large number of matrices calculations [93].
- **Moving particle semi-implicit method (MPS):** Koshizuka and Oka [19]

used a Taylor series to discretise the incompressible Navier-Stokes equation to create the moving particle semi-implicit (MPS) method. Souto-Iglesias et al. [94, 95] have shown that MPS and SPH are essentially equivalent to each other, i.e. the MPS weighting functions can be derived from the SPH kernels and vice versa. They show the same inconsistency problems and the same stability and conservation properties, etc. The differences lie in the implementation, a semi-implicit time marching process is used and there are no calculation of kernel gradients in MPS. The solution has proven to be applicable for free-surface flows via a simulation of the classic dam break case [19], and a modified method for two-phase flows by Natsui et al. [96]. With respect to a particular interest to this project, applications of MPS for waves in open channels and wave impact pressure have been made by Imanian et al. [97] and Khayyer and Gotoh [98] respectively.

This study is focused upon the simulation of violent free-surface flows, which can involve highly non-linear and complex phenomena, extreme deformations of the free surface, and numerous discontinuities. Due to its relative simplicity and robustness as a highly flexible method which is well-suited for such flows, described in the following sections, SPH is chosen as the method to simulate these applications. The method has received significant development towards improving the accuracy and robustness during the past 2-3 decades. Moreover, the number of engineering applications simulated with SPH has significantly increased with ongoing advancements in computational processing power and algorithms.

2.5 Smoothed particle hydrodynamics (SPH)

Smoothed particle hydrodynamics (SPH) [4, 5] is one of the earliest Lagrangian computational methods. In SPH, a point cloud of “particles” represent the physical material of the domain. By convolution with a weighted kernel function, integral interpolations are used to determine variables of the flow field, such as velocity and pressure (more details of the SPH fundamentals are found in Section 3.2).

The SPH integral function can theoretically be applied to any set of governing equations. Therefore, since its conception, in addition to its original application to astrophysics [4,5,68,99], SPH has been successfully developed for an extensive range of research fields including solid mechanics [100–102], thermodynamics [103–105], and ballistics [106–108]. However, for the context of this thesis, the focus is on SPH for fluid mechanics and free-surface flows. SPH describes fluid as either weakly-compressible, or incompressible. The key differences between the two formulations are their method for treating density and resolving the pressure field.

2.5.1 Weakly-compressible SPH (WCSPH)

The original SPH formulation, known as weakly-compressible SPH (WCSPH), resolves pressure via an artificial equation of state (EoS) [6, 109]:

$$P = \frac{\rho_0 c_0^2}{\gamma} \left[\left(\frac{\rho}{\rho_0} \right)^\gamma - 1 \right] + P_0,$$

where, at a point in space, P and ρ are pressure and density respectively, subscript 0 denotes a reference value of the variables. c_0 is a numerical speed of sound, and γ is a constant equal to 7 for water [6]. The EoS is such that a Mach number, M , of approximately 0.1 is used for modelling incompressible flows. Subsequently, density is permitted to theoretically vary within approximately 1% of the reference value such that compressibility effects are $O(M^2)$ [68].

2.5.1.1 Free-surface flows with WCSPH

Monaghan [6] first applied SPH (as WCSPH) to free-surface flows in 1994 demonstrating several applications including a dam break and spilling waves. Shortly afterwards, Monaghan continued to use WCSPH to study gravity currents and solitary waves [110, 111]. SPH has since become increasingly recognised as a suitable and robust method for simulating violent free-surface flows in a large range of applications.

Since Monaghan’s work in the early stages of SPH for free-surface flows, the

behaviour of water waves has been extensively investigated with WCSPH including wave propagation and breaking waves [112–116], green water overtopping [117,118], and landslide generated waves [119–121].

Highlighted here, are recent developments of particular relevance to hydrodynamic engineering application involving free-surface flows. Aghili et al. [122] investigated solitary wave interaction with a submerged horizontal plate at varying depths. Comparing to the experimental data of Hayatdavoodi and Ertekin [123], the SPH model successfully reproduced the wave height at three different locations as it propagates over the plate. However, the authors stated the traditional WCSPH method introduced significant errors in the pressure field and required the application of a density renormalisation filter before achieving the correct plate pressure forces.

Crespo et al. [10] compared their SPH model to experimental data [124] and a mesh-based IH-Foam CFD solution [125] of waves interacting with an offshore oscillating water column. Both the numerical models presented close agreements with the free-surface oscillations within the water column chamber, but it was the SPH results which demonstrated higher accuracy. The IH-Foam model overestimated some of the peak oscillations and did not consistently match the profile.

Altomare et al. [126] demonstrated how SPH can be used for estimating wave impacts on coastal structures where scale physical models of the Zeebrugge Harbour and Blankenberge Marina in Belgium were used for the experimental data comparisons. In both cases, there were initial agreements between numerical and experimental results of the free-surface elevations. However, as wave reflections increased later in the physical and numerical models, disparities became more apparent. The impact forces in both cases also showed the same trend. In other works, Altomare et al. [127] have used second-order wave generation to model long-crested regular and random irregular waves in SPH. An active wave absorption model was implemented, which can prevent wave reflection effects in applications with no physical solid boundaries around the domain such as in offshore engineering. The research is an important step for SPH to move towards simulating realistic sea states.

The dam break case is frequently used in SPH to demonstrate the suitability of the method for transient free-surface flow. Crespo et al. [128], Violeau and Issa [129], and Gómez-Gesteira et al. [130], all achieved simulations where the free-surface profile is in close agreement with laboratory experiments of a dam break with a dry bed.

Many authors investigate the effects of waves propagating and impinging on a structure by use of a dam break with a wet bed. Crespo et al. [131] simulated such an application modelling the experiment of Janosi et al. [132], where the numerical results were able to predict the mixing interface between the two bodies of water originally separated by the lock gate.

Dam break studies involving the impact of an object have also been made. Gómez-Gesteira and Dalrymple [133] were the first to compare 3-D SPH simulations with experimental data of a dam break wave impacting on a column. Compared to the experimental data, their numerical results reproduced accurately the velocity-time history of the wave front before the first impact and correctly predicted the net forces during impact on the front and back of the structure. However, the WCSPH results between these two impact events over-predict the net forcing. More recently, using a significantly finer resolution (800,000 fluid particles compared to 15,000 in [133]), Pan et al. [134] repeated the same numerical experiment but compared their results to the experimental data used by Raad and Bidoae [135]. Comparing the time history of the horizontal force on the column, the WCSPH results here are in closer agreement to the experimental data than that produced from Raad and Bidoae's mesh-based method¹.

The SPH European Research Interest Community (SPHERIC) [136] have devised a series of SPH benchmark test cases including comparisons to the dam break experiment of Kleefsman et al. [137] where a small box is placed in the path of the flow. Crespo et al. [138] showed their numerical results converged towards the experimental wave-height time-history data at three different locations, and the profile

¹The Eulerian-Lagrangian marker and micro cell method (ELMMC) [135] uses an Eulerian background mesh for computing variables and moving Lagrangian "surface-markers" for tracking the free surface.

of pressure forces imposed on the box with time were in general agreement although the peak impacts were highly overestimated. Compared with the VOF model of Kleefsman et al. [137], the pressure-time plots of both numerical methods were very similar and mostly matched the experimental data. However, unlike the SPH results, the VOF model exhibited several pressure spikes due to its method of tracking the free-surface via cell marking on a computational grid. Mayrhofer et al's [139] WC-SPH method with improved boundary conditions (unified semi-analytical boundary conditions) gave better peak-impact predictions than those from Crespo et al. [138]. Their profile of the pressure-time plots were similar to the Kleefsman et al. [137] VOF simulations with differences due to the absence of the air-phase in the SPH model.

SPH has also been shown to be a suitable method for modelling sloshing problems. Souto-Iglesias et al. [140] conducted a series of anti-rolling tank experiments and subsequently modelled them with WCSPH. They produced agreements for phase lags in a rectangular tank without baffles and a C-section shaped tank for the majority of tests. Accurate predictions of the free-surface profile for a rectangular tank with and without baffles were also made including the presence of breaking waves. Where the numerical results did not match very well with the experiments, the authors stated their SPH formulation required improvement and the 2-D model could not capture some strong 3-D non-linearities. Consequently, Souto-Iglesias et al. [26] sought to improve the model, which was achieved by: (i) evolving density via the SPH continuity equation, as in traditional WCSPH, instead of the SPH integral interpolation function, (ii) improving the boundary conditions with implementation of a repulsive force, (iii) including an artificial viscosity to take account for viscous stresses that were previously neglected, and (iv) using a second-order leap frog predictor-corrector time stepping scheme instead of an explicit Euler. However, it should be noted that none of the improvements were particularly new ideas to SPH. Nevertheless, the methodology demonstrated SPH to be able to reproduce accurately the phase lags, moment amplitudes and free-surface shape of all the experiments. However, pressure predictions were not presented, the authors stated the method

produced small density errors (and therefore errors in the pressure field) which would magnify towards the late stages of the simulation.

Federico et al. [29] simulated a series of open-channel flows with SPH including hydraulic jumps of an undular and weak nature. Each of the hydraulic jump cases achieved the theoretical downstream depth, although free-surface oscillations increased with the downstream-upstream depth ratio. Compared to experimental data [141], agreement of horizontally averaged velocity profiles was demonstrated. The authors also numerically reproduced their own experiments of a flash-flood impacting on a bridge. The time history of the vertical and horizontal forces, and pitching moment acting on the bridge were measured. For all three plots, the average trend of the WCSPH results matched well with the filtered experimental data, however severe fluctuations were evident. López et al. [142] also experienced large fluctuations, but for the pressure field in a hydraulic jump. They did however, observe an improved free-surface profile of the flow at different time instants as a consequence of introducing an extra parameter, based on vorticity, to their viscosity model. Both sets of authors did not include the presence of air in their models, implementing the extra phase would better capture the air entrapment in the hydraulic jump.

To determine the potential of cavitation effects occurring, the pressure variations of flow at the top of a stepped spillway was investigated by Husain et al. [28]. Prior to modelling a full spillway, flow over a broad-crested weir was simulated and compared to the experimental data of Hager and Schwalt [143]. Both experimental and numerical velocity profiles were in agreement of each other as well as the curve plots of the pressure head above the weir crest, although the SPH results produced lower magnitudes. For the full spillway, they modelled the experiments of Meireles and Matos [144]. Once again, the SPH results matched the velocity data obtained from the laboratory experiments and the flow depth along the chute slope for discharge rates were consistent with both data sets. Meireles and Matos neglected to look at the pressure in their experiments, but nevertheless the numerical pressure predictions showed comparable behaviour to other experimental spillways of simi-

lar configuration [145, 146]. Although WCSPH results yielded reasonable pressure values, the flow field appeared to be noisy.

2.5.1.2 Drawbacks of WCSPH

The traditional WCSPH formulation is easy to implement and has shown many successes in its early stages [147]. However, the simplicity of the methodology comes with some well-known issues. The problems which make WCSPH an unattractive method arise from the stiff EoS:

- The speed of sound is inversely proportional to the time step, and therefore a very small time step is permitted in accordance to the Courant-Friedrichs-lewy (CFL) condition [148]. This incurs a severe computational cost.
- The previous point motivates the use of an “artificial” speed of sound, much lower than that in reality. However, a value too low can cause the propagation of sound waves across the domain and subsequent instabilities. Thus, the artificial speed of sound is commonly chosen to be at least 10 times larger than the maximum particle velocity, which has been found to reduce such detrimental effects for fluid propagation problems [6, 111].
- The most common problem with WCSPH is the noisy/inaccurate pressure field, which is apparent from much of the aforementioned literature [26, 28, 122, 138, 141, 142]. The EoS is empirical and was originally developed to fit experimental data rather than mathematical derivation of physics. The consequence is that the pressure varies drastically as density deviates from its reference value [149], which arises as a consequence of inaccurate interpolations. False pressures propagate throughout the domain and subsequently lead to instabilities and smaller time steps, increasing the computational time further.

In more recent years however, WCSPH has been much improved over its traditional form. Developments have been towards:

- Reducing noise in the pressure field through velocity [6] and density field re-normalisation procedures [150, 151], or by reducing density field fluctuations with an additional diffusive term [152, 153].
- Increasing the accuracy of SPH interpolations [154–156].
- Addressing the truncation of the SPH kernel interpolation at boundaries, namely improving SPH boundary conditions [14, 157–159]
- Improving particle distributions with tensile instability control (preventing the non-physical clumping of particles) [160] and particle shifting [161–163] (see Section 2.5.2.1).

Le Touzé et al. [164] has also shown accuracy improvements through shifting the numerical spectrum of response with various techniques such as:

- Using an initial particle distribution with a small amount of applied “noise”, where each particle is moved a distance $\pm 0.5\%$ times the initial particle spacing from the original Cartesian particle arrangement. Simulations initialised with a non-Cartesian particle configuration have been observed to evolve with a maintained regular distribution of particles [164, 165].
- Using a choice of sound speed close to the weak-compressibility limit, Mach 0.1, to reduce spurious pressure oscillations.
- Using a higher-order scheme with a periodic “remeshing” of particles, which allows for the avoidance of some correction techniques such as tensile instability control.

Many years of research have been dedicated to developing the WCSPH pressure field and although improved, the modern methods are not without added implementation effort/complexity and computational expense.

The deficiencies of WCSPH has given rise to other SPH formulations. Such alternatives include the use of Riemann solvers [166–168], where traditional particle interactions are replaced with a unique Riemann problem between each particle

pair. This solution improves numerical stability without the use of artificial viscosity. Parshikov et al. [167] used a Godunov-type SPH with approximate Riemann solutions to solve discontinuities in shock-wave problems. Vila et al. [166] employed a similar scheme, referred to at the time as SPH-ALE, which used an approximate Riemann solver to reduce numerical noise in the pressure field. SPH-ALE has been shown to improve stability and produces significantly less noise in the pressure field compared to traditional WCSPH [51, 169]. However, as a hybrid version of WCSPH, an unphysical speed of sound within the equation of state is still present and the scheme is still not truly incompressible.

2.5.2 Incompressible SPH (ISPH)

Incompressible SPH (ISPH), on the other hand, typically enforces incompressibility by keeping density constant and using a pressure Poisson equation (PPE) to resolve the pressure field. A divergence-free velocity field, $\nabla \cdot \mathbf{u} = 0$, arises as a consequence of the constant density field:

$$\nabla \cdot \left(\frac{1}{\rho_0} \nabla P \right) = \frac{1}{\Delta t} \nabla \cdot \mathbf{u},$$

where P is pressure and ρ_0 is the reference density as before in Section 2.5.1, t is time, and \mathbf{u} is velocity. The PPE is generally solved as a linear matrix system, in the form of $[\mathbf{A}] \mathbf{x} = \mathbf{b}$, via an iterative solver algorithm [170]. This is discussed further in Sections 3.3.2 and 4.6.

In an alternative approach, Ellero et al. [171] avoided the use of a PPE and enforced incompressibility through kinematic constraints and Lagrange multipliers to achieve a constant volume. However, this method has not received much development since its inception and is not considered in this study.

2.5.2.1 ISPH formulations

In 1999, Cummins and Rudman [172] avoided the use of a speed of sound via a projection method [173] based upon a Helmholtz-Hodge decomposition [174] where

a vector field can be expressed as a divergence-free component and the curl-free component. In the context of Cummins and Rudman’s work, they project the velocity vector field onto a divergence-free space and subsequently solve the pressure field via a PPE to enforce the incompressibility condition. Here, the pressure is not treated as a thermodynamic variable and subsequently there is no EoS. The method has come to be known as incompressible SPH (ISPH), or ISPH with a divergence-free velocity.

Cummins and Rudman discussed significant advantages of their projection-based method compared to WCSPH, mainly stemming from the elimination of the sound speed variable from the algorithm. The CFL time step constraint was now only dependent upon the fluid velocity field and thus larger. Furthermore, numerical stability is also improved since sound waves are no longer a problem. Lee et al. [175] showed that ISPH with a divergence-free velocity exhibited smooth and near noise-free pressure fields in contrast to traditional WCSPH for a number of 2-D internal flows and dambreak simulations. Despite the larger computational expense per time step of ISPH, due to the larger time steps permitted, the overall simulation times were 2-20 times faster than the WCSPH tests.

A few years after Cummins and Rudman’s SPH projection method, Lo and Shao [176] (and similarly Shao and Lo [177]) developed upon the work to create their own ISPH formulation for the simulation of near-shore solitary wave run-up. Lo and Shao’s “ISPH with density invariance” method, similar to the moving particle semi-implicit method (MPS) [19, 178], uses a prediction-correction time stepping scheme where a temporal velocity is found through an explicit time integration in the prediction stage. The subsequent variations in density, characterised in the continuity equation, are then projected onto a divergence-free space to enforce incompressibility within the correction stage and used as the source term on the right-hand side (RHS) of the PPE. The proposed ISPH formulation opened up the methodology to the application of free-surface flows where particle densities with more than 1% fluctuations below that of the fluid bulk were used to track the free surface. Lee et al. [175] later introduced a simple method for identifying free-surface particles by

computing the divergence of position at each particle.

It is worth noting the work of Souto-Iglesias et al. [94,95], who showed that ISPH and MPS were mathematically equivalent (as mentioned before in Section 2.4.1). Therefore, advancements within the field of MPS should also be considered for ISPH. Authors [179,180] have since applied developments of MPS to enhance the accuracy of the ISPH with density invariance method.

Hu and Adams [181] combined the two aforementioned ISPH methods, enforcing both the density invariance and divergence-free velocity conditions. Particle positions are iteratively adjusted to satisfy the former condition and an intermediate velocity field is projected onto a divergence-free space for the latter. They showed accurate results for various multi-phase and free-surface applications demonstrating a superior ability to represent sharp interfaces with density discontinuities compared to an FDM solution [172].

Xu et al. [161] investigated the three ISPH algorithms, assessing the accuracy and stability of each method. They showed that the original ISPH method [172] with a divergence-free velocity field produced accurate results. However, severe density-error accumulation from particle clustering occurred, particularly for high Reynolds numbers which lead to unstable simulations. Such observations were also made by Cummins and Rudman [172], and Hu and Adams [181]. ISPH with density-invariance [176,177] proved stable for the Reynolds numbers tested (up to 1000), although the fluid field was inaccurately predicted and exhibited a lot of numerical noise. The compound method of Hu and Adams [181] combined the advantages of both divergence-free velocity and density invariance ISPH formulations producing results showing stable and uniform particle distributions and accurate flow fields. However, the requirement to solve two Poisson equations affected the computational efficiency, with simulations taking at least 4 times longer than the other methods.

Following their investigation on the various ISPH methods, Xu et al. [161] proposed using a “particle shifting” technique for ISPH, to be executed at the end of each divergence-free projection step, which moved particles slightly off their respective streamline paths to maintain uniform particle distributions for improved accu-

racy and stability of ISPH simulations. This was based upon the work of Nestor et al. [182], who identified that when the fully Lagrangian Navier-Stokes equations are solved accurately, particles move along streamlines which subsequently causes particle clustering and numerical instabilities. They remedied this effect in their finite volume particle method (FVPM) by proposing a “velocity correction” for each particle based upon the local particle distribution. Xu et al. [161] showed that for a relatively small amount of extra computational effort, their ISPH with divergence-free velocity and particle shifting method gave both accurate and stable simulations for several internal flows. The particle shifting methodology is a significant development in ISPH as it enables the method to be accurately applied to high Reynolds number flows and maintain near noise-free pressure fields. Particle shifting has been shown to also be advantageous to WCSPH [163,183]. Shadloo et al. [162] introduced a similar technique to show comparable improvements for both WCSPH and ISPH. The redistribution of particles by shifting can help prevent, or reduce the effects, of non-physical artefacts in a simulation such as voids and extreme particle clumping.

Following Xu et al. [161], Lind et al. [11] developed the particle shifting method to include free surfaces using a Fickian-based particle shifting technique where particles are seen to be “shifted” from regions of high to low particle concentrations. Simple treatment of the shifting for particles near the free surface allowed for the simulation of violent dam break flows and waves in ISPH with highly accurate pressure fields. The treatment of shifting near the free surface has since been further generalised by Khayyer et al. [184].

2.5.2.2 Free-surface flows with ISPH

ISPH has been used for a wide variety of free-surface flows due to its ability for highly accurate pressure prediction. Section 2.5.1.1 identified some of the successes in WCSPH for free-surface flows within the context of hydrodynamic engineering applications. However, the method’s pressure field has been shown to require improvement. Here, a selection of ISPH literature demonstrates its suitability to meeting the aim of the project.

Khayyer et al. [185, 186], and Khayyer and Gotoh [187], paid attention to enhancing the accuracy of the ISPH with the density invariance method. Such improvements included: (i) theoretically acquiring the exact conservation of linear and angular momentum, (ii) introducing higher order PPE source terms to reduce pressure fluctuations, and (iii) improved accuracy in free surface tracking (although this was around the same time as the method of Lee et al [175] mentioned in Section 2.5.2.1, which is more commonly used now). Their works simulating dam break impacts demonstrate smoother and more accurate pressure predictions of experimental data, and despite the relatively low resolution, the numerical free-surface profiles also manage to resolve several key flow features. In Khayyer et al's work [185], their improved ISPH method was compared with boundary element method (BEM) [188] and VOF/BEM [189] meshed-based methods for simulating a plunging wave breaker, ISPH was demonstrated to be the only method capable of accurately representing the wave height in both the pre and post-breaking stages.

Skillen et al. [9] implemented a simple smoothing function which altered the system of equations within the PPE matrix for particles near the free surface. The function uses a sinusoidal-type transition from the fluid bulk to the Dirichlet boundary condition at the free surface to replace the traditional step function. The authors also used a particle shifting distance coefficient based on CFL-type stability analysis. Their results showed a significant reduction in pressure fluctuations for a number of transient test cases involving objects impacting a free surface.

The Ph.D. thesis of Leroy [190] presents an ISPH method capable of industrial applications. The model developed the ISPH with divergence-free velocity and shifting method to include turbulence and buoyancy models, the implementation of unified semi-analytical wall boundary conditions, allowing for the general modelling of complex geometries, the imposition of open boundaries, and the use of parallel programming for improved computational times. The author compares the model to WCSPH (with state-of-the-art formulation) and VOF solutions for 2-D free-surface flow with a water wheel and dam breaks in both 2D and 3D. For each case, the ISPH results are more accurate and smoother in pressure predictions than those of

WCSPH. Despite the limited comparison between the multi-phase (air-water) VOF model and single-phase SPH methods, ISPH showed similar results to the VOF where air effects were negligible. The comparison also exposed VOF's inferior ability to resolve the free surface. The thesis also presented several confined flows, where ISPH performed very well in comparison to FVM solutions.

Following the introduction of shifting to ISPH for wave propagation [11], Lind et al. [191] applied the methodology to offshore engineering applications. They modelled regular waves and irregular focussed wave groups, of both non-breaking and breaking nature, impacting on either a fixed cylindrical column or taut moored rigid body. Computational savings are made by using a Froude-Krylov (FK) forcing to approximate 3-D wave loadings and therefore requiring 2-D simulations only. By comparison against laboratory experiments [20,192,193], the FK approximation was shown to be sufficiently accurate in predicting the peak forcings on a body subject to non-breaking waves. However, it was concluded that a full 3-D simulation is required to better capture the wave breaking effects and loadings.

Liang et al. [194] also looked at simulating waves with SPH, more specifically, the generation of solitary waves and their subsequent run-up and impact with a vertical wall. Both WCSPH and ISPH simulations were conducted where both achieved the desired wave height, however the ISPH results showed some discrepancies in the free-surface profile both upstream and downstream of the wave crest. Despite this error, the subsequent impact with a vertical wall once again proved ISPH to be superior to WCSPH in accurately portraying the pressure field. On comparison of the time series of force exerted onto the wall with experimental data [195], WCSPH displayed severe pressure fluctuations throughout, for all cases, sometimes with large over predictions. ISPH on the other hand, generally matched the experimental force profile, although there were still some oscillations (likely due to the use of the repulsive boundary condition [196]), they were significantly less pronounced than those of the WCSPH results.

Lee et al. [197] looked at the application of SPH to waterworks and spillways. Similar to literature mentioned in Section 2.5.1.1, the authors here also validated

their model against the dam break experiment of Kleefsman et al. [137]. Comparisons were made between their models of WCSPH and ISPH. Once again, the WCSPH simulation showed severe pressure fluctuations on the two pressure probes at the front of the box (facing the initial water column), so much so that the authors omitted those results from the probes on top of the box. Unlike the work of Crespo et al. [138] (as reviewed in Section 2.5.1.1), the traditional form of WCSPH was used here, and a comparison of the two works shows the original form of WCSPH requires artificial numerics to provide acceptable results. The ISPH simulation on the other hand, provided much smoother results of good agreement with the experiment without the need for empirical numerical fixes. However, the ISPH results did not come without some noticeable discrepancies. The water height at the original position of the column was under predicted for the latter half of the simulation, some particles were lost due to boundary penetration, and the initial impact pressures on the front of the box are highly underestimated (by about a factor of a half). The authors identified that all these issues could be resolved through improving the impermeability of the boundary condition for ISPH, increasing the resolution, and increasing the kernel support radius (i.e. the number of influencing particle neighbours). The latter two factors were restricted due to computational time and resources and so it was apparent that the acceleration of ISPH simulations was required. For that reason, it was only feasible for the authors to simulate their engineering application of a river dam spillway with WCSPH, and not the preferred ISPH.

Just as for WCSPH, the development of ISPH has been increasing in recent years, and the method is steadily becoming more popular for the simulation of free-surface flows. Described in the following are a few recent developments of ISPH towards such applications. Lind et al. [198] has looked at improving the physics of a simulation by use of a multi-phase model consisting of ISPH-water and WCSPH-air phases. This was applied to investigating the behaviour of water-air wave slam onto a rigid flat plate and confirmed the “cushioning-effect” air has in reducing the peak pressure loads from the water, which were accurately predicted. By enforcing the incompressibility of the water phase with ISPH and allowing some compressibility

with the air-phase using WCSPH, the physical speeds of sound for the two phases were maintained. This is not the case in simulations where WCSPH is used for both phases, as the sound-speed ratio between phases is usually inverted [183, 199].

Fourtakas et al. [200] coupled a quasi-arbitrary Lagrange-Euler finite element method (QALE-FEM) [201, 202] with ISPH for wave propagation problems. Their solver uses QALE-FEM to efficiently calculate the far-field wave properties, and ISPH for computing the local/near-field free-surface flow. An interface region between the two methods was created where QALE-FEM formed a pressure/velocity boundary condition for ISPH. This method of coupling allows one to simulate problems such as wave-breaking at higher resolutions with ISPH without the large domain size (and subsequent additional computational expense) required for wave generation.

Several authors [203–206] have recently explored the idea of an Eulerian-based SPH method. Such an investigation may seem, at first, counter-intuitive for the meshless method, but a number of benefits arise from maintaining a fixed regular distribution of particles including improved boundary conditions, reduced computational expense, and the use of higher-order kernels for improved accuracy and spatial convergence-rate [203]. Perhaps most significantly, the transition of an Eulerian to Lagrangian region is made relatively simple, compared to the coupling of separate methods. Following the work of Lind and Stansby [203], Fourtakas et al. [206] demonstrated the effectiveness of an Eulerian-Lagrangian incompressible SPH (ELI-SPH) formulation by simulation of periodic wave propagation. The bottom half of the initial water depth was filled with fixed Eulerian particles, whilst the top half with a free surface was represented as Lagrangian particles, able to move freely as normally in SPH. ELI-SPH demonstrated comparable accuracy against a higher-order potential flow model [207] where no noticeable numerical artefacts were present at the Eulerian-Lagrangian region interface.

Inspired by the combined divergence-free velocity field and density-invariance method of Hu and Adams [181] (see Section 2.5.2.1), Gui et al. [208] aimed to satisfy both conditions but without the expense of solving two PPEs. They did so, by ap-

plying arbitrary weighting parameters to each source term and then combining them into one PPE to solve. Just as in Hu and Adams' method, the advantages of both source terms were combined providing accuracy and stability without any additional subroutines such as shifting. Compared to the strictly density-invariant model, the new model maintained similar global density-errors, and accuracy was improved with reduced pressure fluctuations. The time histories of wave forces were also in agreement to those of a Reynolds-averaged Navier-Stokes (RANS) model [209]. However, the weighting parameters associated with each source term is chosen empirically and so the method would require calibrations for different test cases and possibly encounter difficulties with highly varying flows/domains.

Such developments within the field are not explored in this study, but are certainly of interest to the author and represent potential ideas for future development.

2.5.2.3 SPH boundary conditions for engineering applications

At the boundaries in SPH, there is a truncation of the kernel which is a significant issue. The accuracy of an SPH kernel interpolation is related to the number of neighbouring particles, and for particles near the boundary, their respective kernels extend beyond the computational domain where there are no particles present. This leads to inaccurate interpolations and/or particle penetration of the boundary. The treatment of boundary conditions in SPH (both WCSPH and ISPH) is an ongoing development, where researchers aim to provide treatments which are accurate and robust for complex 3-D geometries directly applicable to engineering problems.

There are numerous approaches for the treatment of rigid wall boundaries. Authors have proposed to fill the truncated area of the kernel with either additional fixed particles [128, 158], virtual fictitious particles [157, 210–212], or mirror image particles [14]. While others apply repulsive forces [111], to prevent penetration of the boundary, or provide a more analytical approach [139, 159, 213, 214]. However, with the extensive range of applications required in the engineering industry, it is not clear which approaches are most suitable.

Other research has focused on the comparison and review of different boundary

conditions [147, 215–217], and developing rigorous frameworks and theory on the subject matter [218, 219].

In this study, an appropriate boundary condition for a 3-D ISPH method must be established which can be applied accurately to a range of free-surface engineering problems.

2.5.2.4 Challenges in ISPH

It has been established that ISPH provides a pressure field which is near noise-free and more accurate than that of traditional WCSPH. However, the method is more complex. The pressure projection step requires more coding effort due to the solution of the PPE, which requires the setup of a matrix system and the subsequent linear solver. It has also been established that boundary conditions for SPH are a significant challenge and an area of ongoing development. For ISPH, the presence of the PPE adds complexity towards the implementation of flexible and accurate boundary conditions.

Whilst there has been significant advances in improving the accuracy of ISPH regarding conservation properties and recovering accuracy [11, 161, 179, 180, 217, 220, 221], there is little research that addresses the large computational expense of the method, which is a significant, if not the most, limiting factor restricting the advancement of ISPH. For most engineering applications, very large numbers of particles (10^7 to 10^9) are required, with the solution of large sparse matrices from the PPE accounting for over 90% of the computational time in a serial code. Therefore, parallelisation of the method is a necessity to allow for the simulation of higher particle numbers to be feasible. The next section of this literature review therefore explores the role of hardware acceleration in SPH.

2.6 Hardware acceleration and parallel programming for SPH

SPH is criticised within the CFD community for its large computational cost. With each particle having tens to hundreds of neighbours, searching for neighbouring particles and computing particle-particle interactions each time step requires a high number of computations. Moreover, the associated computational time increases non-linearly with increasing number of particles [15, 222]. In WCSPH, particle-particle interactions account for approximately 99% of the overall computation time on a single-core CPU. In ISPH, in addition to particle-particle interactions, the solution of the PPE must also be computed which takes up over 90% of the methods computation time. Therefore, the computational expense of an ISPH time step is significantly more expensive than that of the WCSPH time step.

The solution of the ISPH PPE is generally computed with an iterative linear solver (as discussed in Section 4.6) where the PPE matrix elements can be either stored or computed during execution of the solver. The former option of storing the elements incurs a significant computational memory requirement. The latter option was used by Leroy [190] where matrix-vector products were computed within the linear solver, eliminating the need to store the PPE. However, this method is very computationally expensive for simulations of large particle numbers as particle-particle interactions are computed several times within each iteration of the linear solver for the matrix-vector product. Therefore, the first option of storing the PPE matrix is adopted as the aim here is to accelerate the ISPH method. This poses a further challenge concerning the memory requirements of ISPH, which is discussed in Section 2.7. This is contrary to WCSPH where the computational memory requirements are relatively low [223].

There have been successful efforts to reduce the number of computations, such as the elimination of unnecessary particle interactions through neighbourlists [224, 225], but improvements in computation time by such algorithms are limited. Thus, for real engineering problems, the use of hardware acceleration and parallel programming

is the most appropriate option for achieving efficient computation of large particle numbers on the order of millions and above [138].

Hardware acceleration is the specialist use of computing hardware, instead of a single central processing unit (CPU), to execute operations and functions more efficiently. Hardware acceleration is utilised through parallel programming which enables the computation of repetitive calculations over large data sets simultaneously. Hardware acceleration can be achieved through high performance computing (HPC) on multiple CPU cores (ranging into the thousands), or through specialist hardware architecture such as the graphics processing unit (GPU) [138].

2.6.1 HPC with message passing interface (MPI) and open multi-processing (OpenMP)

One of the earliest parallel-computing enabling standards to arise is the message passing interface (MPI), first drafted in 1993 [226] with the first version released in 1994. MPI invokes communication between different processes that are executed in a parallel manner. MPI is powerful as it also allows the communication between thousands of computing processors in a network across heterogeneous architectures (distributed memory).

In the late 1990s, computing manufacturers' ability to increase the processor's clock rate gradually became restricted and so they started looking towards developing multi-core CPU architectures with shared and distributed memory layouts [223]. The open multi-processing (OpenMP) standard [227] was thus designed to take advantage of such architectures. The OpenMP language hosts a set of compiler directives and runtime library routines that enables parallelism between multiple cores sharing the same memory bank on a single processor (shared memory). Implementing OpenMP into existing code (providing the algorithm can be executed in parallel) is simple and acts as an extension to the users supported programming language. However, the scalability of a code with OpenMP is restricted by the number of cores hosted by the processor.

HPC has now evolved into the use of massively parallel CPU systems commonly

consisting of 10,000s of cores (although the largest in the world at the time this study is in the millions) [228], where a mixture of MPI and OpenMP is used to make full use of parallelism between cores and processors.

SPH is ideal for massive parallelisation, and several authors have demonstrated this with WCSPH [210, 229–231], where some achieve over 100 million particles on more than 1000 cores. The literature regarding the parallelisation of ISPH through HPC, on the other hand, is scarce. To date, Guo et al. [232] showed their model achieved an overall efficiency of about 81.3% for 1024 cores and could simulate up to 100 million particles. Yeylaghi et al. [233, 234] also implemented a parallel ISPH with OpenMP and MPI, although their scheme solved the Poisson equation explicitly without the use of a matrix, which affects accuracy and limits the time step size, taking away the advantages of ISPH.

Whilst HPC through OpenMP and MPI is a very powerful form of parallel processing and resolves the memory limitations in ISPH, the efficient implementation of SPH onto a memory distributed system is complex and time consuming and have only been recently addressed. Perhaps the most important challenge for an MPI implementation is domain decomposition and dynamic load balancing, which subdivides the domain into partitions over the number of processors in the computing network, and then subsequently attempts to assign each processor the same amount of computation work. This is especially difficult in SPH because of the moving particles and constantly changing particle-particle connectivity and interactions. Such a problem is further complicated when applying to heterogeneous systems. Efficient domain decomposition and load balancing is very important for reducing memory footprint and communication/latency times between processors. In ISPH, the order of particles in the PPE matrix will change the condition number, and subsequently the solution time. Therefore, domain decomposition is very important in this case [232].

Particles on the edge of a sub-domain in each partition also need to be identified as they may move outside of the sub-domain or interact with particles on neighbouring partitions. In ISPH, particles are advected three times per time step (projection

step, correction step and shifting, see Section 3.3.1) so measures should be taken to reduce the number of times particles migrate between processors. Similar issues also arise with data management of particles between processors.

Communication is often a significant limiting factor in massively parallel CPU systems as the transference of data is usually slow compared to the computation of it [31]. Using hardware of higher memory bandwidth can also be done, but this typically comes at an increased price.

In addition to the complexity of implementing SPH with HPC, massively parallel CPU clusters are large, expensive, and require continuous specialist maintenance [235]. The investment or access of a HPC cluster is not a viable option for many researchers and smaller companies in industry, and so the use of a graphics processing unit (GPU) for scientific computing has become increasingly popular, especially in SPH.

2.6.2 The graphics processing unit (GPU)

GPUs were originally designed for graphics visualisation within the video games industry. Their unique parallel architecture, highly suitable for the rapid processing of large quantities of data, were recognised as an alternative form of hardware acceleration for scientific simulations. Thus the development of general purpose GPUs, and dedicated parallel programming languages such as CUDA (Compute Unified Device Architecture) [12, 236] and OpenCL (Open Computing Language) [13, 237], for such applications soon emerged, providing a cheap, energy efficient and portable substitute to HPC [238].

The main component of a GPU, which attributes to its massively parallel architecture, is the streaming multiprocessor (SM). A GPU houses multiple SMs, where each SM schedules the execution of instructions across hundreds of computing cores concurrently. Further details of GPU architecture are found later in Section 4.2.

The earliest use of a GPU for SPH featured in the work of Amada et al. [239] in 2004, who parallelised the force computation stage on the GPU, but still conducted the initialisation of variables and particle neighbour mapping on the CPU. Their

CPU-GPU model was about two times faster than their CPU-only code for 2,000 particles. Greater speed ups were expected for simulations of more particles.

Shortly after, in 2005, Kolb and Cuntz [240] proceeded to demonstrate that the entire SPH computation could be performed on the GPU and only the loading and storage of data needed to be executed on the CPU. Since the output format of the GPU was in 2D, 3-D simulations were computed and represented as sets of 2-D “slices” (arrays). Each slice was associated with a different particle position, and contained within the slices were possible contributing particles. Any particle found to have no contribution (outside the kernel support radius) were automatically clipped from the slice. However, interpolation errors were apparent as sampling of 3-D quantities was carried out by communication of trilinear interpolation between slices.

The major breakthrough for SPH on the GPU was achieved by Harada et al. [241] in 2007. They kept memory transfer between the CPU and GPU, which hinders the overall speed of computation, to a minimum by creating all the data arrays on the GPU and described a neighbour searching algorithm appropriate for the GPU. A 3-D computational grid, called a “bucket”, was placed over the domain and a voxel (a value on the grid) was computed for each particle. Neighbour searching for each particle was then performed within each of their surrounding voxels. The creation of a bucket for neighbour searching distributed the computational load across GPU processors evenly, increasing efficiency. The authors saw consistent speed ups with their new GPU computational model compared to a CPU implementation where, for a 260,000 particle free-surface flow simulation, a speed up of 28 times was observed.

The three sets of aforementioned authors [239–241] required specialist knowledge of computing graphics, using OpenGL (Open Graphics Library) and Cg (C for Graphics) languages to manipulate the GPU. Shortly after Harada et al. [241], in 2007, the application of GPUs for scientific simulations was given more support when the NVidia Corporation released the first version of the CUDA parallel programming language and framework [12], which made GPU programming more accessible to the general public in the form of a general-purpose GPU (GPGPU) framework. As an

extension of the common programming language C++, users could program and execute parallel functions on supported Nvidia GPU hardware with relative ease. Since then, the application and development of WCSPH with GPUs has become increasingly common.

Hérault et al. [242] investigated the use of CUDA for implementing SPH on the GPU. The authors looked at which parts of the parallelised SPH algorithm were either memory-bound (requiring large number of data accesses compared to computations), or compute-bound (requiring large number of computations with relatively little memory access). The force computation function was found to be compute-bound following inspection of the near-linear speed up of the GPU to CPU. The particle update was realised to be memory-bound with a high memory access to operation ratio of 5 for 4. The creation of the neighbour list was a memory-bound operation as expected because it is primarily a function of ordering data structures. The authors also stated that the speed of memory access of multiple data by the same processor could be improved by re-ordering the memory addresses of such data close together. The work reported a 15.1, 207, and 23.8 times speed up of the neighbour list, force calculation, and Euler step particle update procedures respectively, by an Nvidia GTX 280 GPU, of 240 processors, compared to the CPU code. Although, the CPU code was not optimised and the speed ups would have been halved if done so.

The efficient implementation of WCSPH on the GPU (with CUDA) was then further understood with the work of Crespo et al. [138]. They recognised that an efficient CPU algorithm is not necessarily the most efficient one for the GPU. For instance, on the CPU, the force computation procedure can make use of kernel symmetry (the magnitudes of the kernel, or kernel gradient, between two particles are equal and opposite for the reversed interaction) and reduce unnecessary computations. For the GPU, on the other hand, the work of one instruction scheduling thread is assigned to the interactions of one particle, and so kernel symmetry is not used here because threads cannot write to the same memory address at the same time. As in other publications before them [241, 242], the authors sought to keep

data transfer between CPU and GPU to a minimum by storing all the particle data on the GPU. In doing so, the use of shared memory for particle interactions, within simulations of large numbers of particles, could not be done due to the limited size of the memory. Their code also used algorithms provided by CUDA for optimisation including “radixsort”, to reorder particles and their respective data structures, and “parallel reduction”, to compute minimum and maximum array values. It was discovered that the most time consuming part of the SPH algorithm was the force computation, for both CPU and GPU, and therefore, one’s processing power should be concentrated towards that procedure. The primary aim of the authors was to develop an SPH code capable of computing engineering applications. They managed to reduce the computation time of a 1 million particle dam break simulation from 5 days on a CPU, to just a couple of hours on a GPU. However, they recognised that the memory limitations of a single GPU would need to be addressed with a multi-GPU implementation (although maximum available GPU memory has increased by 3-4 times now).

For free-surface flow applications, SPH models implemented on the GPU have enabled detailed simulations including wave propagation and subsequent interaction with structures [10, 127, 243–245], sloshing [8, 246], ship-motion with an anti-roll tank [247], multi-phase applications in violent hydrodynamics [248], nuclear flows [249], and extreme run-off [250]. Many of the applications listed above required GPU-acceleration for millions of particles because of either the domain size, or the resolution needed to capture particular complex flow phenomena (or both). Additionally, some simulations are required to compute long durations of physical time. GPU speed-ups of over two orders of magnitudes compared to serial CPU codes were observed in many of the publications.

The next stage of parallel computing after the GPU, would be HPC with multiple GPUs. A few authors have demonstrated SPH with a multi-GPU implementation [222, 251, 252], with Domínguez et al. [251] achieving a 1 billion particle simulation on 64 interconnected Tesla M2090 GPUs computed in 91.9 hours for 12 seconds of physical time. Multi-GPU clusters are programmed with a combina-

tion of CUDA (or other GPU computing language), and MPI for communication between GPU devices. The challenges of an SPH multi-GPU implementation are similar to that of massively parallel HPC models as described in Section 2.6.1, with the added complexity of programming for the GPU.

It should be noted that field-programmable gate arrays (FPGAs) serve as another potential form of single-machine hardware acceleration. The devices contain programmable logic blocks where the user can configure the behaviour of the circuitry without physically changing the hardware, which allows for tailoring to specific applications for highly-optimised parallel processing. However, whilst FPGAs have been shown to significantly accelerate SPH within the field of astrophysics [253,254], there has been little research about the implementation of the numerical method on such devices. This is unlike advances regarding SPH on the GPU in recent years. Compared to GPUs, FPGAs require long development times and are less portable. This due to the need to develop the application specific logic blocks for a specific FPGA device [254], which is not ideal for the development of a general engineering software in this study.

Based upon the literature surrounding hardware acceleration for SPH, it is clear the GPU is the most favourable option. Exporting a GPU code is now a cost-effective and energy efficient approach to parallel computing for researchers and the engineering industry. The majority of the world's most powerful and energy efficient HPC computing clusters comprise of GPUs [228]. Implementing ISPH on the GPU will give the method the computational acceleration needed to compete with WCSPH's established success with the GPU. Moreover, this research will provide a stepping stone towards establishing a multi-GPU accelerated ISPH code, which would also take advantage of MPI parallelisation (see Section 2.6.1).

The majority of literature in this section showcase GPU codes written with CUDA. There exists another popular parallel programming framework, OpenCL [13, 237], which is the open industry standard for executing programs across heterogeneous platforms consisting of both CPUs and GPUs. However, it is not chosen as the programming language here for a few reasons. Whilst OpenCL is desirable in the

sense it is portable across all platforms, its generality naturally results in a performance drop compared to CUDA-based applications that exclusively target NVidia GPUs [255]. Although OpenCL can give competitive, if not the same, performance, extra coding adjustments would be required [256]. CUDA has a larger amount of support and development within the community whereby numerous libraries and algorithms now exist for the language. For these reasons, the parallel programming language CUDA is used in this study to execute ISPH on the GPU. It must be said, however, there is no reason for most of the work conducted in this study to not apply to an OpenCL framework written code.

It should also be noted that, due to ISPH's close relation with the MPS method [94, 95], advances in the MPS method on GPUs should also be paid attention to during development of ISPH on the GPU. The MPS method was first implemented on the GPU for 2-D simulations in 2011 by Hori et al. [257] and Zhu et al. [258], where the latter achieved a 26 times speed up over a traditional CPU-based MPS code. MPS on a GPU has since been applied to multi-phase flows [259] and 3-D free-surface flows [260]. Developments in MPS also include GPU-based neighbour list algorithms for MPS by Murotani et al. [261] and the comparison of openMP/MPI parallelised solvers for MPS [262]. Such advances in the method are particularly relevant because a PPE is also solved in MPS, unlike WCSPH.

2.6.3 Open-source software for the GPU

The efficient implementation of an SPH code is a large and time consuming task. The magnitude of the process is even more apparent for ISPH because, unlike WCSPH, it requires multiple particle-particle computation loops for the pressure projection step, including the population of the PPE matrix, and the presence of a linear solver for the solution of the PPE matrix. There exists a range of open-source software for the GPU which can be utilised to reduce the amount of work required to achieve an efficient implementation of ISPH on the GPU.

A few open-source WCSPH solvers, namely GPUSPH [242,263], AQUA_gpusph [223, 264], and DualSPHysics [15, 265], are of recognition within the literature. All

have shown rigorous validation and an extensive record of publications, for example: [10, 126, 247, 266–268]. Of the three, this study makes use of DualSPHysics, for several reasons:

- DualSPHysics, unlike the other two, contain both CPU and GPU implementations of WCSPH, which is useful for ease of implementation and debugging purposes.
- It is highly optimised, and written in CUDA to maximise performance with NVidia GPUs (GPUSPH is also written in CUDA, whereas AQUAgpusph uses a combination of OpenCL and python).
- DualSPHysics is a package with dedicated pre- and post-processing software, which allows for ease of application to engineering problems.

In addition to SPH software, numerous open-source linear algebra libraries for the GPU have emerged [16, 269, 270]. Such libraries are also highly optimised and contain algorithms for matrix and vector manipulation, and a variety of matrix linear solvers and preconditioners which can be applied to the PPE matrix. Taking advantage of these libraries will allow for quick evaluation of the best algorithms for the solution of the ISPH PPE.

2.7 Challenges of ISPH on the GPU

To the best of the author’s knowledge, the only implementation of ISPH on a GPU is from the Ph.D. thesis of Leroy [190], however no investigation or algorithmic implementation is described.

By implementation of an ISPH algorithm run entirely on the GPU, a novel study is made herein. Such a task is not necessarily unique solely because of the recent advances in technology which make it possible, but also from the several associated challenges requiring attention:

- **A Lagrangian PPE matrix:** For mesh-based methods, the coefficient matrix of the PPE can be created just once at the beginning of a simulation (as-

suming no remeshing is required) and reused [30]. Finite volume and difference methods spend the majority of their computational power towards solving the PPE, this is no different in ISPH, but there is the added effort of constructing the matrix each time step due to moving computation points [271]. Moreover, herein it is required to do so, in parallel, on the GPUs streaming multiprocessors, which can be difficult to perform efficiently because of the high number of memory accesses associated with large numbers of particles [258].

- **Memory limitations:** The choice to store the PPE matrix provides a convenience of simple compatibility with many highly optimised open-source linear algebra libraries (see Section 2.6.3) for investigating quick solutions of the PPE without additional coding effort. However, it poses a challenge in that the memory requirements are large and the GPU's physical memory is limited and non-expandable. For an ISPH simulation to take place on the GPU with intermediate data transfers only occurring for output, the memory available is confined to a single GPU device. The modern high-end gaming GPUs host about 10-11GB RAM, whereas the high-end scientific computing GPUs have 12-24GB RAM, but with a significant increase in purchase cost. Whilst a multi-GPU implementation of ISPH would resolve the memory limitations, it is a highly complex task. This study of ISPH on a single GPU will provide a stepping stone towards massively parallel HPC simulations with multiple GPUs. Memory footprint can be reduced with alternative sparse matrix storage methods [272] which are also featured in the works of Guo et al. [232] and Li et al. [260].
- **ISPH boundary conditions on the GPU:** Some authors [194, 197] indicated the need for improved boundary conditions in their ISPH models. Although relatively easy to parallelise, the simpler boundary treatments, such as the use of dummy particles or repulsive boundary forces, present numerical boundary layers due to misrepresentation of mathematical conditions. The mirror particle method [11, 157, 161] or multiple boundary tangent method [9, 211] are such boundary conditions that can provide accurate representation of

the desired mathematics. However, efficient generation of fictitious particles each time step and complex boundaries still remain to be a challenge for the methods. Guo et al. [232] redesigned the mirror particle generation procedure for MPI parallelism, which saw partitions create mirror particles from within their own sub-domain independent of each other. However, this cannot be done on a GPU, as there is no such domain partitioning across the SMs, resulting in large computational overhead. So there remains to be a challenge in establishing a boundary condition that is accurate, able to model complex geometries, and suitable for efficient parallelisation on the GPU.

- **Exploiting fast linear solvers on the GPU:** Fast and scalable linear solvers and preconditioning for a Poisson equation is a well-researched area for finite-element methods with fixed meshes [273,274]. For particle methods however, the solution of a linear system presents a different challenge because the particle connectivity is constantly changing. The solution of large Lagrangian PPE matrices has been conducted through massive parallelisation of MPI, from the work of Guo et al. [232] with ISPH, and Duan and Chen [262] using MPS. The algorithms and libraries for CPU-based applications however, are much more sophisticated than those of their relatively young GPU counterparts. Researchers working with MPS on the GPU [257,258,260] only report the percentage of computation time spent solving the PPE matrix, and do not investigate the effects on solution time induced by moving particles in complex flow fields.

2.8 Conclusions

For the simulation of violent hydrodynamic free-surface flows within the context of engineering applications, this study will use the SPH method. As a meshless method, SPH is well-suited for the simulation of violent fluid flow featuring highly nonlinear phenomena and discontinuities. Moreover, the representation of a free surface can be handled with relative ease. Of the specific SPH formulations, ISPH will be

used due to its near noise-free and highly accurate computation of the pressure field, which is essential within the engineering industry for estimating hydrodynamic forces. The ISPH method however, is very computationally expensive due to the solution of the PPE. Most engineering applications would require several millions of particles, meaning the need to solve large sparse matrices, and it is the associated computational expense which is addressed in this study. It has been established that for ISPH to advance towards large-scale simulations, parallelisation of the method is a necessity.

GPUs have proven to be a powerful and cost-efficient alternative to massively parallel CPU clusters, and so herein the novel parallel implementation of ISPH on the GPU will be studied. The research presents some unique challenges, which will be addressed: (i) the construction of a Lagrangian PPE matrix every time step on the GPU's SMs, (ii) addressing the memory limitations of the GPU for the inherently expensive ISPH method, (iii) Establishing a robust and accurate ISPH boundary condition suitable for GPU parallelisation, and (iv) exploiting fast linear solvers on the GPU in the context of a Lagrangian particle method.

The next chapter presents the numerical methodology of SPH and ISPH used for this study, including an ISPH boundary condition suitable for execution on a GPU.

Chapter 3

SPH Methodology

3.1 Introduction

This chapter presents the basic SPH methodology and state-of-the-art ISPH methodology used in this study. An ISPH boundary condition suitable for parallel processing and implementation on the GPU is also detailed here by adapting and extending the WCSPH method of Marrone et al. [14] for ISPH on the GPU.

3.2 SPH fundamentals

3.2.1 Interpolation and the SPH kernel

In SPH, the domain is discretised into a set of computational interpolation points commonly referred to as “particles”. Particles are free to move according to governing physical equations and each one possesses physical quantities for their individual representational material mass in the domain (velocity, density, pressure etc.).

The SPH formulation is based on the integral interpolant in Eq. (3.1) [4, 5], which expresses any quantity, ϕ , of a particle at its respective position vector, \mathbf{r} with Cartesian coordinates (x, y, z) , in the domain, Ω ,

$$\phi(\mathbf{r}) = \int_{\Omega} \phi(\mathbf{r}') \delta(\mathbf{r} - \mathbf{r}') d\Omega, \quad (3.1)$$

where $\delta(\mathbf{r} - \mathbf{r}')$ is the Dirac delta function [275], and $d\Omega$ is a differential volume

element.

The Dirac delta function is defined as an infinitely valued and infinitesimally thin function with a value of zero across the whole real number line except at its location, \mathbf{r} :

$$\delta(\mathbf{r} - \mathbf{r}') = \begin{cases} \rightarrow \infty, & \mathbf{r} \rightarrow \mathbf{r}', \\ 0, & \mathbf{r} \neq \mathbf{r}' \end{cases}, \quad (3.2)$$

where the total area underneath the distribution is equal to unity and physically represents the density of a point mass:

$$\int_{-\infty}^{+\infty} \delta(\mathbf{r} - \mathbf{r}') d\mathbf{r}' = 1 \quad (3.3)$$

However, computation of a particle summation with the Dirac delta function is not possible because it is infinitesimally narrow. Therefore, in SPH, a weighted smoothing kernel, ω , is used to approximate the δ -function and so the continuous interpolation, Eq. (3.1), is approximated by Eq. (3.4), where $\langle \cdot \rangle$ denotes an approximation:

$$\langle \phi(\mathbf{r}) \rangle = \int_{\Omega} \phi(\mathbf{r}') \omega(\mathbf{r} - \mathbf{r}') d\Omega, \quad (3.4)$$

such that the partition of unity property of the δ -function is retained as

$$\int_{\Omega} \omega(\mathbf{r} - \mathbf{r}') d\Omega \equiv 1 \quad (3.5)$$

An SPH kernel with compact support will determine the influence a particle, j , has on another particle, i , by function of:

- The distance between the particles, $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$.
- The characteristic smoothing length, h , a quantity used to represent the kernel support radius and is determined as a multiple of the original particle spacing, dp . A kernel's radius of influence, as illustrated in Fig. 3.1, is typically at least $2h$ to help maintain numerical stability. For computational efficiency though, kernels are desired to be compactly supported [276], i.e. the weighting term vanishes outside the radius of influence.

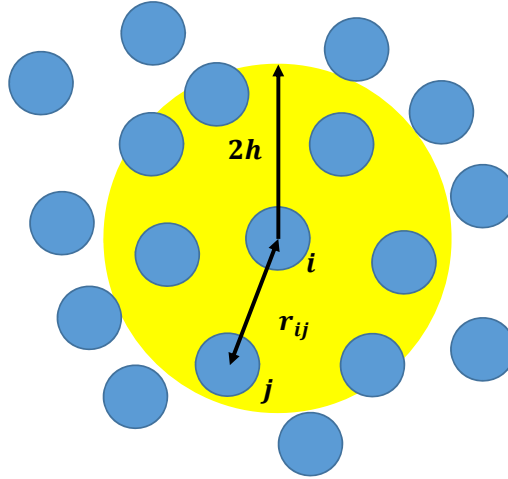


Fig. 3.1: The SPH kernel radius of influence (highlighted in yellow).

To estimate Eq. (3.4) computationally, the integral is replaced by a summation such that the value of any given quantity at an SPH interpolation point (particle), i , is equal to the weighted summation of the quantities at the surrounding interpolation points, j :

$$\phi(\mathbf{r}_i) \equiv \langle \phi(\mathbf{r}_i) \rangle = \sum_j V_j \phi_j \omega(r_{ij}, h), \quad (3.6)$$

where V is the particle volume and

$$\sum_j V_j \omega(r_{ij}, h) \equiv 1 \quad (3.7)$$

For simplicity, the kernel weighting function will hereafter be written as $\omega_{ij} = \omega(r_{ij}, h)$.

In this work, two different kernels common in ISPH literature [9, 11, 184, 190] are used: a fifth-order quintic spline kernel [157],

$$\omega_{ij} = \alpha_D \begin{cases} (3 - q)^5 - 6(2 - q)^5 + 15(1 - q)^5, & 0 \leq q < 1, \\ (3 - q)^5 - 6(2 - q)^5, & 1 \leq q < 2, \\ (3 - q)^5, & 2 \leq q < 3, \\ 0, & q > 3, \end{cases} \quad (3.8)$$

and a fifth-order Wendland kernel [277],

$$\omega_{ij} = \alpha_D \begin{cases} \left(1 - \frac{q}{2}\right)^4 (2q + 1), & 0 \leq q \leq 2, \\ 0, & q > 2, \end{cases} \quad (3.9)$$

where $q = r_{ij}/h$, and α_D is a normalisation factor derived from the condition in Eq. (3.5). The value of α_D for both kernels in 2D and 3D are stated in Table 3.1.

For both kernels, the smoothing length $h = 1.3dp$.

Table 3.1: Normalisation factor (α_D) values

	Quintic spline	Wendland Kernel
2D	$7/(478\pi h^2)$	$7/(4\pi h^2)$
3D	$1/(120\pi h^3)$	$21/(16\pi h^3)$

3.2.2 Kernel gradients

An advantage of SPH is its use of a differentiable kernel to estimate the gradient of a function over an arbitrarily distributed set of points. This means the gradient of any given quantity can simply be expressed as:

$$\langle \nabla \phi_i \rangle = \sum_j V_j \phi_j \nabla_i \omega_{ij}, \quad (3.10)$$

where $\nabla_i \omega_{ij}$ is the gradient of the kernel function between particle i and j with respect to the position of particle i , in a Cartesian coordinate system:

$$\nabla_i \omega_{ij} = \left(\mathbf{i} \frac{\partial}{\partial x_i} + \mathbf{j} \frac{\partial}{\partial y_i} + \mathbf{k} \frac{\partial}{\partial z_i} \right) \omega_{ij}, \quad (3.11)$$

where \mathbf{i} , \mathbf{j} , and \mathbf{k} are unit direction vectors. The gradient of the kernel with respect to the x -, y -, and z -directions, is evaluated with Eqs (3.12), (3.13), and (3.14) respectively.

$$\frac{\partial \omega_{ij}}{\partial x_i} = \frac{\partial \omega_{ij}}{\partial r_{ij}} \frac{x_i - x_j}{r_{ij}} \quad (3.12)$$

$$\frac{\partial \omega_{ij}}{\partial y_i} = \frac{\partial \omega_{ij}}{\partial r_{ij}} \frac{y_i - y_j}{r_{ij}} \quad (3.13)$$

$$\frac{\partial \omega_{ij}}{\partial z_i} = \frac{\partial \omega_{ij}}{\partial r_{ij}} \frac{z_i - z_j}{r_{ij}} \quad (3.14)$$

The form of the gradient operator in Eq. (3.10) however, is generally not used for fluid simulations as higher accuracy can be obtained [68]. By applying Eq. (3.10) to the following identities:

$$\rho \nabla \phi = \nabla (\rho \phi) - \phi \nabla \rho, \quad (3.15)$$

$$\frac{\nabla \phi}{\rho} = \nabla \left(\frac{\phi}{\rho} \right) - \frac{\phi}{\rho^2} \nabla \rho, \quad (3.16)$$

one can obtain the gradient operators [68]:

$$\langle \nabla \phi_i \rangle = \sum_j V_j (\phi_j - \phi_i) \nabla_i \omega_{ij}, \quad (3.17)$$

$$\langle \nabla \phi_i \rangle = \sum_j m_j \left(\frac{\phi_j}{\rho_j^2} + \frac{\phi_i}{\rho_i^2} \right) \nabla_i \omega_{ij}, \quad (3.18)$$

where m is mass, and ρ is density. Divergence operators are obtained similarly.

Both of the two gradient operators are widely used in SPH (WCSPH in particular). Eq. (3.17) is used in WCSPH for the conservation of mass and is able to conserve constant fields, but momentum violation is present with such an operator. Eq. (3.18) on the other hand, satisfies Newton's third law reaction principle (the interaction between particles i and j is equal and opposite to the interaction between particles j and i), and so possesses the advantage of conserving linear momentum. Consequently however, Eq. (3.18) does not ensure zero gradients in constant fields as it does not follow the Taylor expansion [155]. The findings of Oger et al. [155] show that interaction reciprocity (as satisfied by Eq. (3.18)) is not as important as interpolation accuracy. Therefore, Eq. (3.17) is chosen for the gradient operator as it gains significant benefits in accuracy when combined with a renormalisation procedure [155], which Eq. (3.18) gains no advantage from.

3.2.2.1 Kernel gradient renormalisation

A renormalisation procedure for the kernel gradient as explained by Oger et al. [155] is used in this work to improve consistency of the kernel interpolation gradients from zeroth-order to first-order. The normalised kernel gradient, $\nabla_i W_{ij}$, is given by:

$$\nabla_i W_{ij} = \mathbf{L}(\mathbf{r}) \nabla_i \omega_{ij}, \quad (3.19)$$

where in 3D for each particle i , $\mathbf{L}(\mathbf{r})$ is the inverse of a 3×3 matrix as in Eq. (3.20). Herein, the inverse is computed using a compact version of the method of minors, cofactors, and adjugate as shown in Appendix A.

$$\mathbf{L}(\mathbf{r}) = \begin{pmatrix} \sum V_j (x_j - x_i) \frac{\partial \omega_{ij}}{\partial x_i} & \sum V_j (x_j - x_i) \frac{\partial \omega_{ij}}{\partial y_i} & \sum V_j (x_j - x_i) \frac{\partial \omega_{ij}}{\partial z_i} \\ \sum V_j (y_j - y_i) \frac{\partial \omega_{ij}}{\partial x_i} & \sum V_j (y_j - y_i) \frac{\partial \omega_{ij}}{\partial y_i} & \sum V_j (y_j - y_i) \frac{\partial \omega_{ij}}{\partial z_i} \\ \sum V_j (z_j - z_i) \frac{\partial \omega_{ij}}{\partial x_i} & \sum V_j (z_j - z_i) \frac{\partial \omega_{ij}}{\partial y_i} & \sum V_j (z_j - z_i) \frac{\partial \omega_{ij}}{\partial z_i} \end{pmatrix}^{-1} \quad (3.20)$$

The operator is used when calculating the divergence of velocity (Eq. (3.31)) and similarly for when computing the pressure gradient (Eq. (3.33)).

Oger et al. [155] found that they needed to use a mixture of the pressure gradient operators to maintain stability when a free surface is present in their WCSPH scheme. Instabilities were prevented by using the renormalized kernel gradient with Eq. (3.18) for particles near the free surface and the renormalized kernel gradient with Eq. (3.17) for everywhere else. Despite these original findings regarding renormalisation, only the antisymmetric pressure gradient (Eq. (3.17) is used here, even in the presence of a free surface. Throughout this study, no instabilities related to the renormalisation method were encountered. Although unconfirmed, the instabilities might be prevented here due to the use of the projection step scheme.

3.2.3 Discretisation error

The error from approximating Eq. (3.1) with Eq. (3.6) is on the order of $O(h^2)$ [68]. For instance, taking the first two terms (and the error) from a Taylor series expansion

of a general variable function $\phi(\mathbf{r}')$ around the point \mathbf{r} with differentiable function $\phi(\mathbf{r})$:

$$\phi(\mathbf{r}') = \phi(\mathbf{r}) + \nabla\phi(\mathbf{r}) \cdot (\mathbf{r}' - \mathbf{r}) + O(|\mathbf{r}' - \mathbf{r}|^2), \quad (3.21)$$

and applying Eq (3.6), the expansion expressed in SPH terms is:

$$\langle\phi(\mathbf{r})\rangle = \phi(\mathbf{r}) \int_{\Omega} \omega(\mathbf{r}-\mathbf{r}')d\Omega + \nabla\phi(\mathbf{r}) \cdot \int_{\Omega} (\mathbf{r}' - \mathbf{r}) \omega(\mathbf{r}-\mathbf{r}')d\Omega + O(|\mathbf{r}' - \mathbf{r}|^2) \quad (3.22)$$

Three steps are then applied to the terms on the RHS of Eq. (3.22):

1. Substituting the condition of Eq. (3.5) into the first term reduces it to $\phi(\mathbf{r})$.
2. Since the smoothing kernel is to be an even function about the position, \mathbf{r} , the second integral evaluates as an odd function [278] and vanishes,

$$\int_{\Omega} (\mathbf{r}' - \mathbf{r}) \omega(\mathbf{r} - \mathbf{r}')d\Omega = 0 \quad (3.23)$$

3. An assumption is made in that the order of $|\mathbf{r}' - \mathbf{r}|$ is similar to that of the kernel smoothing length.

Thus, second-order accuracy in space is demonstrated:

$$\langle\phi(\mathbf{r})\rangle = \phi(\mathbf{r}) + O(h^2) \quad (3.24)$$

However, the proof is only applicable to regions of full (compact) kernel support [156]. Errors are also likely to increase with particle disorder due to the Lagrangian nature of SPH. This has been partially addressed with the particle-shifting methodology as explained in Section 3.3.5.

3.3 ISPH methodology

For incompressible flows, the Lagrangian Navier-Stokes equations express conservation of mass and momentum according to:

$$\nabla \cdot \mathbf{u} = 0, \quad (3.25)$$

$$\frac{d\mathbf{u}}{dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{u} + \mathbf{f}, \quad (3.26)$$

where \mathbf{u} is velocity, t is time, ρ is density, P is pressure, ν is kinematic viscosity, and \mathbf{f} is the acceleration due to any external forces in the system such as gravity. To solve these equations using an ISPH algorithm, a projection method [173] and a pressure Poisson Equation (PPE) are used to enforce incompressibility.

3.3.1 A projection-based ISPH algorithm with shifting

In ISPH, there are a few ways to construct the PPE:

- **ISPH with a divergence-free velocity field:** Cummins and Rudman [172] presented a projection based method which enforces incompressibility and constructs the PPE through a divergence-free velocity field on the RHS of the PPE. True to a Lagrangian scheme, particles move along streamlines in this form of ISPH which, whilst this may be physically accurate, it can cause irregular particle spacings and consequently, numerical instabilities.
- **ISPH with a density invariance:** Shao and Lo [177] introduced an ISPH method with an imposed density invariance. The algorithm uses a relative density difference between initial and temporal fluid densities of each particle on the RHS of the PPE. The method mitigates the particle clustering seen in the divergence-free velocity ISPH. However, a divergence-free velocity field is not maintained and the formulation does not truly represent an incompressible fluid.

- **ISPH with velocity and density control:** Following the other two forms of ISPH, Hu and Adams [181] decided to combine them to achieve the best of both methods. They solve a PPE with density invariance to advect particle positions, and then a second PPE with a divergence-velocity field for updating particle velocities. The advantages of both the former two formulations were demonstrated with the solution of the two different PPEs. However, solving a single PPE is a time consuming and memory intensive process. The requirement to solve two PPEs for every time step is likely to be too impractical for large-scale 3-D engineering applications compared to the other methods.

For this study, the ISPH with a divergence-free velocity field method is implemented due to its representation of a truly incompressible fluid. To prevent the occurrence of particle clustering and subsequent numerical instabilities, particle shifting is employed following Xu et al. [161]. Therefore, their projection-based ISPH algorithm with shifting is used. The scheme follows from Chorin’s original projection method [173] and Cummin and Rudman’s [172] use of it in SPH. In the following description the superscript refers to the time step, and the subscript refers to a particle. For a particle, i , at any particular time step, n , the pressure projection step goes as follows:

1. Initially, a particle position, \mathbf{r}_i^n , is moved to an “intermediate” position, \mathbf{r}_i^* , using \mathbf{u}_i^n :

$$\mathbf{r}_i^* = \mathbf{r}_i^n + \Delta t \mathbf{u}_i^n \quad (3.27)$$

2. The intermediate positions are subsequently used to calculate an intermediate velocity from the viscous term of the momentum equation only:

$$\mathbf{u}_i^* = \mathbf{u}_i^n + (\nu \nabla^2 \mathbf{u}_i^n) \Delta t, \quad (3.28)$$

such that,

$$(\nu \nabla^2 \mathbf{u}_i^n)_i = \sum_j V_j \frac{(\nu_i + \nu_j) \mathbf{r}_{ij}^* \cdot \nabla_i \omega_{ij}}{(r_{ij}^*)^2 + \eta_s^2} \mathbf{u}_{ij}^n, \quad (3.29)$$

as explained further in Section 3.3.3.

3. Free-surface particles are identified and their pressure values are set to zero in the next step. Further details of this are given in Section 3.3.4.
4. The next step is the solution of a pressure Poisson equation (PPE) [173], which gives the pressure, p , for the following time step, $n + 1$:

$$\nabla \cdot \left(\frac{1}{\rho} \nabla P^{n+1} \right)_i = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}_i^*, \quad (3.30)$$

where the Laplacian operator used on the LHS of the equation is given in Section 3.3.3, and the divergence of velocity, $\nabla \cdot \mathbf{u}_i^*$, is calculated as:

$$\nabla \cdot \mathbf{u}_i^* = \sum_j V_j (\mathbf{u}_j^* - \mathbf{u}_i^*) \cdot \nabla_i W_{ij}, \quad (3.31)$$

where $\nabla_i W_{ij}$ refers to the renormalised kernel gradient, as in Eq. (3.19), and is also used later in Eq. (3.33).

A derivation of Eq. (3.30) can be found in Appendix B. The PPE is solved as a linear matrix system in the form of $[\mathbf{A}] \mathbf{x} = \mathbf{b}$, and is described further in Sections 3.3.2 and 4.6. The numerical solution of the matrix is one the key challenges of implementing an efficient ISPH algorithm both generally and on the GPU. Free-surface particles provide Dirichlet conditions to the system, and must be applied here to provide a unique solution, this is detailed later in Section 3.3.4. At solid boundaries, Neumann conditions are applied as in Section 3.4.3.

5. The momentum equation (Eq. (3.26)) is completed by using the resulting pressure values from Eq. (3.30) for the pressure gradient and including external forces to give a new velocity at time step $n + 1$:

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^* - \Delta t \left(\frac{\nabla P_i^{n+1}}{\rho} - \mathbf{f}_i^n \right), \quad (3.32)$$

where the pressure gradient term, ∇P_i^{n+1} , is calculated as:

$$\nabla P_i^{n+1} = \sum_j V_j (P_j^{n+1} - P_i^{n+1}) \nabla_i W_{ij} \quad (3.33)$$

Herein, the external force, \mathbf{f}_i^n , is always gravity, which is a conservative vector field and hence consistent with the Helmholtz-Hodge decomposition when included in the projection step [172] in Eq. (3.32). Furthermore, there is a numerical advantage with this approach in that with the kernel gradient normalisation (Eq. (3.19)), any linear pressure field is balanced by gravity to within machine precision.

6. The new particle positions are calculated:

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \Delta t \left(\frac{\mathbf{u}_i^{n+1} + \mathbf{u}_i^n}{2} \right) \quad (3.34)$$

This is where Cummin and Rudman's projection step ends. However, to prevent particle clustering and ensure near noise-free pressure fields, particle shifting is implemented. Following Nestor et al. [182], the method of particle shifting in ISPH was first proposed by Xu et al. [161] and later extended for the application of free-surface flows by Lind et al. [11].

7. Shift particles to avoid clustering, and maintain an ordered particle distribution (see Section 3.3.5):

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^{n+1} + \delta \mathbf{r}_{s,i}, \quad (3.35)$$

where $\delta \mathbf{r}_{s,i}$ is the particle shifting distance.

3.3.2 The PPE matrix

The pressure field is found implicitly through the solution of the PPE (Eq. (3.30)). To solve the PPE, the equation is expressed as a linear matrix system in the form of $[\mathbf{A}] \mathbf{x} = \mathbf{b}$ where:

- $[A]$ is a sparse coefficient matrix of $N \times N$ dimensions, where N is the number of particles in the system.
- \mathbf{x} is a vector consisting of N elements, representing the system solution, in this case it is the pressures of each particle.
- \mathbf{b} is a vector consisting of N elements, representing the source terms of the system, here it is the divergence of velocity of each particle.

The individual elements of the matrix will be denoted by two numeric subscripts, representing the elements row and column number respectively. The matrix element notation is illustrated in Fig. 3.2 within the context of ISPH for a 5 particle system. Each row will belong to one i particle, and each column represents an interacting j particle. Similarly for vectors \mathbf{x} and \mathbf{b} , the subscript denotes the particle number. For example, for the matrix elements associated with particle 2 interactions are $A_{20}, A_{21}, A_{22}, A_{23}, A_{24}$, and elements $x_{m,2}$ and $b_{m,2}$ for the two vectors. The actual values of the PPE matrix elements are presented in the following sections of the chapter.

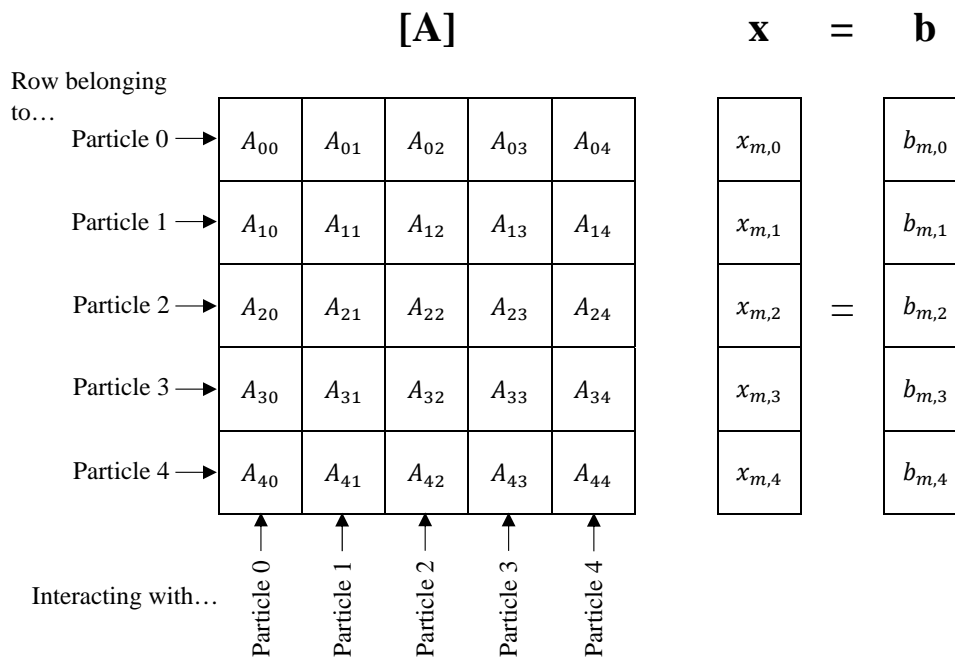


Fig. 3.2: PPE matrix element notation for ISPH. Example for a 5 particle system.

In larger systems, the majority of the elements in the matrix system will be

of null value, due to a compact kernel support limiting the number of interactions per particle, resulting in a sparsity property. Without boundary conditions, the matrix is initially symmetric because of the symmetry of the kernel for pairwise particle interactions. However, the inclusion of boundary conditions creates a non-symmetric matrix, which is explained in Appendix C.

Herein, the matrix system will include all fluid particles and some, if not all boundary particles, as explained later in Section 4.5.2.1 leading to very large sparse matrices at high resolutions. Such large matrices with a potential for highly-irregular particle distributions (especially in violent flows) may present ill-conditioned systems. Sections 4.5.2.1 and 4.6 explain how the PPE matrix system is treated and solved on the GPU.

3.3.3 Laplacian operator

The operator of Morris et al. [157], commonly employed in ISPH [161, 175, 190] is used to approximate the Laplacian during computation of the viscous diffusion term in Eq. (3.28) and the LHS of the PPE in Eq. (3.30). The operator is based upon Brookshaw's [279] approximation of the Laplacian for the viscous diffusion term:

$$(\phi_1 \nabla^2 \phi_2)_i = \sum_j V_j \frac{(\phi_{1,i} + \phi_{1,j}) \mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{r_{ij}^2 + \eta_s^2} \phi_{2,ij}, \quad (3.36)$$

where $\eta_s = 10^{-5}$, a constant to prevent singularities at $r_{ij} = 0$ during computation.

3.3.4 Free-surface identification

Identification of the particles on the free surface is important in ISPH for providing the Dirichlet boundary conditions within the PPE system. Employed in this work is the tracking method of Lee et al. [175], where for each particle, the divergence of position is calculated and the presence of a truncated kernel support is used to identify the free surface. The divergence of a particle's position, $\nabla \cdot \mathbf{r}_i$, is computed

as:

$$\nabla \cdot \mathbf{r}_i = \sum_j V_j \mathbf{r}_{ij} \cdot \nabla_i \omega_{ij} \quad (3.37)$$

With a complete kernel support, Eq. (3.37) is equal to 2.0 in a 2-D domain, and 3.0 in a 3-D domain. Numerical tests indicate that in 2D, a divergence of position value equal to or below 1.6 identifies a particle on the free surface. For 3D, values equal to or below 2.6 are used. The near-free-surface smoothing criterion of Skillen et al. [9] is also used here to smooth the transition of the PPE system from bulk fluid to free-surface regions. Therefore, for the linear system (Eq. (3.30)) in the form of $[\mathbf{A}] \mathbf{x} = \mathbf{b}$ (where \mathbf{x} is a vector of particle pressures, P), the entries are modified as follows:

$$A_{ii}P_i + \sum_j \alpha_{m,i} A_{ij}P_j = \alpha_{m,i} b_i, \quad (3.38)$$

where,

$$\alpha_{m,i} = \begin{cases} 0 & \text{if } \nabla \cdot \mathbf{r}_i \leq \alpha_l, \\ \frac{1}{2} \left[1 - \cos \left(\pi \frac{\nabla \cdot \mathbf{r}_i - \alpha_l}{\alpha_u - \alpha_l} \right) \right], & \text{if } \alpha_l < \nabla \cdot \mathbf{r}_i < \alpha_u, \\ 1 & \text{if } \nabla \cdot \mathbf{r}_i \geq \alpha_u \end{cases} \quad (3.39)$$

α_l and α_u are constants to describe the lower and upper limits of divergence of position for near free-surface regions in the fluid. In 2D, $\alpha_l = 1.6$ and $\alpha_u = 1.8$. In 3D, $\alpha_l = 2.6$ and $\alpha_u = 2.8$.

3.3.5 Particle shifting

Following Nestor et al. [182], Xu et al. [161] introduced particle shifting into ISPH as a method to avoid particle clustering. Particle shifting takes place every time step after the projection step, where particle positions are “shifted” (adjusted) to avoid particle disorder and then the hydrodynamic variables are corrected by Eq (3.40), a Taylor series approximation. In this work, particle velocities are corrected after shifting. For an arbitrary variable, ϕ , the fluid properties are updated according to

$$\phi_{i'} = \phi_i + (\nabla \phi)_i \cdot \delta \mathbf{r}_{ii'} + O(\delta \mathbf{r}_{ii'}^2), \quad (3.40)$$

where i and i' denote the positions of particle i before and after shifting respectively, and $\delta\mathbf{r}_{ii'}$ is the vector between the two positions. The second-order error term in Eq. (3.40) arises from the Taylor series expansion and can be found the same way as the discretisation error described in Section 3.2.3. One can further quantify the error further by seeing that shifting a particle a distance of no more than dp and correcting the hydrodynamic variables through the Taylor series expansion, the error introduced to the velocity field following shifting is second-order in particle-spacing. Without correction of the hydrodynamic variables, it again follows from a Taylor expansion that the velocity error is first-order after shifting. Assuming a fixed mass of fluid, the error in velocity then translates to a first-order error in momentum. Herein, the particle shifting distance is restricted to be an order of magnitude less than the particle spacing, $0.1dp$, to stay well within the aforementioned error bounds.

The method of Particle shifting here violates conservation of momentum in the discrete particle system [161]. However, as established with Eq. (3.17) the method is already non-conservative. Nevertheless, the use of particle shifting has been seen to improve accuracy, stability, and convergence properties in SPH methods [11,161,183,184]. An alternative particle shifting approach was proposed by Adami et al. [280] who shift particles using a transport velocity obtained from a slightly modified momentum equation. There is no correction of hydrodynamic variables undertaken, even though particles are moved with a non-Lagrangian velocity. More recently, Oger et al. [281] addressed this issue and have managed to preserve consistency and conservation within the shifting methodology through an ALE formalism, where transport with a non-Lagrangian velocity is also (correctly) accompanied by an appropriate change in particle mass.

This work uses the improved shifting for free-surface flows algorithm of Lind et al. [11] based on Fick's law of diffusion where particles move from high to low particle concentration regions. A particle's shifting distance, $\delta\mathbf{r}_{s,i}$, is calculated as

$$\delta\mathbf{r}_{s,i} = -D\nabla C_i, \quad (3.41)$$

where, $D = 0.5h^2\Delta t$ a diffusion coefficient, and ∇C_i is the concentration gradient

expressed in Eq. (3.42). Included within the calculation of the concentration gradient is a pairing instability term, F_{ij} [160]:

$$\nabla C_i \approx \sum_j V_j (1 + F_{ij}) \nabla_i \omega_{ij}, \text{ where } F_{ij} = R_1 \left(\frac{\omega_{ij}}{\omega(dp, h)} \right)^{R_2}, \quad (3.42)$$

where dp is the initial particle spacing distance, and R_1 and R_2 are coefficients taken as 4.0 and 0.2 respectively. Pairing instability arises from the gradient of the kernel derivative [282] and negativity in the kernel Fourier transform [283]. This is apparent in spline-based kernels, such as the quintic spline (Eq. (3.8)), when the support radii (h/dp ratio) is sufficiently large [283]. Despite the efforts of shifting to maintain a regular particle distribution, particles still clump together, hence the use of the pairing instability term. The Wendland kernel however, is not affected by such a phenomenon [283].

Using a particle shifting technique based on Fick's law has the consequence that the shifting of particles on, or near, the free-surface would rapidly diffuse away from the fluid bulk. Therefore, Lind et al. [11] also controlled the amount of shifting, normal to the free surface, of particles at, and nearby the free-surface (in 2D) via:

$$\delta \mathbf{r}_{s,i} = -D \left[\frac{\partial C_i}{\partial s} \mathbf{s} + \alpha_{shift} \left(\frac{\partial C_i}{\partial n} - \beta \right) \mathbf{n} \right], \quad (3.43)$$

where \mathbf{s} and \mathbf{n} are tangential and normal vectors respectively, α_{shift} is a constant in the range $[0, 1]$ that controls the amount of normal diffusion at the free surface, and β is an equilibrium term equal to $\frac{\partial C_i}{\partial n}$ for particles near an unperturbed plane free surface. For transient and violent flows, shifting normal to the free surface can be eliminated entirely, $\alpha_{shift} = 0$. Mokos et al. [183] extended Eq. (3.44) to 3-D simulations:

$$\delta \mathbf{r}_{s,i} = -D \left[\frac{\partial C_i}{\partial s} \mathbf{s} + \frac{\partial C_i}{\partial s_b} \mathbf{s}_b + \alpha_{shift} \left(\frac{\partial C_i}{\partial n} - \beta \right) \mathbf{n} \right] \nabla C_i, \quad (3.44)$$

where \mathbf{s}_b is the bi-tangent vector defined as a tangent to two points on a curve.

Whilst not presented in the implementation of this study, the work of Khayyer

et al. [184] is worth noting here as it builds upon the Lind et al. formulation presented. The tuning of the sensitive parameters in Eq. (3.44) are empirical so Khayyer et al. [184] devised a new shifting formulation, optimised particle shifting (OPS), which computed an accurate and consistent normal concentration gradient of the free surface, and eliminated the use of parameters. OPS proved successful for a number of analytical test cases displaying reduced errors and a smoother free-surface profile when compared against method of Lind et al. [11] However, numerical experiments have shown the scheme cannot handle violent flows effectively.

3.3.6 Time step constraint

For the simulations in Chapter 5, a fixed time step is used, restricted by the Courant-Friedrichs-Lewys (CFL) condition [148]. The time step, Δt , is determined by the criterion that it is less than or equal to the convection time on the smoothing length, h [284]:

$$\Delta t \leq C_{CFL} \frac{h}{u_{max}}, \quad (3.45)$$

where C_{CFL} is the CFL condition number, a constant in the range [0.1, 0.2] in this study, and u_{max} is the expected maximum velocity of the flow.

3.4 An ISPH boundary condition suitable for GPUs

In SPH, boundaries (wall and inflow/outflow) pose a problem in that there is a truncation of the kernel which, if not treated, will cause inaccurate interpolation and penetration of the boundary by moving fluid particles. Fig. 3.3 shows an example of a fluid particle's truncated kernel at a solid boundary. For ISPH on the GPU, the desirable criteria for an SPH boundary condition are:

- Accurate computation of desirable mathematical boundary conditions: $\mathbf{u} \cdot \mathbf{n} = 0$, and $\partial P / \partial n = \rho \mathbf{f} \cdot \mathbf{n}$.
- Relatively simple implementation.

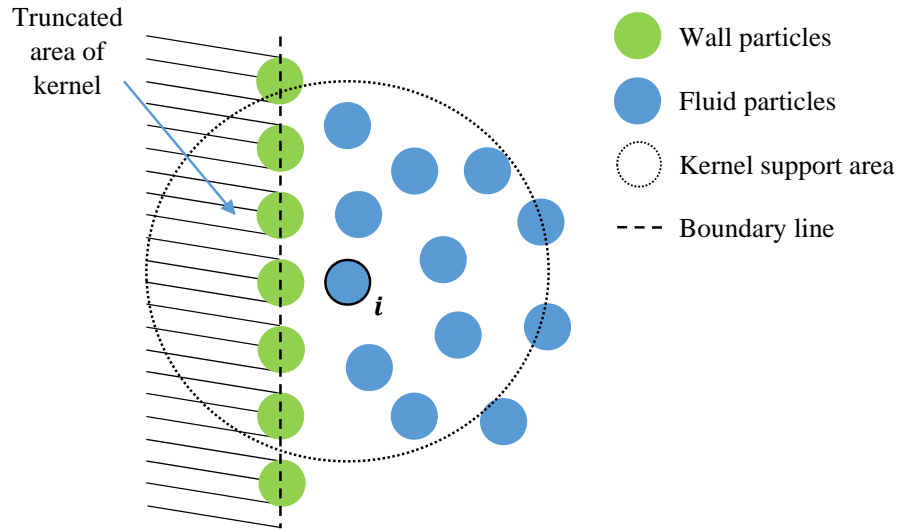


Fig. 3.3: The truncation of the kernel for a particle, i , near a solid boundary.

- Suitable for parallelism.
- Robust with regards to complex geometries.

The definition of a complex geometry here may include solid boundaries of curved shapes, and non-right-angled corners. To account for the kernel truncation at boundaries, the SPH boundary types employed for ISPH are typically mirror particles [157, 170] and fixed dummy particles [175, 177]. Unified semi-analytical boundary conditions [159, 285] are also used, however, although accurate, the method is complex and time consuming to implement and therefore not considered here.

3.4.1 Mirror particles

The mirror particle method sees fluid particles reflected across the boundary to generate “mirror particles” which possess the properties of their original counterparts. Fig 3.4 illustrates a mirror particle generated from a fluid particle at a distance, dr , perpendicular to the boundary. The method of reflection ensures the desired mathematical conditions to be exact on the boundary line when interpolating between a fluid particle and its mirror. However, the generation of mirror particles are required every time step and its most efficient implementation for a parallelised code is not yet known. Furthermore, difficulties arise when representing complex geometries with mirror particles.

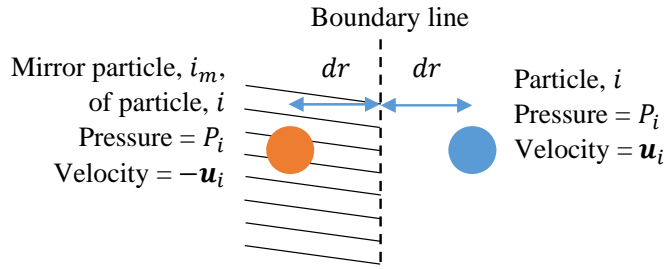


Fig. 3.4: A fluid particle (blue), i , at a distance, dr , perpendicular to the boundary line generates the mirror particle (orange), i_m .

3.4.2 Fixed dummy particles

The method of fixed dummy particles includes extra particle layers to the boundary to fill the kernel as shown in Fig. 3.5. Solid boundaries are represented by "wall particles" which remain fixed on the physical boundary line. "fixed dummy particles" are placed in layers perpendicular to the boundary behind the wall particles. Throughout the simulation dummy particle boundary layers are prescribed the same pressure and velocity as their corresponding normal direction wall particles. The wall particles are treated the same as fluid particles but without advection. Compared

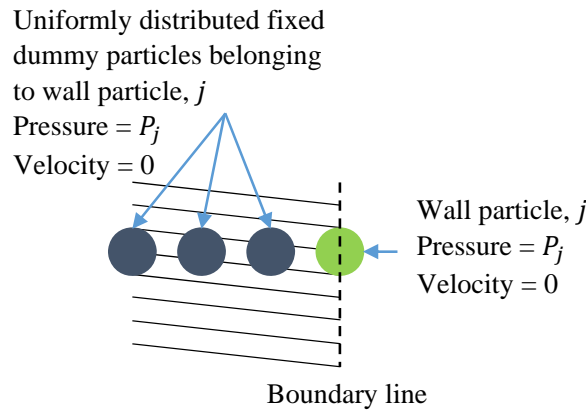


Fig. 3.5: An example of fixed dummy particles for a stationary boundary ($\mathbf{u} = 0$). Fixed dummy particles (grey) are uniformly distributed and possess the same velocity and pressure as their corresponding normal direction wall particle (green).

to mirror particles, fixed dummy particles are simple to implement (generated once at the beginning of a simulation), can be easily used with complex geometries, and straight forward to parallelise. However, satisfaction of the mathematical bound-

ary conditions are inaccurate due to uneven interpolation of properties at the wall caused by the extra dummy particle layers.

3.4.3 Adapting the Marrone et al. boundary condition for ISPH

Both mirror and fixed dummy particles meet only some of the aforementioned desirable boundary condition criteria. The experiences of such boundary conditions through numerical experiments are summarised in Table 3.2. An additional boundary condition, as proposed by Adami et al. [158], is also included as a simple alternative to the fixed dummy particle boundary condition, where a zeroth order kernel summation is used to extrapolate velocity and pressure values for fixed dummy particles. Although the Adami et al. [158] boundary condition has been evaluated as an improvement over the standard fixed dummy particles, it can never fully satisfy the mathematical boundary conditions due to the use of extrapolation.

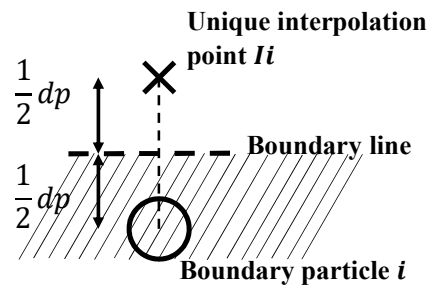
Presented here is a new boundary condition for ISPH that combines the advantages of the different types of boundary conditions. Here, dummy particles are employed to complete kernel summations, whilst boundary conditions are satisfied using the method of Marrone et al. [14] (and similarly Bouscasse [286]), which is modified for ISPH.

Fig. 3.6 illustrates the Marrone et al. [14] boundary condition where each fixed dummy particle is mirrored into the fluid, perpendicular to the boundary to give a unique interpolation point (UIP). For each fixed dummy particle, i , the associated UIP generated is denoted as Ii , and interpolates properties from surrounding fluid particles, j , where the interaction is denoted as $(Ii)j$.

The mirroring process for the generation of UIPs is executed once at the beginning of the simulation during setup and detailed in Section 4.4.3. For moving rigid boundaries, the velocity vector of a boundary particle is used to move its UIP. At each time step, a moving least squares (MLS) interpolator, Eq. (3.46), is used to estimate the velocities and pressures at the location of UIPs. The MLS interpolator

Table 3.2: Experiences of different boundary conditions regarding the satisfaction of desirable SPH boundary condition criteria.

	Mirror particles	Fixed dummy particles	Adami et al. [158] - extrapolated dummy particles
Accurate $\mathbf{u} \cdot \mathbf{n} = 0$ and $\partial P / \partial n = \rho \mathbf{f} \cdot \mathbf{n}$	Yes - straightforward to enforce	No - Numerical boundary layer created	Improved accuracy and convergence compared to fixed dummy particles but less accurate than mirror particles
Simple to implement	Not as simple as fixed dummy particles - extra effort for mirror particle generation, memory management, and complex geometries	Yes - straightforward, treated similarly to fluid particles	Yes - Only one extra summation required for extrapolation of velocity and pressure of dummy particles
Suitable for parallelism	No - efficient parallel generation of mirror particles on GPU unknown	Yes - parallelism of boundary particles similar to fluid particles	Yes - parallelism similar to fixed dummy particles
Robust for complex geometries	No - Issues arise with geometries with singularities (sharp corners)	Yes - placement of particles specified in pre-processing and then treated similarly to fluid particles	Yes - Similar to fixed dummy particles

Fig. 3.6: A fixed dummy particle, i , is mirrored perpendicular to the boundary line to create a unique interpolation point (UIP), (I_i) .

is:

$$\begin{cases} \omega_{(Ii)j}^{MLS} = \mathbf{M}_{Ii}^{-1} \mathbf{e}_1 \cdot \mathbf{c}_{(Ii)j} \omega_{(Ii)j}, \\ \mathbf{c}_{(Ii)j}^T = [1, (x_{Ii} - x_j), (y_{Ii} - y_j), (z_{Ii} - z_j)], \quad \mathbf{e}_1^T = [1, 0, 0, 0], \\ \mathbf{M}_{Ii} = \sum_j \mathbf{c}_{(Ii)j} \otimes \mathbf{c}_{(Ii)j} \omega_{(Ii)j} V_j \end{cases} \quad (3.46)$$

The velocity at the UIP, Ii , and its corresponding dummy particle, i , is then described as:

$$\begin{aligned} \mathbf{u}_{Ii} &= \sum_{j \in \text{fluid}} \mathbf{u}_j \omega_{(Ii)j}^{MLS} V_j, \\ \mathbf{u}_i &= \begin{cases} \mathbf{u}_{Ii} - 2\mathbf{u}_{(Ii)norm}, & \text{Free-Slip,} \\ -\mathbf{u}_{Ii}, & \text{No-Slip,} \end{cases} \end{aligned} \quad (3.47)$$

where $\mathbf{u}_{(Ii)norm}$ is the velocity component in the direction perpendicular to the boundary. For pressure:

$$\begin{aligned} P_{Ii} &= \sum_{j \in \text{fluid}} P_j \omega_{(Ii)j}^{MLS} V_j, \\ P_i &= P_{Ii} + \partial P_i / \partial n, \end{aligned} \quad (3.48)$$

where $\partial P_i / \partial n = \rho \mathbf{f} \cdot \mathbf{r}_{i(Ii)}$.

Applying the Marrone et al. boundary condition to the ISPH PPE fluid particles involves a nested loop. Usually, the PPE is expressed in terms of fluid particles only, where boundary particle pressures are expressed as equivalent fluid particle pressures. This method results in a $npf \times npf$ matrix¹. The LHS of the PPE for a fluid particle, i , is:

$$\sum_j 2V_j \frac{(P_i - P_j) \mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{\rho(r_{ij}^2 + \eta_s^2)} \quad (3.49)$$

However, if the fluid particle interacts with any boundary particles, the summations with the Marrone et al. boundary condition involves extra terms,

$$\begin{aligned} &\sum_{j \in \text{fluid}} 2V_j \frac{(P_i - P_j) \mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{\rho(r_{ij}^2 + \eta_s^2)} + \\ &\sum_{j \in \text{boundary}} 2V_j \frac{\left(P_i - \sum_{k \in \text{fluid}} \left[P_k \omega_{(Ij)k}^{MLS} V_k \right] - \partial P / \partial n \right) \mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{\rho(r_{ij}^2 + \eta_s^2)} = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}_i^* \end{aligned} \quad (3.50)$$

¹ npf = number of fluid particles in simulation

Eq. (3.50) is complex to implement and slow in execution due to the nested particle sweeps. Instead, a more elegant solution involves allocating a row for every particle in the simulation, giving an $np \times np$ sparse matrix². Therefore, the nested loop can be taken out of Eq. (3.50) to become a separate equation for boundary particles and the $\partial P/\partial n$ term is moved to the RHS of the PPE. The LHS of the PPE for fluid particles is then expressed exactly as written in Eq. (3.49).

To summarise, two sets of equations are defined that construct the linear system:

$$\sum_{j \in \text{all}} 2V_j \frac{(P_i - P_j) \mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{\rho(r_{ij}^2 + \eta_s^2)} = \sum_{j \in \text{boundary}} 2V_j \frac{\mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{\rho(r_{ij}^2 + \eta_s^2)} \rho \mathbf{f} \cdot \mathbf{r}_{j(Ij)} + \frac{1}{\Delta t} \nabla \cdot \mathbf{u}_i^* \quad (3.51)$$

$$P_i - \sum_{j \in \text{fluid}} P_j \omega_{(Ii)j}^{MLS} V_j = 0 \quad (3.52)$$

Eq. (3.51) is used for fluid particles and Eq. (3.52) for boundary particles. Both equations can be included in the same matrix because they are of the same form, $[\mathbf{A}] \mathbf{x} = \mathbf{b}$. When expressing Eq. (3.51) as Eq. (3.38), elements A_{ii} and A_{ij} are equal to the coefficients of the pressures on the LHS of the equation; $b_{m,i}$ is equal to the RHS of the equation, and $\alpha_{m,i}$ equals the criterion specified in Eq. (3.39). When expressing Eq. (3.52) as Eq. (3.38), A_{ii} and $\alpha_{m,i} = 1.0$, $A_{ij} = \omega_{(Ii)j}^{MLS} V_j$, $b_{m,i} = 0$. This method allows for simple parallelism across all particles during population of the PPE matrix, which is described in Section 4.5.2.4.

3.4.3.1 Boundary setup

This section describes the mirroring process for obtaining each boundary particle's UIP). Fig. 3.7 details the steps, suitable for parallelisation, to find the positions of each boundary particle's UIP:

1. As illustrated in Fig. 3.7a, particles are placed on the physical boundary line and distinctly defined, these belong to the set Ω_{Phys} . Extra layers of boundary particles are then placed, these belong to the set Ω_{Dum} . This particle arrangement is the same as the WCSPH fixed dummy particle setup.

² np = total number of particles in simulation

2. Each Ω_{Dum} particle is linked to its closest Ω_{Phys} particle and this group of particles is defined as $\Omega_{G,i}$, relating to the Ω_{Phys} particle i . For example, in Fig. 3.7b, particles 5-12 all belong to the set Ω_{Dum} , and their closest Ω_{Phys} particle is particle 19. The group of particles can then be referred to as $\Omega_{G,19}$.
3. The direction vector, $\mathbf{d}_{n,i}$, perpendicular to the boundary for each $\Omega_{G,i}$ group of particles is defined as $\sum_{j \in \Omega_{G,i}} (\mathbf{r}_i - \mathbf{r}_j)$. Subsequently, a mirror point for a group, $\Omega_{G,i}$, can be created at the position, $\mathbf{r}_{m,i} = \mathbf{r}_i + \mathbf{d}_{n,i}$. Fig. 3.7b demonstrates this for $\Omega_{G,19}$, where $\mathbf{d}_{n,19} = \sum_{j=5}^{12} (\mathbf{r}_{19} - \mathbf{r}_j)$. A mirror point is then created by adding the resulting direction vector, $\mathbf{d}_{n,19}$, to particle 19's position.
4. The direction vectors are normalised and then multiplied by a distance of half the original particle spacing, $0.5dp$. Fig. 3.7c illustrates the rescaled direction vector for $\Omega_{G,19}$, and the subsequent repositioning of the mirror point.
5. Every particle in a group $\Omega_{G,i}$ is then mirrored across the group's respective mirror point to find the position of each particle's UIP, $\mathbf{r}_{Ii} = 2.0\mathbf{r}_{m,i} - \mathbf{r}_i$. The resulting UIPs for group $\Omega_{G,19}$ are shown in Fig. 3.7d.

It should be noted that in step 4 of the process, the physical boundary is effectively moved by $0.5dp$. This will however converge to the desired location of the boundary with particle spacing refinement. The mirroring algorithm gives flexibility to the ISPH code because once implemented there is no-hard coding of the physical boundary required for specific test cases.

3.5 Conclusions

This chapter has described the mathematical model for this study. The key fundamentals of the SPH method are explained, and the ISPH algorithm for simulation of the Navier-Stokes equations identified. The model has further developed that of Lind et al. [11] as the method has been proven successful for violent free-surface hydrodynamic applications. This study has a key difference in the method regarding

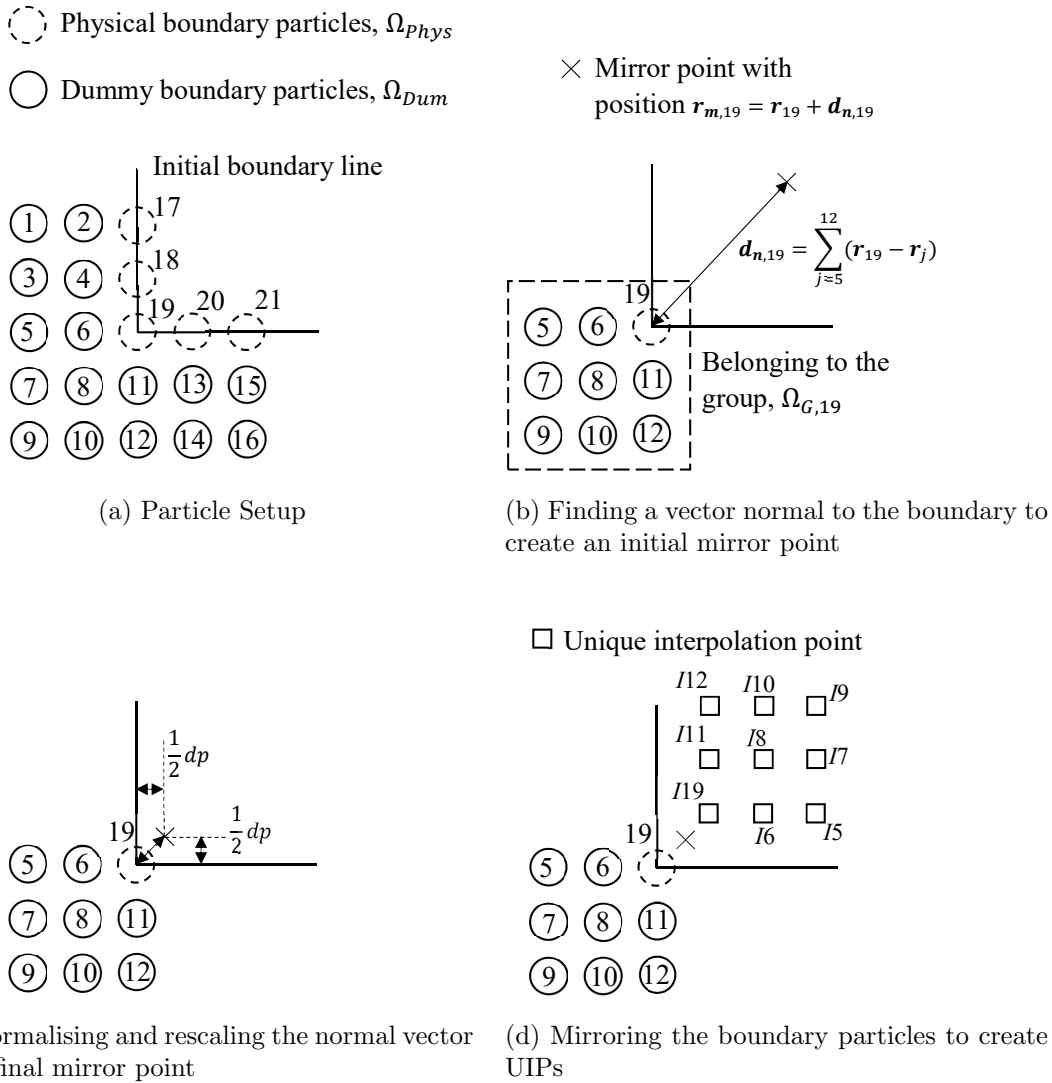


Fig. 3.7: The boundary mirroring process to find the positions of each boundary particle’s UIP.

the boundary conditions. The method of Marrone et al. [14] has been adapted to establish an accurate and robust ISPH boundary condition suitable for parallelism on the GPU. This is the first application of the Marrone et al. [14] boundary condition to ISPH, which is more complex to implement due to the population of the PPE matrix. Introducing the MLS kernel interpolation for pressure (Eq. (3.52)) into the system of equations for boundary particles avoids a nested loop during computation and allows for simple parallelisation in populating the matrix as discussed later in Section 4.5.2.4.

After discussing the graphics processing unit (GPU), the next chapter details the implementation of ISPH on the GPU.

Chapter 4

Implementing ISPH on the GPU

4.1 Introduction

The novel algorithmic implementation for ISPH on the GPU is detailed in this chapter. A new engineering tool is created by conversion of the open-source WCSPH code DualSPHysics v4.0 [15] into an ISPH code implemented for the GPU. Prior to presenting the implementation of ISPH on the GPU, a brief summary of GPU hardware architecture and the programming of such devices is given.

4.2 The graphics processing unit (GPU)

The graphics processing unit (GPU) originated from within the computer graphics industry for real-time visualisation of video game graphics and image processing. The specialist hardware's massively parallel architecture is designed to process large quantities of data where the same computation is required for each data element.

The general-purpose GPU (GPGPU) has since been created to parallelise code that would normally be computed on the CPU in addition to producing computer graphics. Thus, with the introduction of GPU dedicated parallel programming languages, CUDA (Compute Unified Device Architecture) [12, 236] and OpenCL (Open Computing language) [13, 237], GPGPUs have since been used for accelerating a variety of applications including scientific computations. The compute-intensive capabilities of GPU hardware are well-suited for computing large N-body simulations

involving pair-wise calculations [287] and therefore, meshless Lagrangian methods such as SPH.

In addition to computational performance, GPUs are being used as a cost-effective approach to HPC, where a single high-end gaming GPU device (such as the Nvidia GTX 10 series) can be purchased for less than £1000. Such an investment is relatively cheap when considering the cost of a small CPU cluster which costs £10,000s as well as associated running costs (e.g. power consumption, cooling, and maintenance). GPUs require little to no specialist maintenance, relatively low-purchase cost, and are generally more energy efficient, with a higher performance-per-watt rating i.e. they use less energy per math instruction [238, 288].

Figure 4.1 shows the basic concept of how a GPU would be used in the sequence of program execution. In the context of simulation computing, the GPU would deal with all, if not the majority, of data processing in the core of the program and the CPU manages the handling of input/output data.

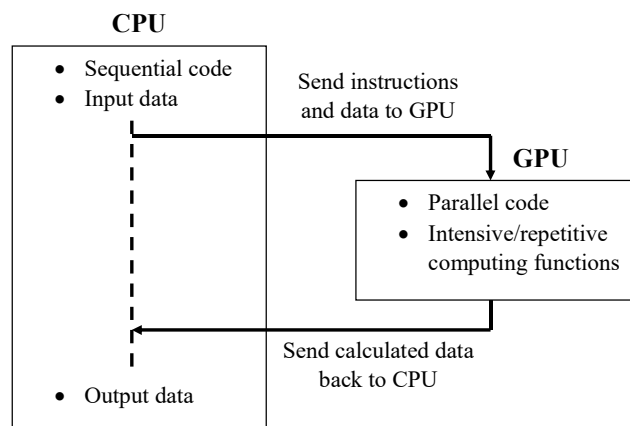


Fig. 4.1: A basic model for the interaction between the CPU and GPU.

4.2.1 GPU architecture

The large-data processing power of GPU hardware is enabled by its unique architecture, which is rather different to that of the common CPU. A parallel CPU architecture has a limited number of multi-threaded cores (typically 4-16 core in a non-specialist CPU), able to efficiently handle a large complex instruction set including calculations, memory fetching, input/output, and branch prediction.

The GPU's core purpose on the other hand, is to process problems of high arithmetic intensity, so the flow control mechanisms and data cache of the hardware need not be as sophisticated as the CPU's. This allows for more space on the chip to be devoted to housing algorithmic logic units (ALUs) for performing arithmetic operations.

Nvidia are one of the largest manufacturers of GPUs and are one of the only companies to produce GPUs specifically for scientific computing. They are the developers of the CUDA programming language, which is exclusive to Nvidia GPUs and detailed further in Section 4.2.4.

The architecture of every Nvidia GPU model is different, however they all have the same general components for parallel processing. The main component is a streaming multiprocessor (SM), which is depicted in Fig. 4.2. showing an example of a modern GPU's SM micro-architecture. An SM houses up to 100s¹ of CUDA computing cores and 1000s of threads. The multiprocessor schedules and executes threads in groups of 32, known as a "warp", and each CUDA core is defined as having an ALU and a single-precision floating-point unit (FPU) to compute one warp at a time.

All the threads in a warp can execute the same set of instructions simultaneously, and similarly, multiple cores within a SM can execute concurrently. As each warp terminates, through completing the assigned set of instructions, a new warp is scheduled and executed by the SM. The architecture respectively uses a mixture of SIMT (single instruction, multiple threads) and SIMD (single instruction, multiple data) parallelism through the CUDA cores and threads to achieve massively parallel computations. Such a method of parallelism is of great benefit to SPH, as one particle's interactions is typically computed through one thread. This means if a GPU has 2500 CUDA cores (a reasonable number for the modern GPU) each executing a thread warp, then theoretically $2500 \times 32 = 80,000$ particles can be computed simultaneously. This is a simplified example however, as the number of threads per core can be limited by the size of data being processed (see Section 4.2.2).

¹Usually in multiples of 32, up to 192 cores in modern architectures.

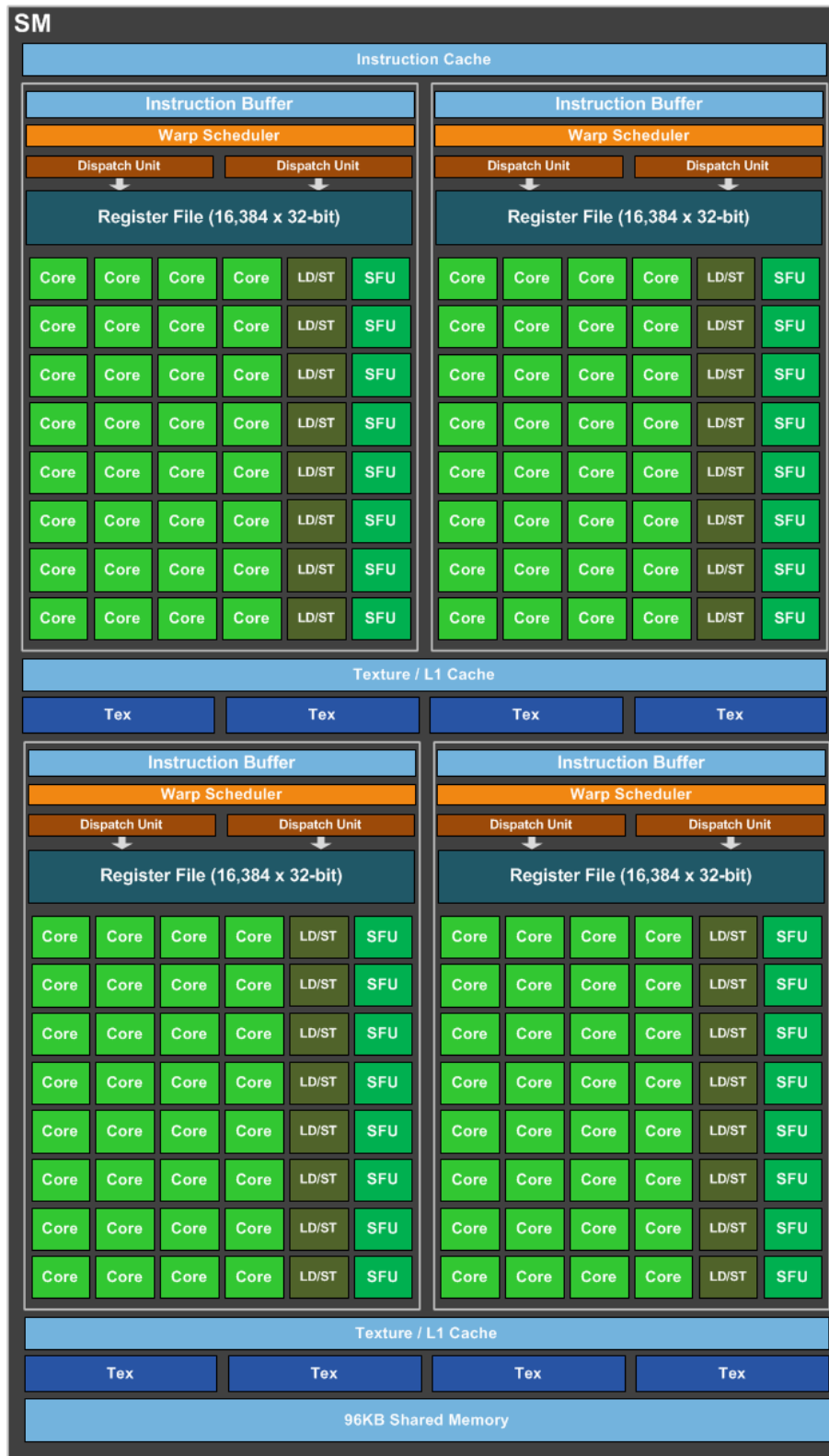


Fig. 4.2: An example of an SM microarchitecture. This particular model is the GP104 SM from the GTX 1080 [289]. “LD/ST” stands for load/store unit. “SFU” stands for special function unit.

In practice, the programmer divides the work across threads, thread blocks, and a grid. Thread blocks are groups of threads, and the grid is made up of thread

blocks of identical size. SMs execute thread blocks with the appropriate number of warps.

Most GPUs are intended for single precision calculations due to their original purpose of graphics visualisation where accuracy is not such a concern. Double-precision calculations are possible, but require more sophisticated algorithms and the theoretical number of available threads at any one time is halved. Some Nvidia architectures are however, specifically designed for use of double-precision calculations, with special double-precision cores built into the SM.

In terms of executing code, parallel functions called CUDA kernels are invoked by the CPU, which in turn prompts the GPU to execute them through the SM(s) on blocks of threads (warps) concurrently. A CUDA kernel in the context of GPU programming, is not to be confused with the SPH kernel, but is a parallel-programming GPU function. Further details of CUDA kernel programming are found in Section 4.2.4).

4.2.2 The GPU memory model

The GPU has a unique memory structure very different to that of the CPU. Fig. 4.3 shows a diagram of the hierarchical memory layout of the GPU. The memory types are:

- **Global memory:** This is the main memory on the GPU, analogous to random-access memory (RAM) on the CPU, and has storage on the order of gigabytes (GB) (Modern GPUs typically range between 4-12 GB, some exceptions give 16 or 20 GB, but at a greater cost). It is this memory space which takes part in GPU-CPU and CPU-GPU data transference. Data within global memory resides there for the lifetime of the program or until the memory addresses are freed up by the user. In the context of ISPH on a single GPU, all the particle data, neighbour list data, the PPE matrix, and linear solver must be able to fit here. All cores on each SM have access to the global memory, and are able to access and overwrite data. Therefore, the synchronisation of cores during parallel computation should be taken into account. Situated the

furthest away from the computing threads, accessing global memory within a kernel function is slower than accessing any of the other memory types.

- **Registers:** In contrast to global memory, GPU registers are the fastest memory type as they are situated immediately adjacent to the computing threads. The access time is largely hidden by active threads so the memory latency is minimal. However, the memory space is very limited in size with only a few hundred kilobytes (KB) per SM distributed exclusively and evenly across each thread within the SM. Register memory only exists as long as the lifetime of the thread execution.
- **L1 cache memory:** Whenever register memory is filled up, variables are “spilled” into the L1 cache. This memory space is located on the SM and accessible by all threads on the SM and typically holds 48-96 KB. The transfer rate between L1 cache and the threads is slower than that of the registers, but still relatively fast compared to using global memory (about an order of magnitude faster).
- **Shared memory:** Each thread block has access to its own private shared memory on an SM, and each thread within the block is able to access it. Data must be copied over to the shared memory where it lasts the lifetime of the block execution. Shared memory resides next to the L1 cache on the SM so has similar memory access latencies. However, because the memory space is user-controlled, unlike the L1 cache, memory access patterns can be optimised for faster execution times. Correct utilisation of shared memory can result in significant speed ups, however one must be careful of memory bank conflicts² which can incur very high latencies between threads.
- **Local (L2 cache) memory:** Local memory is the back-up memory space for when the L1 cache is full. The local memory is accessible by all thread blocks within each SM on the GPU and is therefore much larger (about 1-

²When 2 or more threads within the same warp access the different data elements in the same memory bank

2 megabytes (MB)). However, since the memory is no longer on the SM, the memory latencies are much higher and more similar to those of accessing global memory.

- **Texture memory:** Texture memory in modern architectures reside on the SM. Although it is fast for the storing and transferring of 2-D data arrays for graphics, the manipulation of numerical data can be complex due to their unique formatting based on spatial locality.
- **Constant memory:** As the name suggests, constant memory is used to store variables that will not, and cannot, change throughout the lifespan of the kernel. Constant memory shares the same memory banks as global memory but access is much quicker as it is cached.

To summarise, for the execution of a kernel, variables and arrays are transferred to the different memory spaces for fast parallel computation. Threads will first use their respectively allocated register spaces. When the registers are full (or the data cannot fit onto the memory space e.g. large arrays), the temporary memory data is allocated to L1 cache. Similarly once the L1 cache space is full, local memory is used. If more data is still required after filling local memory, threads access global memory. Moving through the hierarchy, from threads to global memory, memory spaces get larger, but accessing data gets slower due to the physical location from the threads. Fig. 4.4 depicts the number of clock cycles required for threads to access the data within each memory space.

GPU memory works off temporal locality of data for low latency multiple accesses to data. The GPU's high memory-bandwidth also contributes towards efficient parallelism [290]. This is another key difference between the GPU and CPU, as the latter is limited by slow memory data accesses compared to a high processing speed due to a simpler architecture configuration.

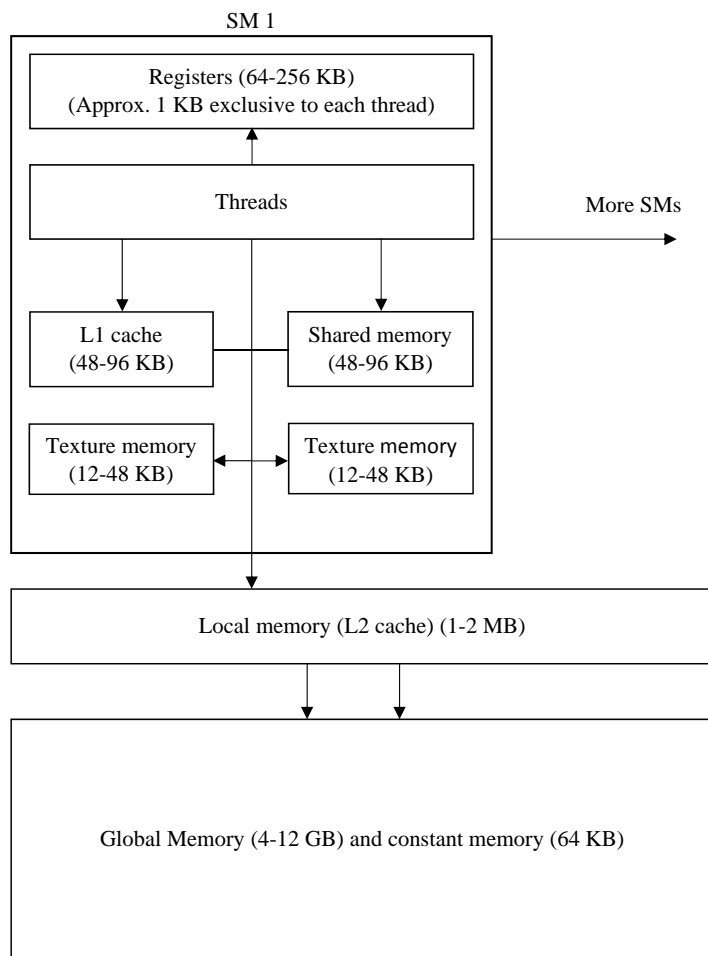


Fig. 4.3: The GPU memory model. Figures are representative of a range of modern Nvidia GPU architectures.

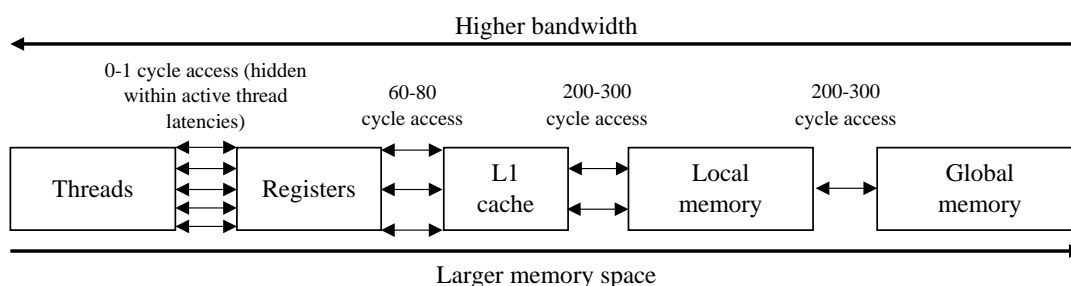


Fig. 4.4: GPU memory access times.

4.2.3 GPU limitations

Despite the unique architecture of the GPU, tailored towards parallelism of compute-intensive applications such as SPH, there are some inherent weaknesses one must consider when programming.

Although the memory structure is tailored for massively parallel computation, a

bottleneck of the GPU for scientific simulations, besides the overall limited memory space, is the small sizes of the faster accessing memory types such as registers. There is a risk of threads requiring access to the local and global memory spaces which are an order of magnitude slower. Therefore, data locality must be managed carefully and high-level optimisations usually require detailed deconstruction of the program/algorithm. Full optimisation of an algorithm though, is specific to each GPU model/architecture.

The DualSPHysics code (described in Section 4.4) can fit all the necessary data to compute a 35 million particle simulation on a 6 GB GPU [15]. However, such a figure is a gross overestimation for the number of particles capable with ISPH on a GPU because the storage of the PPE matrix and linear solver needs to be accounted for. Unfortunately, the maximum global memory cannot be upgraded unlike CPU RAM. One could avoid this problem by transferring blocks of data between the CPU and GPU when needed, but this solution is potentially complex and troublesome to program. Moreover, Hérault et al. [242] and Domínguez et al. [291] have already established that CPU-GPU transfers (and vice versa) at each time step takes up a significant portion of execution time, concluding that data exchange should be kept to a minimum throughout the simulation. The transference of data between the CPU RAM and GPU global memory has such an impact because it is limited by the Peripheral Component Interconnect (PCI) express bus, which is significantly slower than the internal GPU memory. Additionally, for GPU-CPU transfer there is latency in operations where the CPU must wait for all current GPU operations to finish execution, then a signal must be sent back to the CPU to commence data exchange.

The cores of a GPU are less sophisticated so capabilities for data caching and flow control are limited. The use of flow-control statements such as “if” and “switch” promotes “warp divergence” where threads within the same warp follow different execution paths. Ideally, the whole warp would execute the same instructions simultaneously. However, if different levels of branching within a warp occur, then latencies will arise from threads executing varying numbers of instructions, delaying

the block.

The use of transcendental functions, such as square root, can also slow a program down. On the GPU there is a special unit for such operations called the “special function Unit” (SFU) (as seen in Fig. 4.2). There is typically only 1 SFU for every 4-8 cores, and so each thread warp for each of those cores must wait for the SFU to be free before executing their respective computations. Such a case is also similar for double precision computations. The majority of GPUs only have 1 double precision computation for every 24 or 32 cores. Some scientific computing tailored GPUs do have a specific double precision core for every 3 regular CUDA cores though, but at a significant purchase cost increase (approximately 3-6 times more expensive).

4.2.4 The CUDA parallel programming framework

CUDA [12] is a programming language developed by Nvidia. It is an extension of C/C++ and can be used for programming on any CUDA-enabled Nvidia GPU. The CUDA platform provides the user with two mutually exclusive types of possible application programming interface (API) for programming on the GPU, the CUDA driver API, and the CUDA runtime API. The former uses low-level programming allowing for more control over features such as contexts, module loading, and the lifetime of kernels. However, the code is relatively complex including the configuration and launching of kernels, and debugging. The CUDA runtime API, on the other hand, is generally easier to use. Kernel launching, initialisation, and context and module management are all done implicitly. It should be noted that between the two APIs there is no noticeable performance difference as the kernels are the same [236]. The main difference is the increased functionality of the driver API such as certain GPU device queries. With the runtime API, the code is generally easier to manage, and more portable as CUDA kernels can be compiled into executables without the need to distribute CUDA binary files. Therefore, the runtime API is used herein as does the DualSPHysics code detailed in Section 4.4.

There is some terminology within the CUDA programming framework also used herein:

- The CPU and GPU are referred to as the “host” and “device” respectively.
- There are three types of CPU/CUDA kernel functions:
 - A `host` function is executed on the host (CPU) and can be only called by the host.
 - A `_global_` function is a GPU kernel executed across the specified number of threads on the device, and is usually called by the host. In later versions of CUDA (version 5.5 and higher), `_global_` functions can be called by other `_global_` functions for an extra level of parallelism, known as dynamic parallelism. However, such a technique is not explored in this study.
 - A `_device_` function is a GPU thread subroutine executed on the device and can only be invoked by the device.

In the CUDA programming model, the configuration of threads used to execute a `_global_` kernel is shown in Fig. 4.5. As stated in Section 4.2.1, there is a hierarchy whereby a grid is made up of thread blocks, which use a number of threads per block to execute the kernel.

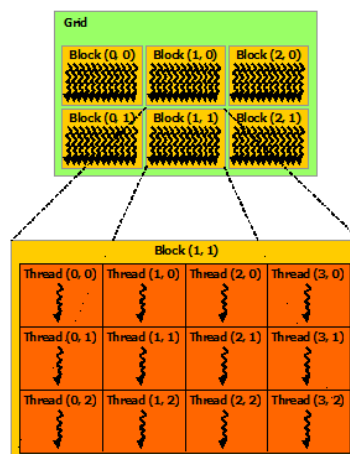


Fig. 4.5: The CUDA programming model [236].

When a `_global_` kernel function is “launched”, or in other words, to be invoked by the host for execution on the device, the number of blocks on the grid and number of threads per block is specified. Each thread is then given a unique thread address for the device to refer to during execution. Within the kernel, each thread address

is converted, with use of CUDA code standard for all applications, such that all the threads specified by the kernel launch grid are enumerated. In SPH, each enumerated thread address is used to refer to particle i . The line of code in Alg. 1 refers to the CUDA code used for the enumeration of a thread address, also known as a thread index number.

Algorithm 1 Acquiring the thread index number and associated particle

```
1:  $i = \text{blockIdx.y} * \text{gridDim.x} * \text{blockDim.x} + \text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x};$ 
```

In early versions of CUDA (less than version 6.0), the memory must be managed explicitly. Therefore, for a device array pointer, device memory must be allocated to it, and then initialised by either a CUDA kernel, or copying data from the equivalent host array. At the end of intended use, the GPU memory must be freed (similar to CPU programming). Allocating memory for large arrays on the device is relatively slow so it is beneficial to allocate all device memory at the beginning of a program. Here, the memory required for particle data and the storage of the PPE matrix is allocated prior to simulation execution (see Sections 4.4.1 and 4.7).

Since CUDA version 6.0, there is the option of unified memory management, which provides a single memory address for data that can be accessed by both the host and device. However, the feature is just a programming simplification and bears no implication of a physical memory bank shared between CPU and GPU hardware. The data is still required to travel between the CPU and GPU, where in this case a deficiency in performance is normally incurred. Therefore, explicit memory management is used here as it is compatible with all CUDA versions and Nvidia GPUs.

The CUDA programming features used in this study are largely governed by the use of the DualSPHysics v4.0 code [15], which is compatible with CUDA versions 4.0 and upwards. DualSPHysics is explained in more detail in Section 4.4.

The rest of the chapter is devoted to explaining the novel methodology of implementing ISPH on the GPU. The next section summarises how the challenges of ISPH on the GPU, mentioned in Section 2.7, are addressed before presenting the details in the subsequent sections.

4.3 Addressing the challenges of ISPH on the GPU

In Section 2.7, several challenges associated with implementing ISPH on the GPU are identified. These difficulties are investigated and addressed within this thesis. The challenges are:

1. Constructing a new Lagrangian PPE matrix every time step on GPU streaming multiprocessors.
2. Implementing the inherently expensive ISPH method onto the limited memory of the GPU.
3. Establishing an accurate and robust ISPH boundary condition suitable for parallelisation on the GPU.
4. Exploiting fast, scalable linear solvers on the GPU for the Lagrangian ISPH PPE matrix.

The foundation for the implementation of ISPH on the GPU comes from open-source software, providing a base to build upon. The WCSPH code DualSPHysics [15] is converted to solve ISPH whilst maintaining the software's efficient data management, particle reordering, and neighbour list algorithms. The DualSPHysics code conversion is detailed in Sections 4.4 and 4.5. In Section 4.6.4, the open-source ViennaCL [16] linear algebra library is utilised for fast solutions of the PPE matrix on the GPU. By using the well-developed and validated open-source software, implementation time is saved. Furthermore, both sets of codes are highly optimised for use on the GPU, where efficient memory management is included.

Section 4.5.2 shows how the parallel construction of the ISPH PPE matrix every time step on the GPU can be achieved by establishing the order of particles in the matrix, pre-calculating the number of interactions for each particle, and the use of separate CUDA kernels for boundary and fluid particles. The matrix is also stored in compressed sparse row (CSR) storage format to reduce memory consumption.

In addition to the use of memory optimised open-source software and the CSR storage format, the GPU memory limitations for ISPH are addressed with a mixed precision storage and calculation model in Section 4.7. A method for estimating the memory required to store the PPE matrix, prior to the simulation, is also presented.

Section 3.4 presented the mathematical formulation of the Marrone et al. [14] boundary condition adapted for ISPH. Throughout Sections 4.4 and 4.5, the implementation of the boundary condition into DualSPHysics is detailed, and the process of parallel computation with it.

The ViennaCL linear algebra library provides a number of different linear solvers and matrix preconditioners implemented on the GPU. Such flexibility in the library is useful for exploring the different options to solve the PPE matrix. Section 4.6.7 exposes the complexity of solving a Lagrangian PPE matrix on the GPU, as it is shown the library's maximum independent set (2) algebraic multigrid (MIS(2)AMG) preconditioner requires modification in order to successfully solve the ISPH PPE. During validation of the code in Section 5.6.3, an investigation of preconditioners shows how they are affected by violent flow characteristics.

The following section describes the open-source WCSPH code DualSPHysics v4.0, and its conversion to solve ISPH on the GPU.

4.4 WCSPH DualSPHysics v4.0 and elements requiring conversion to ISPH

DualSPHysics [15] is an open-source WCSPH code aimed for application to real-life hydrodynamic engineering problems. Development of version 4.0 of the code, used here, is a collaborative effort of researchers from several institutions, namely Universidade de Vigo (Spain), The University of Manchester (UK), EPHYTECH SL (Spain), Science & Technology Facilities Council (UK), Instituto Superior Tecnico, Lisbon (Portugal), Università degli studi di Parma (Italy), and Universiteit Gent - Flanders Hydraulics Research (Belgium). Originally developed from the FORTRAN code SPHysics [292, 293], the software has yielded many successes in its history

where published literature has featured the use of the code either in its release form, or by modification for specific applications [294]. To date, the software has been downloaded at least 35,000 times worldwide.

4.4.1 Software package structure

The code uses either C++/OpenMP for execution on the CPU only, or C++/CUDA for a CPU/GPU hybrid approach. The option allows one to execute CPU-based SPH simulations even when a CUDA-enabled GPU is not available. The codes for the CPU and GPU implementations use similar structure and algorithms, which simplifies the understanding and debugging of the CUDA (GPU) side and allows for straightforward comparison between CPU and GPU results. Both variants of the code share CPU procedures for initial simulation configuration and data output. There is also dedicated pre- and post-processing software in the DualSPHysics package which works regardless of which hardware is used for the SPH computation.

Fig. 4.6 shows a flow diagram of the DualSPHysics package simulation process. In pre-processing, the user defines the initial configuration of the case via an *eXtensible Markup Language* (XML) file. In the file, parameters such as the particle spacing, smoothing length, reference density, and gravity are defined in addition to the initial geometry of fluid and boundary (stationary and/or moving). The DualSPHysics dedicated pre-processing software “GenCase”, then reads the XML file and creates the particles for the specified geometry outputting to a binary file. The resulting binary file can subsequently be read by DualSPHysics to initialise the simulation setup.

Once pre-processing by GenCase is completed, DualSPHysics reads the data and initialises the test case by loading particles and assigning initial data etc. If the CPU solver is specified, then the SPH computation is performed in either serial, or parallel (with OpenMP), on the CPU. However, if a GPU is to be used, then all the initialised case data is transferred to the GPU for a parallel computation governed by CUDA. In both cases, output data is saved occasionally (at a specified

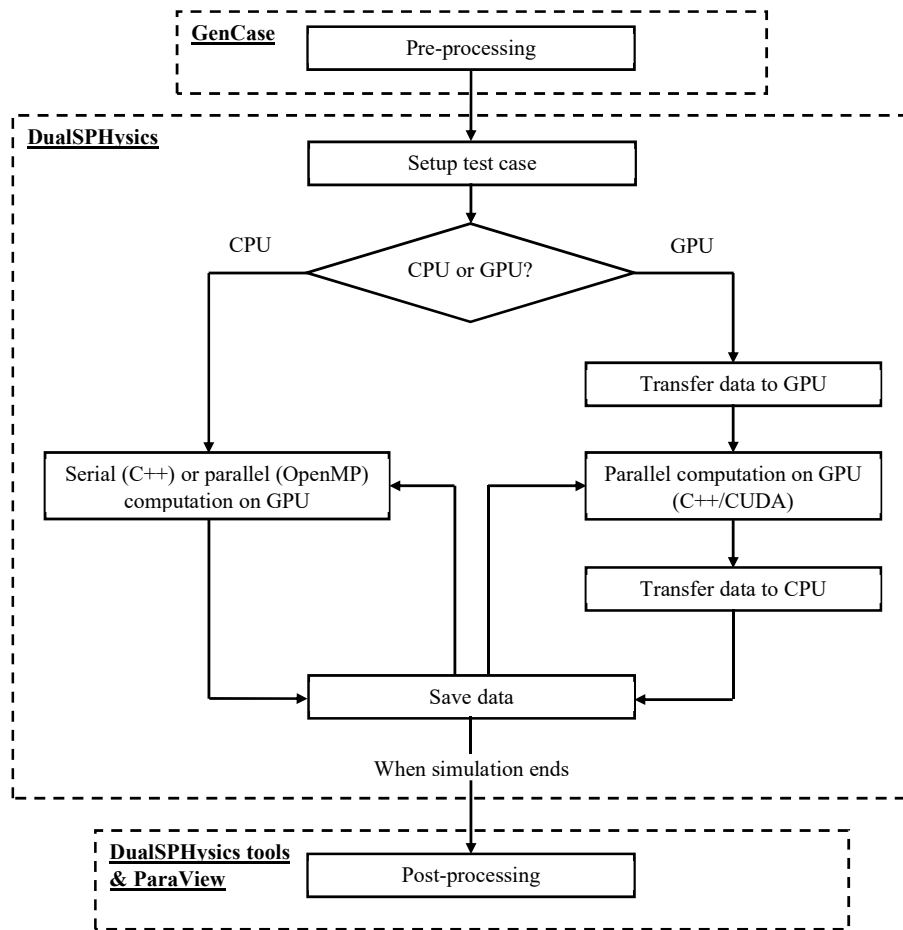


Fig. 4.6: Flow diagram the DualSPHysics software package [15].

simulation time interval) in binary file format for efficiency³. For GPU simulations, the relevant information must be transferred back to the CPU. After the simulation has finished, post-processing can take place.

The software package provides numerous dedicated post-processing tools which converts the binary files, containing particle positions and properties, into visualization toolkit (VTK) files for visualisation of various properties of the data. In this project, once the VTK files have been obtained, the ParaView [295] software is used for data visualisation and post-processing.

In the DualSPHysics program model, the force computation comprises of three parts: (i) neighbour list, (ii) force computation, and (iii) system update. The latter two parts deal with computing the necessary derivatives and updating the hydrodynamic variables respectively. Changing these for ISPH on the GPU is a significant

³Binary files are efficient in terms of, memory storage, fast data access, maintaining data precision, and portability across all computing platforms.

amount of work and is detailed in Section 4.5. Included in the system update is also data output (when required). This does not need modification.

The following three subsections detail important information about the neighbour list, particle sorting, boundary conditions, and time stepping scheme in DualSPHysics required for converting from a WCSPH algorithm to ISPH.

4.4.2 Neighbour list and particle sorting

In SPH, a neighbour list is a method for storing the local neighbours of a particle. In the domain, the total possible number of interactions is $O(np^2)$, where np is the number of particles. However, due to the limiting kernel support radius, many interactions, and thus computations, are unnecessary. A neighbour list reduces a particle's neighbour search radius from the whole domain, to just their respective local area.

The neighbour list used in DualSPHysics is the cell-linked list as described by Domínguez et al. [225] and illustrated in Fig. 4.7. The computational domain is divided into square cells (for 2D, or cubic cells for 3D) and particles are placed into cells based upon their positions in the domain and then reordered into a 1-D array based upon their cell and position within that cell. For example, all particles in cell 0 will be placed in the ordered list before all particles in cell 1 and so on. During force computations, particles now need only search for neighbours in their respective and surrounding cells.

Another common neighbour list in SPH is the Verlet list, which stores the actual neighbours of each particle, eliminating the need to search for particles altogether during the force computation stage. However, studies [225, 296] have shown that despite the more direct approach of computing (non-zero) kernel summations, the performance speed up of the Verlet list compared to the cell-linked list is marginal. On a GPU, the Verlet list only seems beneficial for simulations of less than a million particles [296], which is insufficient for this project. Moreover, the larger memory requirements of the Verlet list limits the maximum number of particles to a few million (for WCSPH) on the GPU. Therefore, the DualSPHysics cell-linked list is

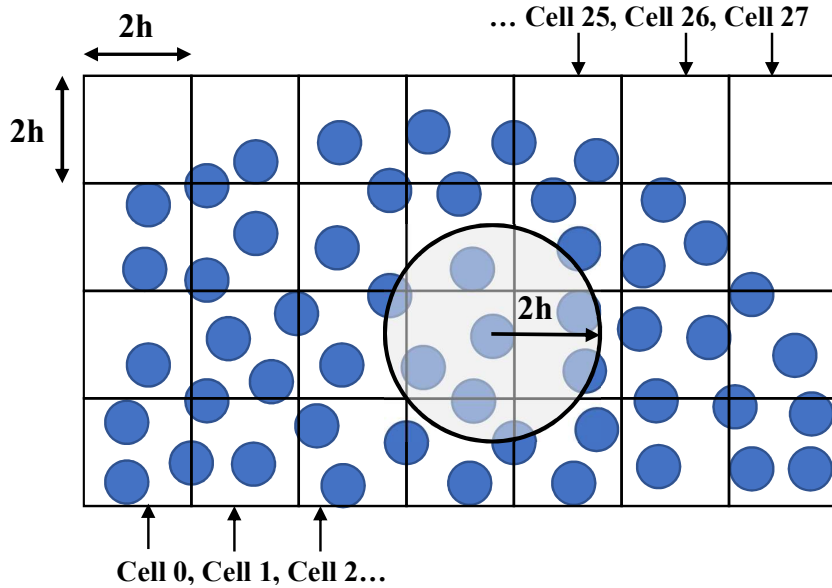


Fig. 4.7: The cell-linked list grid in 2D.

kept intact here with the exception of the cell size for when the quintic spline is used (Eq. (3.8)), which has a support radius of $3h$. Otherwise, it is kept as $2h$ for simulations using the Wendland kernel (Eq. (3.9)).

In addition to the placement of particles in cells, DualSPHysics will also sort the particles according to their positions within the cells to improve GPU memory data locality (see Section 4.2.2). Another constraint to the particle ordering is the particle type as shown in Fig. 4.8.

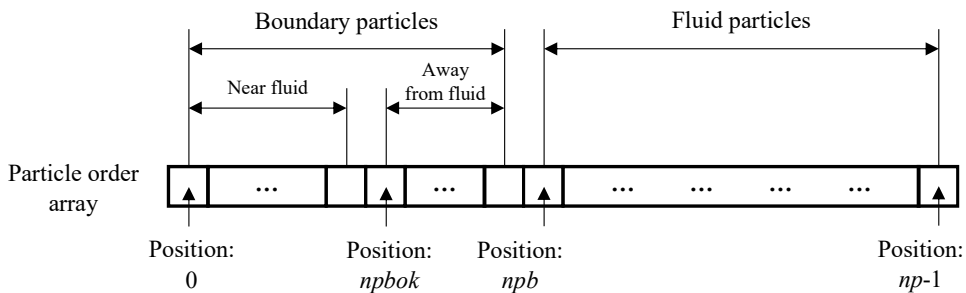


Fig. 4.8: The order of particle groups in DualSPHysics.

A simulation of np total particles, consists of npb boundary particles, and $npf = np - npb$ fluid particles. The boundary particles are placed into positions 0 to $npb - 1$ of the particle order array, and then followed by the fluid particles in positions npb to $np - 1$. Boundary particles are also separated into two categories, those particles

near the fluid, and those outside the local vicinity of the fluid. During the particle sorting at each time step, a halo equal to twice the kernel support radius is built around the fluid domain. Any boundary particles outside of the halo are excluded from computation for the rest of the time step, as they do not influence the fluid. The number of boundary particles that do influence the fluid is denoted by $npbok$, and such particles are placed in the particle order array before those that do not affect the fluid.

The separation of differing particle types saves computational time and also gives rise to separate cell-linked lists for the fluid and boundary particles, which is useful for reducing unnecessary computations, including for the implementation of the Marrone et al. [14] boundary condition (see Sections 3.4 and 4.4.3), which requires for interaction with fluid particles only.

Particle sorting [225] will rearrange all fluid and boundary related particle data arrays (this is part of achieving data locality). Therefore, every particle is also given a unique particle ID number, so specific particles can be distinguished even if their position in the particle order array has changed.

4.4.3 Extending the DualSPHysics boundary condition

DualSPHysics uses fixed dummy particles (also known as dynamic boundary particles [128] as stated in their documentation). However, the Marrone et al. [14] boundary condition, as described in Section 3.4, is implemented for ISPH on the GPU.

For implementation of the Marrone et al. [14] boundary condition in DualSPHysics, the code's fixed dummy particles require unique interpolation points (UIPs). These are obtained with the mirroring process detailed in Section 3.4.3.1.

In addition to the mirror process, new arrays also need to be stored for the boundary condition's UIPs:

- A 3-tuple array for storing their positions.
- An array for storing the cell numbers. The cell number here is obtained from

existing DualSPHysics code used to find the cell number of fluid or boundary particles.

- A 4-tuple array for storing their MLS coefficients.

UIPs are not included within the neighbour list and particle sorting algorithm (Section 4.4.2). Instead, for efficient summation of neighbouring fluid particles, their cell numbers are simply inserted into an existing DualSPHysics function used for fetching the cell-linked list of particles within surrounding cells.

Since the UIPs are not included in the particle order array, their array data is ordered according to boundary particle IDs. Therefore, the position of data for any particular UIP stays fixed throughout the whole simulation, and can be easily accessed according to the corresponding boundary particle ID.

4.4.4 Time stepping scheme in DualSPHysics

DualSPHysics has the option of using a Predictor-Corrector (Symplectic) time stepping scheme for WCSPH, which requires modification for ISPH. To understand the changes required for ISPH, a more detailed version of the GPU branch in Fig. 4.6 is depicted in Fig. 4.9, showing the main steps of the WCSPH DualSPHysics code:

- First, the initial configuration and simulation setup is executed on the CPU. All the necessary memory for the simulation is allocated on the GPU and then the initial data is transferred from the CPU to the GPU.
- The simulation then takes place entirely on the GPU. There are three main procedures: (i) neighbour list creation, (ii) force computation, and (iii) system update. The three procedures are the same for both the Predictor and Corrector stages using the same particle sweep algorithm (highlighted in blue).
- The desired particle data is transferred back to the CPU at specified intervals for output and post processing.

The conversion process requires a significant amount of modification for ISPH on the GPU and is explained in the following section.

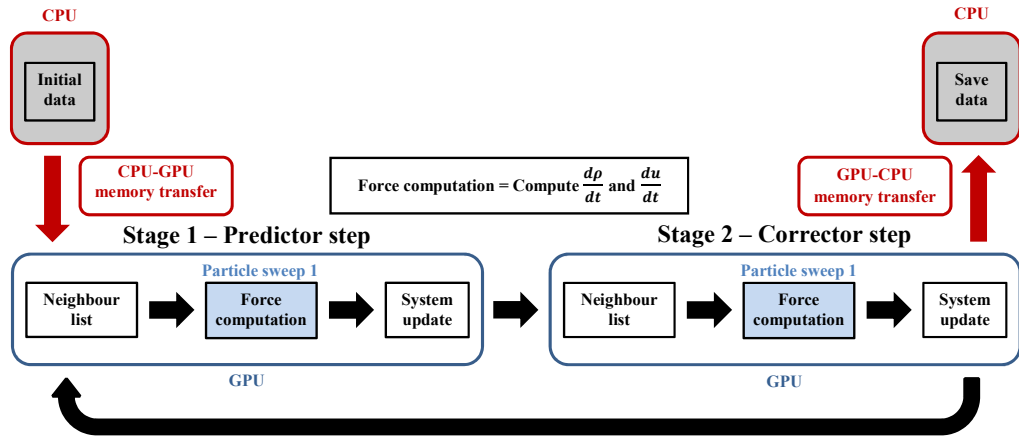


Fig. 4.9: Flow diagram of the key repeating steps in the DualSPHysics Predictor-Corrector time step on a GPU: neighbour list, force computation, system update.

4.5 The pressure projection implementation

For implementation of the pressure projection step (from Section 3.3.1) some key changes to the process depicted in Fig. 4.9 are required. The predictor-corrector elements of the time step need to be modified, the “predictor” step is also renamed to the “intermediate” step for the pressure projection time step. The setup and solution of the PPE matrix needs to be inserted between the intermediate and corrector steps. Particle shifting, which is an important element of the ISPH algorithm to maintain stability, then takes place after the corrector step. These changes are shown in Fig. 4.10. Hence, the “intermediate” step involves functions for the initial advection, viscous force computation, and intermediate velocity \mathbf{u}^* (Eqs (3.27) and (3.28)). The “corrector” step includes the pressure gradient computation and position/velocity time step integration. (Eqs (3.32) and (3.34)).

The four stages are therefore: (i) intermediate step, (ii) setup and solve PPE matrix, (iii) corrector step, and (iv) particle shifting. All four stages require separate particle sweeps that must be distinct from each other.

To include the boundary conditions, after transferring all the initial data from the CPU to the GPU, the fixed dummy particles are mirrored to find their unique interpolation points. This only needs to be performed once, even in the presence of rigid moving boundaries (see Section 3.4).

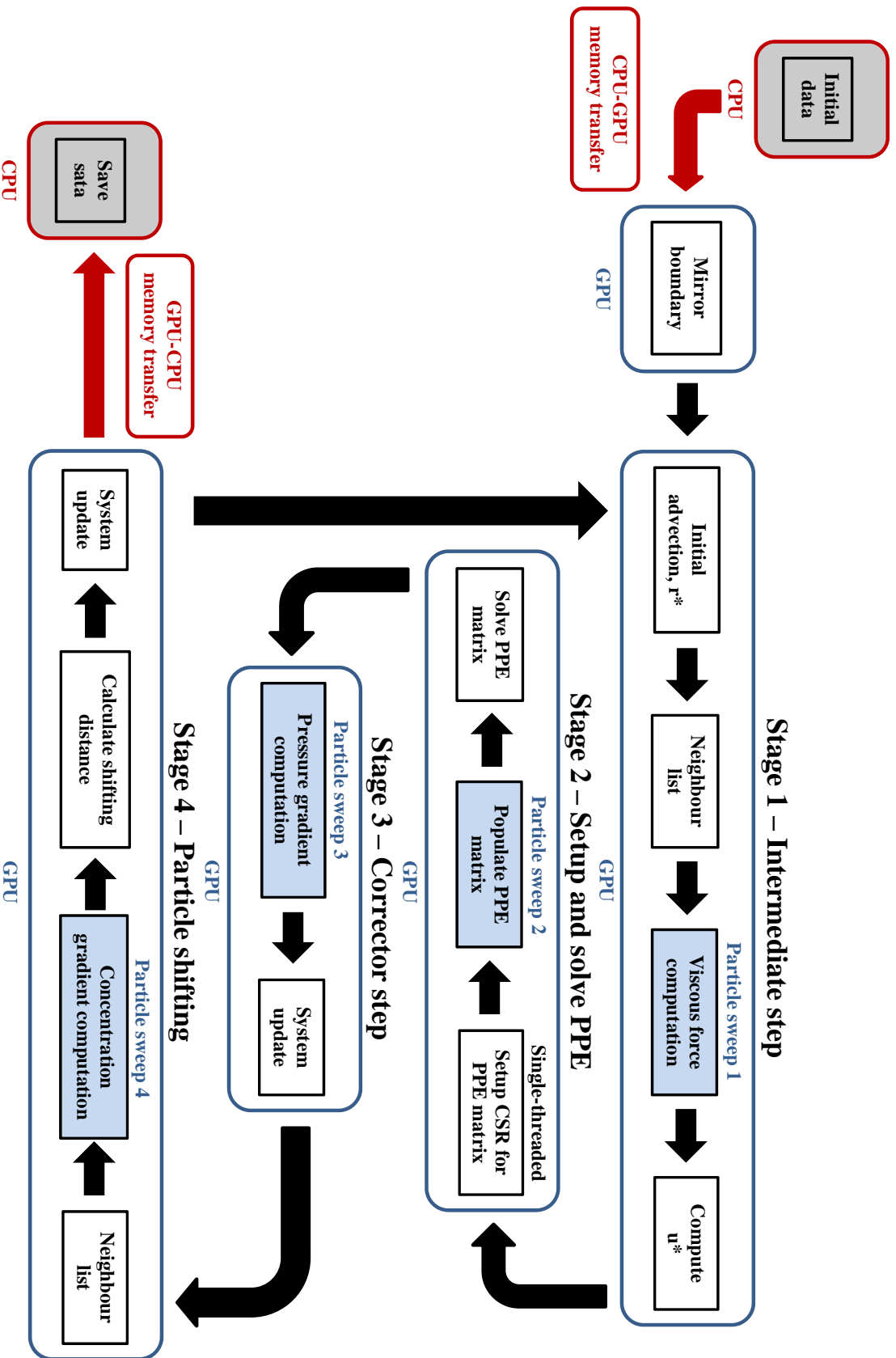


Fig. 4.10: Flow diagram of the ISPH projection step on the GPU implemented into DualSPHysics. Highlighted are the 4 distinct particle sweeps in the simulation. CSR denotes Compressed Sparse Row storage format for matrices.

4.5.1 Stage 1 - Intermediate step

In Particle sweep 1, fluid and boundary particles require different computations. The order and function of the particle sweeps are computed as follows:

1. A sweep for i boundary particles computes the MLS coefficients (Eq. (3.46)) for each of the UIPs.
2. A second sweep for the i boundary particles is then executed which calculates the imposed boundary velocities for slip/no-slip conditions. These are computed using the MLS coefficients found in the previous sweep.
3. A single particle sweep for fluid particles calculate the acceleration due to viscous forces (Eq. (3.28)), the kernel gradient correction variables (Eq. (3.20)), divergence of position (Eq. (3.37)), and the number of interactions per particle used later for the PPE matrix setup in Stage 2 (see Section 4.5.2.3). All these quantities can be computed at this stage using the same SPH kernel calculations to reduce the number of arithmetic operations later in the algorithm.

4.5.2 Stage 2 - Setup and solve PPE matrix

In order to perform Particle sweep 2, which populates the PPE matrix in parallel, it is first required to establish the ordering of particles within the matrix, free-surface particles, and matrix storage array lengths.

4.5.2.1 The order of particles in the matrix

For moving computational points in a Lagrangian system, the PPE matrix generated for each time step is different. This can be significant for violent and transient flows. Due to the changing connectivity between particles, ill-conditioned matrix systems will result from placing particles within the matrix every time step in the same sequence as their particle ID numbers. Ultimately, this leads to slow convergence and longer solution times. Therefore, to reduce such negative effects, the arrangement of

particle ID numbers within the matrix is changed every time step using the following procedure.

For each time step, in Stage 1, a new cell-linked list is generated and the same ordered particle list is used to dictate the order of particles within the matrix (as described in Section 4.4.2). The particle reordering algorithm used for optimisation in the WCSPH DualSPHysics code [15] is re-used here to order the particles for entry into the matrix. It should be noted that boundary particles away from the fluid as determined in Section 4.4.2 are also excluded from the matrix as with the rest of the time step. Thus, the PPE matrix dimensions for each time step are $(npbok + npf) \times (npbok + npf)$.

With a cell dimension equal to the kernel support radius, this arrangement of particles produces a sparse matrix where the majority of entries lie within a diagonal band, an attempt to maintain similar system condition numbers and solution times between time steps. However, this is not always possible depending on the flow evolution where such a case is observed in Section 5.6.3.

4.5.2.2 Free-surface particles

Free-surface particles are identified with the criterion specified in Section 3.3.4 and are used to provide a unique solution to the matrix system by forcing $P = 0$ at the surface. Free-surface particles are identified in Stage 1 (see Section 4.5.1) prior to populating the matrix, to avoid computing the particles' matrix entries, saving computational time.

4.5.2.3 Computing matrix storage array lengths

Due to the sparsity of the PPE matrix, the compressed sparse row (CSR) storage format [272] is used to reduce the memory requirements by storing only the non-zero entries of the matrix.

For example, in a uniformly distributed configuration of particles with an average kernel support of 44 neighbours (using the quintic spline); including the main diagonal of the matrix, a conservative maximum number of values stored is $45npm$,

where npm is the number of particles included in the matrix. This is significantly less than storing the entire matrix including the zero-value entries which would require npm^2 entries. Herein, the number of non-zero entries in the matrix is denoted by Nnz .

The CSR storage format comprises three arrays:

- `aValues` - contains the non-zero entries of the matrix $[\mathbf{A}]$. Array length = Nnz .
- `column` - contains the corresponding column index of each non-zero entry. Array length = Nnz .
- `rowPtr` - contains pointers to the memory location of entries that start a new row in the matrix. Array length = $npm + 1$, where the last entry is used to store Nnz .

Due to the CSR storage format, populating the matrix in parallel cannot be done immediately because the positions of the particles and their neighbouring data in the arrays, `aValues` and `column`, to store each value are unknown. Therefore some preparation is required prior to parallel matrix population on the GPU. In Stage 1 of the main loop, free-surface particles are identified, and the number of interactions per particle are calculated (see Section 4.5.1) and stored in the `rowPtr` array: `rowPtr[i] = Number of interactions for particle i` . The `rowPtr` quantities for the interactions are then utilised within a single-threaded GPU function, which is sequential in nature at the beginning of Stage 2 (Labelled “Setup CSR for PPE matrix” in Fig. 4.10) to find the number of non-zero entries in the matrix and the actual values required for `rowPtr`. Algorithm 2 shows the operation, which loops through in order, and concatenates the `rowPtr` values. An extra entry is added for every row to take account of the main diagonal. For identified free-surface particles, the number of interactions for those particles is ignored. The function will also simultaneously calculate the number of non-zero entries in the matrix and the total number of free-surface particles, which is used for the algebraic multigrid preconditioner during the matrix solution process (see Section 4.6.7.3).

Algorithm 2 Compute_Nnz_and_rowPtr()

```

1: unsigned Nnz = 0;                                ▷ Number of non-zero entries
2: unsigned NFSP = 0;                                ▷ Number of free-surface particles
3: for i particles included in PPE matrix do      ▷ Single-threaded sequential loop
4:   if i is free-surface particle then
5:     rowPtr[i] = 0;
6:     NFSP = NFSP + 1;
7:   end if
8:   unsigned NnzOld = Nnz;
9:   Nnz = Nnz + rowPtr[i] + 1;                       ▷ +1 to include particle i itself
10:  rowPtr[i] = NnzOld;
11: end for
12: rowPtr[npm] = Nnz;                                ▷ npm = number of particles in matrix

```

In addition to the CSR arrays, two arrays `bValues` and `x`, of length npm are used to store the right-hand side of the PPE equation (Eq. (3.30)) and the matrix solution respectively.

4.5.2.4 Populating the PPE matrix in parallel on a GPU

Once the values for `rowPtr` and Nnz are obtained, the matrix is ready to be populated in parallel. Fig. 4.11 shows a basic example of the CSR array values, `column` and `aValues`, for fluid particles (in the bulk fluid and on the free surface) and boundary particles.

- The position `rowPtr[i]`, in the `column` and `aValues` arrays, for particle i stores the non-zero entry, A_{ii} , of the matrix main diagonal. Subsequent positions before entries for particle $i + 1$ are allocated for the neighbours of particle i . For example, Fig. 4.11a shows the highlighted kernel support radius and neighbour list of particle 17. `rowPtr[17]` denotes the position of the matrix main diagonal for the particle's row in the matrix, $A_{17,17}$, then its neighbouring particles (particles 5, 18, 11, 14, 15, 16, 18, 19 and 20) are placed in the positions immediately afterwards.
- For a free-surface particle, such as particle 18, the main matrix diagonal for that particle, $A_{ii} = 1.0$, which is a Dirichlet boundary condition in the linear system.

- Boundary particles follow the same procedure. However, the kernel support radius, neighbour list, and free-surface criterion (see Section 4.5.1) is taken about the boundary particle's unique interpolation point. For example, in Fig. 4.11b, unique interpolation point $I8$ (of boundary particle 8) has a neighbour list of fluid particles only (see Section 3.4), (particles 16, 17, 19, 20, and 22).
- The matrix entries are as explained in Section 3.4 (Eqs 3.51 and 3.52).

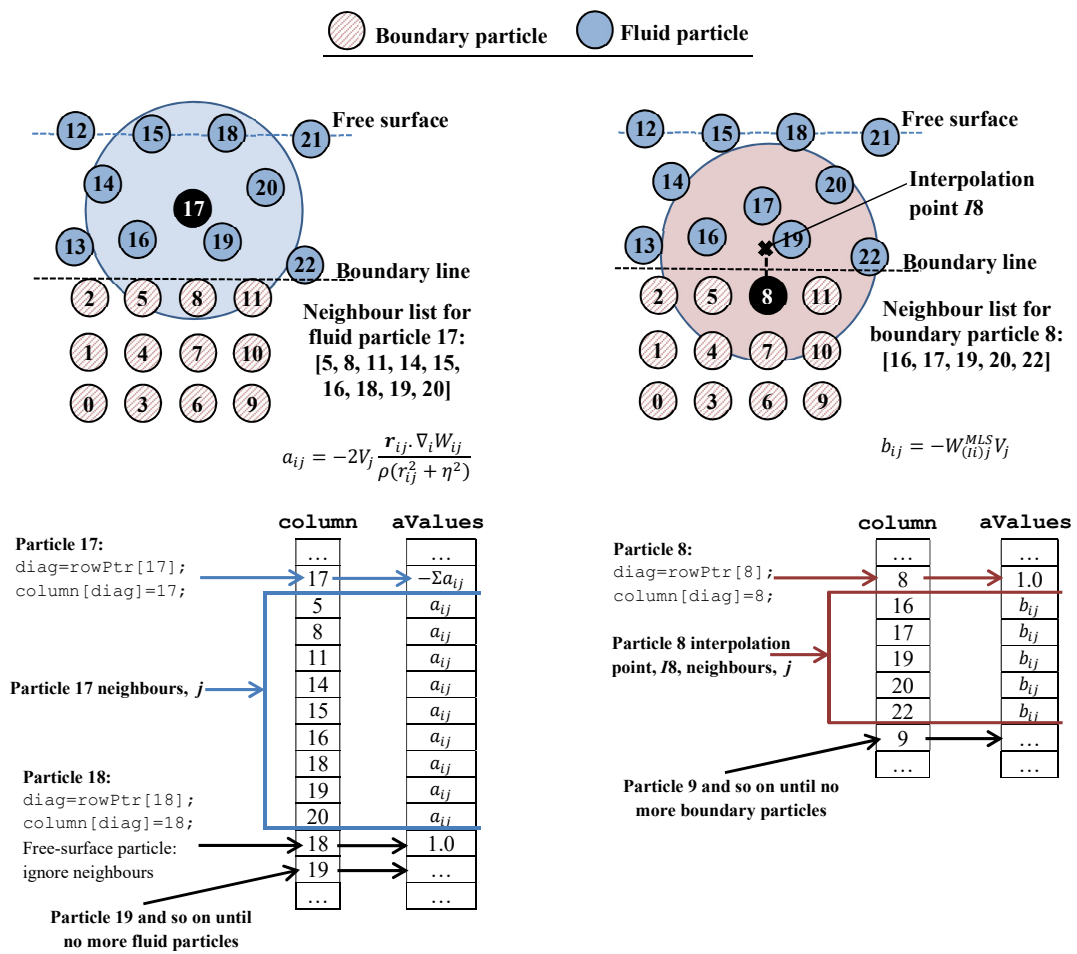


Fig. 4.11: An example of the CSR matrix arrays `column` and `aValues` for a fluid particle, 17, a boundary particle, 8, and a free-surface particle, 18.

The equivalent algorithmic code for populating the CSR matrix arrays on the GPU, as discussed for Fig. 4.11, is shown in Algorithms 3, 4, and 5, which are explained in the following.

The population of the matrix on the GPU is executed with two `_global_` functions, one for fluid particles and the other for boundary particles. Both perform the same basic instructions shown in Algorithm 3, which identifies the location of entries for each particle in the CSR arrays. Lines 5-7 of Algorithm 3 executes particles interactions for i particles not on the free surface. After fetching the neighbour list of a particle, a `_device_` function is called for particle interaction computations dependent on the particle type (lines 5-7, Algorithm 3; Eq. (3.51) for fluid, Eq. (3.52) for boundary). Separate `_global_` functions are called depending on if a particle i is a fluid or boundary particle because of the differing neighbour list data required for each particle type. Fluid particles compute interactions about their own positions (see Eq. (3.51)), whereas boundary particles compute kernel interactions about their respective unique interpolation points and with fluid particles only (see Eq. (3.52)). As a result, the majority of boundary particles will contribute less entries to the matrix than the majority of fluid particles. Thus the execution time of boundary particle matrix population is less than for fluid particles and so the use of two `_global_` functions reduces latency for multiple threads of different types in a parallel scheme (in a GPU this prevents branch divergence and latency between threads within the same warp). The `_device_` functions called for fluid

Algorithm 3 `_global_` PopulateMatrix

```

1: i=GetThreadIndex;                                ▷ CUDA Thread indexing
2: if i in PPE matrix then
3:   unsigned diag=rowPtr[i];                        ▷ “diag” = element (i,i)
4:   column[diag]=i;                                ▷ First entry is always for element (i,i)
5:   if i is NOT on free surface then
6:     GETNEIGHBOURLIST                             ▷ Fetch neighbour list for particle i
7:     PARTICLEINTERACTIONS                          ▷ Populate matrix for particle i
8:   else
9:     aValues[diag]=1.0;                            ▷ For free-Surface particles
10:  end if
11: end if

```

and boundary particles are described in Algorithms 4 and 5 respectively. Here, the interactions of a boundary particle i refer to the interactions of its unique interpolation point with fluid particles. Each function cycles through the neighbour list of particle i in order to populate the matrix. The variable `diag` represents the

position in the `aValues` and `column` arrays for element (i, i) in the matrix, and the variable `index` is used to represent the positions for all other elements (i, j) of the matrix starting at position `diag + 1`. For each neighbour interaction, the associated matrix equations for particle i are computed (see Eqs (3.51) and (3.52):

- For fluid particles (Algorithm 4):
 - The Laplacian operator (LHS of Eq. (3.51)) is computed, this value is added to the main diagonal (A_{ii}), `aValues[diag]`, and subtracted for the non-diagonal elements of the matrix (A_{ij}), `aValues[index]`. The variable `index` is then incremented to consider the next interaction.
 - The divergence of velocity and $\partial P/\partial n$ (if j is a boundary particle) (RHS of Eq. (3.51)) are calculated and added to `bValues[i]`.
- For boundary particles (Algorithm 5):
 - The main diagonal element (A_{ii}), `aValues[diag]`, is set to, and stays as, 1.0.
 - The MLS Kernel (Eq. (3.46)) multiplied by the volume of particle j is calculated and subtracted from the non-diagonal elements of the matrix (A_{ij}), `aValues[index]` (see Eq. (3.52)). The variable `index` is then incremented to consider the next interaction.

Algorithm 4 `_device_ PopulatePPEFluidInteractions`

```

1: unsigned index=diag + 1                                ▷ Represents elements (i,j)
2: for j in neighbour list do                             ▷ Including fluid and boundary particles
3:   if i interacts with j then
4:     Calculate Laplacian between i and j                 ▷ LHS of Eq. (3.51)
5:     aValues[diag] = aValues[diag] + Laplacian;         ▷ Element (i,i)
6:     aValues[index] = -Laplacian;                       ▷ Element (i,j)
7:     column[index] = j
8:     index = index + 1;                                  ▷ Increment for the next neighbour
9:     Calculate DivergenceOfVelocity;                    ▷ RHS of Eq. (3.51)
10:    Calculate dp/dn;                                    ▷ RHS of Eq. (3.51)
11:    bValues[i] = bValues[i] + DivergenceOfVelocity + dp/dn;
12:   end if
13: end for

```

Algorithm 5 `_device_ PopulatePPEBoundInteractions`

```

1: unsigned index=diag + 1                                ▷ Represents elements (i,j)
2: aValues[diag] = 1.0;                                  ▷ Element (i,i) = 1.0
3: for j in neighbour list do                          ▷ Including fluid particles only
4:   if (Ii) interacts with j then
5:     Calculate MLSKernel between Ii and j              ▷ Eq. (3.46)
6:     aValues[index] = -MLSKernel*(Volume of j);        ▷ Element (i,j)
7:     column[index] = j
8:     index = index + 1;                                ▷ Increment for the next neighbour
9:   end if
10: end for

```

4.5.3 Stage 3 - Corrector step

In Stage 3, the third particle sweep of the pressure projection step calculates the acceleration due to the pressure gradient, $\nabla P_i^{n+1}/\rho$, for every fluid particle. This is used to compute the fluid particle velocities for the next time step (Eq. (3.32)). This particle sweep is also used to prevent particle penetration through the boundaries. If a fluid particle is within $1dp$ of any boundary particles, the closest boundary particle is recorded for each fluid particle. Between Eqs. (3.32) and (3.34), any fluid particles identified as being within $1dp$ of the boundary are checked for boundary penetration. If a fluid particle is approaching a boundary, the fluid velocity component normal to the boundary is set equal to the normal velocity component of the boundary. This procedure prevents spray particles penetrating into the boundary as the Marrone et al. [14] boundary condition in the PPE will not generate the necessary pressures to prevent penetration.

4.5.4 Stage 4 - Particle shifting

In Stage 4, all interactions are considered to calculate the concentration gradients of fluid particles in the final particle sweep (Eq (3.42)). The particles are then shifted as described in Section 3.3.5.

This concludes the implementation of the pressure projection step into Dual-SPHysics on the GPU. The solution of the ISPH PPE matrix on the GPU is described in the next section.

4.6 Solving the PPE matrix on the GPU

The solution of the PPE (Eq. (3.30)) is the most computationally expensive, and arguably, most complex part of the ISPH algorithm. It is however, very important in ISPH for enforcing incompressibility and obtaining the method's highly desirable accurate and stable pressure field. The PPE is commonly solved by representing the equation through the linear matrix system:

$$[\mathbf{A}] \mathbf{x} = \mathbf{b}, \quad (4.1)$$

such that $[\mathbf{A}]$ is a large sparse $N \times N$ matrix, and \mathbf{x} and \mathbf{b} are vectors with N elements. Obtaining the values of each element in the matrix $[\mathbf{A}]$ and vector \mathbf{b} are described in detail throughout Sections 3.3.2, 3.3.4, and 3.4.3. The goal of this Section is to evaluate the solution, \mathbf{x} , of the linear system and therefore the pressure field. With the exception of the symbols defined in Eq. (4.1), the mathematical symbols used in this section (4.6) are to be considered separate to the rest of this thesis to allow for consistency of notation with external literature.

The solution of linear systems, including those of large sparse matrices, is widely studied in mathematics [272, 297, 298]. In CFD, engineers seek to solve the system with a linear solver which is efficient and robust, as it can largely influence the use of one's code for varying applications. As with many other applications, the advances in computing over the years has also prompted the use of hardware acceleration techniques for achieving fast linear solvers [288, 299, 300].

Whilst the research on solving sparse matrices is theoretically applicable to both meshed- and meshless-based systems, benchmarks and investigations of solving the latter are in short supply unlike those of the former. This is because solutions of these linear systems in CFD have mainly developed with applications employing an Eulerian approach. In general, the matrix systems generated in meshless methods are more complex than those in meshed methods due to:

- **The number of neighbours per data point and therefore non-zero en-**

tries in the matrix: For a simple comparison, an FDM central differencing scheme has a 5-point stencil (in 2D), and thus the same number of non-zero entries per cell in the matrix. ISPH on the other hand, using the quintic spline gives approximately 45 non-zero entries per particle in the matrix. Expansion into 3D amplifies the difference further with 7 and 250 non-zero entries for FDM and ISPH respectively. A high number of non-zero matrix entries generally means a high number of computations required for solution, and potentially larger matrix condition numbers.

- **The distribution of data points:** Meshless methods, and in particular particle methods such as ISPH, have the potential for the organisation of data points to be highly irregular. This contributes towards the properties of an ill-conditioned matrix, and thus increases the complexity of the system. Moreover, a new matrix is created at each time step due to moving computational points. This means that the best method for solving a particular matrix system at one time step may not necessarily be the best for the solution of the system at the next. Such is not the case for methods with a fixed mesh.

To reduce the complexity of solving such large and/or ill-conditioned matrix systems, preconditioning is usually employed.

4.6.1 Matrix preconditioning

A preconditioner is a transformation matrix, $[\mathbf{M}]$, of the matrix, $[\mathbf{A}]$, such that the linear system is reduced to an equivalent problem but of reduced solution complexity [300]. A linear system can be treated in multiple ways with a preconditioner [301], however here it is applied as:

$$[\mathbf{M}_1]^{-1} [\mathbf{A}] [\mathbf{M}_2]^{-1} \mathbf{y} = [\mathbf{M}_1]^{-1} \mathbf{b}, \text{ where } \mathbf{y} = [\mathbf{M}_2] \mathbf{x}, \quad (4.2)$$

such that $[\mathbf{M}] = [\mathbf{M}_1] [\mathbf{M}_2] \approx [\mathbf{A}]$. The idea is that the new system, Eq. (4.2), possesses the same solution as Eq. (4.1), but can be solved more efficiently because it has a lower condition number. The use of preconditioning techniques are almost

mandatory now due to the size of the linear systems being solved, usually involving no less than several millions of equations. They are widely recognised as the key ingredient to improving performance and reliability of linear system solvers [301].

The choice of available preconditioning techniques is vast due to the infinite number of possible systems arising from scientific computing, so there is much development for creating efficient and robust preconditioners. Benzi [301] provides an extensive review of preconditioning techniques up to the early 2000s, presenting literature results of various methods' performance which show that the speed-ups can vary drastically depending on the problem and preconditioner. Identifying the most suitable matrix preconditioners for ISPH alone would warrant an extensive study. The study of combined implementation on the GPU is recommended for further study in Chapter 7. The preconditioners explored in this thesis are limited to the Jacobi and maximum independent set (2) algebraic multigrid (MIS(2)AMG) preconditioners from the ViennaCL library, which are all discussed in Section 4.6.4. The Jacobi preconditioner is chosen to be part of this study because it is commonly used in ISPH literature [9, 11, 170, 232]. Algebraic multigrid preconditioners have been shown to give fast solution convergence and high scalability for increasing linear system sizes in CFD [302, 303], so their applicability to ISPH on the GPU (with large numbers of particles) is investigated here.

For the actual solution of Eq. (4.1), methods can be broadly categorised into two areas: direct methods and iterative methods.

4.6.2 Direct methods

Direct methods are often based upon the factorization of the coefficient matrix resulting in upper and/or lower triangular matrices. The most well-known methods are perhaps the Gaussian elimination and LU decomposition.

The Gaussian elimination makes use of elementary row operations to manipulate the lower triangle entries of the matrix to equal zero. The solution vector can subsequently be solved one element at a time through the simple sequence of linear equations left in the matrix's upper triangle.

LU decomposition finds lower and upper triangular matrices, $[\mathbf{L}]$ and $[\mathbf{U}]$, such that $[\mathbf{A}] = [\mathbf{L}][\mathbf{U}]$. The system can then be solved through two stages via $[\mathbf{L}]\mathbf{y} = \mathbf{b}$, and subsequently $[\mathbf{U}]\mathbf{x} = \mathbf{y}$. The key advantage of the LU decomposition compared to the Gaussian elimination is if one requires to solve the same matrix multiple times with different source-terms then the upper and lower triangular matrices can be pre-calculated and re-used for the differing \mathbf{b} vectors [304].

Direct methods are generally suited to dense linear systems, but can also benefit some sparse matrices of specific non-zero-element structure [298]. They are popular in industry because of their robustness and reliability. Mathematically speaking, they can solve a linear system exactly using a fixed and finite amount of work. In the context of computing the solution however, precision errors can lead to erroneous results. Moreover, during the process of computing the triangular factors, zero-elements may turn into non-zero elements, also known as “fill-ins”. The problem of fill-ins are two fold, namely the extra computational work required to complete the solution, and the additional memory storage requirements which may not exist in some cases. The development of direct methods for addressing these issues are well researched⁴ [301].

The main challenge for direct methods are the solution of very large linear systems as the problem becomes increasingly expensive to solve. Although the exact answer can be achieved (in the absence of rounding errors), direct solvers’ poor scaling deems the computational time and memory requirements too impractical. Thus for such systems, iterative methods are commonly employed.

4.6.3 Iterative methods and Krylov subspace methods

The idea of an iterative method is to start with an initial approximation of linear system solution, \mathbf{x}_0 , and systematically improve the approximation towards the exact answer by use of solving a simpler system “near” the matrix, i.e. a new system that is an approximation of the original [298].

⁴Pivoting, reordering and bandwidth reducing, and scaling strategies are such techniques improving computation time and addressing some issues of direct solvers

For instance, the initial guess can be obtained by solution of $[\mathbf{K}] \mathbf{x}_0 = \mathbf{b}$, where $[\mathbf{K}]$ is a coefficient matrix closely related to $[\mathbf{A}]$. The Gauss-Seidel method provides such an example where $[\mathbf{K}]$ is a lower triangular matrix with the elements equal to their corresponding values in $[\mathbf{A}]$. The solution of Eq. (4.1) can then be represented as its initial approximation, and the correction for the approximation, \mathbf{x}^c :

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{x}^c. \quad (4.3)$$

Therefore, the linear system can be expressed as:

$$[\mathbf{A}] (\mathbf{x}_0 + \mathbf{x}^c) = \mathbf{b}. \quad (4.4)$$

Rearrangement of Eq. (4.4) yields a new linear system:

$$[\mathbf{A}] \mathbf{x}^c = \mathbf{b} - [\mathbf{A}] \mathbf{x}_0. \quad (4.5)$$

The solution of the new system is also approximated, commonly by the use of $[\mathbf{K}]$ again:

$$[\mathbf{K}] \mathbf{x}_0^c = \mathbf{b} - [\mathbf{A}] \mathbf{x}_0. \quad (4.6)$$

The solution of Eq. (4.6) is then used to find a new approximation, $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{x}_0^c$, that is closer to the exact answer i.e. Eq. (4.3). The process of Equations (4.4) to (4.6) is then repeated until the solution has converged to within some measurement of error [298], for which there are various ways of determining the residual.

It should be noted that the use of a convergence criterion within the linear solver implies a cut-off in the numerical precision of solution which also implies some allowance of compressibility effects. The methodology prevents a truly incompressible flow. However, such criterion are widely used in CFD for incompressible flows [272, 298], and it is presented later in Sections 5.3 and 6.3.1 that a high precision of accuracy and convergence is achieved here for two incompressible flow test cases with analytical solutions.

For large sparse systems, iterative methods are usually favoured over direct meth-

ods. Providing the appropriate iterative solver is employed, convergence to a solution has the potential to take significantly less time whilst using a fraction of the memory required by a direct solver [298, 304]. Iterative solvers also preserve the coefficient matrix in its original form and therefore the sparsity. Consequently, the risk of fill-ins are eliminated and rounding errors, which tend to hinder convergence, minimised [305].

Within the taxonomy of iterative solvers, the so-called Krylov subspace methods have yielded much success for the solutions of large sparse matrix systems [272, 298]. The subset of subspace methods are based upon projection methods⁵ where solution approximations of the Krylov subspaces, \mathcal{K}_m , extracted from the matrix $[\mathbf{A}]$ (as a real number set \mathbb{R}) are found. The Krylov subspaces are defined as the span of repeated multiplications of the initial residual, \mathbf{r}_0 , and system matrix:

$$\mathcal{K}_m([\mathbf{A}], \mathbf{r}_0) = \text{span} \{ \mathbf{r}_0, [\mathbf{A}] \mathbf{r}_0, \dots, [\mathbf{A}]^{m-1} \mathbf{r}_0 \}, \quad (4.7)$$

where $\mathcal{K}_m([\mathbf{A}], \mathbf{r}_0)$ is the m th Krylov subspace.

Variations of methods arise from the choosing of a subspace, \mathcal{L}_m , orthogonal to \mathcal{K}_m , and the choice of preconditioning method (see Section 4.6.1). Further details on the theory and origins of Krylov subspace methods are detailed by Saad [272].

The preservation of the system matrix means that the Krylov subspaces can be constructed by means of matrix-vector multiplications through a single function, and so the system matrix does not require storage. Leroy [190] took advantage of this property for ISPH. However, the matrix elements of each matrix-vector product would also require a particle summation each time, which is computationally inefficient for large linear systems requiring many solver iterations for convergence per time step. This study, on the other hand stores the matrix for computational efficiency and compatibility with open-source linear algebra libraries (see Section 4.6.4).

In theory, Krylov subspace methods are able to converge within a finite number of iterations [301, 306]. In practice however, the computational precision behaviour of the chosen algorithm plays a role in the convergence such that performing a higher

⁵Mathematical definition, not to be confused with Chorin's [173] projection method for CFD.

number of iterations does not necessarily result in increased solution accuracy. This is the reason for varying residual tolerance criteria with different solvers. For very large or highly ill-conditioned (or a combination of both) systems, the convergence behaviour of Krylov subspace methods can be erratic, or may not even find a solution at all. Therefore, preconditioned versions of the solvers are employed, which are generally more reliable and can reduce the solution time even further.

Two common Krylov subspace methods suitable for the solution of non-symmetric matrix systems are used in ISPH literature [170,190,232,284] the generalized minimal residual (GMRES) method, and the bi-conjugate gradient stabilized (Bi-CGSTAB) method.

4.6.3.1 Generalized minimal residual method (GMRES)

Saad and Schultz [307] introduced the generalized minimal residual method (GMRES) iterative solver as a more flexible form of the minimal residual method (MINRES) [308] algorithm able to solve non-symmetric systems. GMRES has the ability to converge to an exact solution with m number of iterations equal to the dimension of the problem matrix, N [307]. However, although there is guaranteed convergence, two problems arise when it comes to the solution of large systems:

- It would be too impractical for the solver to converge after N iterations when N is large. Naturally there is a longer solution time for larger systems, and GMRES is no exception.
- With each iteration, the memory storage requirements increase because there is an orthogonalisation, using Arnoldi's method [309], which results in a new vector every iteration. The additional vector also incurs extra computations per iteration where the number of performed multiplications is proportional to the square of the number of iterations [304].

Preconditioning can be used to reduce the number of required iterations for convergence, potentially alleviating some of the computational expense issues. However, there is still no guarantee that computational resource usage will be kept to a reasonable amount. Therefore, a popular method to limit the computational requirements

of GMRES is to have the solver “restart” after every m iterations. Such a method is known as GMRES(m). When the m th iteration is reached, the solver restarts the Arnoldi process with the latest solution approximation as the new initial solution guess. Saad and Schultz recognised that the performance of GMRES(m) substantially improved with values of m higher than 5 and stated the rate of convergence increased with m . However, the best definition of m is still not clear. Depending on the problem, GMRES(m) may be significantly more expensive, in terms of number of required iterations, than GMRES($m + 1$), or vice versa [298]. In some cases, the solver may not ever converge. There is no indicative “best” value for m , other than that derived from experience. In ISPH, a new system will be created every time step due to moving particles, which could potentially give highly variant systems throughout a simulation. For large particle numbers, the behaviour of GMRES(m) could be too unpredictable in practice.

4.6.3.2 Bi-conjugate gradient stabilized method (Bi-CGSTAB)

The bi-conjugate gradient stabilized method (Bi-CGSTAB) was designed by van der Vorst [310] to find a variant of the conjugate gradients-square method (CG-S) [311] with improved convergence properties. Bi-CGSTAB is just as popular as GMRES for the solution of non-symmetric systems [300].

The algorithm relies on a short iteration step to reduce rounding-errors within iterations. Another advantage is the finite memory requirements being $7N$ vs $(m + 2)N$ of the GMRES(m) algorithm. This means for GMRES(m) to be competitive, in terms of storage requirements, m should be equal to 5 or less, which may not yield an acceptable performance from the solver (see Section 4.6.3.1). The convergence behaviour of Bi-CGSTAB is “smoothed” by use of a polynomial defined at each step with the goal of “stabilising” the residual. This means the achieved solution does not possess a minimum property (residual) in the current Krylov subspace, but nevertheless the convergence can be faster than GMRES [300].

For implementation on a GPU with limited memory storage, Bi-CGSTAB is the favourable choice due to the well-defined memory requirements and its relatively

short and simple algorithm. Algorithm 6 shows the preconditioned Bi-CGSTAB method [310].

Algorithm 6 Preconditioned Bi-CGSTAB algorithm [310]

```

1:  $\rho_0 = \alpha = \omega_0 = 1; v_0 = p_0 = 0;$ 
2:  $\mathbf{r}_0 = \mathbf{b} - [\mathbf{A}] \mathbf{x}_0;$  ▷ Initial residual;  $\mathbf{x}_0$  is an initial guess
3:  $\bar{\mathbf{r}}_0 = \mathbf{r}_0;$ 
4: for  $i = 1$  to max number of iterations do
5:    $\rho_i = (\bar{\mathbf{r}}_0, \mathbf{r}_{i-1}); \beta = (\rho_i / \rho_{i-1}) (\alpha / \omega_{i-1});$ 
6:    $\mathbf{p}_i = \mathbf{r}_{i-1} + \beta (\mathbf{p}_{i-1} - \omega \mathbf{v}_{i-1});$ 
7:   Solve  $\mathbf{y}$  from  $[\mathbf{M}] \mathbf{y} = \mathbf{p}_i;$ 
8:    $\mathbf{v}_i = [\mathbf{A}] \mathbf{y};$ 
9:    $\alpha = \rho_i / (\bar{\mathbf{r}}_0, \mathbf{v}_i);$ 
10:   $\mathbf{s} = \mathbf{r}_{i-1} - \alpha \mathbf{v}_i;$ 
11:  Solve  $\mathbf{z}$  from  $[\mathbf{M}] \mathbf{z} = \mathbf{s};$ 
12:   $\mathbf{t} = [\mathbf{A}] \mathbf{z};$ 
13:   $\omega_i = ([\mathbf{M}_1]^{-1} \mathbf{t}, [\mathbf{M}_1]^{-1} \mathbf{s}) / ([\mathbf{M}_1]^{-1} \mathbf{t}, [\mathbf{M}_1]^{-1} \mathbf{t});$ 
14:   $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha \mathbf{y} + \omega_i \mathbf{z};$ 
15:  if  $\mathbf{x}_i$  is accurate enough then quit;
16:   $\mathbf{r}_i = \mathbf{s} - \omega_i \mathbf{t};$ 
17: end for

```

From Algorithm 6, it can be seen that Bi-CGSTAB is well-suited for parallelisation (this is true for other iterative schemes). The method consists of mainly matrix-vector and vector-vector multiplications, which have well established algorithms for the GPU. The relatively slow computations belong to the inner product additions, as the process is inherently serial in nature, but the CUDA “parallel reduction” kernel can still be used for parallel computation.

4.6.4 Open-source linear algebra libraries and ViennaCL

Due to the size of linear systems increasing with advances in computational processing power over the years, the development of robust and efficient linear solvers, and preconditioners, in the context of high performance computing has become a thriving area of study. There are numerous studies, such as [273, 288, 312, 313], describing algorithms and performance comparisons of the solution methods on a GPU.

In recent years, GPU-implemented open-source libraries for linear solvers and matrix operations have become available such as cuSPARSE [270], PETSc [314],

PARALUTION [315] CUSP [269], and ViennaCL [16]. The CSR matrix storage format employed here is widely used and compatible with the majority of the available libraries. In this study, the ViennaCL library [16] is coupled with DualSPHysics to precondition and solve the PPE matrix on the GPU. Like DualSPHysics, the library is implemented in C++, OpenMP, and CUDA, therefore direct speed up comparisons between CPU and GPU versions of the new ISPH code are simple. There is a high range of functionality with a variety of linear solvers and matrix preconditioners (including the Jacobi and algebraic multigrid) and available for the execution on the CPU or GPU. Benchmark comparisons [316] against Nvidia’s own sparse linear solver library, cuSPARSE [270], show the ViennaCL library performs similarly or better.

The library is simple to implement into DualSPHysics for both Windows and LINUX versions. The library is a series of header files which are easily portable between different computers. Only the library folder file path is required to be specified as an additional include directory during DualSPHysics compilation, and then the necessary “`#include file_name.h`” pre-processor directives inserted into the appropriate DualSPHysics C++/CUDA files.

For this work, the method of solving the PPE linear system is limited to using, from the library, the Bi-CGSTAB linear solver with either the Jacobi preconditioner, or the maximum independent set (2) aggregation algebraic multigrid preconditioner (MIS(2)AMG). In Section 7.2.2, solution time comparisons using the two different preconditioners for a range of problem sizes are presented. The next two sections provide some more detail about these preconditioners.

4.6.5 The Jacobi Preconditioner

Of all preconditioning techniques, the Jacobi preconditioner is one of the simplest. As shown in Fig. 4.12, a coefficient matrix (Fig. 4.12a) will have a corresponding Jacobi preconditioner matrix (Fig. 4.12b) containing all zeros, except the main diagonal, which is equal to that of the coefficient matrix. The Jacobi preconditioner enforces the principle of diagonal dominance in the coefficient matrix. For ISPH,

2	-1		-1			
-1	3	-1		-1		
	-1	3	-1		-1	
-1		-1	4	-1		-1
	-1		-1	3	-1	
		-1		-1	3	-1
			-1		-1	2

2						
	3					
		3				
			4			
				3		
					3	
						2

(a) Coefficient matrix

(b) Jacobi preconditioner matrix

Fig. 4.12: An example coefficient matrix and the corresponding Jacobi preconditioner matrix. Blank elements indicate a value of zero.

the Jacobi preconditioner is advantageous in that its element values can be taken straight from the coefficient matrix without the need for storage. This is possible because the matrix is preserved when using an iterative method (see Section 4.6.3). Furthermore, the application of the preconditioner to the coefficient matrix is simple and fast. Anzt et al. [317] showed that for several different Krylov-type solvers on the GPU, the Jacobi-preconditioned versions could give speed-ups of up to two orders of magnitude. Although, it was found that the robustness of the Bi-CGSTAB solver showed no improvements with its preconditioned counterpart for the several hundred matrix systems tested.

4.6.6 Algebraic multigrid as a Preconditioner

Multigrid methods partially fall within the iterative-type solver classes (but can sometimes be interpreted well within a class of their own), which use a hierarchy of systems to solve the problem matrix. Of the various multigrid methods available, the algebraic multigrid (AMG) [318] is used here.

AMG methods start with the values of the system coefficient matrix and use a chosen smoothing operator to construct and interpolate different “levels” of a grid hierarchy. In the hierarchy, the original matrix level (level 0) is referred to as the finest level and the last calculated level is known as the coarsest level (level

a , where a is the number of coarse levels after the system matrix). These levels are illustrated in Fig. 4.13, where coarser representations of the same system are successively obtained. To achieve the next coarse level in the hierarchy, the most influential system points from the current level are identified and then interpolated onto the coarser, smaller matrix level. Each successive coarser level is a simpler approximate representation of the previous finer level that is theoretically quicker to solve. The idea of AMG methods is for the coarsest (and therefore smallest) matrix level to be relatively easy and quick to solve. The solution is then used as a close approximation to the next system level, which will therefore also be quick to solve. Each system is solved in this way up to the original matrix level. By providing appropriate approximations and interpolations throughout the levels, less iterations are required to reach convergence for the larger matrix levels than if there was no grid hierarchy.

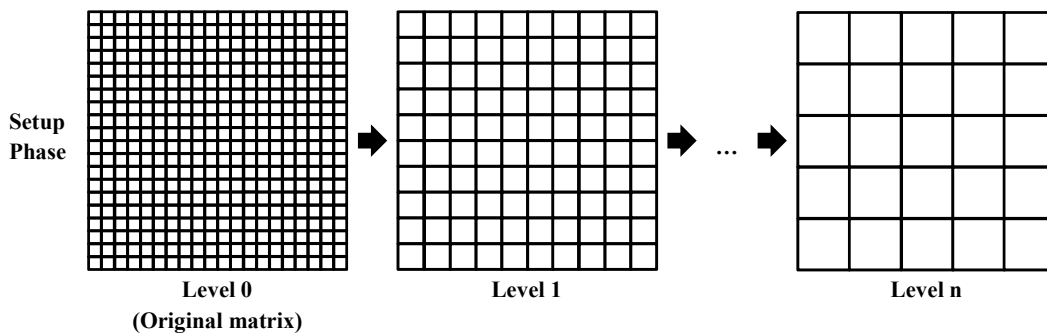


Fig. 4.13: AMG matrix levels

The advantage of AMG methods is the property of mesh independence, where the factor by which the error is reduced with each iteration no longer depends upon the resolution of the problem [319,320]. As a result, convergence with an appropriate multigrid solver often requires relatively few iterations compared to other methods. This is thanks to the grid hierarchy which targets the reduction of both low and high frequency errors (linear system error components in the direction of small and large-value system eigenvectors). Iterative methods and other preconditioners are quick to minimise the high frequency components, but struggle with the low [272].

Axelsson and Vassilevski [321, 322] showed that AMG methods can also be applied as a preconditioner in combination with an iterative solver (such as Bi-

CGSTAB) to provide rapid convergence. AMG preconditioners also often provide a higher degree of robustness for a wider range of problems compared to their solver counterparts [318]. However, it should be noted that there is a downside of multigrid preconditioners, compared to the Jacobi, which are the associated computational expenses in setting up and storing the multiple levels.

Various AMG methods exist defined by the chosen smoothing operator and coarsening strategy of defining which matrix entries (also known as the “coarse aggregates”) belong to each successive coarse level. The AMG method used here is called the maximum independent set (2) AMG.

4.6.7 Modifying the ViennaCL MIS(2)AMG preconditioner for ISPH

The ViennaCL library’s CUDA implementation of the maximum independent set (2) AMG (MIS(2)AMG) [323] preconditioner is used herein. The MIS(2)AMG is an AMG preconditioner specifically designed for GPU execution and the only AMG-type preconditioner available at the present time for the GPU. However, modification to the algorithm within the library has been required in order for the preconditioner to setup correctly for the application of ISPH.

To understand why the modification is required for ISPH, it is useful to first understand the AMG setup phase and the difference between a standard sequential aggregation coarsening strategy and the parallel MIS(2)AMG aggregation strategy.

The aim of the AMG setup phase is to obtain all the AMG matrix levels in the grid hierarchy. Fig. 4.14 illustrates the process of obtaining a successive coarser level (in this case from levels 0 to 1) in the context of ISPH, visualising the domain. The finest level of the hierarchy (level 0) is the original ISPH PPE matrix, containing all particles. Each particle here occupies its own region of the domain, as illustrated in Fig. 4.14a. These regions are also known as “aggregates” and are denoted by C_{num} , where subscript num is the region number. In level 0, the number of aggregates is equal to the number of particles in the system.

The next matrix level in the hierarchy (level 1) is obtained through identifying

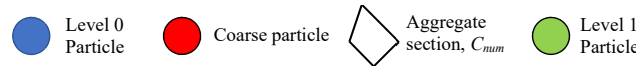
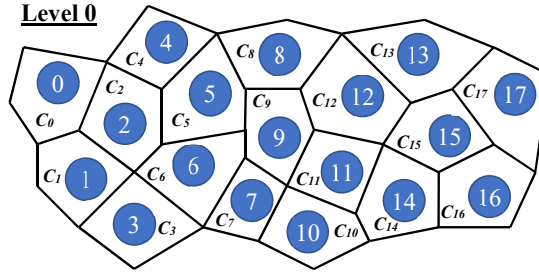
“coarse particles” in the system and forming aggregates around these points only. The algorithmic identification of coarse particles/aggregates are explained later in Sections 4.6.7.1 and 4.6.7.2. These new aggregate regions are called “coarse aggregates” and are used to form the next matrix level, where the number of coarse aggregates is equal to the current number of identified coarse particles. For instance, in Fig. 4.14b particles 2, 9, and 15 have been identified as coarse particles of level 0, and the surrounding non-coarse particles, also referred to as “fine” particles, are associated to one of the three coarse particle neighbours, where coarse particle 2 associates with fine particle neighbours 0, 1, 3, 4, 5, and 6, coarse particle 9 associates with fine particle neighbours 7, 8, 10, 11, and 12, and coarse particle 15 associates with fine particle neighbours 13, 14, 16, and 17.

In preparation for constructing the next matrix level, the next step of the process, as depicted in Fig. 4.14c, redefines the coarse aggregate regions as C_0 , C_1 , and C_2 , one for each coarse point within the current matrix level. Each aggregate now encompasses a different coarse particle and its associated fine particle neighbours. For example, the coarse particles 2, 9, and 15, and their respective associated fine particle neighbours, are assigned to coarse aggregate regions C_0 , C_1 , and C_2 respectively.

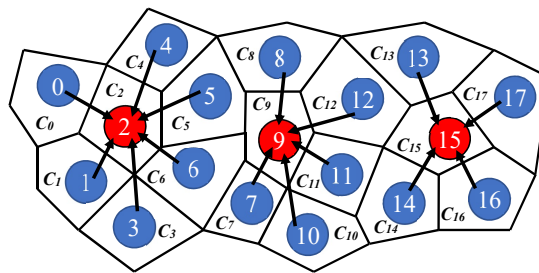
Finally to complete the construction of the next (coarser) matrix level, for each coarse aggregate region the information of the particles within is interpolated to create a new system data point (particle) associated with that aggregate. This is depicted in Fig. 4.14d, where there are new “level 1 particles”, 0, 1, and 2, for coarse aggregate regions, C_0 , C_1 , and C_2 respectively. The original system, as depicted in Fig. 4.14a, is now represented but with fewer data points (particles).

The example in Fig. 4.14 presents a small system of particles, where only one additional level to the original matrix is obtained. In practice, large-scale simulations are likely to give rise to AMG hierarchies of several matrix levels which can be obtained through repetition of the process in Fig. 4.14.

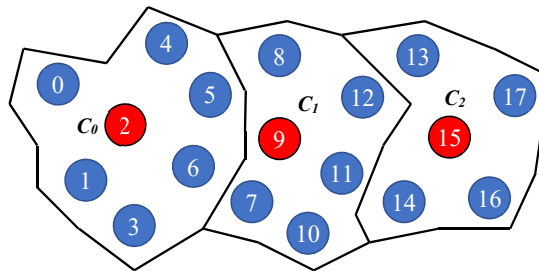
The mathematical/computational algorithm for obtaining successive coarse levels, corresponding to Fig. 4.14 can be split up into 4 stages:

**Level 0**

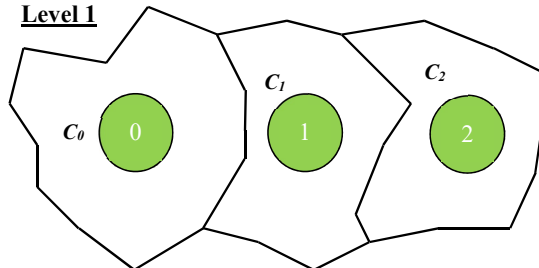
(a) Step 1: In level 0, there is an aggregate section for every particle. This is the original matrix, stored in memory for level 0.



(b) Step 2: Identify coarse particles (red). Each non-coarse (fine) particle (blue) is associated with one coarse particle neighbour only.



(c) Step 3: Define new “coarse aggregate” sections from coarse particles and associated fine particles.

Level 1

(d) Step 4: Define level 1 by interpolation of particle information to a create new “level 1 particle” for each aggregate. This data is stored in memory for level 1.

Fig. 4.14: Obtaining AMG matrix level 1 from level 0, in the context of ISPH.

- **Stage 1:** The construction of coarse and fine points through a chosen coarsening strategy. This stage has been illustrated with steps 1-3 of Fig. 4.14.
- **Stage 2:** Building the interpolation matrix, $[\mathbf{P}]$ from the coarse points ob-

tained in stage 1. This stage and the following two are all part of step 4 of Fig. 4.14.

- **Stage 3:** Constructing the next coarse matrix level $(a+1)$, $[\mathbf{A}^{a+1}] = [\mathbf{P}]^T [\mathbf{A}^a] [\mathbf{P}]$, where the superscript of the system matrix denotes the level number.
- **Stage 4:** Constructing the solution and RHS vectors for the next coarse level $(a + 1)$, \mathbf{x}^{a+1} and \mathbf{b}^{a+1}

Stages 2-4 can be computed in both serial or parallel and the algorithms are common amongst various AMG methods. For stage 1 however, the process of coarsening strategies is inherently sequential, so the parallel MIS(2)AMG aggregation coarsening strategy is more complex than previous techniques. The differences are described in the following two sections explaining both the sequential and parallel GPU algorithms (from the ViennaCL library) in the context of ISPH.

4.6.7.1 The sequential aggregation coarsening strategy

The aim of the coarsening strategy is to identify which points on the grid matrix level are defined as “coarse” or “fine”. In the context of ISPH, the points are particles. There are numerous ways to complete such a process, but presented here is the basic sequential aggregation coarsening strategy.

Fig. 4.15 shows the sequential process for identifying the point types. The example uses shows a portion of a coefficient matrix with the row and column numbers denoted by the associated particle number. The non-zero element matrix entries marked are with a cross and indicate an interaction between a particle i , of row i , and a particle j , of column j . Throughout the process, particles can be defined as one of three states: “Coarse”, “Fine”, or “Undecided”, denoted by “C”, “F”, and “U” respectively. The state of each particle can be seen in each column in the figure.

At the beginning of the process (Fig. 4.15a, step 1), all particles are marked as undecided (U). The identification of coarse and fine particles is carried out by going through each matrix row, in sequence, using the two rules:


- **Rule 1:** If all the neighbouring j particles of particle i , are associated with particles of undecided state only, then the particle i is marked as coarse, and its neighbouring j particles are marked as fine. This is demonstrated in Fig. 4.15b (step 2) where for particle (row) 0, the neighbours (non-zero) entries in that row are for particles 0 and 1, which are both in an undecided state. Therefore, particle 0 (particle i) is marked as coarse, and particle 1 is marked as fine.
- **Rule 2:** If particle i has at least one neighbouring j particle marked as coarse or fine, or is marked as coarse or fine itself, then there is no change of state for any particles and the process moves onto the next particle row in the matrix. For instance in Fig. 4.15c (step 3), consider particle (row) 1 with neighbours 0, 1, and 2. Although particle 2 is in an undecided state, the other two particles are coarse or fine, so the marking process is ignored and the procedure moves on to consider particle (row) 2. This is a similar case for the next particle (row), where particle 2 is neighbours with particle 1 which is fine, so no changes of state are made here.

The example in Fig. 4.15 can be finished by looking at Fig. 4.15d (step 4). For particle (row) 3, only particle 3 is included as a non-zero entry and is of an undecided state. Therefore, rule 1 is used and particle 3 is marked as coarse. Similarly for particle (row) 4, the neighbours (non-zero entries) belong to particles 2 and 4, which are both undecided, so particle 4's state becomes coarse, and particle 2 becomes fine. The marking process is such that each fine particle will be associated with only one coarse particle.

Repeating the process for all particles is such that once completed, every particle is marked as either coarse or fine. The final step of the coarsening strategy is to group the points together into “coarse aggregates” (see Section 4.6.7, Fig. 4.14c), where the number of coarse aggregate regions is equal to the number of coarse particles. The coarse particles are enumerated in sequence, for example in Fig. 4.15e, coarse particles 0, 3 and 4, are labelled as belonging to coarse aggregate regions, C_0 , C_1 , and C_2 respectively. The fine particles are assigned to the same coarse aggregate region as their respective coarse particle neighbour i.e. fine particle 1 is neighbours

Legend

State: C = Coarse, F = Fine, U = Undecided

 Denotes a non-zero matrix entry i.e. an interaction between particles i and j

Particle number	j =	0	1	2	3	4
	State	U	U	U	U	U
i = 0						
i = 1						
i = 2						
i = 3						
i = 4						

(a) Step 1: Make all particles “undecided” (U). Crossed boxes indicate matrix entries between particles.

Particle number	j =	0	1	2	3	4
	State	C	F	U	U	U
i = 0						
i = 1						
i = 2						
i = 3						
i = 4						

(b) Step 2: Row 0 initially all undecided points, particle 0 = coarse (C), particle 1 = fine (F).

Particle number	j =	0	1	2	3	4
	State	C	F	U	U	U
i = 0						
i = 1						
i = 2						
i = 3						
i = 4						

(c) Step 3: Rows 1 and 2 are ignored as they contain non-undecided particles.

Particle number	j =	0	1	2	3	4
	State	C	F	F	C	C
i = 0						
i = 1						
i = 2						
i = 3						
i = 4						

(d) Step 4: Particle 3 = coarse, Particle 4 = coarse, and its neighbour, particle 2 = fine.

Number of coarse particles (3) = number of coarse aggregates (3)

Coarse aggregate region	C ₀	C ₁	C ₂
Particle number (State)	0 (C)	1 (F)	4 (C)
		2 (F)	3 (C)

(e) Step 5: Coarse particles 0, 3, and 4 and their neighbours are labelled C₀, C₁, and C₂ respectively.

Fig. 4.15: Sequential aggregation coarsening strategy example.

with coarse particle 0, so belongs to coarse aggregate region C₀, and fine particle 2 is neighbours with coarse particle 4, and therefore assigned to coarse aggregate region C₂.

The sequential coarsening strategy is now complete, the enumerated coarse aggregates are used to build an interpolation matrix, [P], with dimensions, n × c_a, where c_a is the number of coarse aggregate regions. For instance the interpolation matrix for the example in Fig. 4.15 is constructed from the Fig. 4.15e and shown in Fig. 4.16.

The columns and rows represent the coarse aggregate indices and particles respectively. There can be numerous variations of the interpolation matrix entry values, depending on the interpolation strategy, but for simplicity, each non-zero

		Coarse Aggregates		
		C ₀	C ₁	C ₂
Particle number	$i = 0$	1		
	$i = 1$	1		
	$i = 2$			1
	$i = 3$		1	
	$i = 4$			1

Fig. 4.16: The interpolation matrix corresponding to the example in Fig. 4.15. Blank entries indicate a zero entry.

element is of value 1, here.

The interpolation matrix, and its transpose, are multiplied against the current matrix level, to build the next, $[\mathbf{A}^{a+1}] = [\mathbf{P}]^T [\mathbf{A}^a] [\mathbf{P}]$. The coarse aggregates are similarly used to communicate back up the grid hierarchy when solving the system.

The sequential nature of the coarsening algorithm is such that particles in the matrix marked as coarse are well distributed throughout the system. This is facilitated by the “skipping” of particles with neighbours already marked as coarse or fine, which is crucial to understanding where the parallel MIS(2)AMG algorithm requires modification for ISPH.

4.6.7.2 The parallel MIS(2)AMG aggregation coarsening strategy and modification for ISPH

The parallel MIS(2)AMG algorithm by Bell et al. [323] finds a maximal independent set (MIS) in the matrix system graph using a variant of Luby’s parallel algorithm [324]. The particles included in the MIS are referred to as MIS nodes (of coarse state), otherwise they are called non-MIS nodes (of fine state). Just as in the sequential coarsening strategy algorithm, explained in Section 4.6.7.1, the parallel MIS(2)AMG algorithm also aims to organise particles of a current matrix level into coarse aggregate regions for the next level. The algorithm was specifically designed to take advantage of the GPU’s unique parallel computing architecture.

When applied to ISPH however, the ViennaCL library’s MIS(2)AMG algorithm requires modification to prevent a potential error due to the meshless nature and moving computation points of the method. When at least two coarse particles share exactly the same neighbours as each other (and including each other), the original algorithm has the potential to overwrite data such that a coarse aggregate region

exists without any particles present within, which ultimately leads to divisions of zero during the solution of the PPE matrix.

Just as in the sequential aggregation coarsening strategy algorithm, all particles begin with an undecided state. Then each particle is given a random weighting and a parallel lexicographical ordering algorithm is used to find the MIS nodes, where each particle's state is determined based upon their respective neighbours' state, random weighting and particle/row matrix number. The ordering scheme is repeated until there are no more undecided state particles left. Resulting coarse nodes here are likely to be different to that of the sequential coarsening algorithm at this point because the parallel algorithm considers the influence of neighbours for each i particle as completely independent to all others. This means that particles within each others kernel support radii can be identified as coarse.

Next, the fine particles are reset to an undecided state. All particles in the matrix are now in a state of either coarse, or undecided. The coarse particles are enumerated into their aggregate indices in the same way as that in the sequential algorithm (similarly to that in Fig. 4.15e). The undecided particles are then assigned to coarse aggregates in parallel. This procedure is known as the propagation of coarse aggregate indices. It is here that the MIS(2)AMG algorithm requires modification for ISPH.

In the ViennaCL library's parallel MIS(2)AMG GPU algorithm, the undecided particles (non-MIS nodes) are organised into the aggregate regions, in parallel, by assigning every j particle neighbour of coarse particle i , the same coarse aggregate region number, C_{num} , as that of particle i . The algorithm does not take into account whether any of the j neighbour particles are also of coarse state and will overwrite/re-assign the j particles current coarse aggregate region regardless. If two coarse particles share the same particle neighbours as each other, and are therefore within each other's neighbourhoods, the propagation of coarse aggregate indices algorithm has the potential to assign all the involved particles into just one of the two associated coarse aggregate regions, leaving the other one without any particles.

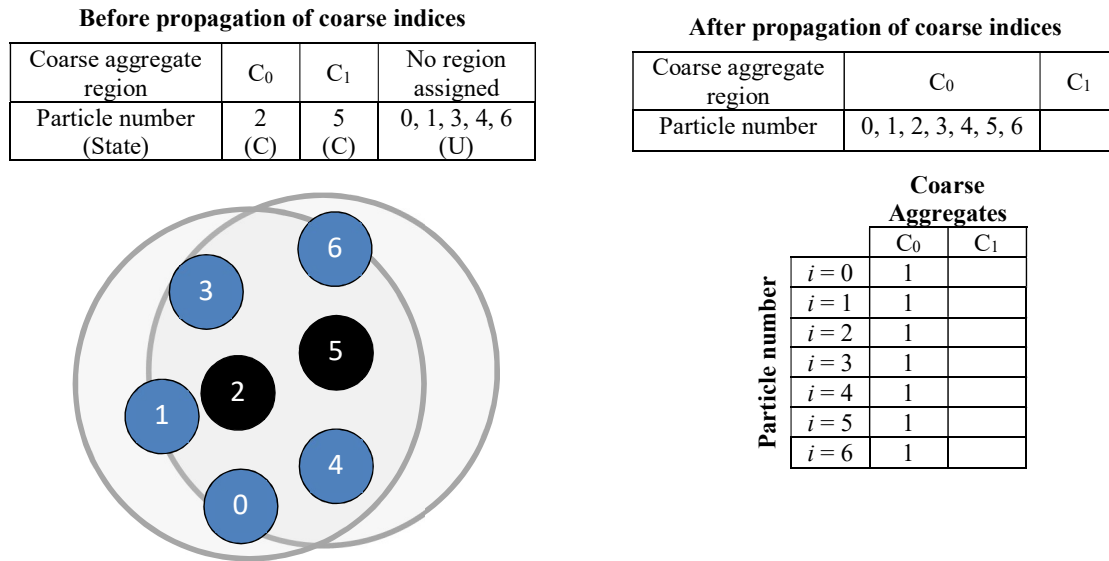
For example, in Fig. 4.17a, particles 2 and 5, in the system have been identified

with a coarse state, they have coarse aggregate indices C_0 and C_1 respectively. The two coarse particles also share exactly the same neighbours, particles 0, 1, 3, 4, and 6, which before the propagation process, do not have a coarse aggregate region assigned. The propagation of coarse aggregate indices is executed for both particles 2 and 5 at the same time in parallel. Data-write race-conflict⁶ between threads can easily occur here and there is a possibility that data is overwritten such that a coarse aggregate region will no longer contain any particles. Fig. 4.17b shows this resultant arrangement of particles amongst coarse aggregate regions, where all particles originally assigned to region C_1 have been re-assigned to C_0 within the same instance of the parallel algorithm execution. The resultant coarse aggregates are C_0 , with all particles in the system, and region C_1 containing no particles at all. The consequence of this is that the interpolation matrix, $[\mathbf{P}]$, will have a column, assigned to a particular coarse aggregate region, with no non-zero entries such as that for C_1 in Fig. 4.17b. Empty columns in the interpolation matrix leads the resultant matrix system in the next level to have empty rows and therefore divisions by 0 during execution of the linear solver.

In ISPH, the error is possible because the particles are constantly moving. In transient or violent flows, particle distributions may become highly irregular (even with shifting) at the free surface. To resolve the issue here, a conditional-if statement which allows propagation of coarse aggregate indices to non-coarse particles only is inserted. This prevents the occurrence of an interpolation matrix with entirely empty matrix columns associated with a coarse aggregate index, increasing the reliability of the parallel MIS(2)AMG preconditioner for ISPH.

The procedure is not usually an issue in meshed-based methods as the stencil, and therefore number of cell “neighbours” (as an analogy to SPH), is fixed. This is demonstrated in Fig. 4.18a, which shows a simple 1-D case. Cells 1 and 2 have been identified as coarse cells (this is possible due to the parallel lexicographical ordering algorithm) and have cell neighbours $[0, 1, 2]$, and $[1, 2, 3]$ respectively. In a parallel GPU framework, the propagation of coarse aggregate indices results stored

⁶When two, or more, individual threads try to write data into the same memory address at the same time, this leads to an uncertainty of which value the memory address ultimately takes.



(a) The coarse aggregate indices of particles before propagation.

(b) Possible resulting propagation of coarse aggregates indices and corresponding interpolation matrix.

Fig. 4.17: A system of particles with identified 2 coarse particles, which have the same neighbours as each other.

in cells 0, 1, and 2 will be assigned to coarse aggregate regions C_0 , associated with coarse cell 1, and cells 2, 3, and 4 will be assigned to coarse aggregate region C_1 , associated with coarse cell 2. Dependant on the order of thread execution there are two resulting possibilities for the assignment of cells to coarse aggregate regions as seen in Fig. 4.18b, where a thread assigning coarse aggregate regions to neighbours of cell 1, may overwrite the associated region of cell 2, and another thread executing the same task but for cell 2, may re-assign cell 1's coarse aggregate region. This is harmless however, as there will still be matrix entries for both the columns of C_0 and C_1 in the interpolation matrix, preventing the error as seen for ISPH.

4.6.7.3 ViennaCL MIS(2)AMG parameters

The ViennaCL library requires several parameter inputs for the MIS(2)AMG preconditioner. The parameters and their values used in this study (for Chapter 5) obtained from numerical experiments are listed as follows:

- **Aggregation interpolation type:** The parameter determines whether aggregation or smoothed aggregation is used to interpolate solution approxima-

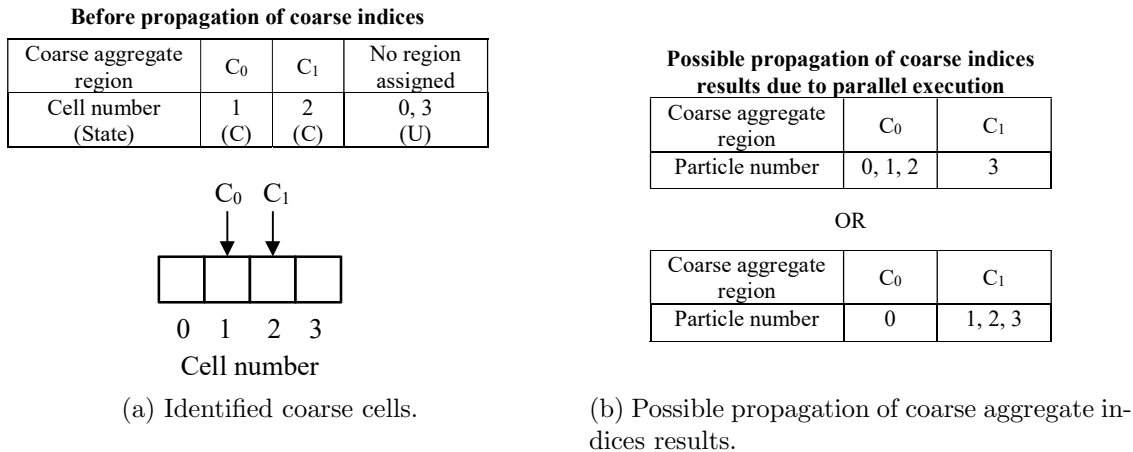


Fig. 4.18: Simple 1-D meshed method example for the propagation of coarse aggregate indices resulting from the parallel MIS(2)AMG algorithm

tions between matrix levels. The standard “aggregation” method uses an interpolation matrix, $[\mathbf{P}]$, of ones as demonstrated in Fig. 4.16. In the ViennaCL library, “smoothed aggregation” also applies a weighted Jacobi iteration to the approximation solution between matrix levels in addition to the interpolation matrix. Smoothed aggregation aims to improve the quality of interpolation between matrix levels. From numerical experiments, the solution of the PPE matrix using a smoothed aggregation method generally requires less solver iterations and is thus used here.

- **Jacobi smoother weight value:** This parameter is to be used in conjunction with the smoothed aggregation. It defines the relaxation value for the Jacobi smoother. The parameter would ideally be 1.0 for higher accuracy, but this value leads to an error from empty entries in the main diagonal of a matrix level (and subsequent divisions of zero) when there are free-surface particles. Therefore, a value of 0.999999 is used here, which has been found to provide a balance between accuracy and computation speed of results.
- **Pre-smoothing steps:** The number of times the Jacobi smoother is applied to a fine level before restricting the residual to the coarse level. Numerical experiments have found that applying such steps have increased the preconditioner setup time and solution so there are no pre-smoothing steps used

herein.

- **Post-smoothing steps:** The number of times the Jacobi smoother is applied to the coarse grid correction when interpolating back to a fine level. Numerical experiments have shown that the use of one post-smoothing step significantly improves the solution time over having no steps, and is therefore used here. Values more than one have shown the extra steps increase the number of solver iterations required.
- **Coarse level cut-off:** Defines the lower threshold for the number of coarse aggregates in a matrix level before the preconditioner setup stops creating new matrix levels. In the ViennaCL library, an exception is thrown if this threshold is not reached in the setup phase and so the simulation execution is halted. The algorithm is such that all free-surface particles will always be identified as coarse particles, so by using the number of such particles each time step for the coarse level cut-off parameter, the prevention of the exception is achieved dynamically. The number of free-surface particles in a time step is computed during the serial GPU function (see Alg. 2) at the beginning of Stage 2 of the pressure projection.

The final challenge to address for the implementation of ISPH on the GPU are the associated memory limitations. The penultimate section of this chapter details some of the techniques used to treat this.

4.7 Addressing the GPU memory limitations

The memory usage of the new code is not completely optimised because of the limited time to conduct this study. However, presented here are the methods implemented to address the challenges of memory limitations for ISPH on the GPU:

- To aid the issue of memory limitations on the GPU, mixed precision storage is used. With the exception of particle positions and CSR matrix arrays, all the particle data is stored in single precision. Particle positions, \mathbf{r} , are stored

in double precision to maintain accuracy in the kernel. Numerical experiments have concluded that the CSR matrix arrays are required to be stored in double precision in order for the linear solver to converge for simulations of approximately more than 500,000 particles.

- The majority of the code computations are performed in single precision, taking advantage of high single-precision FLOPS produced from Nvidia GPUs. For the CSR matrix arrays, calculations are performed with `float` register variables. The final values of the PPE matrix entries are cast to `double` for the matrix arrays. The library then performs the solving of the matrix using double precision. To minimise error in calculating the inverse of matrices for the MLS kernel and kernel gradient normalisation variables (Eqs (3.46) and (3.20)), `double` registers are created to perform the calculation and then cast to `float` for storage.
- Similar to the WCSPH DualSPHysics code, all memory is allocated before the main simulation execution to save time. In ISPH, it is unknown how many non-zero entries are required for the PPE matrix because of the moving computational points. However, given the kernel support size, it is possible to estimate the maximum value number of non-zero entries, Nnz_{max} , required for the matrix and therefore the memory allocation size of the CSR matrix arrays, `aValues` and `column`. In Chapter 5, as an upper bound for matrix memory storage, $Nnz_{max} \approx 1.5N_{n,max}$, is used where $N_{n,max}$ is the maximum number of neighbours a particle has within a uniformly distributed Cartesian arrangement. For example, for a 2-D case using the quintic spline, $N_{n,max} = 44$ and therefore $Nnz_{max} = 70$. The memory for the CSR arrays are subsequently allocated as follows:

```

- aValues=new double[ $Nnz_{max} \times np$ ];
- column=new unsigned[ $Nnz_{max} \times np$ ];

```

This estimation is usually sufficient as the use of shifting will maintain an even distribution of particles throughout most of a simulation.

4.8 Conclusions

This chapter described the key details of the novel methodology of this thesis, implementing ISPH on the GPU. An overview of GPU programming was first given before explaining the changes required for conversion of the WCSPH DualSPHysics code [15] to an ISPH algorithm where an algorithm for the parallel population of the matrix on the GPU was included. Methods for solving the ISPH PPE were discussed with the conclusion of combining DualSPHysics with the ViennaCL [16] open-source linear algebra library for fast solutions of the linear matrix system. The implementation of the library's MIS(2)AMG preconditioner also requires modification for ISPH. Finally the associated computational memory limitations of ISPH on the GPU have been acknowledged. All the challenges identified in Section 2.7 for ISPH on the GPU have been addressed.

The next chapter presents several test cases to validate the methodology of Chapters 3 and 4. The performance of the new code is also assessed.

Chapter 5

Validation Tests

5.1 Introduction

This chapter presents a series of test cases and numerical experiments to validate and test the performance of the new GPU-accelerated ISPH code, Incompressible-DualSPHysics, created from the implementation methodology of ISPH on the GPU presented in Chapters 3 and 4.

5.2 Technical specifications of hardware and software

The ISPH algorithm has been implemented for both the CPU (serial and multi-threaded OpenMP) and GPU in DualSPHysics version 4.0. The CPU experiments were run on an Intel(R) Xeon(R) CPU E5-2640 v3 (16GB RAM, 2.60GHz) with 8 cores (16 threads). Two different Nvidia GPUs are used for comparison, the Nvidia GeForce GTX 1070, and the Nvidia Tesla K40c. A summary of the GPU device properties are found in Table 5.1.

Run time comparisons between all devices are made in Sections 5.4 and 5.6 where a simulation time is defined as the time taken from the beginning of the “Mirror boundary” set-up (see Fig 4.10) to computing a desired number of time steps (to be stated for each case). The runtimes presented are the averages of 10 simulation runtimes for each resolution.

Table 5.1: GPU properties. GFLOPS stand for floating-point operations per second on the order of giga (10^9).

GPU	GeForce GTX 1070	Tesla K40c
Compute capability	6.1	3.5
Multiprocessors (CUDA cores)	15 (1920)	15 (2880)
Clock rate (GHz)	1.78	0.88
GFLOPS (single precision)	6463	5040
GFLOPS (double precision)	202	1680
Global memory (GB)	8	12
Memory bandwidth (GB/s)	256	288

The ViennaCL library’s Bi-CGSTAB solver requires the specification of the relative solver tolerance. It has been found for higher resolution test cases, the tolerance setting should be adjusted in order to achieve the desired accuracy of results. The adjustment is required more when using the Jacobi preconditioner at higher resolutions than the MIS(2)AMG preconditioner, but is still needed for both. The preconditioner and solver tolerance value used for each test case will be stated.

5.3 Impulsively started plate

The case of an impulsively started plate sees the instantaneous movement of a solid boundary against a body of fluid. This demanding test with an analytical solution is used to show that Incompressible-DualSPHysics produces accurate results for the free surface as well as demonstrating convergence of the methodology. The flow is also assumed to be inviscid and without gravity.

Fig. 5.1 shows the initial setup of the test case; a vertical plate moves instantaneously with a constant velocity $u_p = 0.2$ m/s against a body of still water. Symmetry is used to impose the horizontal boundary indicated with a dashed line.

Roberts [325] generalised Peregrine’s [17] analytical solution of the flow to suit different plate velocities. The solution describes the progression of the free-surface elevation, η , of the fluid at a position, x , relative to the moving plate for a water depth, d , as described in Eq. (5.1). Herein, comparisons are made with Peregrine’s solution where the water depth is 0.5 m measured from the dashed centreline in Fig. 5.1:

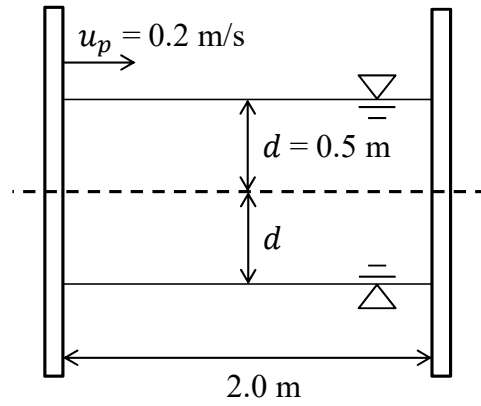


Fig. 5.1: The initial geometry of the impulsively started plate test case. The centre line (dashed) indicates a symmetry between the top and bottom half of the setup.

$$\eta = -\frac{2u_p t}{\pi} \ln \left(\tanh \left(\frac{\pi x}{4d} \right) \right). \quad (5.1)$$

Three resolutions with initial particle spacings, 0.025 m, 0.0125 m, and 0.00625 m are used to demonstrate the convergence of the L_2 error norm, of the free-surface elevation in the top half of the flow given by Eq. (5.2). The Jacobi preconditioner is used for this test case with a solver tolerance of 10^{-5} for all resolutions. Fig. 5.2 shows that linear convergence is achieved with an order of convergence of 1.18.

$$L_2 = \sqrt{\frac{\sum_i (\text{Theoretical} - \text{ISPH})^2}{\sum_i \text{Theoretical}^2}} \quad (5.2)$$

Fig. 5.3 shows particle positions at $t = 0.6$ s near the plate and free surface for the finest resolution, $dp = 0.00625$ m. The free-surface particles coloured in red are included within the L_2 error norm calculation. Due to a singularity of the solution at the boundary, particles within $1dp$ of the moving plate (indicated with the dashed line) and particle spray, detached from the main body of fluid, are omitted from the results. The theoretical free surface calculated from Eq. (5.1) is plotted with the solid line.

Fig. 5.4 shows a snapshot of pressure field, corresponding to that in Fig. 5.3, in the upper half of the simulation. Spray caused by approximation of the singularity and impulsive motion of the plate can be seen near the tip of the water jet. The pressure field plotted shows a smooth distribution without instabilities. This case

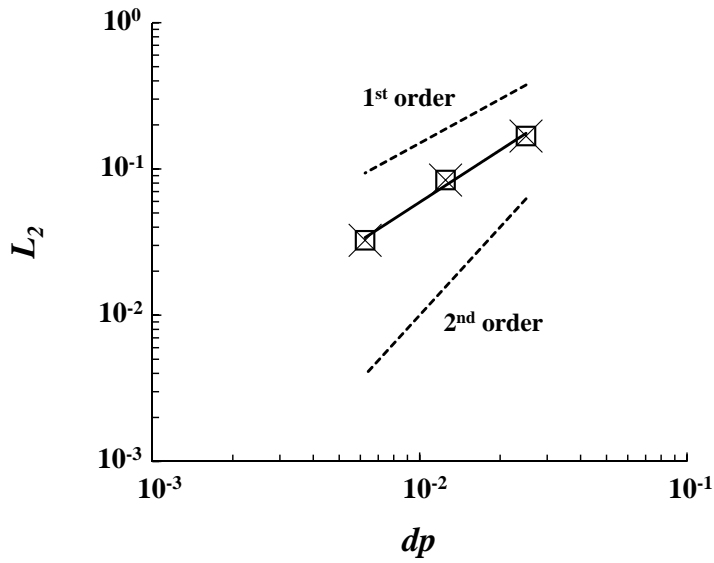


Fig. 5.2: Impulsively started plate convergence plot for both CPU (crossed marker) and GPU (square marker) codes; the relative L_2 error norm of the free surface elevation for $u_p = 0.2$ m/s, $d = 0.5$ m at $t = 0.6$ s. The solid trendline shows a convergence rate of 1.18 and the dashed lines above and below represent first and second order convergence respectively.

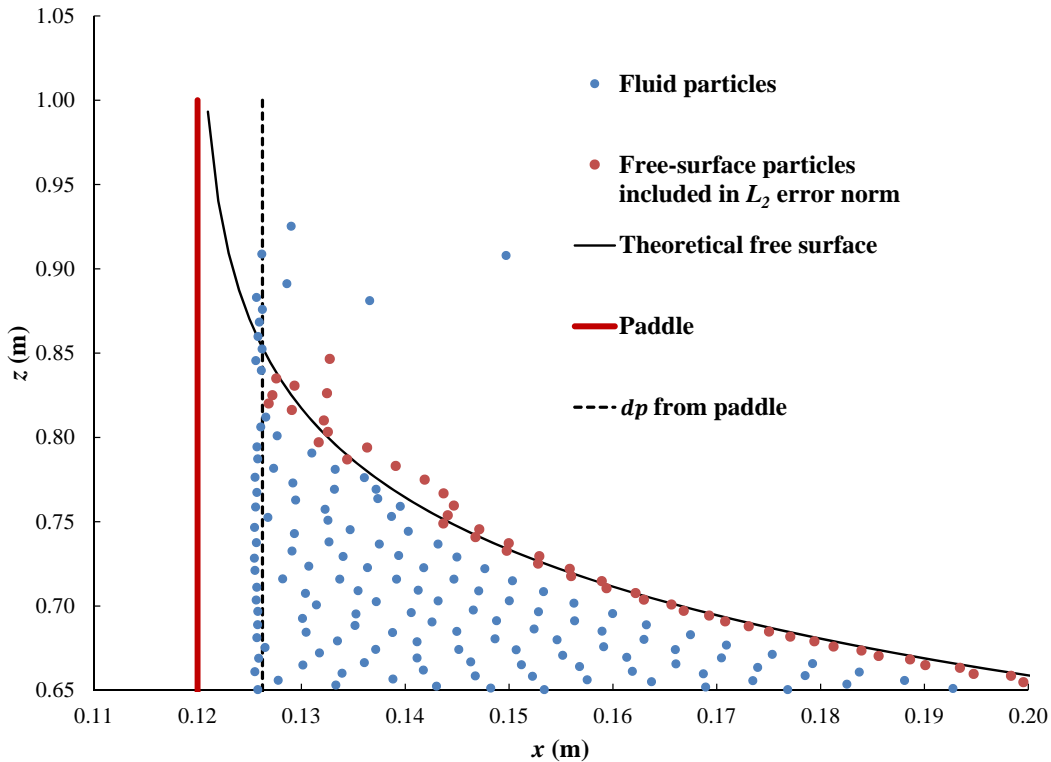


Fig. 5.3: Impulsively started plate ($dp = 0.00625$ m) close-up of fluid particle (blue) positions near the plate and free surface. Particles coloured in red are included within the calculation of L_2 (Eq. (5.2)). The solid line shows the theoretical free surface according to Eq. (5.1).

demonstrates that the new solver can produce accurate results for an impulsively moving plate.

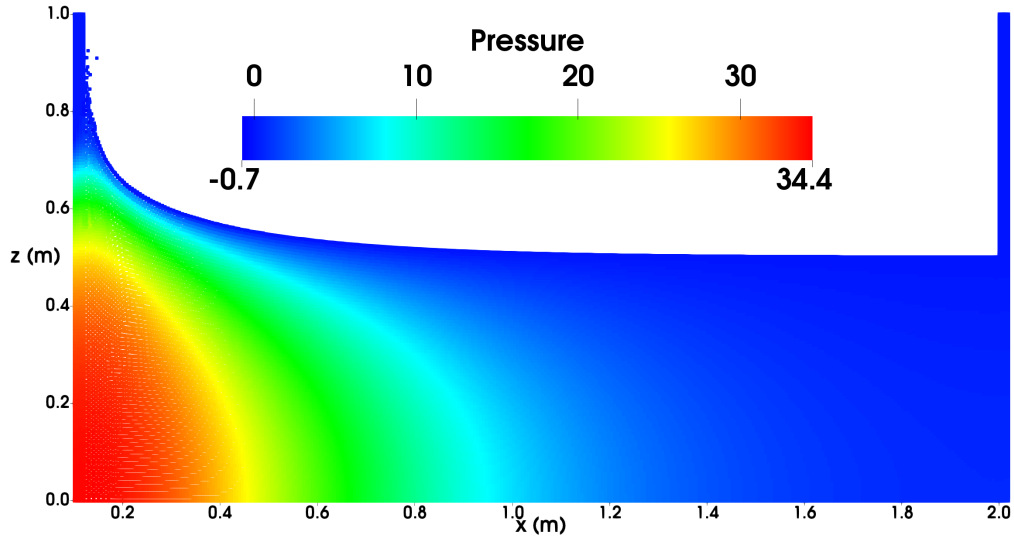


Fig. 5.4: Impulsively started plate pressure plot for $u_p = 0.2$ m/s, $h = 0.5$ m at $t = 0.6$ s. Here, the initial particle spacing is 0.00625 m. Units in Pa.

5.4 2-D incompressible flow around a moving square in a rectangular box

The case of 2-D incompressible flow around a moving square in a rectangular box is the 6th benchmark test case [18] proposed by the SPH European Research Interest Community (SPHERIC). The case further demonstrates the stability and flexibility of the Incompressible-DualSPHysics code.

Fig. 5.5a shows the geometry of the test case. The fluid domain covers a 10 m x 5 m area and is surrounded by solid boundaries enforcing Neumann boundary conditions. In the left handside of the domain is a 1 m x 1 m square which is initially at rest, then accelerates to a final maximum velocity of 1 m/s using the prescribed motion, as in Fig. 5.5b, provided with the benchmark test [18]. The density of the fluid is 1 kg/m³, so therefore its kinematic viscosity, ν , can be adjusted for different Reynolds numbers, Re equal to $1/\nu$.

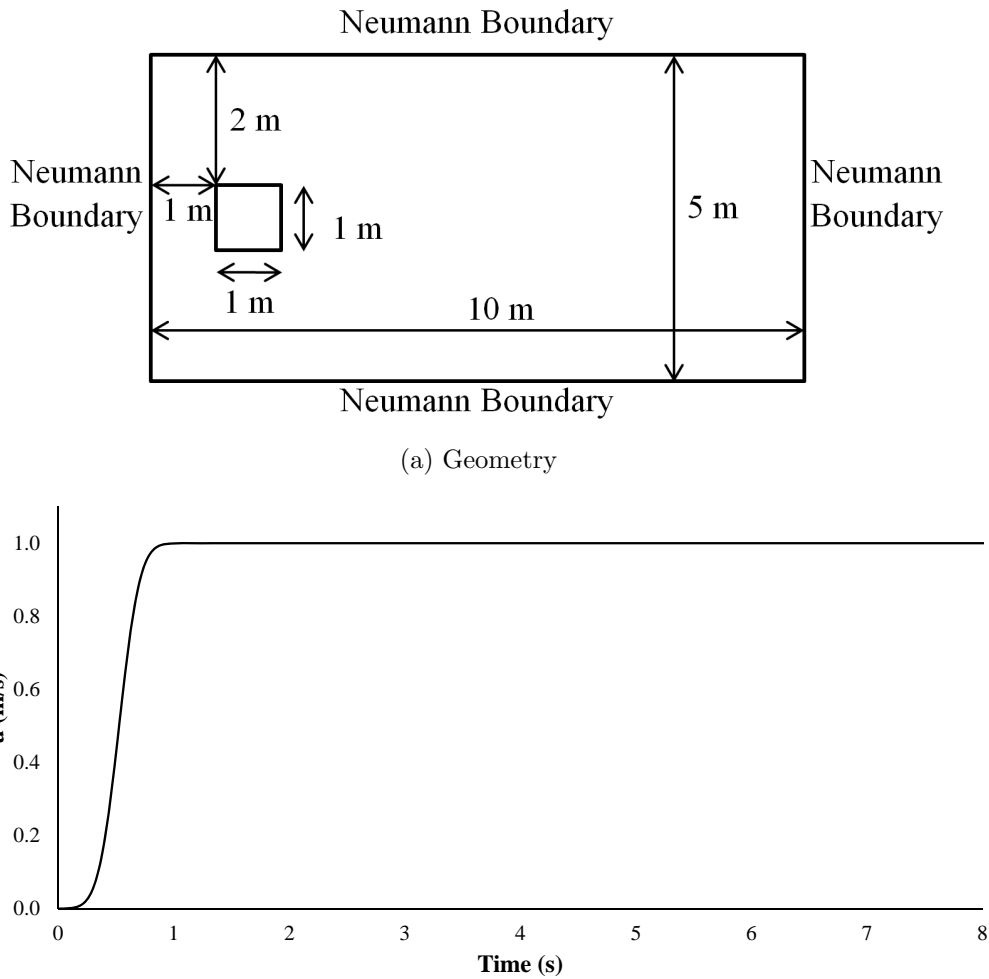


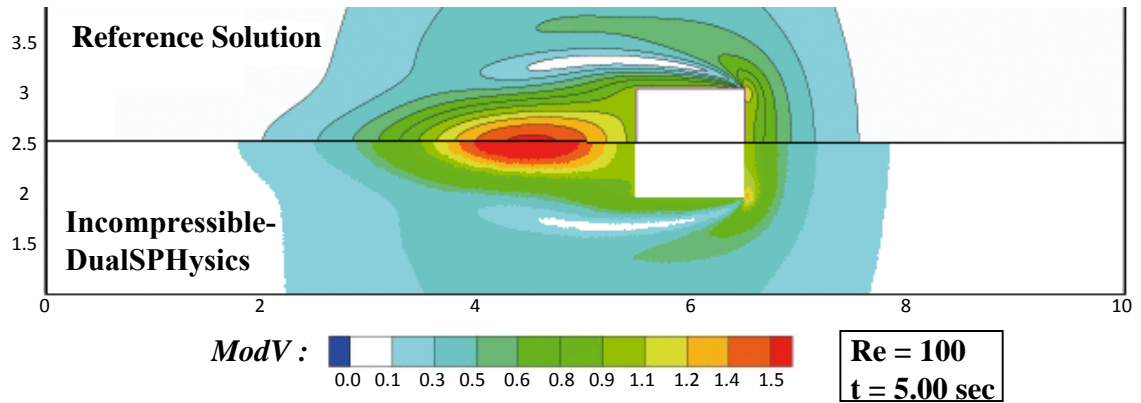
Fig. 5.5: The setup for the 6th SPHERIC benchmark test case: 2-D incompressible flow around a moving square in a rectangular box.

Despite the creation of a singular matrix system in the absence of a Dirichlet boundary condition, a solution to the PPE matrix was found by combining the linear solver, set to a tolerance of 10^{-6} , with a Jacobi preconditioner. The linear system here is such that a Krylov subspace method converges to a solution [326, 327].

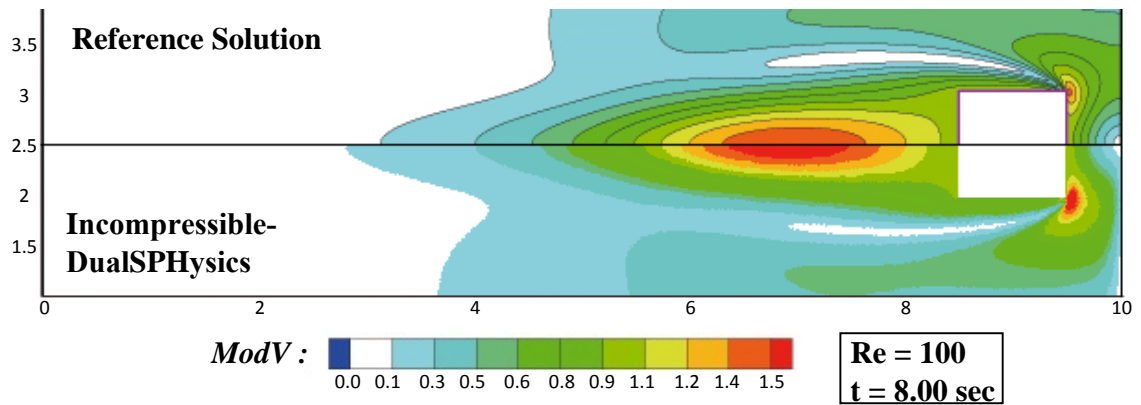
5.4.1 $Re=100$

Approximately 500,000 particles were used to compute 8.0 s of physical time (4,001 time steps) on the GTX 1070 GPU in 40 minutes. Figs 5.6a and 5.6b compare the velocity fields of the ISPH solution with a reference finite difference solution [18] at times, $t = 5.0$ s and $t = 8.0$ s respectively. Both figures show general agreement between the two methods, where the contours are of similar shape and size. The

main difference is seen at the front corners of the square, however resolving flow around external corners in SPH is generally a problem [211].



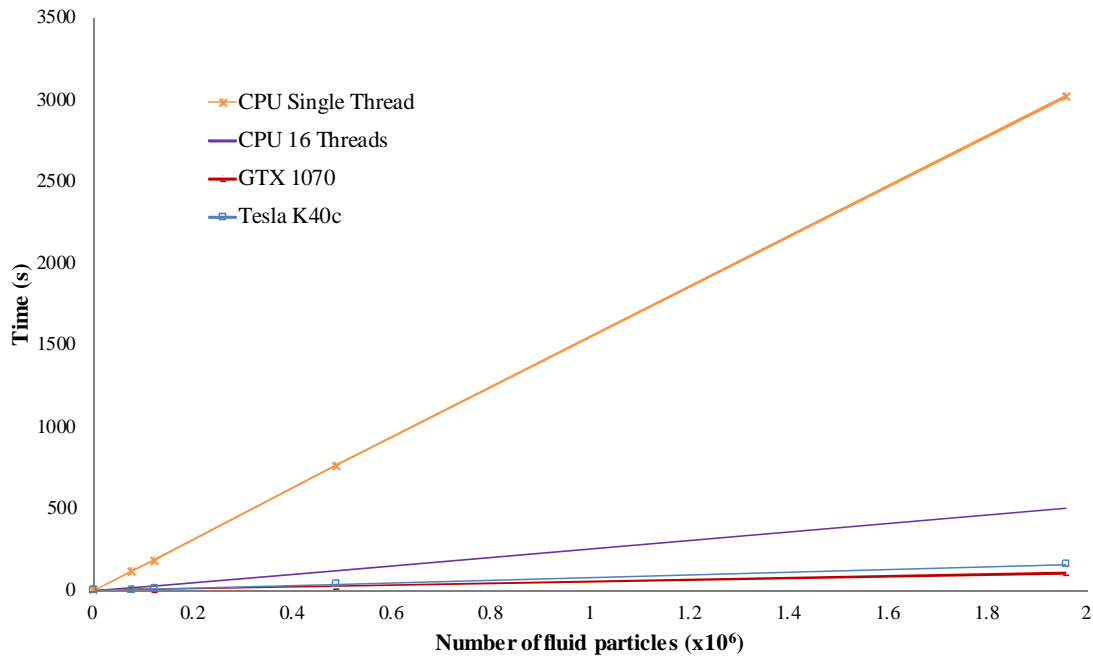
(a) $t = 5.0$ s



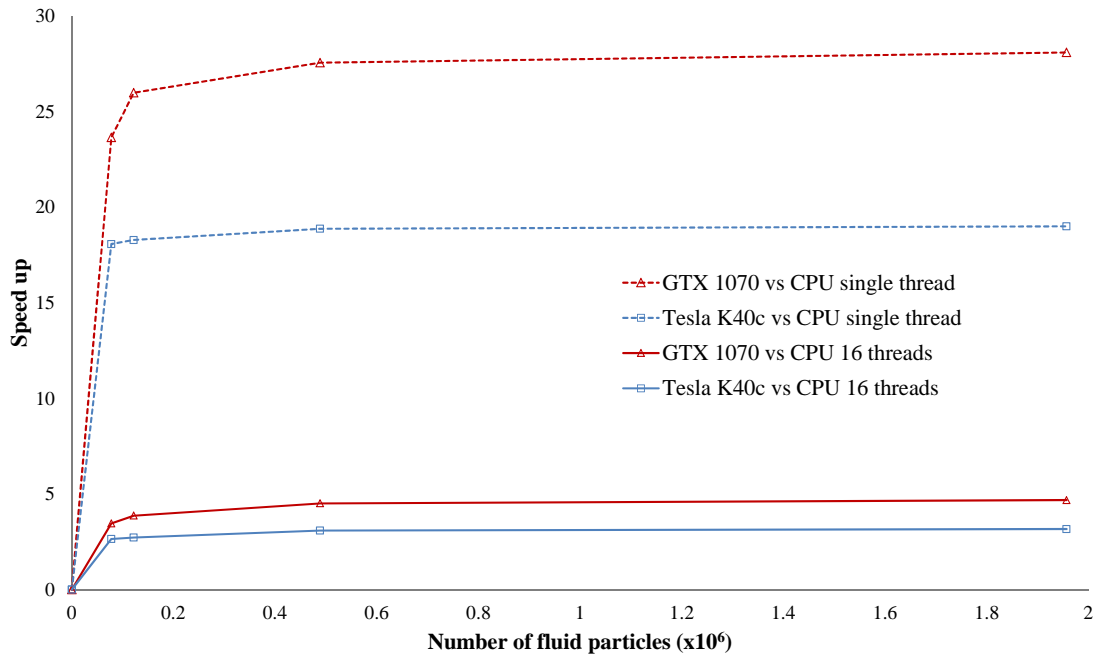
(b) $t = 8.0$ s

Fig. 5.6: 2-D incompressible flow around a moving square in a rectangular box velocity magnitude field comparison between Incompressible-DualSPHysics and a finite difference reference solution [18]. Units in m/s.

Fig. 5.7a presents the average times to compute the first 50 time steps of the test case for various resolutions up to nearly two million particles. The two GPU devices demonstrate a clear improvement over both the CPU single and 16-threaded runtimes where their respective speed ups are illustrated in Fig. 5.7b. The GTX 1070 GPU provides the best performance, for the highest resolution it gives a 27.5 times and 4.5 times speed up over the CPU single thread (serial) and 16-threaded (parallel) codes respectively.



(a) Run times for the first 50 time steps



(b) GPU speed up

Fig. 5.7: Runtime comparisons for computing the first 50 time steps of the 2-D incompressible flow around a moving square in a rectangular box test case. A Jacobi preconditioner is used for all tests.

5.4.2 $Re=1$ million

To emphasise the stability of the code, the test case is repeated but for $Re = 1$ million ($\nu = 10^{-6}$ m²/s). The GTX 1070 GPU completed the experiment in 3 hours

and 40 minutes. Fig. 5.8 shows snapshots of the flow evolution. The particle ID numbers are displayed at 2 second intervals to demonstrate flow characteristics. As the box moves forward, particles in its path are pushed to either side and vortices are created by the front two corners of the box. These vortices are fractal in nature and move down the side of the box before residing in the box's wake. Vortex shedding is observed in Figs 5.8c and 5.8d. Although there is no reference solution, the behaviour here is typical of similar flows in CFD [328, 329].

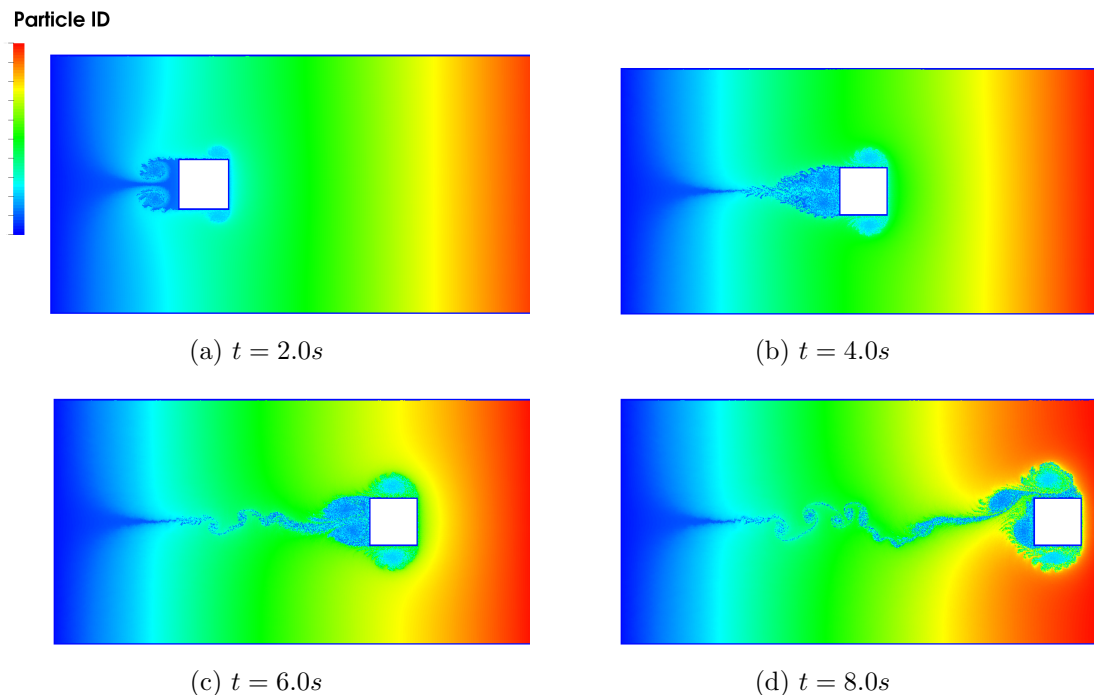


Fig. 5.8: 2-D incompressible flow around a moving square in a rectangular box, $Re = 1$ million. Particle are coloured according to ID number.

5.5 Dambreak

In this section, the classical SPH dambreak test is used to investigate the performance gains from accelerating ISPH with the GPU for free-surface flows in 2D and 3D. The solver tolerance for all results presented was set to 10^{-8} , a value found to be necessary for avoiding pressure fluctuations during impact.

5.5.1 2-D validation

The dambreak test by Koshizuka and Oka [19], as shown in Fig. 5.9, is simulated here. The geometry of the tank here however, is extended in the z -direction to account for spray after impact on the RHS wall of the tank. A column of fluid with width, L , and height, $2L$, where $L = 0.146$ m, is placed on the LHS of a tank with a width of $4L$ and height of 2 m. At $t = 0$ s, the fluid accelerates with gravity, $g = 9.81$ m/s². A free-slip condition is applied to all solid boundaries, and the fluid is modelled as water such that $\nu = 10^{-6}$ m²/s and $\rho = 1000$ kg/m³.

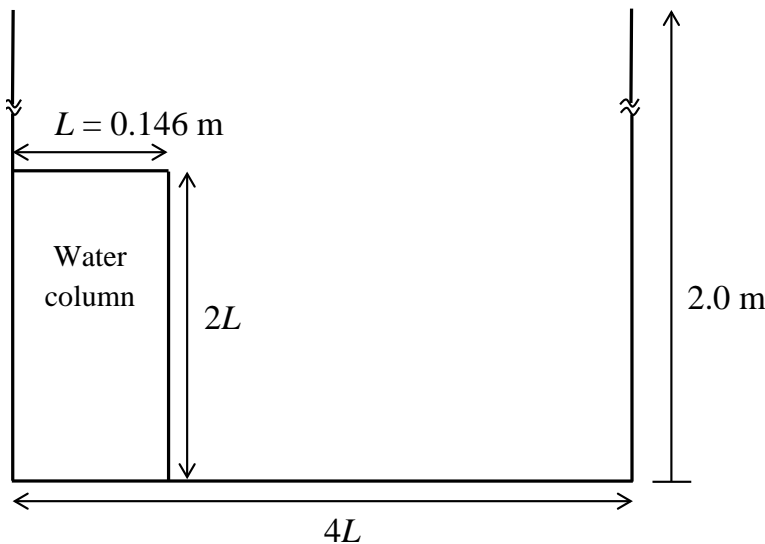
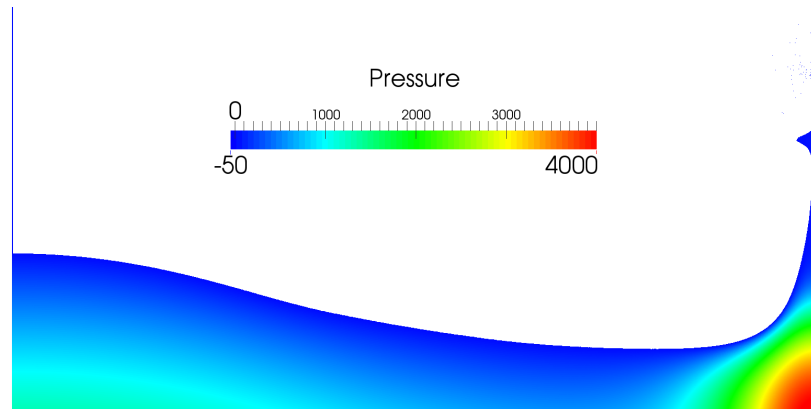


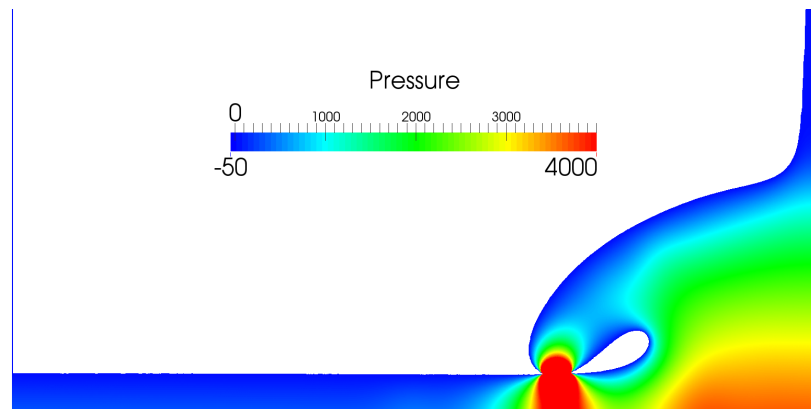
Fig. 5.9: Dambreak test geometry. A replication of the experiment of Koshizuka and Oka [19], but with extended tank walls.

The following validation test uses 680,000 fluid particles and a Jacobi preconditioner. Fig. 5.10 shows snapshots of the pressure field during critical points of the flow evolution. Figs 5.10a and 5.10b display the impact on the right-hand wall and subsequent overturning wave event respectively. For SPH, distortions in the pressure field are most likely to occur during impact events, but here the pressure distribution remains smooth and noise-free throughout the simulation. Fig. 5.10c shows a close-up snapshot of the fluid overturning just before impacting on itself. The smooth pressure distribution near the free surface is clearly evident. Moreover, the use of a high resolution, enabled by the GPU, and the PPE in ISPH with $P = 0$ enforced on the surface highlights the two distinct free surfaces of the overturning

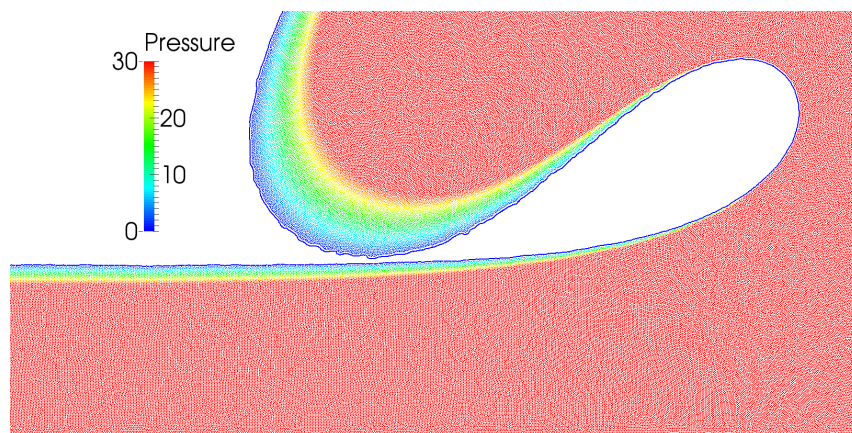
wave's tip and fluid on the tank bottom. Each of these free-surface locations are within a kernel support radius of each other. Such a noise-free pressure field would not be observed in WCSPH as the free-surface particles would affect each other's kernel summations.



(a) Impact with wall, $t = 0.3$ s



(b) Initial flip over impact, $t = 0.759$ s



(c) Close-up snapshot of the flip-over just before impact, $t = 0.758$ s

Fig. 5.10: 2-D dambreak using 680,000 fluid particles. Pressure is plotted in Pa.

The leading toe position of the dambreak is compared with the Koshizuka and

Oka's experimental data [19] in Fig. 5.11. The non-dimensionalised values of toe position, X/L and time, $t^* = t(2g/L)^{0.5}$, are plotted against each other where X is the x-position of the toe. Although the ISPH simulation shows the flow to move faster than the experimental data, there is agreement for the rate of change of position. The discrepancy may come from excluding the release gate as occurred in the experiment [131].

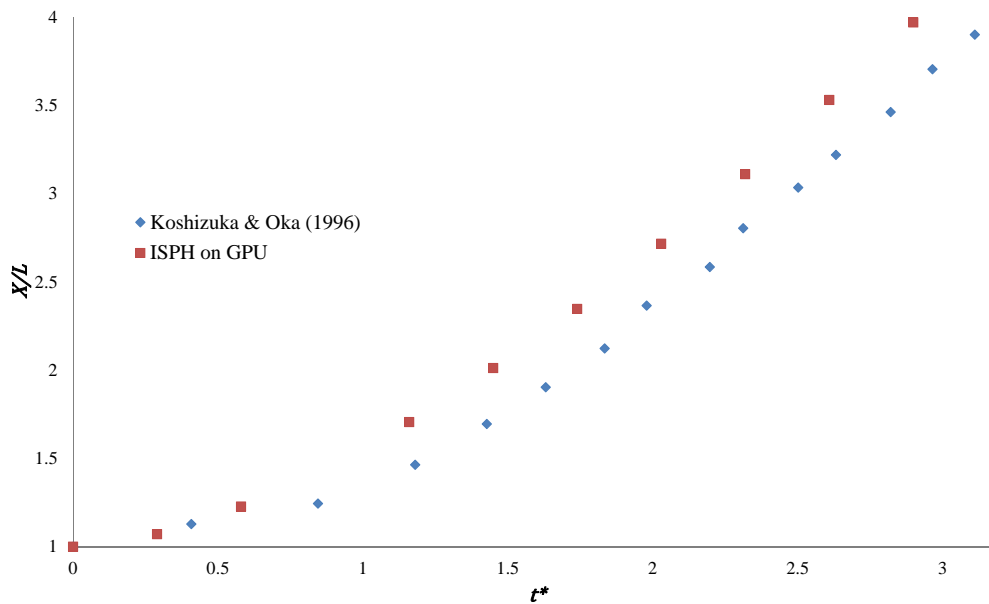


Fig. 5.11: Comparison of the dambreak toe with experimental data of Koshizuka and Oka [19].

5.5.2 3-D validation

For 3-D simulations, the computational expense increases significantly. From 2D to 3D, the number of neighbours, for a particle in a uniformly distributed arrangement, using the quintic spline increases from 44 to 274 and, for ISPH, this means a large increase in memory expenditure due to the PPE matrix. Therefore, the maximum number of particles that can be computed on the GPU is limited. For the same number of particles, a WCSPH simulation is not subject to the additional memory expenditure and subsequent particle limitation.

To reduce the memory usage for 3-D simulations, the Wendland kernel with a smoothing length, $h = 1.3dp$, and support size, $2h$, is used resulting in 80 neighbours for a particle in a uniformly distributed arrangement.

To validate the use of the Wendland kernel for 3-D simulations, the toe comparison from Fig. 5.11 is repeated in 3D. The geometry from Fig. 5.9 is used and extended in the y -direction by 0.2 m, the height of the tank walls are also reduced to 0.3 m. A dp of 0.002 m was used resulting in a total of 384,687 particles. A snapshot of the 3-D simulation at $t = 0.3$ s is shown in Fig. 5.12. Spray can be observed in the figure as the fluid travels up the wall post-impact. Although other ISPH literature [11, 175] exhibits such fragmentation, the accuracy of the simulation regarding this is unclear due to the lack of quantitative experimental data and difficulties in obtaining accurate qualitative comparisons.

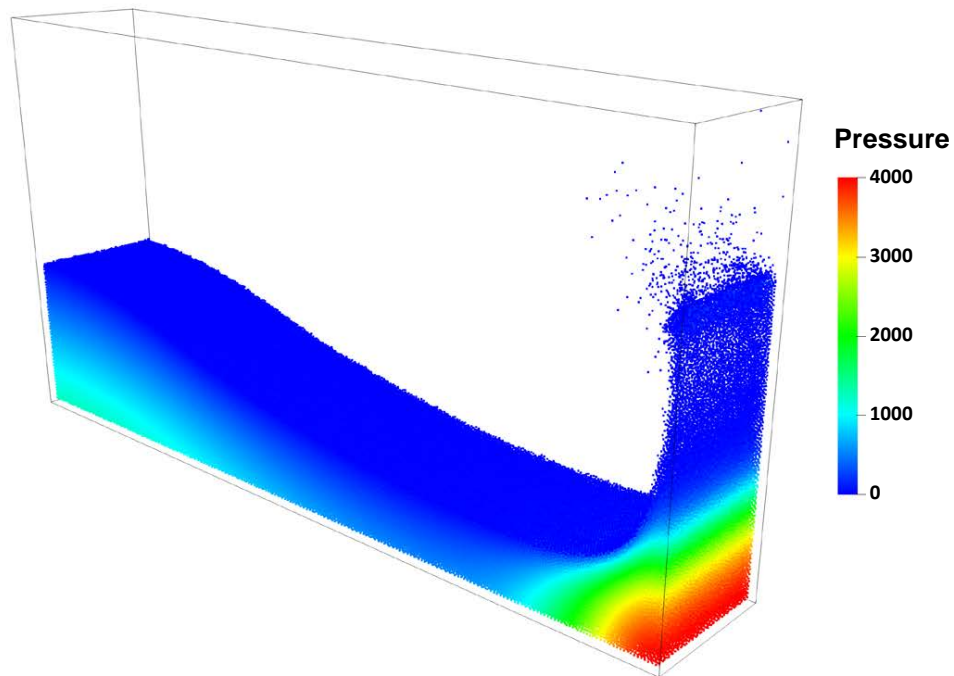


Fig. 5.12: 3-D dambreak simulation, $t = 0.3$ s. The geometry is the same as that in Fig. 5.9 and extended in the y -direction by 0.2 m. $dp = 0.002$ m. Pressure is plotted in Pa.

Fig. 5.13 shows the toe position results of the 3-D ISPH simulations using the Wendland kernel and the quintic spline, which are once again compared to the experimental results of Koshizuka and Oka [19]. Both kernels produce results which are near identical to each other and to the 2-D validation in Fig. 5.11. The figure shows, for 3-D simulations, the Wendland kernel can produce similar results for a reduced computational expense. For instance, the memory reserved for the `aValues` array to store the non-zero entries of the PPE matrix is 367 MB ($Nn_{z_{max}} = 125$) and

1,174 MB ($Nn_{z_{max}} = 400$) for the Wendland kernel and quintic spline respectively. Therefore, in the following section, the Wendland kernel is used for performance analysis of 3-D simulations.

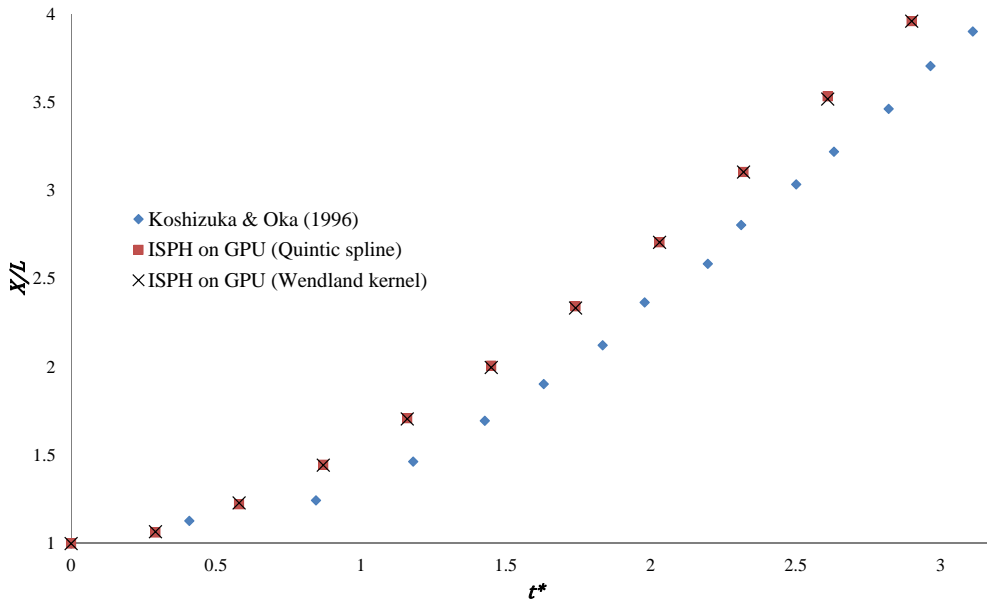


Fig. 5.13: Comparison of the quintic spline and Wendland kernel for the position of the dambreak toe in 3D with experimental data of Koshizuka and Oka [19].

5.6 Performance Analysis

Here, the performance of the new Incompressible-DualSPHysics code is analysed using the 2-D and 3-D dambreak test cases in Section 5.5. Firstly, the number of particles that can be simulated on different GPUs is presented. Then, runtime comparisons are conducted. This is followed by analysis of the effect of the matrix preconditioning choice for a full simulation. Finally, the different stages of the full solution algorithm are profiled.

5.6.1 Maximum particles for GPU RAM

Tables 5.2a and 5.2b show the approximate maximum total number of particles that are able to be computed for GPUs limited to 2, 4, 8, and 12 GB of RAM with the Jacobi and MIS(2)AMG preconditioners respectively. The largest GPU RAM size is able to compute up to approximately 10 million particles in 2D and 6.1 million

particles in 3D using the Jacobi preconditioner. The use of the MIS(2)AMG preconditioner reduces the maximum allowable number of particles to 3.6 million and 2.4 million (for 2D and 3D respectively) because it requires extra memory usage to store the components of the AMG. The values presented here are approximate because the linear solver’s memory requirements will vary with the number of particles and non-zero entries in the matrix per time step, depending on the test case. With just 4 GB of GPU RAM, Incompressible-DualSPHysics can still compute up to approximately 3 million particles in 2D and 2.2 million particles in 3D (when $Nnz_{max} = 1.5N_{n,max}$, see Section 4.7).

Table 5.2: Approximate maximum total number of particles (in millions) that can be computed for a various GPU RAM size. 2-D simulations use the quintic spline, 3-D simulations use the Wendland kernel.

	(a) Jacobi preconditioner			(b) MIS(2)AMG preconditioner			
	4 GB	8 GB	12 GB		4 GB	8 GB	12 GB
2D	3.0	6.2	10.0	2D	1.3	2.2	3.6
3D	2.2	3.9	6.1	3D	0.9	1.4	2.4

5.6.2 Dambreak runtime comparisons

Fig. 5.14 shows the times taken to complete the first 10 time steps of the 2-D dambreak simulation (as in Section 5.5.1) for various resolutions using either the Jacobi or MIS(2)AMG preconditioners. For the Jacobi preconditioner runs, up to approximately 4.3 million fluid particles were computed on the GTX 1070 GPU, and up to 9.8 million fluid particles on the Tesla K40c GPU. For the MIS(2)AMG preconditioner, the highest number of fluid particles computed on the GTX 1070 GPU was 2 million, and for the Tesla K40c, up to 2.7 million were computed. For clarity, Fig. 5.15 shows the MIS(2)AMG preconditioner plots only. The full range of CPU runs were unable to be computed due to memory allocation issues with the linear solver library.

Both GPUs show clear improvements in execution time over the CPU single and 16-threaded simulations. Regardless of the device used, for simulations using the Jacobi preconditioner, the overall simulation runtime increases exponentially with

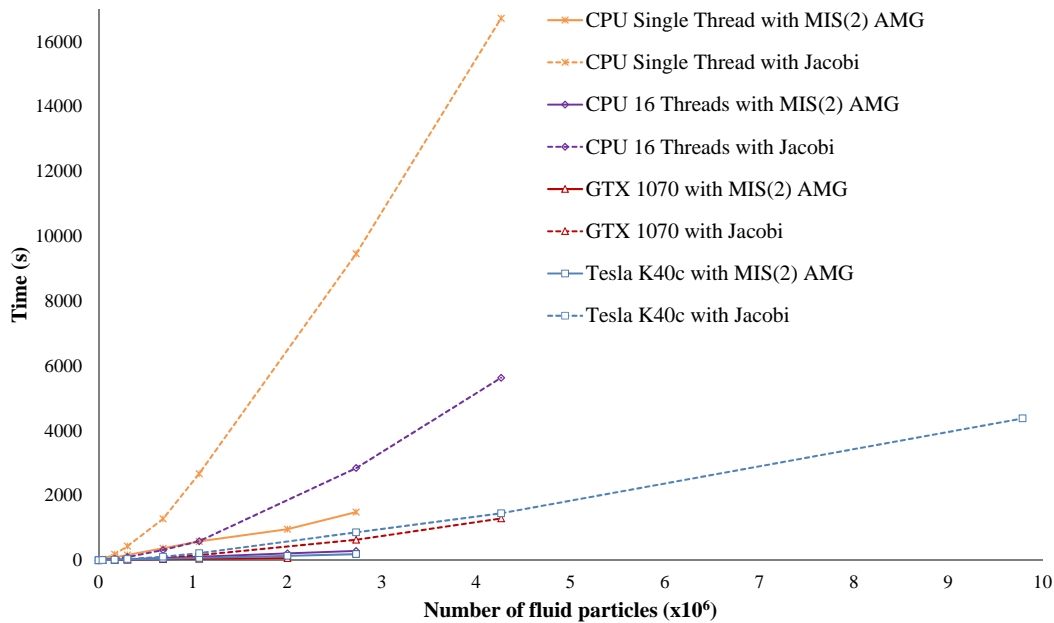


Fig. 5.14: Runtime comparisons for the first 10 time steps of the 2-D dambreak case (as in Section 5.5.1) with the Jacobi and MIS(2)AMG preconditioners.

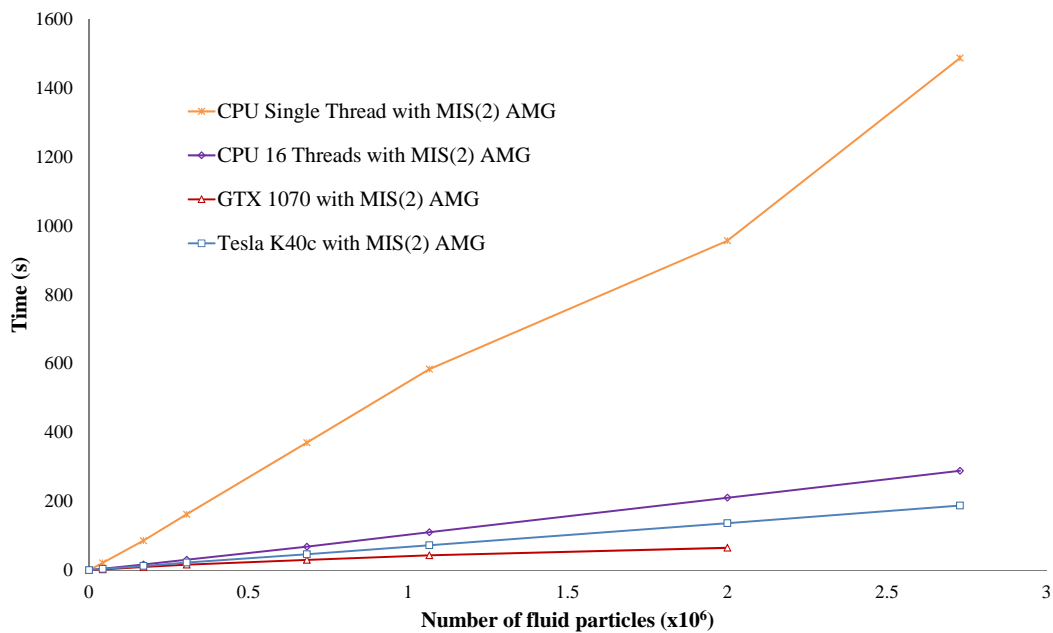


Fig. 5.15: As in Fig. 5.14 showing the MIS(2)AMG plots only.

increasing number of particles. On the other hand, for the MIS(2)AMG, the trend between runtime and number of particles appears to be near linear. Therefore, for this case the use of the AMG preconditioner results in quicker simulation times, although the number of particles are limited comparative to the Jacobi.

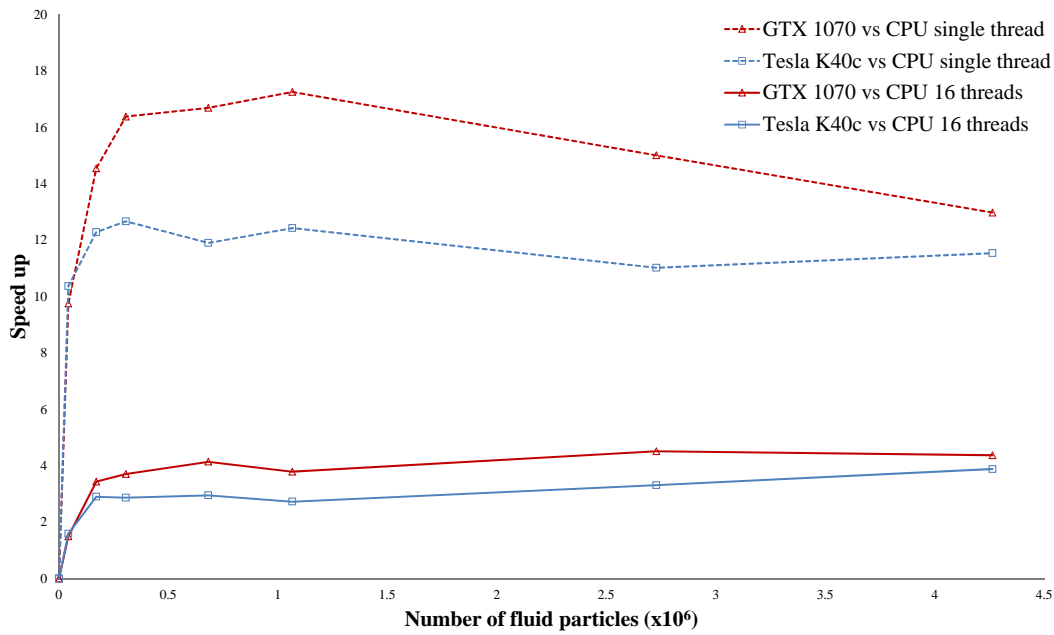
The GPU speed ups over the CPU single and 16-threaded runs are plotted in

Fig. 5.16a for the Jacobi preconditioner runtimes, and Fig. 5.16b for the MIS(2)AMG preconditioner runtimes. Simulations executed on both GPUs are performed fastest by the GTX 1070 GPU, which is aided by its higher clockspeed. Comparing simulations using the Jacobi preconditioner, it is shown GPUs can provide speed ups varying between 10-17.3 times compared to a CPU single-threaded device, and 2.7-4.5 times speed ups against CPU multi-threaded devices. The use of the MIS(2)AMG preconditioner produces lower speed ups, 6-16 times vs CPU single-threaded and 1.1-3.2 times vs CPU 16-threaded. However, it does give a significant reduction in overall runtime here, the runtime for approximately 1 million fluid particles on the GTX 1070 with the MIS(2)AMG preconditioner, compared with the CPU single-threaded run with the Jacobi preconditioner, a typical ISPH solver setup, a speed up of 61.7 times is produced.

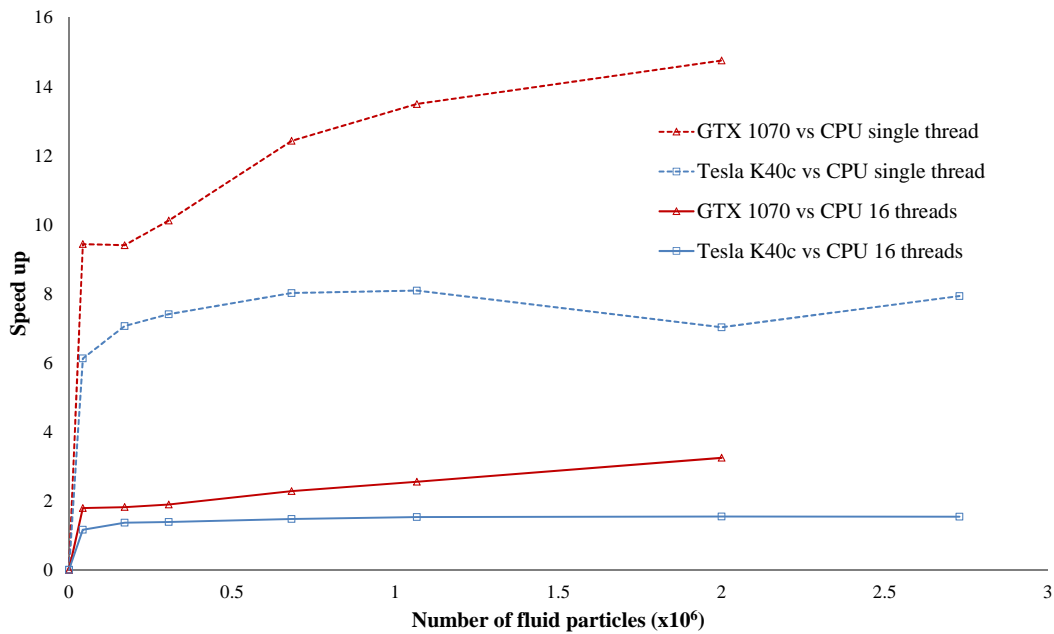
Ten-time step runtime comparisons are also made for the 3-D case (from Section 5.5.2) with different extensions of the geometry in the y -direction in Fig. 5.17. Plots for the Jacobi preconditioner only are shown in Fig. 5.18 where there is an additional plot showing runtimes performed using the quintic spline on the Tesla K40c GPU for comparison against the Wendland kernel.

For the Jacobi preconditioner runs, up to approximately 3.4 and 5.1 million fluid particles were computed on the GTX 1070 and Tesla K40c GPUs respectively. Using the quintic spline limited the Tesla K40c to a maximum of approximately 1.6 million fluid particles. For the MIS(2)AMG preconditioner, runs of up to approximately 1.0 and 1.8 million fluid particles were executed on the GTX 1070 and Tesla K40c GPUs respectively. Once again, the GTX 1070 GPU performs the best for its allowable number of particles.

Both preconditioners display near-linear trends for the runtime against the number of fluid particles where, unlike the 2-D case, the Jacobi preconditioner provides the faster solution times. For accurate representation of real engineering applications, simulations need to be in 3D. So here, the use of the Jacobi preconditioner over the MIS(2)AMG preconditioner is more favourable as it also does not require any additional memory usage.



(a) Jacobi preconditioner



(b) MIS(2)AMG preconditioner

Fig. 5.16: 2-D dambreak GPU speed ups

The quintic spline plot (solid black line) in Fig. 5.18 shows that simulations are limited to approximately 1.6 million fluid particles on the Tesla K40c GPU due to an increased kernel support radius and therefore approximately 3 times more memory usage compared to the Wendland kernel runs. Fig. 5.24a shows that using the Wendland kernel is also 2-3 times faster than the quintic spline.

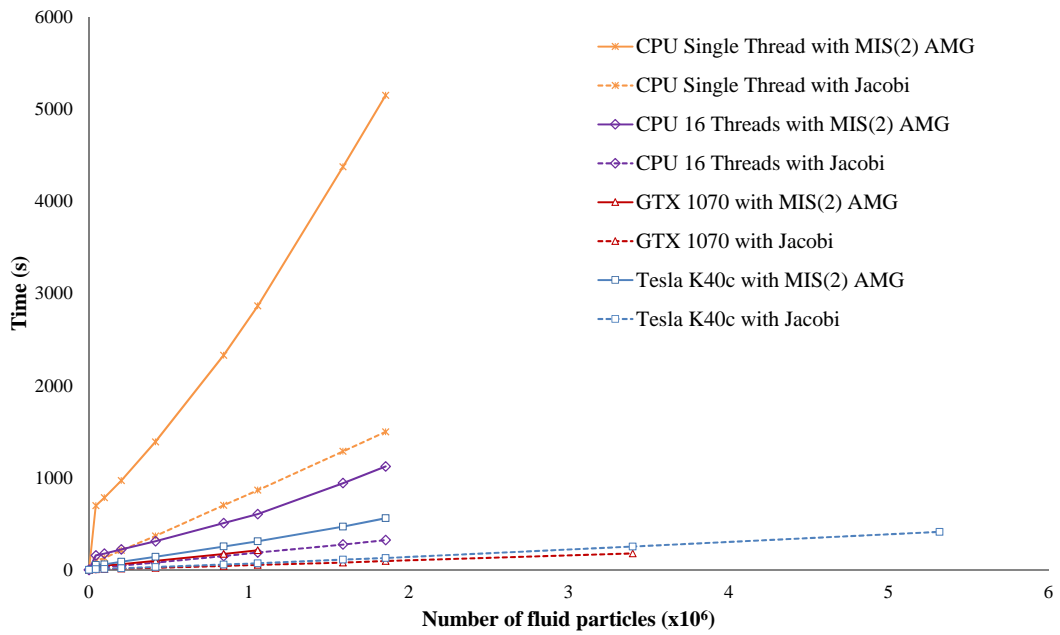


Fig. 5.17: Runtime comparisons for the first 10 time steps of the 3-D dambreak case (as in Section 5.5.2) with the Jacobi and MIS(2)AMG preconditioners.

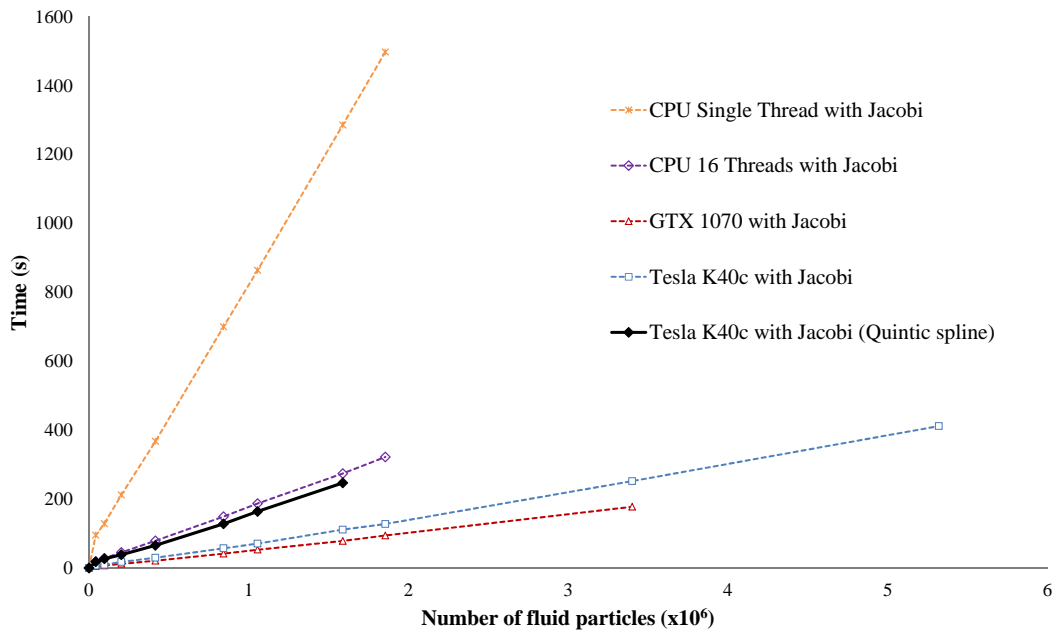
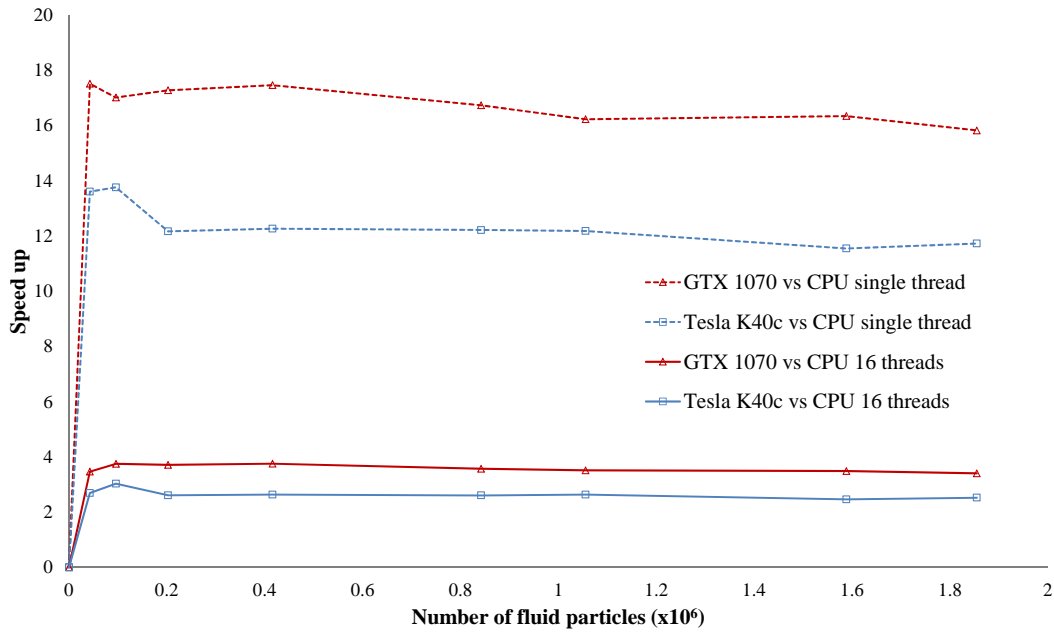


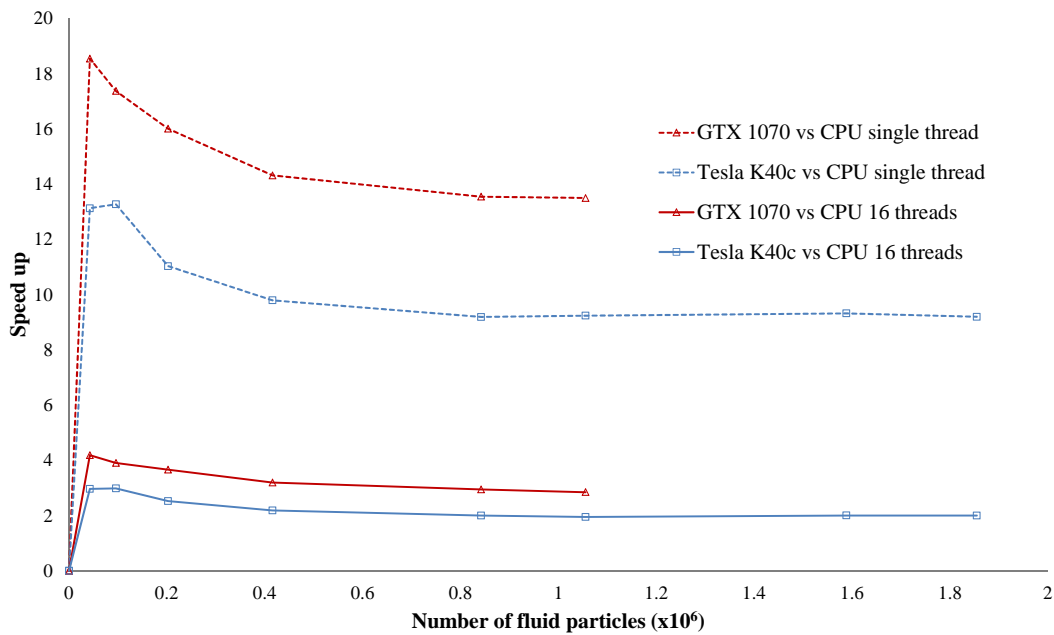
Fig. 5.18: As in Fig. 5.17 showing the Jacobi plots only, and an additional plot for runs using the quintic spline.

The GPU against CPU speed-ups for all runs, seen in Figs 5.24a and 5.24b (for Jacobi and MIS(2)AMG preconditioners respectively), reach a peak speed-up for fluid particle numbers of less than approximately 100,000 and then drop to a value with little variation for increasing numbers of fluid particles. The highest speed-ups are observed from the Jacobi preconditioner runs, the Tesla K40c is about 12

and 2.5 times faster than the CPU single and 16-threaded runs respectively, and approximately 16 and 3.5 times speed ups are observed from the GTX 1070.



(a) Jacobi preconditioner



(b) MIS(2)AMG preconditioner

Fig. 5.19: 3-D dambreak GPU speed ups

5.6.3 Preconditioner performance for ISPH

In ISPH, the performance of the linear solver will change with the evolution of the flow due to moving computational points. The runtimes presented in Fig. 5.15 should therefore only be an indicative representation and not be assumed for every test case. The 2-D dambreak was repeated for a full 1.0s of physical time (10,000 time steps) to compare the linear solver's performance throughout the simulations using the Jacobi and MIS(2)AMG preconditioners. Fig. 5.20 compares the number of solver iterations executed per time step of the simulation between the two preconditioners. The solid black line on the graph (between time steps 2,000 and 3,000) marks the instant at which the flow strikes the right-hand wall of the tank. Before the impact event, the results here correlate to those in Fig. 5.15, the MIS(2)AMG preconditioner requires less solver iterations and therefore less time to solve the PPE (and the overall time step/simulation). However, after the impact event, the number of iterations required with the MIS(2)AMG preconditioner rapidly increases and becomes highly unpredictable. The time required to setup the preconditioner was also observed to increase substantially (milliseconds to seconds). On the other hand, the number of iterations required with the Jacobi preconditioner decreases.

The change in behaviour with each preconditioner originates from fragmentation of the fluid after the impact event. Such fragmentation is displayed in Fig. 5.21a, where a snapshot of the simulation shows the entire domain. The main bulk of the fluid stays at the bottom of the tank, but there is also a body of water (along with spray) in the upper half of the domain separated from the main fluid bulk after impact with the RHS wall. Fig. 5.21b shows a close-up shot of the fragmentation encircled in Fig. 5.21a. The MIS(2)AMG preconditioner takes significantly longer to setup because it treats all of the fluid particles as one connected system, where in fact, there are at least two effectively independent systems. The linear solver then subsequently takes longer to find a solution. Such an occurrence was not observed in the dambreak simulations of Guo et al. [232] because they did not extend their domain in the z -direction. It should also be noted that the HYPRE BoomerAMG preconditioner [330] used in their work is more developed, with a

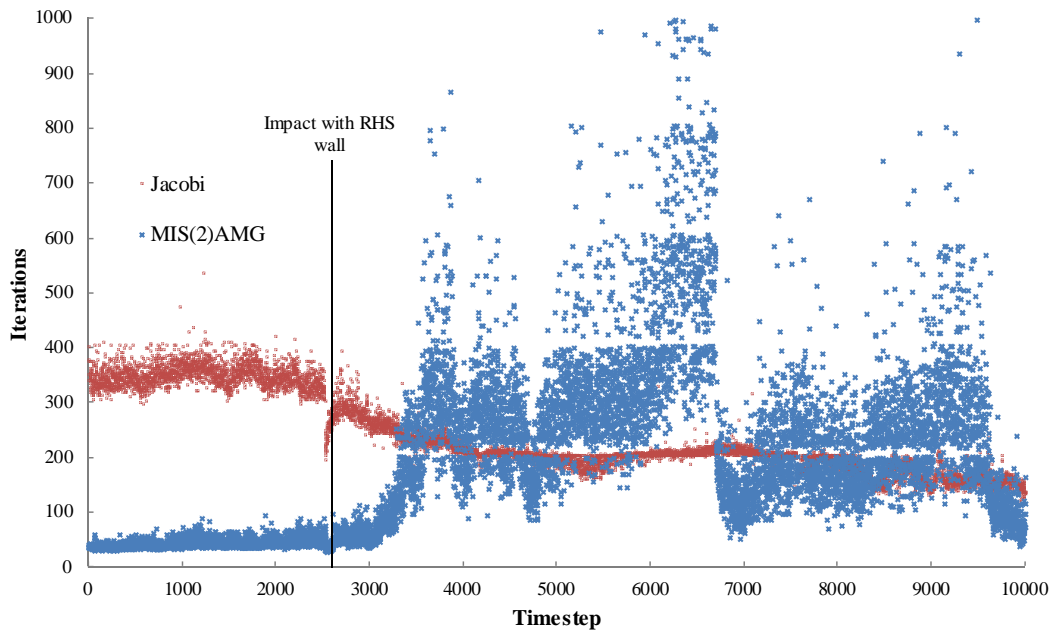


Fig. 5.20: Comparison of the number of solver iterations taken per time step during the 2-D dambreak simulation between the Jacobi and MIS(2)AMG preconditioners. Here, $dp = 10^{-3}$, resulting in 42,632 fluid particles.

greater range of functionality and robustness on CPUs than the ViennaCL library's AMG preconditioners.

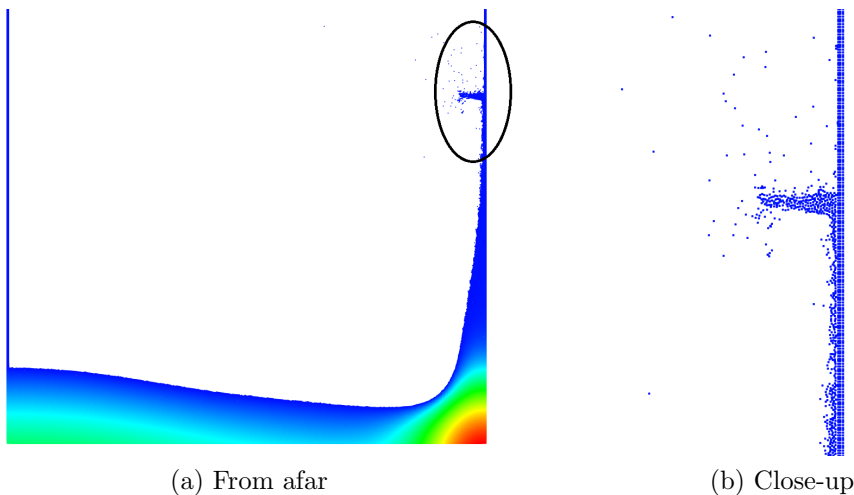


Fig. 5.21: A snapshot of the 2D dambreak simulation from afar and a close-up of the area encircled. These images show the fragmentation that occur in the flow leading to more than one linear system within the computational domain. The snapshot correlates to time step 3500 in Fig. 5.20. Multiple independent systems of fluid are more evident with higher resolutions.

Fig. 5.22 plots the cumulative time spent solving the PPE matrix per time step for the simulations in Fig. 5.20, i.e. the time spent using the ViennaCL library, with the two preconditioners. For the MIS(2)AMG, this includes the setup time of

the preconditioner. The total time spent solving the PPE matrix with the Jacobi preconditioner was 0.03 days, and when using the MIS(2)AMG, a cumulative time of 3.8 days was spent.

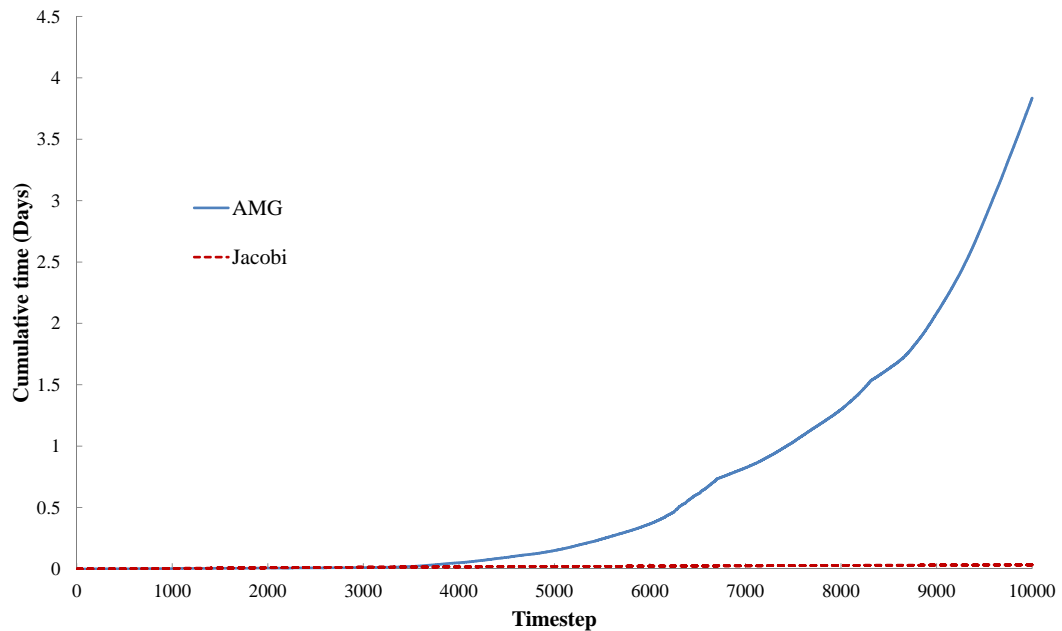


Fig. 5.22: Comparison of the cumulative time spent solving the matrix for the simulation as in Fig. 5.20.

Concluding, although the use of the Jacobi preconditioner initially gives slower solution times, its ability to deal with fragmentation ultimately results in a lower cumulative time spent solving the PPE matrix, and subsequently a faster total simulation execution time. The dambreak however, is a particularly unique flow with an extreme case of fragmentation, for the majority of 2-D flows with ISPH, the MIS(2)AMG preconditioner may be favourable in terms of execution time, although this would need to be investigated more thoroughly.

Comparisons here are for just a single case in 2D and 3D, different combinations of preconditioners and solvers will need to be investigated for a variety of cases. Moreover, algorithmic libraries for the GPU are still in their early stages relative to CPU libraries, there is still much needed development for robust and optimised linear solver algorithms for the GPU.

5.6.4 Profiling

A short profiling study is conducted to assess how well ISPH has been implemented onto the GPU. All results here are taken from the 2-D and 3-D dambreak simulations similar to those presented in Figs 5.14 and 5.17 with approximately 1 million fluid particles and using the Jacobi preconditioner.

Fig. 5.23 shows pie charts where each one presents the time spent on each stage of the pressure projection step as a percentage of the total time spent in the main ISPH loop over the first ten time steps of the dambreak case. Here, Stage 2 of the algorithm has been split up into Stage 2a, the time spent setting up and populating the PPE matrix, and Stage 2b, the time spent solving the matrix, that is the amount of time spent in the linear solver library.

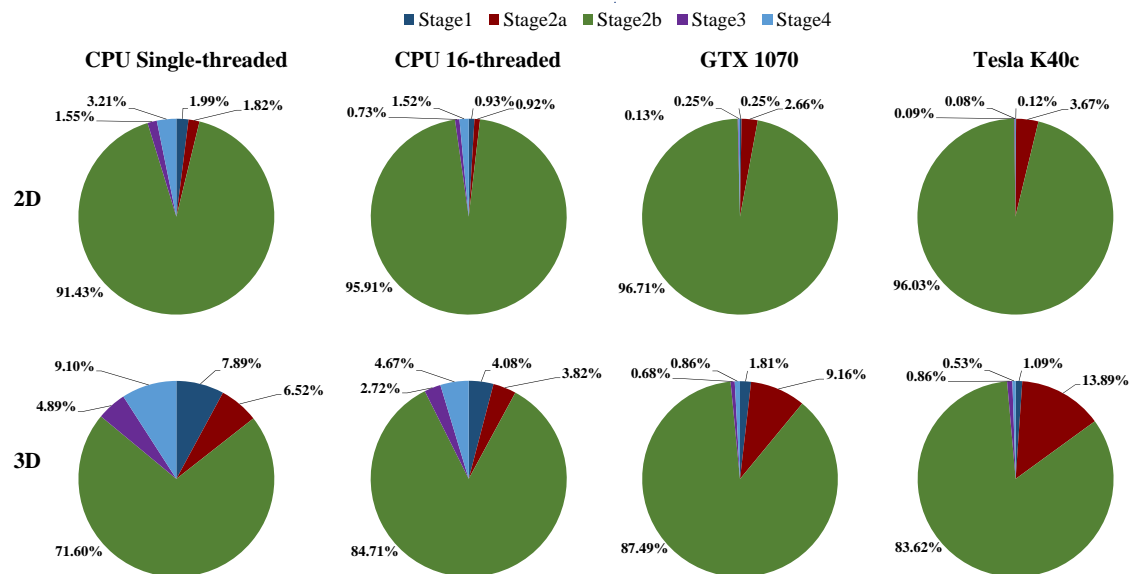


Fig. 5.23: Percentage of time spent on the 4 stages of the algorithm out of the ISPH pressure projection time step for different devices. The first 10 time steps of the of the dambreak case (Section 5.5), in 2D and 3D, are tested using approximately 1 million fluid particles and the Jacobi preconditioner. Stage 2 is split up into two parts, Stage 2a and Stage 2b, for the population and solving of the PPE matrix respectively.

It is clear that solving the PPE matrix (stage 2b) takes up the highest percentage of time in ISPH, which agrees with other researchers [190, 232]. Therefore, the majority of computational effort should be concentrated towards solving the PPE matrix, as done here. For the 2-D simulations, from CPU single-threaded to GPUs,

the percentage of time spent solving the PPE matrix increases from 91.4 % to about 96-97 %, and for 3-D simulations the increase is from 71.6 % to 83-88 %. The CPU 16-threaded runs also display similar behaviour to the GPUs, reinforcing the parallel algorithm benefits for ISPH.

The figure is also ordered such that from left to right shows increasing number of cores, suggesting higher levels of parallelism. This is further confirmed with the percentage of time spent on Stages 1, 3, and 4 decreasing with increasing numbers of cores. For example, for the 3-D simulation, the percentage of time spent on the 3 stages (1, 3, and 4) is 21.88, 11.47, 3.35, and 2.48 % for the CPU single-threaded (1 core), CPU 16-threaded (8 cores), GTX 1070 (1920 CUDA cores), Tesla K40c (2880 CUDA cores) runs respectively.

An important aspect of the algorithm that hinders the potential speed up gains from the GPU lies in Stage 2a, which accounts for the single-threaded sequential function for setting up the CSR arrays and particle sweep 2 for populating the PPE matrix in parallel. The trend between percentage of time spent on particle sweep 2 and the number of cores will be similar to that explained for stages 1, 3, and 4. However, the sequential function (Algorithm 2) relies on the clockspeed and memory hardware of the computing device. The CPU single-threaded and 16-threaded runs use the same hardware, therefore the time spent on the serial function should theoretically be exactly the same. The reduction in overall percentage of time spent in Stage 2a (6.52 to 3.82 % for the 3-D simulation) comes from a quicker execution of particle sweep 2 with the CPU-16 threads. Both GPUs show a larger percentage of time spent on Stage 2a because they have significantly lower clockspeeds (see Section 5.1) and therefore execution of the sequential function takes longer. This is a contributing factor to explain why the Tesla K40c GPU results in slower simulation times compared to the GTX 1070 GPU.

The analysis of the results from Fig. 5.23 can be further clarified with Table 5.3, which shows the speed ups achieved for each stage of the ISPH algorithm. The components of the algorithm which consistently achieve the highest GPU speed ups are Stages 1, 3, and 4. Speed ups in these stages against the CPU single-

threaded run range from approximately 139.1-502.8 times and 65.6-197.6 times in 2D and 3D respectively. When compared to the CPU 16-threaded runs, speed ups are approximately an order of magnitude lower. Such speed ups are expected in these stages as they consist of mainly particle sweeps (see Section 4.5) similar to the original DualSPHysics code [15]. As observed in Fig. 5.23, the overall potential speed of the algorithm is hindered by the single-threaded sequential function in Stage 2a, which is confirmed here in Table 5.3 where the stage exhibits the lowest GPU speed ups (ranging from 0.7-11.8 times) in the whole algorithm. Lower speed ups here achieved by the Tesla K40c confirms the single-threaded serial function relies upon the clockspeed of the hardware. Parallelising the single-threaded serial function will allow the speeds shown for Stage 2a to be more similar to the other particle sweep stages (1, 3, and 4). The GPUs achieve speed ups in the range of 2.6-16.3 times for the solution of the PPE matrix in Stage 2b. This is an order of magnitude lower than the particle sweep stages, and thus hinders the overall speed ups of the ISPH algorithm previously shown in Figs 5.16 and 5.19. Concentrating research efforts towards solving the PPE matrix more efficiently on the GPU will gain the most improvements in overall simulation speed up and computation time. This will be demonstrated in Section 5.6.5.

Table 5.3: GPU speed ups for each stage of algorithm with approximately 1 million particles in 2D and 3D. Results are obtained from the same data used Fig. 5.23. CPU (1) and CPU (16) refer to CPU single and 16-threaded respectively.

		Stage 1	Stage 2a	Stage 2b	Stage 3	Stage 4
2D	GTX 1070 vs CPU (1)	139.1	11.8	16.3	198.3	222.2
	Tesla K40c vs CPU (1)	205.4	6.2	11.8	203.8	502.8
	GTX 1070 vs CPU (16)	14.3	1.3	3.8	20.4	23.1
	Tesla K40c vs CPU (16)	21.2	0.7	2.7	21.0	52.2
3D	GTX 1070 vs CPU (1)	66.9	10.9	12.5	110.7	162.8
	Tesla K40c vs CPU (1)	83.5	5.4	9.9	65.6	197.6
	GTX 1070 vs CPU (16)	7.6	1.4	3.3	13.6	18.4
	Tesla K40c vs CPU (16)	9.5	0.7	2.6	8.0	22.3

Table 5.4 shows statistics of the GPU implementation for the CUDA kernels executed for each particle sweep in the ISPH pressure projection algorithm (Fig. 4.10) in the first time step of the simulation. The functions are named in the format of “Function-Particle type” such that “PS” stands for particle sweep, “MLS” is the

function for determining the MLS coefficients for each boundary particles unique interpolation point (Eq. (3.46)), and “F” and “B” are i particles of fluid and boundary particle type respectively. Four measures are presented here:

- **Theoretical Occupancy:** A measure of the upper limit for the average percentage of warps active during the kernel execution derived from the compilation of the code and the GPU device capabilities. A “warp” is a block of 32 threads, if at least one thread in a warp is being used, then the warp is defined to be active. Only very simple CUDA kernels with low register (local memory) usage can achieve 100 % occupancy. Here the theoretical occupancies range from 25 to 50 %, values similar to the force computation kernels from the original WCSPH DualSPHysics code when the same problem was run.

The original DualSPHysics code used the same particle sweep function for the predictor and corrector phase (see Fig. 4.9), and this function was modified to compute PS1-F and PS3-F during the ISPH conversion process with a conditional if-statement to separate the two different computations. Therefore, more registers than necessary are used for both PS1-F and PS3-F, which results in a reduced theoretical occupancy. However, the inefficiency does not affect the overall computation time greatly as seen from Fig. 5.23. Implementing the two as separate dedicated kernel functions will increase the theoretical occupancy and efficiency of the algorithm.

- **Achieved Occupancy:** The measured percentage of warps active during the kernel execution. All kernel functions here achieve a value the same as or very close to the theoretical occupancy. This means, the code has been implemented such that the device can hide latency between computations. Therefore, lower register usage and so higher theoretical and achieved occupancies have been identified as a method to increase performance within each kernel for the same ISPH methodology.
- **Branch divergence:** A branch instruction is defined as an instruction with a conditional result where the outcome may vary. The branch divergence

is therefore defined as the average percentage of branch instructions executed within a warp of threads which resulted in different code paths for participating threads in the same warp. Branch divergence introduces latency and can have a significant impact in the context of GPUs, therefore it is ideally avoided. Here, the branch divergence has similar values to the WCSPH DualSPHysics run for the same problem [248].

- **Computational throughput (single and double precision):** The computational throughput is an important measure of performance, specifying the number of Giga-floating-point operations per second (GFLOP/s) computed kernel. The single precision computational throughputs of kernels range from 73.3-522.6 GFLOP/s. The theoretical single-precision peak performance of the GTX 1070 GPU is 6,463 GFLOP/s, which means each kernel achieves less than 10% GPU efficiency. However, the low value of efficiency is due to the use of mixed precision computations in the algorithm and is thus an unfair metric here. For the double precision computational throughputs of kernels, a range of 15.3-49.0 GFLOP/s are achieved. Compared to the GPU's theoretical double-precision peak performance of 202 GFLOP/s, double precision throughput efficiencies range from 7.6-24.2%. The highest throughput efficiencies are usually achieved through small and simple CUDA kernels [236]. The CUDA kernels here for computing particle sweeps are larger and more complex comparative to other GPU applications (e.g. matrices manipulation). Reassessing the implementation of particle sweep CUDA kernels to increase efficiency is an area of further study, however as mentioned before, to improve the overall simulation time, efforts should be concentrated more towards solving the matrix efficiently.

In the first time step of the simulation from Table 5.4, the total floating-point operation count (FLOP count) across all particle sweeps was 43.14×10^9 for single-precision operations and 6.85×10^9 for double precision operations. The integer math operation count was recorded to be 18.35×10^9 . A total of 2.17 GB of RAM

Table 5.4: Theoretical and achieved occupancy and branch divergence percentages, and single and double precision computational throughput for the CUDA kernels (parallel GPU functions) executed for each particle sweep in the algorithm. Results are taken from the first time step of the 3-D dambreak simulation (see Section 5.5.2) and performed on the GTX 1070 GPU.

Function	Stage 1			Stage 2		Stage 3	Stage 4
	MLS-B	PS1-B	PS1-F	PS2-B	PS2-F	PS3-F	PS4-F
Theoretical occupancy (%)	37.5	50.0	25.0	50.0	37.5	25.0	50.0
Achieved occupancy (%)	37.4	49.9	25.0	50.0	37.5	25.0	50.0
Branch divergence (%)	1.58	1.22	0.29	1.48	0.29	0.38	0.26
Single precision throughput (GFLOP/s) [Efficiency (%)]	73.3 [1.1]	233.3 [3.6]	149.4 [2.3]	77.0 [1.2]	94.0 [1.5]	256.6 [4.0]	522.6 [8.1]
Double precision throughput (GFLOP/s) [Efficiency (%)]	43.6 [21.6]	35.1 [17.4]	31.2 [15.5]	16.6 [8.2]	15.3 [7.6]	42.9 [21.2]	49.0 [24.2]

was allocated to the GPU at the beginning of the simulation by DualSPHysics, this excludes memory allocated by the linear solver library.

5.6.5 Solver initialisation

In the preceding sections of this chapter, all simulation results are based upon using an initial zero solution, $\|\mathbf{x}_0\|_2 = 0$, for the Bi-CGSTAB linear solver at each time step. This section shows that by initialising the solver with an approximate solution guess, the required computational time can be reduced. Here, simulations with solver initialisation uses the pressure field obtained from the previous time step as an initial solution guess. At the first time step, where there is no available preceding pressure field, $\|\mathbf{x}_0\|_2 = 0$.

The 2-D and 3-D dambreak runtime comparisons for the GTX 1070 GPU and single and 16-threaded CPU experiments, from Section 5.6.2, are repeated with the use of solver initialisation. Only runtimes with the Jacobi preconditioner are considered here.

Fig. 5.24 compares the results of the runtimes using solver initialisation (SI),

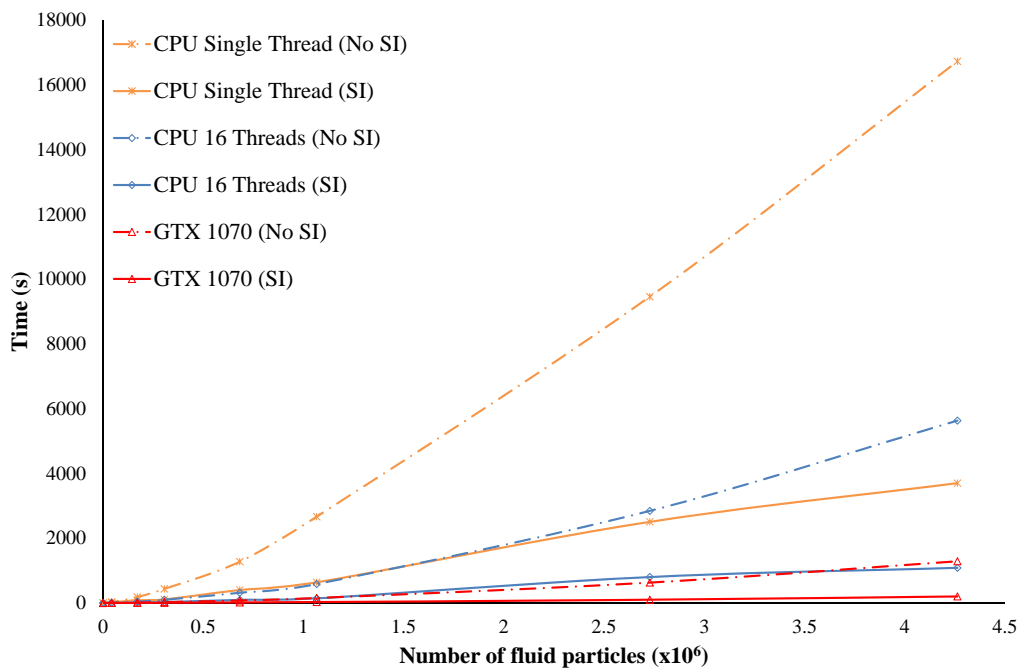
denoted as “SI” in the plots, against those without (No SI), from Section 5.6.2. For all experiments, in 2D and 3D, initialising the solver provides reductions in overall runtimes. For the previous 2-D experiments, the Jacobi preconditioner showed an exponential-type increase in runtimes with increasing particle numbers. However, with the use of solver initialisation, the trend now appears to scale linearly. The 3-D cases show linear scalability for both with and without solver initialisation.

The use of solver initialisation supposedly reduces the time taken to solve the PPE matrix, and subsequently overall runtimes, because it provides a more suitable approximation to the system solution than $\|\mathbf{x}_0\|_2 = 0$. Therefore, the linear solver requires less iterations (and computational time) to reach convergence. This is based on the assumption that the overall pressure field from one time step is similar to the next.

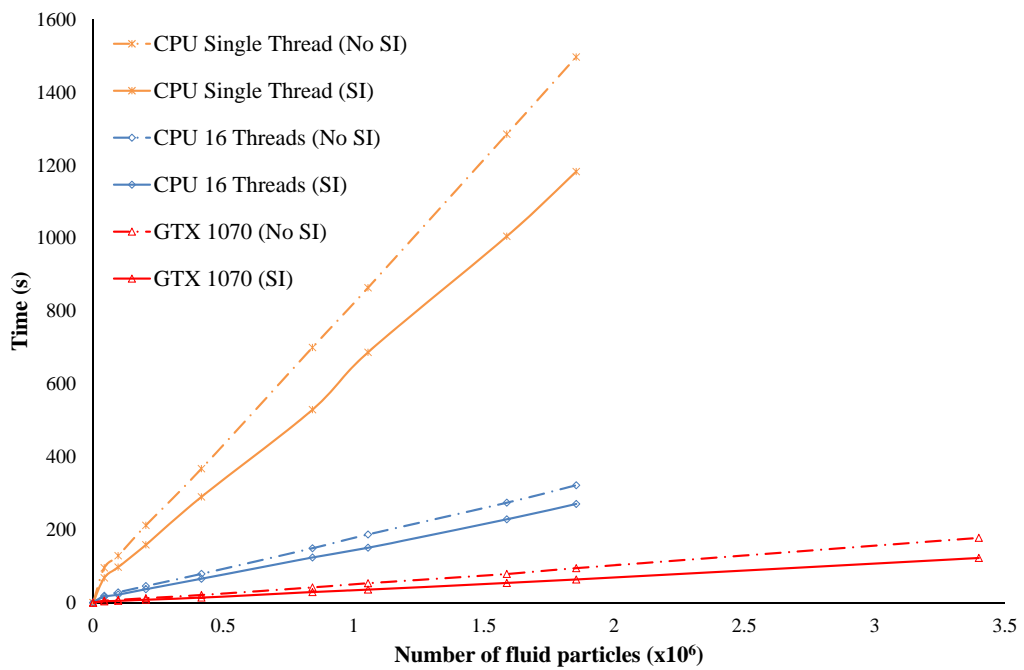
The reduction of runtimes, as a percentage, gained by solver initialisation are plotted in Fig. 5.25. In both 2D and 3D, the GPU benefits more from solver initialisation than the CPU with a single or 16-threaded implementation. In 2D, for fluid particles numbers of over 200,000, the overall runtimes for the GPU are reduced by approximately 80-85% of the original runtimes without solver initialisation. The single and 16-threaded CPU runtimes are improved by 70-80%.

For the 3-D cases, percentage reductions are lower, as the Jacobi preconditioner already scales linearly without initialising the solver solution. Runtime reductions of 30-35% are observed for the GPU, compared to 20-30% and 15-22% for the single and 16-threaded CPU experiments respectively.

The higher reductions in overall runtimes for the GPU, compared to those of the CPU, subsequently improves the speed ups observed between the two hardware as shown in Fig. 5.26. In 2D, the GPU-CPU single thread speed ups increase from 13.0-17.3 times to 18.3-25.3 times for particle numbers of 1-4.5 million. Similarly, GPU-CPU 16-threaded speed ups with solver initialisation range 4.6-8.1 times compared to the original 3.8-4.5 times. In 3D, for up to 2 million particles, it is shown that the use of solver initialisation improves the GPU speed ups from 15.8-17.5 times to 18.2-21.0 times and 3.5-3.8 times to 4.2-4.7 times against the CPU single and



(a) 2-D runtimes



(b) 3-D runtimes

Fig. 5.24: Dambreak runtimes using solver initialisation (SI) and comparing to results obtained from Figs 5.14 and 5.17 without the use of SI (No SI).

16-threaded runtimes respectively.

Fig. 5.27 looks at the percentage of time spent in the different stages of the algorithm, as in Fig. 5.23, for the GTX 1070 GPU and compares the difference between with and without solver initialisation. The measured time spent in all

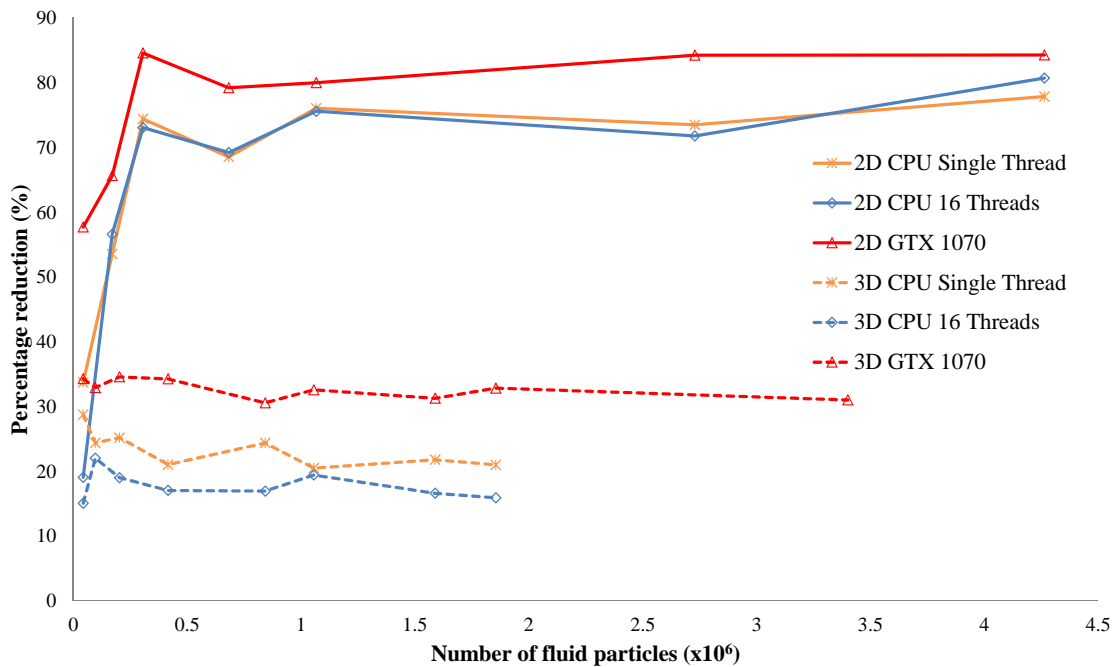
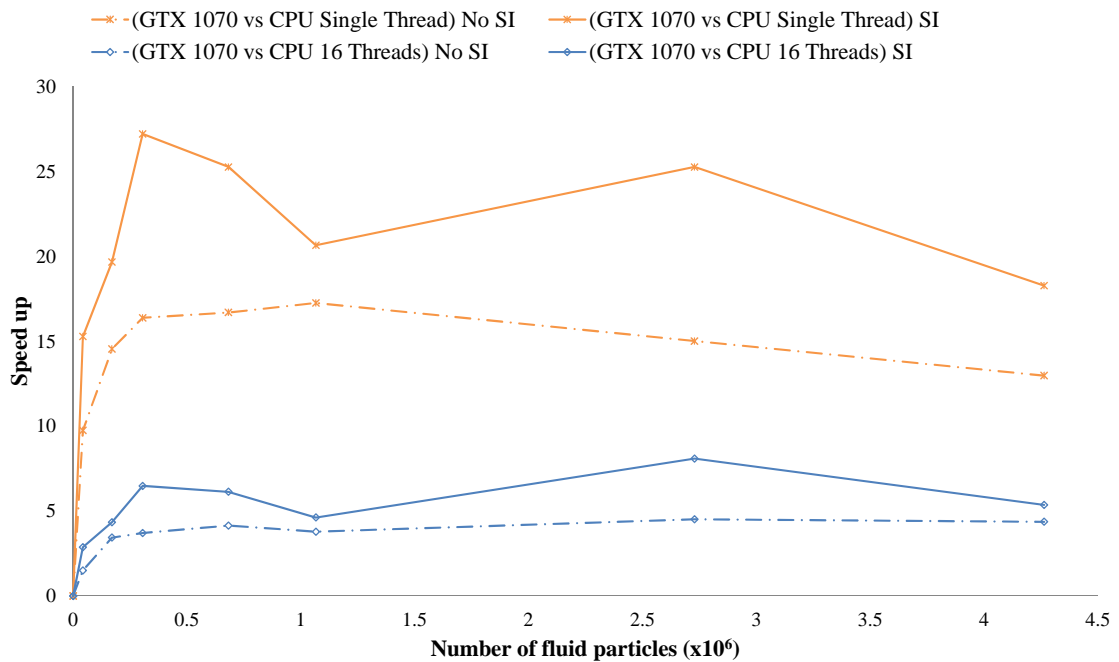


Fig. 5.25: Runtime reductions, as a percentage, achieved by the use of solver initialisation.

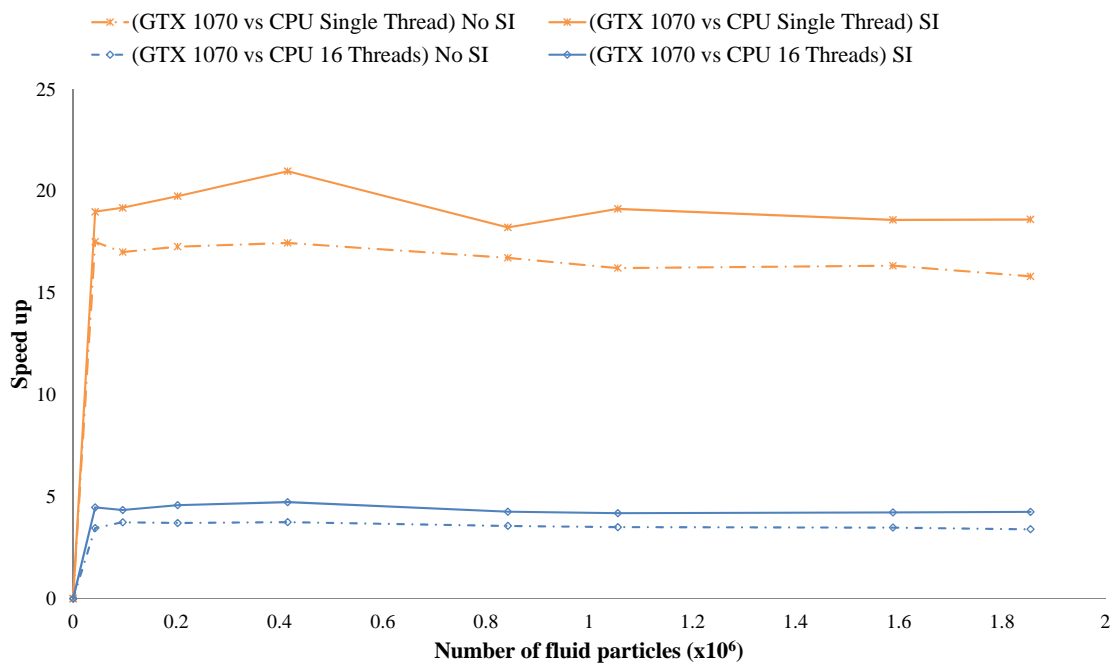
stages other than solving the PPE matrix, i.e. Stages 1, 2a, 3, and 4, are similar between the runs whether solver initialisation is used or not. Overall simulation-time reductions are made within Stage 2b only, where the percentage of time spent solving the PPE is now 83.81% in 2D (from 96.71%) and in 3D, the percentage is reduced from 97.49% to 80.88%. As established in Section 5.6.4, the processing power of the GPU is well utilised as the majority of its computational time is spent solving the PPE in Stage 2b. However, here the reduction in time spent in that stage is an indication of improvement to the numerical methodology or algorithm. As numerical methods for solving the ISPH PPE advance, the evolution of the pie chart should see the percentage reduce further.

5.7 Conclusions

The performance and accuracy of the new GPU-accelerated ISPH code, Incompressible-DualSPHysics, has been investigated in this chapter. The code's accuracy, robustness, and stability has been proven through a variety of test cases: an impulsively started plate, 2-D incompressible flow around a moving square in a rectangular box,



(a) 2-D Speed ups



(b) 3-D Speed ups

Fig. 5.26: New GPU-CPU speed ups with use of solver initialisation (SI) and comparing to results obtained from Figs 5.14 and 5.26b without the use of SI (No SI).

and 2-D and 3-D dambreaks. A series of runtime comparisons show, for particles numbers of between 1-4.5 million in 2D, GPU (GTX 1070) speed-ups of 18.3-25.3 times and 4.6-8.1 times against single-threaded and 16-threaded CPU runtimes respectively. In 3D for up to 2 million particles, GPU-CPU speed ups of approximately

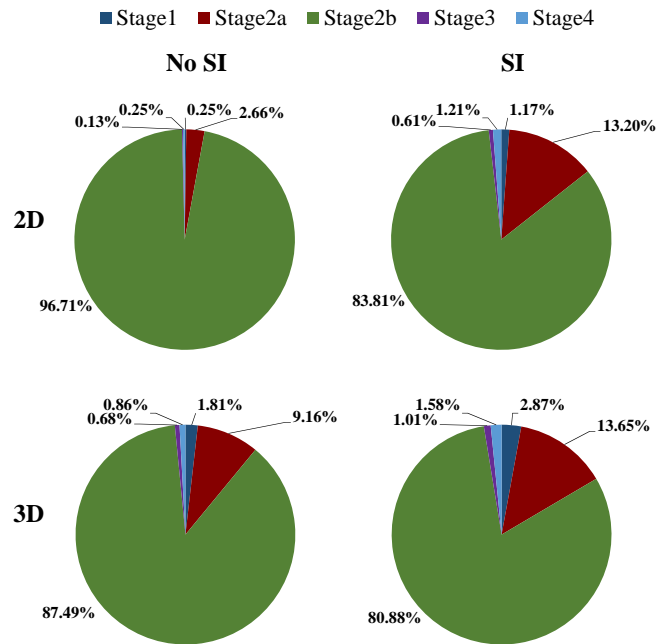


Fig. 5.27: Percentage of time spent on the 4 stages of the ISPH pressure projection step algorithm on the GTX 1070 GPU using solver initialisation (SI) compared to results obtained from Fig. 5.23 without use of solver initialisation (No SI).

18.2-21.0 times are achieved compared to the CPU single-threaded runtimes, and 4.2-4.7 times versus the CPU 16-threaded experiments. A profiling study shows the efficiency of the WCSPH DualSPHysics code has been maintained throughout its conversion to an ISPH algorithm.

Performance investigations concerning different components of the methodology were also made:

- **Kernel:** On comparison of the quintic spline with the Wendland kernel for the 3-D dambreak simulation, both kernels gave near identical results for the propagation of the toe front. However, the latter provides the advantages of lower memory consumption and quicker solution times.
- **Preconditioner:** The Jacobi and MIS(2)AMG preconditioners were also compared where the former proves favourable showing superior speed solution times and an apparent robustness for highly fragmented flows. The MIS(2)AMG preconditioner has potential but the algorithm requires more development for ISPH.
- **Solver initialisation:** When using the pressure field of the previous time

step for the linear solvers initial solution guess, runtimes were reduced by up to 85% and 35% for 2-D and 3-D simulations respectively.

The penultimate chapter of this thesis fulfills the aim of the project by demonstration of the ISPH on the GPU code for a real violent hydrodynamic engineering application: breaking wave-structure interaction.

Chapter 6

Application of ISPH to a Numerical Wave Basin

6.1 Introduction

This penultimate chapter demonstrates how the new Incompressible-DualSPHysics code, accelerated with a GPU, can simulate fully 3-D engineering applications involving violent free-surface flows.

Offshore environments are amongst some of the most aggressive that man-made structures must experience. Ships, wind turbines, and offshore platforms are subject to violent hydrodynamics including extreme waves, wave-breaking, and greenwater overtopping and slamming. For engineers, the design of such applications through laboratory experiments is limited. So the desire for the development of a numerical wave basin has long been sought after as an engineering tool to simulate complex wave-structure interactions. Such a concept has become increasingly realisable with recent advancements of computational power involving parallel processing and improving numerical schemes and algorithms.

The intention is that a numerical wave basin code may simulate all nonlinear effects associated with steep, possibly breaking, waves on bodies which may be fixed or dynamically responding. Ideally the simulations would be two-phase for slam effects where the influence of compressible air and aeration may be significant [198, 331]. In this study however, simulations are restricted to a single-phase (water). Numerical models of particular note include the volume-of-fluid code OpenFOAM [1, 66],

the weakly-compressible smoothed particle hydrodynamics (WCSPH) codes DualSPHysics [15] and GPU-SPH [242, 263] and the mixed finite-volume/particle code PICIN [332]. As previously mentioned in Section 2.4.1, finite-volume methods show disadvantages with mass dissipation at the free surface and the need for a mesh that adapts to the body shape which may be moving, requiring an ALE-type approach or overset meshing. Even then, flows of a fragmented nature such as breaking-waves, cannot be dealt with effectively. The PICIN solver method [332] uses a hybrid Eulerian-Lagrangian approach with a body-fitted mesh and moving interpolation points (particles), offering a higher degree of flexibility, but at the expense of some complex interpolation procedures. Fragmentation of the fluid domain is possible with the PICIN method, however breaking-waves are yet to be demonstrated.

Section 2.5 has already highlighted some successes of SPH for violent hydrodynamic free-surface flow applications demonstrating that the method is well-suited for the simulation of breaking waves. Unfortunately, the large number of particles required for the large domain of a 3-D numerical wave basin can significantly increase the already high computational expense of the method. There are some hybrid methods which address this expense, where a near-field SPH model is coupled with either, a finite-volume [333, 334], or highly efficient nonlinear potential flow solver in the outer domain [200].

The new Incompressible-DualSPHysics code developed from this study addresses the expense through GPU-acceleration and, in this chapter, used to model a 3-D numerical wave basin for simulating breaking-wave impacts with vertical cylinder structures. The advantage over successful GPU-accelerated WCSPH codes [15, 223, 263] is the near noise-free pressure field of ISPH. It will be seen in Section 6.3 that a single GPU is capable of simulating over 5 million particles in a reasonable time. However, further development to the methodology described in Section 3 is required to maintain the accuracy at the free surface for long-duration flows such as wave propagation.

This chapter is structured as follows: in Section 6.2, the numerical wave basin methodology is presented in three subsections describing new extensions to the

methodology from Chapter 3 for increased accuracy of wave-structure impact, the numerical domain of the test cases presented, and the wave generation model. For the numerical wave basin, focused wave groups are simulated because they allow for the direct generation of steep waves responsible for peak loadings on a structure, eliminating the need to simulate a random sea state for long periods of physical time. Section 6.3 then presents the results of focused wave groups impacting a cylinder compared to the experimental data of Zang et al. [20], examining the role of hydrostatic and non-hydrostatic pressures to the force exerted on the cylinder. Finally, Section 6.4 draws conclusions on the work conducted.

6.2 A numerical wave basin

6.2.1 Extensions to the methodology

For development of a numerical wave basin, changes in parameters and extensions to the ISPH methodology presented in Chapter 3 are required for the application of wave-structure impacts. A summary of these changes is listed as follows:

- As a result of the investigations made in Chapter 5, the Wendland kernel (Eq. (3.9)) with a h/dp ratio of 1.3 is used for all simulations in this chapter. The kernel provides similar accuracy for a lower computational expense, in terms of both time and memory, compared to the quintic spline (Eq. (3.8)). The GPU memory assigned for the storage of the PPE matrix is $Nnz_{max} \times Np$ where for the case of wave propagation, $Nnz_{max} \approx 1.0N_{n,max} = 80$ (see Section 4.7).
- Free-surface particles are identified with a divergence of position value, $\nabla \cdot \mathbf{r}$ as in Eq. (3.37), less than 1.5 in 2D, or 2.5 in 3D. The near-free-surface smoothing criterion of Skillen et al. [9], as in Section 3.3.4, is not applied to the PPE here. Numerical experiments revealed the shifting methodology of Skillen et al. [9] is unable to maintain an accurate free-surface for long durations of physical time in slow-flow regions.

- The PPE matrix is solved using the Bi-CGSTAB iterative solver with a Jacobi preconditioner and a solver tolerance of 10^{-6} . The initial solution guess of the solver for the first time step is set to $\|\mathbf{x}\|_2 = 0$. The linear solver solution from there on is initialised with the pressure field from the previous time step which can significantly reduce overall runtimes by up to 35% in 3D (see Section 5.6.5).
- Additional correction terms are added to the Laplacian Morris operator [157] (Eq. (3.36)) for use of the Schwaiger operator [156] to improve accuracy at near-free-surface regions. This is explained further in Section 6.2.1.1.
- As detailed later, in Section 6.2.1.2, a kernel-weighted normal is used when shifting particles near the free-surface to reduce error propagation during wave simulations.
- A variable time step size is employed, as described in Section 6.2.1.3, to reduce overall numerical error growth.
- Two extensions to the boundary conditions are made: (i) boundary particles above the free-surface in any given time step are excluded from kernel summations to improve the accuracy at the interface between the fluid's free-surface and solid boundary, and (ii) instead of the Cartesian-positioned boundary particles generated by GenCase in pre-processing (see Section 4.4.1), a radial arrangement of particles is adopted for improved fluid-structure interaction of waves against a vertical cylindrical structure. Further details of these extensions are given in Section 6.2.1.4.

6.2.1.1 Laplacian operator

Lind et al. [11] showed the Schwaiger operator possessed significant reductions in error near the free surface, when compared to the commonly used Morris operator, and is therefore advantageous for modelling long duration flows such as wave propagation. Therefore, instead of the Morris operator as described in Section 3.3.3, the Schwaiger operator [156] is used to approximate the Laplacian terms during the

computation of the acceleration due to viscous forces (Eq. (3.28)) and LHS of the PPE (Eq. (3.30)):

$$(\phi_1 \nabla^2 \phi_2)_i = \frac{\text{tr}(\mathbf{\Gamma})^{-1}}{n_d} \left\{ \sum_j V_j (\phi_{1,j} + \phi_{1,i}) (\phi_{2,j} - \phi_{\phi,i}) \frac{\mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{r_{ij}^2} - [\nabla(\phi_{1,i} \phi_{2,i}) - \phi_{2,i} \nabla \phi_{1,i} + \phi_{1,i} \nabla \phi_{2,i}] \cdot \left(\sum_j V_j \nabla_i \omega_{ij} \right) \right\}, \quad (6.1)$$

where ϕ_1 and ϕ_2 are variables, n_d is the number of dimensions, and $\mathbf{\Gamma}$ is a tensor defined as

$$\mathbf{\Gamma}_{\psi_1 \psi_2 \psi_3} = \sum \frac{\mathbf{r}_{ij} \cdot \nabla_i \omega_{ij}}{r_{ij}^2} \Delta \chi_{\psi_1} \Delta \chi_{\psi_2} \Delta \chi_{\psi_3}, \quad (6.2)$$

such that χ_{ψ_1} , χ_{ψ_2} , and χ_{ψ_3} stand for coordinate directions.

The Schwaiger operator is simply implemented into the existing code with an additional GPU-kernel for the computation of the acceleration due to viscous forces, and an extra computation term during the population of the PPE matrix. The first term within the curly brackets on the RHS of Eq. (6.1) is equivalent to the Morris operator, therefore the remaining terms can be included as an extra computation step.

The sum of kernel gradients, $\sum_j V_j \nabla_i \omega_{ij}$, requires there to be an extra particle sweep to the algorithm depicted in Fig. 4.10, but the impact on the overall computation time is relatively low compared to the solution of the PPE matrix as determined from the profiling study in Section 5.6.4.

6.2.1.2 Particle shifting

There are two differences to the shifting technique presented in Section 3.3.5:

- Since the Wendland kernel is used here, a tensile instability term of Eq. (3.42) is not required [190, 283].
- When shifting particles near the free surface, Khayyer et al. [184] confirmed the need for corrected free-surface normal vectors. Here, a kernel-weighted normal is employed as used by Lind et al. [191]. Numerical experiments showed that,

for wave propagation, the weighting helps reduce error propagation at the free surface. The kernel-weighted normal to the free surface, $\hat{\mathbf{n}}_{s-shift,i}$, of a particle can be found by:

$$\hat{\mathbf{n}}_{s-shift,i} = \frac{\mathbf{n}_{s-shift,i}}{|\mathbf{n}_{s-shift,i}|}, \text{ where } \mathbf{n}_{s-shift,i} = \sum_j V_j(-\nabla C_j)\omega_{ij}, \quad (6.3)$$

such that the concentration gradient, ∇C_i , of a particle is equal to the sum of the kernel gradient, $\sum_j \nabla \omega_{ij}$. The tangential direction to the free surface is subsequently computed from the new smoothed normal direction. The shifting distance of particles near the free surface (evaluated as $1.5 \leq \nabla \cdot \mathbf{r} \leq 1.7$ in 2D, or $2.5 \leq \nabla \cdot \mathbf{r} \leq 2.7$ in 3D) are restricted by elimination of shifting entirely in the kernel-weighted normal direction to the free surface.

The process requires an additional particle sweep for the kernel summation. However, as explained in Section 6.2.1.1, the added computational time is relatively low compared to the overall simulation time.

6.2.1.3 Variable time step size

A variable time step size based upon the CFL condition is employed to reduce overall simulation numerical-error growth. The largest particle velocity at each time step, $u_{max,n}$ is used to dictate the following time step size, Δt_{n+1} :

$$\Delta t_{n+1} = C_{CFL} \frac{h}{u_{max,n}}, \quad (6.4)$$

where C_{CFL} is a constant equal to 0.2 for all simulations. The initial time step size, $\Delta t_0 = C_{CFL}h$, at time, $t = 0$, for all simulations.

Nvidia's CUDA "parallel reduction" algorithm is used to obtain the maximum of an array of particle velocities for $u_{max,n}$.

6.2.1.4 Boundary conditions

For the rectangular numerical wave basin with a vertical cylinder, as described later in Section 6.2.2, Eq. (3.47) is used to impose no-slip boundary conditions on the

surface of the tank bottom, cylinder and far right-hand wall, and free-slip conditions on the side walls, down the length of the tank, and the piston wavemaker.

Two additions to the Marrone et al. boundary condition described in Section 3.4 are presented here for improved accuracy at the free and solid surface interfaces during wave-structure impact:

Modification 1: The first improvement is the exclusion of unnecessary boundary particles from the computation in each time step. Fig. 6.1 shows the boundary particles included in a time step for a particular fluid domain with and without the improvement. Fig. 6.1a shows the original configuration where, near the free surface, fluid particles will interact with boundary particles above the mirrored free-surface line, which results in inaccuracies in the kernel summations. Fig. 6.1b shows the improved configuration where certain boundary particles are excluded and the shape of the fluid domain is more accurately mirrored within the boundary particle region. The treatment has also been carried out by Bouscasse [286], however their determination of such particles for exclusion is computed differently.

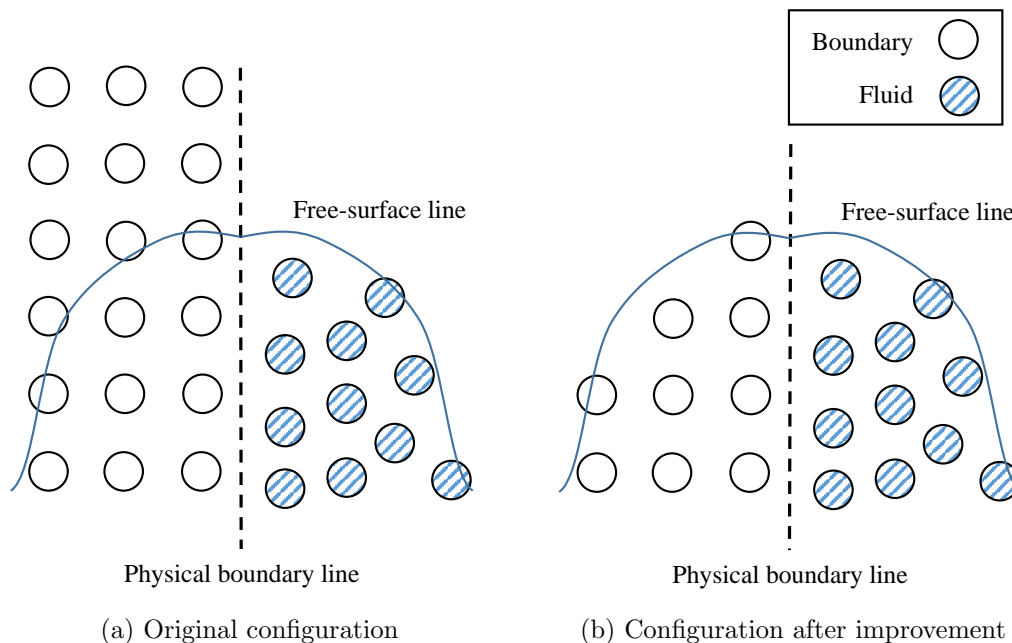


Fig. 6.1: Improving the boundary condition by exclusion of boundary particles above the free-surface line for each time step.

Excluded boundary particles are determined by evaluation of their $\nabla \cdot \mathbf{r}$ value at unique interpolation points (UIPs). The computation fits into the projection step,

between Eqs (3.27) and (3.28), as follows:

1. After computing Eq. (3.27), execute a particle sweep for each boundary particle's unique interpolation point, i , which will calculate: MLS interpolation variables using neighbouring fluid particles only (this is no different to the methodology explained in Section 4.5), and $\nabla \cdot \mathbf{r}_i$ using both neighbouring fluid and boundary particles.
2. Any unique interpolation point whose value of $\nabla \cdot \mathbf{r}_i$ less than 1.0 in 2D, or 1.5 in 3D, is deemed to be outside of the fluid domain. The associated boundary particles will now be referred to as “excluded” particles and denoted by the set \mathbb{E} . A second particle sweep takes place for the unique interpolation points of the non-excluded boundary particles, with neighbouring boundary particles only, to correct $\nabla \cdot \mathbf{r}_i$ as:

$$\nabla \cdot \mathbf{r}_i = (\nabla \cdot \mathbf{r}_i)_0 - \sum_{j \in \mathbb{E}} \frac{m_j}{\rho_j} \mathbf{r}_{ij} \cdot \nabla \omega(r_{ij}), \quad (6.5)$$

where $(\nabla \cdot \mathbf{r}_i)_0$ is the original value of $\nabla \cdot \mathbf{r}$ computed in step 1. The second particle sweep results in a further refinement of which boundary particles are marked as excluded or not.

3. From here on, excluded boundary particles are not involved in any computations for the rest of the time step. The time step now continues with Eq. 3.28.

To demonstrate the effect of excluding such boundary particles near the fluid domain free surface, a simple 2-D still water test case (fluid domain is rectangular, 1.0 m wide by 0.5 deep, $dp = 0.005$ m) is conducted where the pressure field is observed after the first time step. Fig. 6.2 shows the non-dimensional absolute error in the pressures of the column of particles adjacent to one of the tank walls. As expected there is no error at the free surface between the original and improved boundary conditions because the pressure here is enforced. However, just below the free surface there is an error spike for the original conditions due to the effect of

boundary particles above the horizontal free-surface line. Excluding such particles, however, eliminates the error spike.

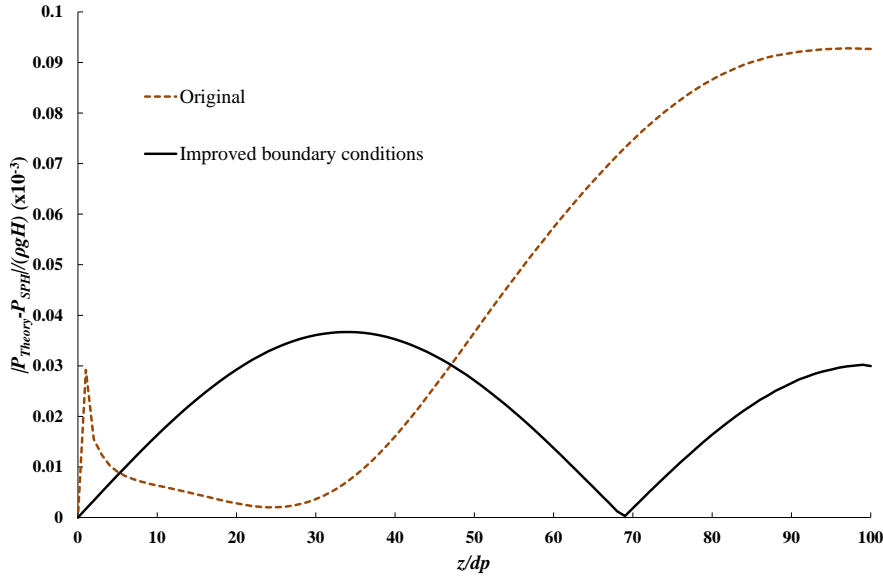
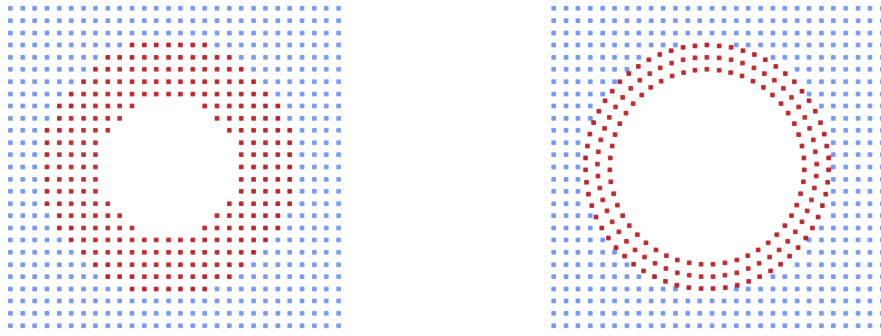


Fig. 6.2: Non-dimensional absolute error of pressure from fluid particles directly adjacent to the boundary on one side of the tank in a 2-D still water case. $dp = 0.005$ m, still water depth $d = 0.5$ m. The theoretical pressure at the bed is 4900 Pa.

Modification 2: The second improvement, is not related to the mathematical formulation of the boundary condition, but is instead concerned with the arrangement of particles for modelling a cylindrical column (as required for simulation of the test cases, see Fig. 6.6). The DualSPHysics pre-processing software, GenCase (as introduced in Section 4.4.1), generates the setup of the case by specifying a Cartesian grid with a regular spacing, equal to dp in each direction, and then placing particles on the nodes within specified geometry shapes (at specified locations in the domain) defined by the user. This means the surface of the column will have a “step” type arrangement as shown in a Fig 6.3a. Such a representation of the column can result in particle penetration of the physical boundary line and thus produce noisy pressures detrimental to the accuracy of results. Therefore, before simulation execution, the cylindrical column particles are arranged in terms of polar coordinates about the column centre, as shown in Fig. 6.3b, for accurate representation of the geometry.

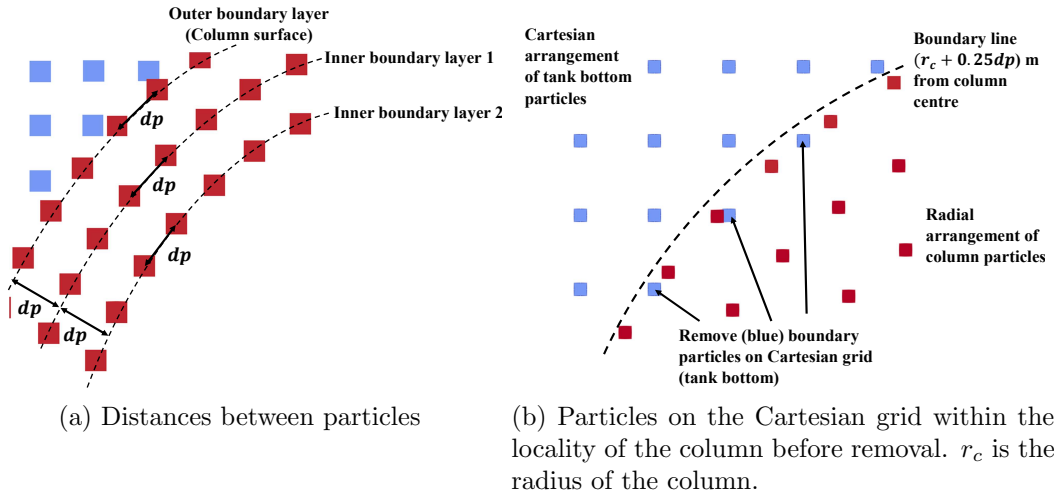
Fig. 6.4 illustrates the detailing of the new arrangement. Each inner boundary layer of the column is $1dp$ apart and the distance between each adjacent boundary particle within the same layer is $1dp$ (as in Fig. 6.4a).



(a) Cartesian arrangement from DualSPHysics

(b) New radial arrangement

Fig. 6.3: Plan view of different arrangements of particles for representation of a cylindrical column. Red indicates a column particle, blue indicates a boundary particle on the tank bottom.



(a) Distances between particles

(b) Particles on the Cartesian grid within the locality of the column before removal. r_c is the radius of the column.

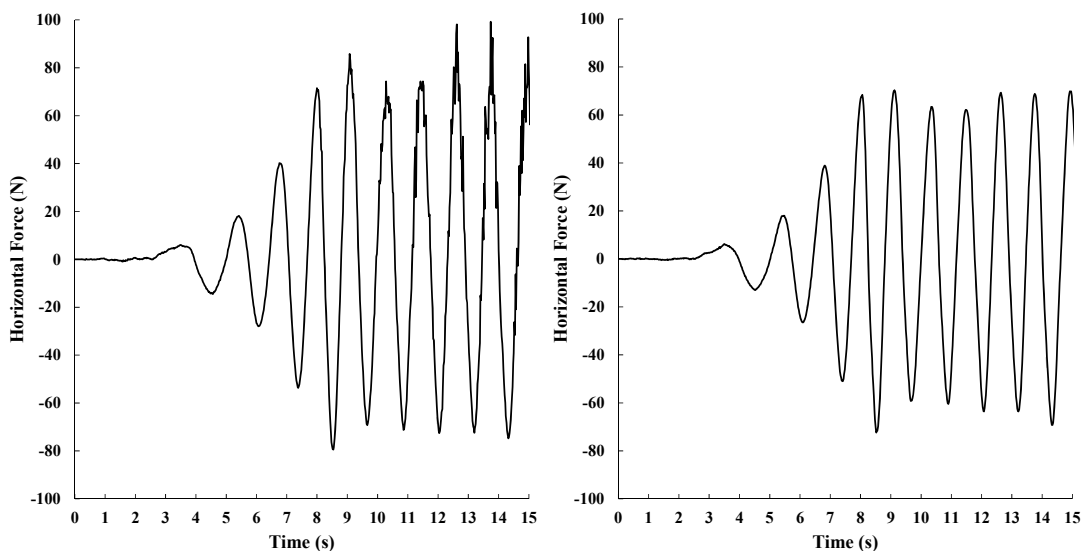
Fig. 6.4: Implementation details for the radial arrangement of the column. Plan views are used for clarity of the particle distributions.

The introduction of radially arranged column particles to a surface of boundary particles placed over a uniform Cartesian grid can give rise to a highly irregular particle distribution at the Cartesian/polar coordinate (tank-bottom/column) interface. This is depicted in Fig. 6.4b, where the irregularity of particles is reduced by removing any tank-bottom boundary particles on the Cartesian grid, within a distance $r_c + 0.25dp$ m of the column centre where r_c is the radius of the column. Generated fluid particles within the same distance of column centre are also removed at the beginning of the simulation. Although the process described does not eliminate the irregularity and is non-conservative of volume, it is however sufficient enough to create an arrangement that does not generate any noticeable non-physical effects during simulation. Fluid particles in the vicinity of the column quickly become

regularised with shifting following the start of the simulation.

Positions of unique interpolation points associated to column particles in the new arrangement are simply computed from pre-defined information about the column i.e. the column radius, centre, and size. Once the column boundary particles and unique interpolation points are generated, no additional coding effort is required for the implementation of the boundary condition as described in Section 3.4.

The benefits of the new particle arrangement are demonstrated in Fig. 6.5, which shows the total horizontal force (obtained using the method described later in Section 6.3.2.2) on the two different particle arrangements for the column subjected to regular wave loading (where the tank geometry used is that described in detail in the next section). For the original Cartesian arrangement, as generated by Gen-Case, in Fig. 6.5a, the time series plot is smooth for the first few wave impacts. However, thereafter, fluctuations begin to appear and increase with time. The plot for the radial particle arrangement in Fig. 6.5b on the other hand, shows no signs of fluctuation for the whole simulation.



(a) Cartesian arrangement from DualSPHysics

(b) New radial arrangement

Fig. 6.5: Total horizontal force on a column (with different particle arrangements) subject to regular wave loading.

6.2.2 Numerical wave tank geometry

The numerical wave basin model is validated by comparison to the physical experiments of Zang et al. [20], which investigated the interaction of non-breaking and breaking focused waves with a cylinder. Fig. 6.6 illustrates a plan view of the numerical wave basin of height (z -direction), 0.7 m. The width (y -direction) of the basin is 1.5 m and is symmetrical across the xz -plane. A piston wavemaker on the LHS spans across the width of the tank and is initially placed -12 m in the x -direction from the right hand tank wall. The piston generates focused waves by moving in the x -direction according to a pre-defined movement, as described in Section 6.2.3. The tank is therefore extended (in the x -direction) an additional -0.2 m from the initial piston placement to account for wavemaker motion in the negative x -direction. A cylindrical column of radius, $r_c = 0.125$ m, and height, 1.0 m, is centred such that it lies on the tank centreline and 7.52 m away from the initial position of the piston.

A numerical damping zone (shaded) is 3 m in length (approximately twice the largest wavelength generated here) is located at the far end of the tank, 9 m away from the piston and is described further in Section 6.2.3.

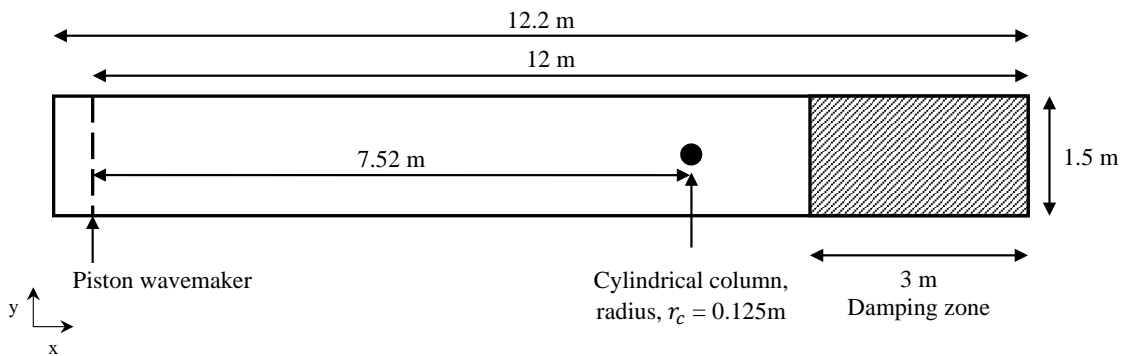


Fig. 6.6: Plan view of the numerical wave basin geometry

The geometry of Fig. 6.6 is similar to the 2-D ISPH simulations by Lind et al [191], who also conducted comparisons to the data of Zang et al. [20], and their results are in Section 6.3. However, Lind et al. [191] avoided the need to explicitly include a column by using the Froude-Krylov forcing approximation to estimate the 3-D loads based on 2-D plane incident waves. Their model was limited to 2D because the computational expense of a 3-D model on a single CPU core was too

impractical.

6.2.3 Focused wave generation

A focused wave is created from an irregular wave group such that the superposition of each component will give a desired wave height, or peak wave amplitude, at a specified focal point away from the wavemaker. Wave breaking will occur if the peak amplitude of the focused wave is sufficiently large. The motion of the piston-type wavemaker is prescribed from linear theory.

The wave spectrum is split into $N = 100$ components, where each wave component has an associated amplitude, a_N , wave number, k_N , frequency, f_N , angular frequency, $\omega_N = 2\pi f_N$, phase, ϕ_N , required piston amplitude, s_N , and power spectral density, S_N . The spectral frequencies, are defined by a JONSWAP spectrum and are in the range of $(0.5 - 3.0)f_p$, where f_p is the frequency at the spectral peak:

$$S_N = \left(\frac{f_p}{f_N}\right)^5 \exp\left(-\frac{5}{4}\left(\frac{f_p}{f_N}\right)^4\right) \gamma_0^r, \quad (6.6)$$

where

$$r = \exp\left(-\frac{(f_N - f_p)^2}{2\sigma^2 f_p^2}\right), \quad (6.7)$$

using $\gamma_0 = 3.3$, and $\sigma = 0.07$ if $f_N \leq f_p$ else $\sigma = 0.09$. Subsequently, wave component amplitudes can be found as calculated from Tromans et al's "NewWave" group [335]:

$$a_n = \frac{A_N S_N \Delta f}{\sum_N (S_N \Delta f)} \quad (6.8)$$

where A_N is a specified peak amplitude of the wave group components which superimpose at a focal point, x_f , and Δf is the frequency step equal to $(3.0 - 0.5)f_p/N$.

The generation of each wave component requires the piston to move by an amplitude distance, s_N , which is found using the respective wave component variables [21]:

$$s_N = a_N \frac{\sinh(2k_N d) + 2k_N d}{2(\cosh(2k_N d) - 1)}, \quad (6.9)$$

where d is the still water depth and k_n is found by solution of the dispersion equation:

$$(\omega_N)^2 = gk_N \cdot \tanh(k_N d) \quad (6.10)$$

The phase angle ϕ_N allows for the superposition of each component at a specified focal point. A component's phase angle is given as:

$$\phi_N = \omega_N t_f - k_N x_f, \quad (6.11)$$

where t_f is the focal time equal to $2x_f/c_g$ such that c_g is the group celerity found as:

$$c_g = \frac{\omega_p}{2k} \left(1 + \frac{2kd}{\sinh(2kd)} \right), \quad (6.12)$$

such that $\omega_p = 2\pi f_p$, and k is found from Eq. (6.10) with ω_p .

The velocity of the piston wavemaker is therefore given as:

$$u_p = \sum_N s_N \omega_N \cos(-\omega_N t + \phi_N), \quad (6.13)$$

This process can be applied to generate waves of small amplitude, defined by $A_N/d \lesssim 0.1$. However for larger waves, such that $A_N/d \gtrsim 0.1$, non-linear interactions affect the outcome of the desired wave, and so for the cases presented in Section 6.3.2, the specified peak amplitude value of the wavemaker is slightly adjusted in order to achieve the actual desired value of A_N for experimental comparison [191].

To reduce wave reflection effects from the end of the domain at the focal point, a numerical damping zone [11], as depicted in Fig. 6.6, exponentially dissipates the wave energy across a damping length, L_D , with the following equation:

$$A_i = A_{0,i} \left(1.0 - e^{-2.0(L_D - (x_i - x_D))} \right), \quad (6.14)$$

where x_i is the x -position of a particle, x_D is the starting point of the damping zone, and $A_{0,i}$ and A_i are the values of a given variable before and after the damping formula is applied. Here, the variables to be damped are the fluid velocity given

by the computation of Eq. (3.32), and the particle shifting distance following its evaluation.

6.3 Results comparisons and analysis

This section compares Incompressible-DualSPHysics' new 3-D numerical wave basin model to the focused wave-cylinder impact experiments of Zang et al. [20] and the numerical results of Lind et al. [11], who used a 2-D ISPH model with a Froude-Krylov approximation for the prediction of 3-D loads on the cylinder. Unique insight to the physical processes of the wave-structure interaction are also demonstrated from the results of the 3-D Incompressible-DualSPHysics model.

6.3.1 Validation of free-surface elevation vs linear theory

Prior to the experimental case study, validation of the generated waves is demonstrated by a 2-D spatial-refinement study for the wave tank in Fig. 6.6, where all motion in the y -direction is prevented and the cylinder is absent, so the free-surface elevation can be measured, undisturbed, to confirm generation of the focused wave-form. The free-surface elevation at a given point (x_{fs}, y_{fs}) in space, for each time step, is determined by extracting all free-surface particles (i.e. using Eq. 3.37) within the bounds of $(x_f \pm 2dp, y_f \pm 2dp)$ and taking the average z -coordinate position.

Fig. 6.7 compares the ISPH results, with multiple particle spacings in the range of $dp = 0.00125$ - 0.02 m, for the free-surface elevation at the focal point ($x_{fs} = 7.52$ m) predicted with linear theory. The peak amplitude of the JONSWAP wave group at the focal point is equal to 0.05 m, and the water depth is the same as that in the experiments, 0.505 m. There is close agreement for all resolutions, where the refinement of the particle spacing demonstrates clear convergence towards the peak free-surface elevation of the analytical solution as seen in the inset of the figure.

Fig. 6.8 shows the L_2 error norm for the peak free-surface elevation at the focal point for each of the resolutions. Two ratios of peak amplitude, and still water depth, to particle spacing are displayed next to the respective resolution. For the

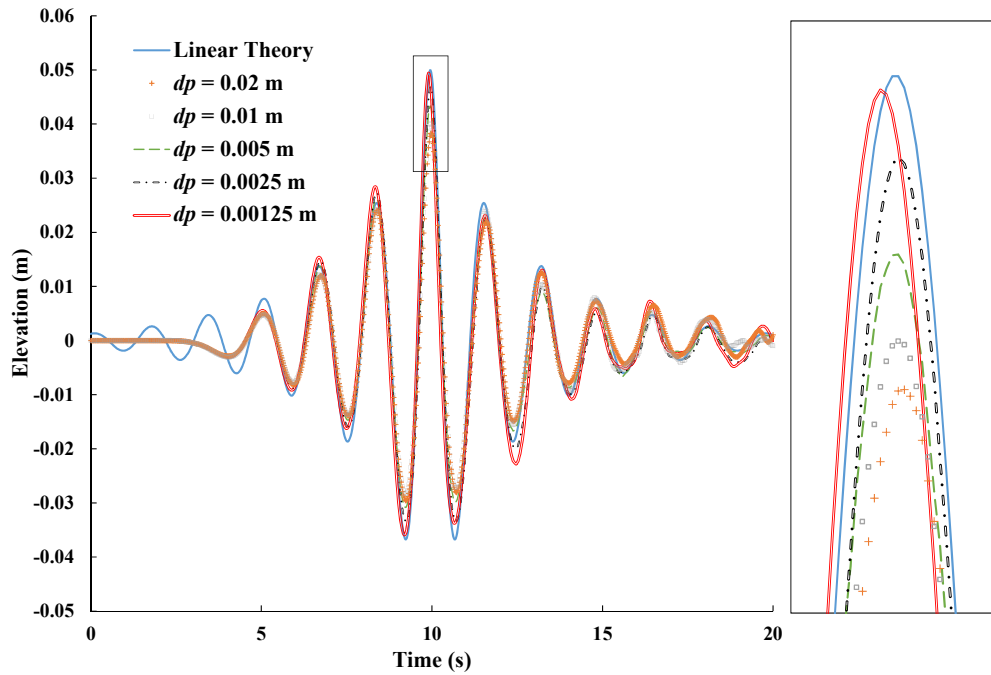


Fig. 6.7: Spatial convergence of the ISPH model with linear theory for the free-surface elevation of a JONSWAP wave group. $A_N = 0.05$ m. The RHS of the plot shows enlarged peak elevations.

coarsest two particle spacings, $dp = 0.02$ and 0.01 m, the water depth cannot be resolved exactly in addition to a relatively low number of particles representing the wave. Therefore, the convergence rate of the L_2 error norm with the refinement of dp is calculated from those resolutions, $dp = 0.005$, 0.0025 , and 0.00125 m, which can represent the domain geometry exactly. The result is a convergence rate of near second order (1.78), as represented by the solid black line.

For the comparisons with experimental data in the next section, $dp = 0.0125$ m ($d/dp = 40.4$) for all cases. Ideally, the chosen particle spacing represents the still-water-level exactly and provides an appropriate number of particles to resolve the wave height (as discussed for Fig. 6.8), but it is the finest resolution allowed within the chosen GPUs memory, for the required dimensions of the 3-D wave basin in Fig. 6.6. Nevertheless, it will be seen the resolution is sufficient to provide accurate results. Information on the hardware and computation time for the experimental comparisons are discussed in Section 6.3.4.

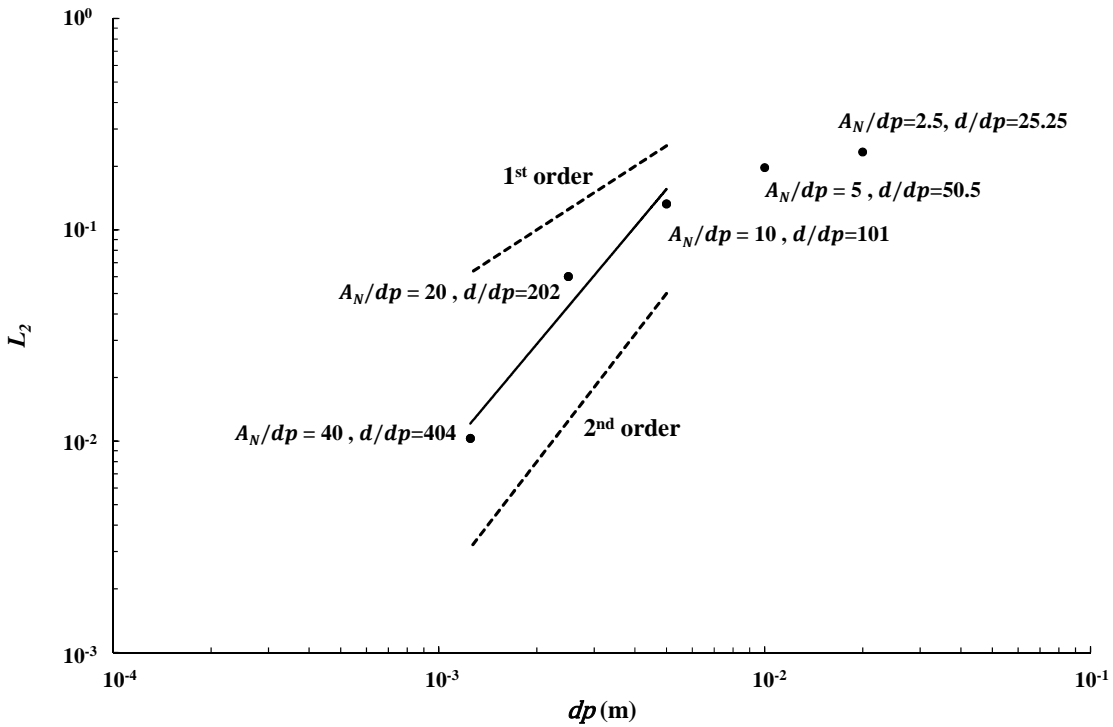


Fig. 6.8: Convergence rate of the L_2 error norm of the peak free-surface elevation at the focal point for the simulation in Fig. 6.7. $A_N = 0.05$ m, $d = 0.505$ m.

6.3.2 Experimental comparisons

Four focused wave group experiments from the work of Zang et al. [20] are chosen for simulation. Table 6.1 shows the peak frequency and wave height, H , of each focused wave group from the chosen experiments. For consistency with the data, the same naming convention as [20] is used. Two non-breaking wave groups, F3 and F16, are chosen, and also two breaking wave groups, F14 and F15.

Table 6.1: Focused wave groups.

Wave group	f_p (Hz)	H	Type
F3	0.61	0.12	Non-breaking
F16	0.61	0.23	Non-breaking
F14	0.82	0.14	Breaking
F15	0.82	0.22	Breaking

Zang et al. [20] provides experimental data for the free-surface elevations, in the absence of the cylinder, and the total horizontal forcings on the cylinder when it is present. The results of the 2-D ISPH with Froude-Krylov (FK) approximation model of Lind et al. [191] are also included.

To improve post-processing time, python scripts were created for execution in Paraview to obtain the results in the following sections. Refer to Appendix D for more details.

6.3.2.1 Comparison of free-surface elevation with experiment

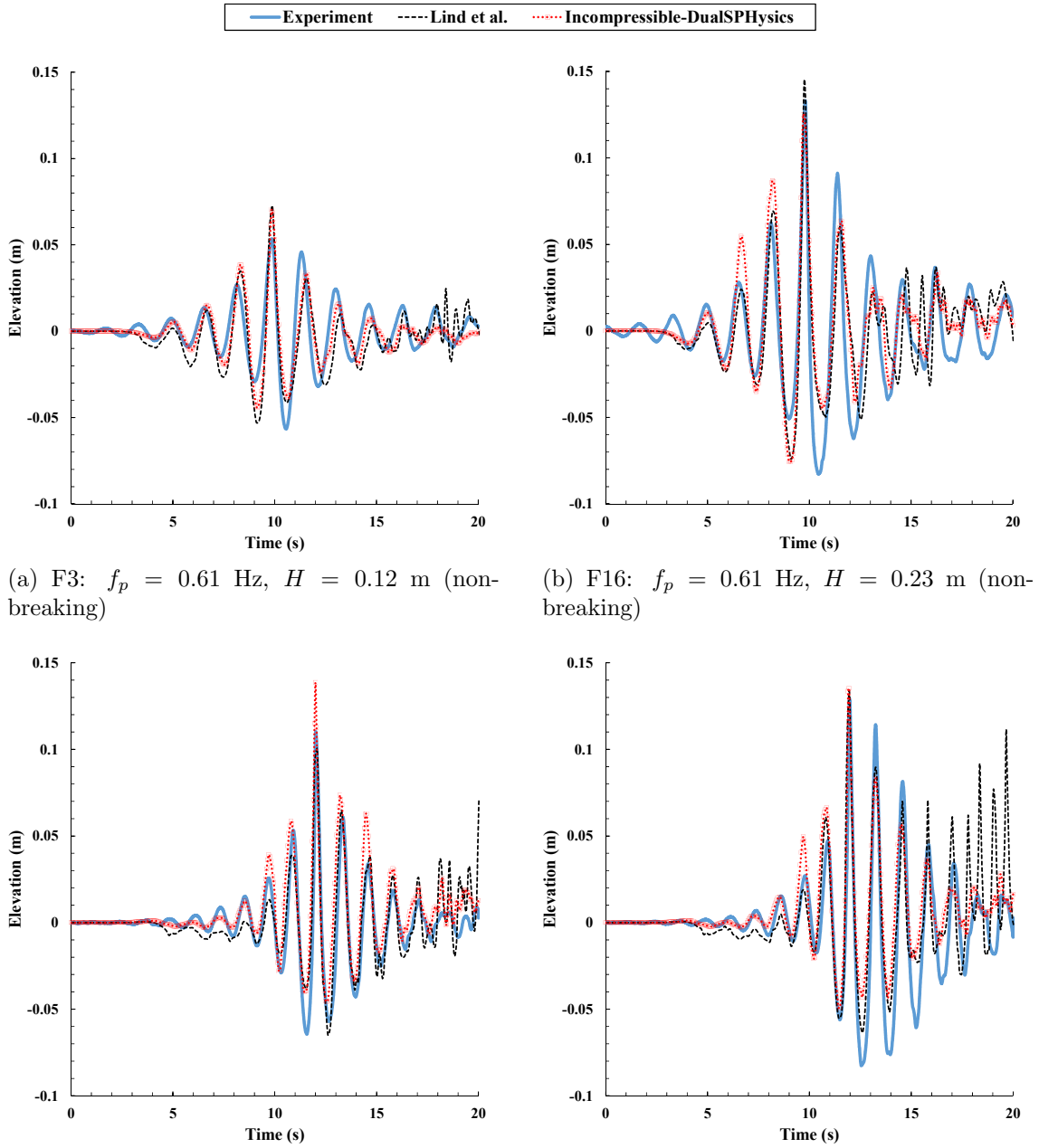
Before comparing 3-D simulations with experiments that include the cylinder, they are compared with experimental data where the cylinder is absent.

Fig. 6.9 compares the free-surface elevation, at the position where the cylinder-front would be ($x_{fs} = 7.395$ m, $y_{fs} = 0$ m) for the different focused wave groups, with the experimental data and the 2-D ISPH results of Lind et al. [191].

The Incompressible-DualSPHysics results, for all wave groups, are very similar to those of the 2-D simulations of Lind et al. [191], which is expected as the two ISPH methodologies are similar and the problem is essentially 2-D due to the absence of the cylinder. Throughout all simulation cases, Incompressible-DualSPHysics maintains the mean still water-level within one particle spacing. For cases F16 and F15 (Figs 6.9b and 6.9d respectively), the peak free-surface elevations measured from this work match those of the experimental data. Both the F3 and F14 (Figs 6.9a and 6.9c respectively) cases however, over-predict the highest elevations by more than one particle spacing. The discrepancy in case F3 is likely due to insufficient particle numbers representing such a relatively small wave height, and the error in case F14 is attributed to “spray”-like particles in a fragmented free-surface due to breaking as seen in Fig. 6.10 which shows a simulation snapshot at the time, $t = 12.0$ s, when the peak free-surface elevation is measured. The wave is in post-breaking at this stage.

6.3.2.2 Total horizontal force on cylinder

The simulations in Section 6.3.2.1 are repeated with the presence of the cylinder. Fig. 6.11 shows simulation snapshots of the breaking-wave case F15 at times $t = 11.7$, 11.9, and 12.0 s, corresponding to the instants before, at, and after the cylinder experiences the peak total horizontal force. For clarity, only fluid particles on the



(a) F3: $f_p = 0.61$ Hz, $H = 0.12$ m (non-breaking)

(b) F16: $f_p = 0.61$ Hz, $H = 0.23$ m (non-breaking)

(c) F14: $f_p = 0.82$ Hz, $H = 0.14$ m (breaking)

(d) F15: $f_p = 0.82$ Hz, $H = 0.22$ m (breaking)

Fig. 6.9: Free-surface elevation of the various focused wave groups at the position of the cylinder-front.

free surface ($P = 0$) are shown, where they are coloured by their velocity magnitudes. The cylinder surface particles are coloured by their pressure.

In Fig. 6.11a, before the cylinder experiences the peak total forcing, the oncoming wave starts to break before the cylinder. The breaking-wave reaches the cylinder at $t = 11.9$ s, as seen in Fig. 6.11b, where the water-surface elevation and velocities increase around the stagnation point. The shape of the wave around the cylinder at this instant, is such that a hydrostatic pressure gradient across the width of the

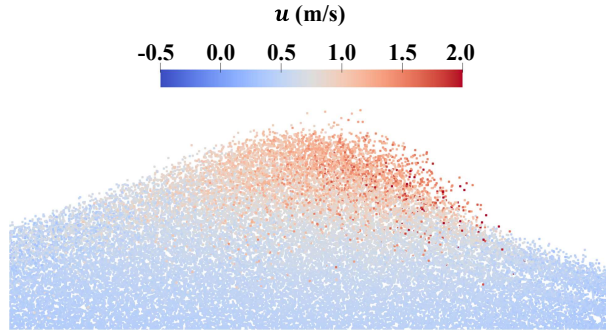


Fig. 6.10: Post-breaking event of focused wave group F14 at $t = 12.0$ s.

structure is clearly present and responsible for the maximum loading which is discussed later. Just after the initial impact, in Fig. 6.11c, fluid particles separate away from the water-surface due to the interaction with the structure. Such fragmentation cannot be so easily or efficiently represented by OpenFOAM [336] and PICIN [38] models. The presence of spray, as seen in the experiments [20], is well-captured here, but under resolved. Fluid velocities and cylinder pressures during the peak total forcing event are analysed further in the Section 6.3.3.

The total horizontal force, $F_{total,x}$, on the cylinder can be obtained in post-processing by use of each particle on the cylinder surface, defined as belonging to the set Ω_{cs} , and their corresponding pressure values, as obtained from the solution of the PPE (Eq. (3.30)):

$$F_{total,x} = \sum_{i \in \Omega_{cs}} P_i dp^2 \cos \theta_i^c, \quad (6.15)$$

where θ_i^c is the angular position of a particle from the centreline, as illustrated in Fig. 6.12, calculated using the cosine rule. $\theta^c = 0$ faces the propagating wave-fronts in the negative x -direction relative to the structure.

Fig. 6.13 shows the total horizontal force experienced with time by the cylinder for each case. The time axes are adjusted to match the timing of the peak force event (of the numerical simulations) at $t = 9.7$, 9.6 , 11.95 , and 11.9 s for cases F3 (Fig. 6.9a), F16 (Fig. 6.9b), F14 (Fig. 6.9c), and F15 (Fig. 6.9d) respectively.

Prior to the peak force event, Incompressible-DualSPHysics shows reasonable agreement with the experimental data for all cases. Most discrepancies before the

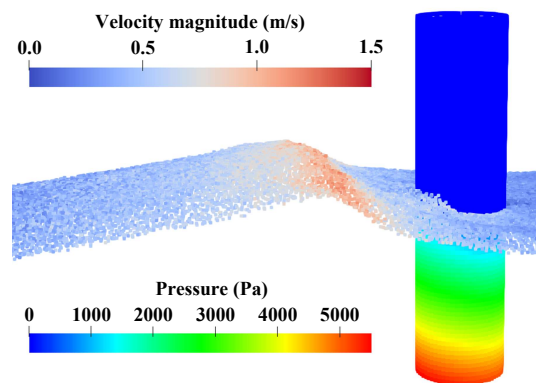
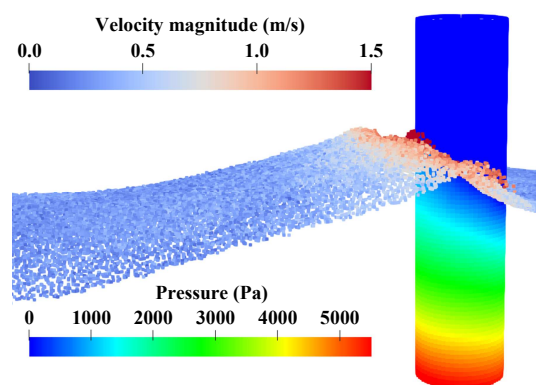
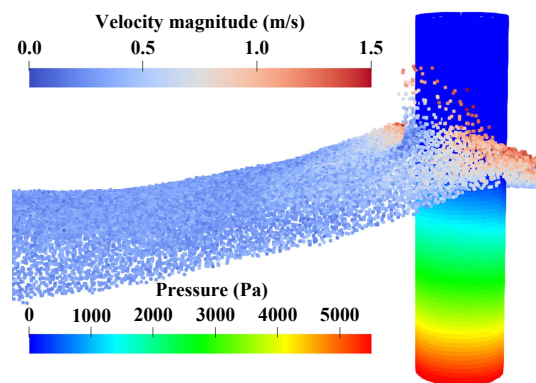
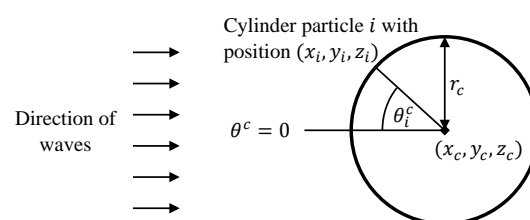
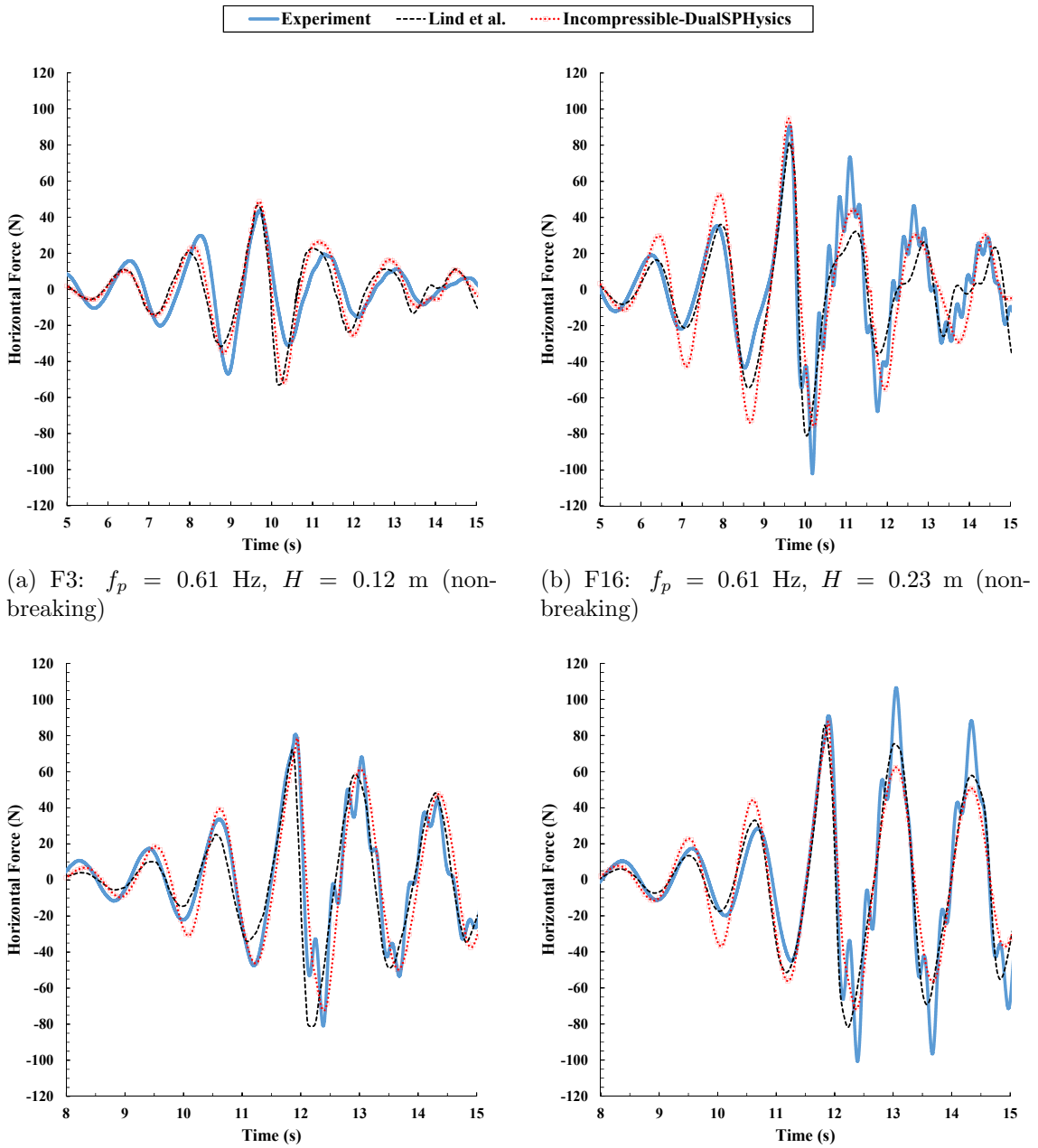
(a) Before peak total forcing, $t = 11.7s$ (b) Time of peak total forcing, $t = 11.9s$ (c) After peak total forcing, $t = 12.0s$

Fig. 6.11: Simulation snapshots of breaking-wave case F15.

Fig. 6.12: θ^c around the cylinder.



(c) F14: $f_p = 0.82$ Hz, $H = 0.14$ m (breaking) (d) F15: $f_p = 0.82$ Hz, $H = 0.22$ m (breaking)
 Fig. 6.13: Total horizontal force on the cylinder subject to the various focused wave groups.

peak event show an overprediction by the numerical model, but these correspond with lower amplitude waves (see Fig. 6.9) which would be represented by very few particles as discussed in Section 6.3.2.1. Accuracy of these forcings can be improved with finer particle spacings.

The actual peak force events show excellent correspondence with the loadings recorded in the experiments, and in some cases, better than the 2-D FK results. After these peak impacts, the experimental data in the three steep wave cases (F16,

F14, and F15) appear to exhibit higher frequency oscillations, which are not captured by either of the numerical models. The features are thought to originate from the response of the cylinder following the high-impact event. This could be confirmed by replacing the current ISPH model's fixed structure with a model which moves in response to the flow mechanics. Nevertheless, the 3-D ISPH model captures the general profile of the experimental force data. Comparing the 2-D FK and 3-D models, the results are generally similar, further validating the FK approximation technique for predicting peak focused-wave forces used by Lind et al [191]. In some places however, the 3-D model shows closer agreement with the experiments. This is expected since the actual fluid-structure interaction is simulated. For the 2-D simulations, the absence of the cylinder means the validity of the model deteriorates as time progresses after the initial waves pass the focal point. The last 3-5 seconds of each plot show fluctuations in the results of Lind et. [191] due to numerical errors at the free surface propagating over many fixed time steps. This is not observed in the Incompressible-DualSPHysics model, using a variable time step, for this length of simulated physical time.

Perhaps the most noticeable discrepancy between numerical and experimental data comes from case F15, where both the ISPH models failed to capture the experiment's peak total force on the column around $t = 13.0$ s. Attempts to reduce the disparity by increasing the tank dimensions and changing parameters associated with the ISPH formulation produced similar results. Increasing the resolution of the simulation was not possible due to GPU memory limitations. If this is not due to particle spacing or dynamic structural response from the cylinder, then more information and data from the experiments (such as photos of the wave form) would be required to determine the cause of the discrepancy.

6.3.3 Investigating hydrostatic and non-hydrostatic pressure during peak loading

The full 3-D numerical model allows for detailed analysis concerning the flow mechanics of non-breaking and breaking waves interacting with the cylinder, which

cannot be identified easily in experiments or with the 2-D ISPH model. To examine the pressure fields immediately adjacent to the cylinder, the surface of the structure herein will be visualised in a 2-D θ^c - z plane.

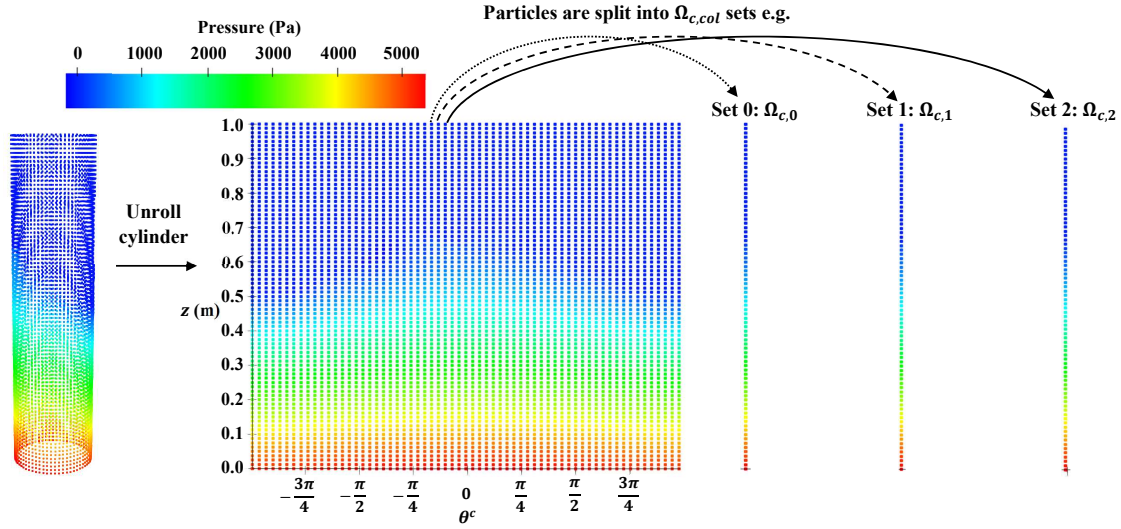
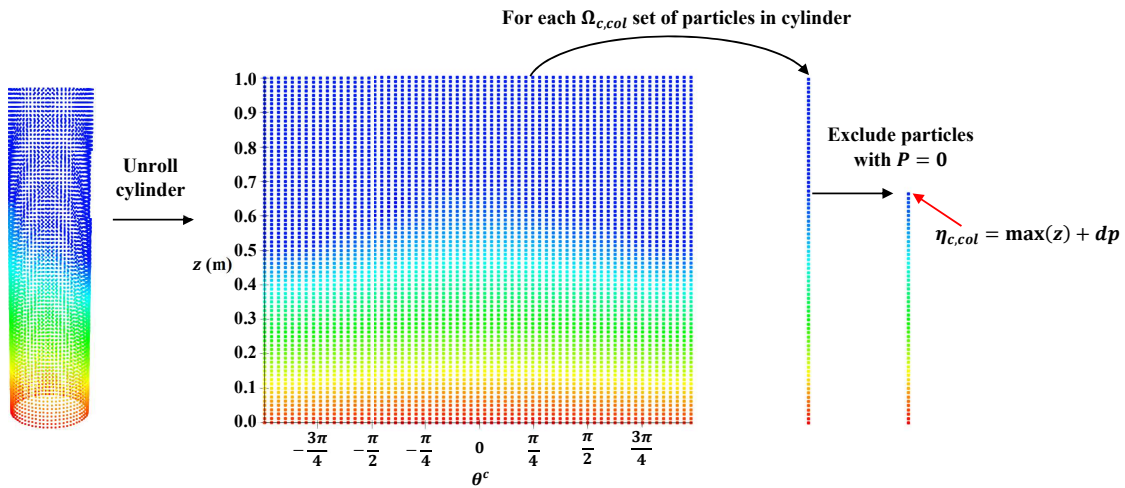
The mechanics of a steep wave impinging upon an object is a complex problem [21] which cannot be de-constructed analytically with ease. Herein, the pressures on the cylinder are split into components of hydrostatic and non-hydrostatic pressure to investigate the effects of non-breaking and breaking waves impacting on the cylinder.

The two components of pressure about the cylinder are found by first determining the free-surface elevations around the structure using the process depicted in Fig. 6.14. Cylinder surface particles within the same vertical plane i.e. they have the same x - and y -position coordinate, are considered to be of the same $\Omega_{c,col}$ set as illustrated in Fig. 6.14a, where subscript col denotes a unique set number within the cylinder. Each $\Omega_{c,col}$ set of particles is observed independently and possesses a free-surface elevation value, $\eta_{c,col}$. As shown in Fig. 6.14b, the particles in a set on, or above, the free surface, i.e. $P_i = 0$, are excluded. Therefore, the free-surface elevation is found as the highest z -coordinate of a particle in the set of remaining particles plus the original particle spacing, dp^1 .

Particles with $P = 0$ are also excluded for visualisation purposes where the free-surface elevation around the cylinder can be qualitatively observed using the remaining particles without knowledge of their pressures.

The method for evaluating the free-surface profile around the column is checked by mapping all fluid particles, within a kernel support of the cylinder surface, over a snapshot of the cylinder particles below the free-surface (with $P \neq 0$). This is illustrated in Fig. 6.15 with the case F14 at the time, $t = 11.95$ s, of the peak horizontal force on the cylinder. The free-surface elevation identified around the cylinder matches the position of the surrounding particles. Any errors are within a particle spacing and will reduce with refinement.

¹Alternatively, the lowest z -coordinate of particles with $P = 0$ within the $\Omega_{c,col}$ set can also be taken as the set's free-surface elevation. This technique may be more robust for cases of plunging breakers due to multiple points of contact with the cylinder.

(a) Splitting cylinder surface particles into $\Omega_{c,col}$ sets(b) Finding the free-surface elevation in each $\Omega_{c,col}$ setFig. 6.14: Evaluating the free-surface elevation around the cylinder where subscript *col* represents a unique set within the cylinder.

Once $\eta_{c,col}$ for a $\Omega_{c,col}$ set is found, the hydrostatic pressure, $P_{H,i}$, of a particle, i , within the set is calculated as:

$$P_{H,i} = \rho g (\eta_{c,col} - z_i) \quad (6.16)$$

The non-hydrostatic component, $P_{NH,i}$ is then found as the pressure (obtained from the PPE solver) minus the hydrostatic part:

$$P_{NH,i} = P - P_{H,i} \quad (6.17)$$

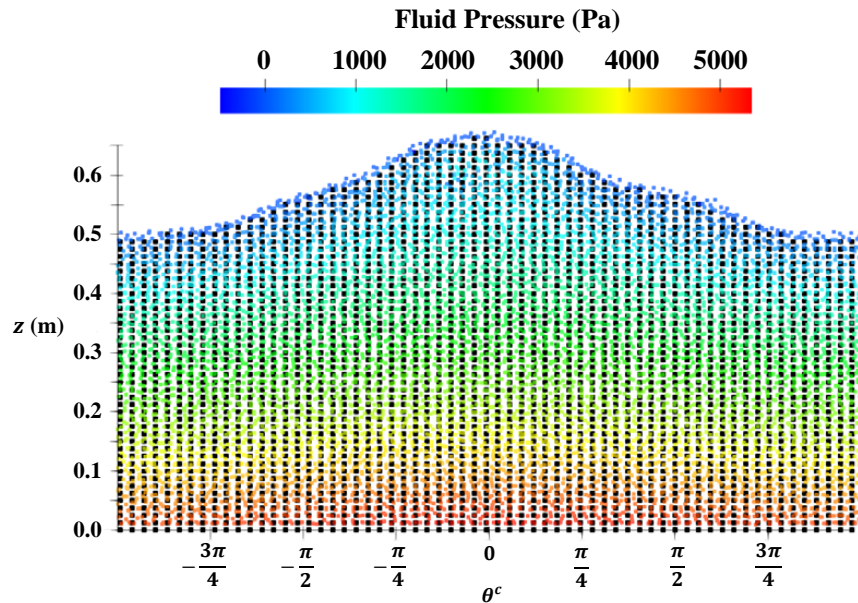
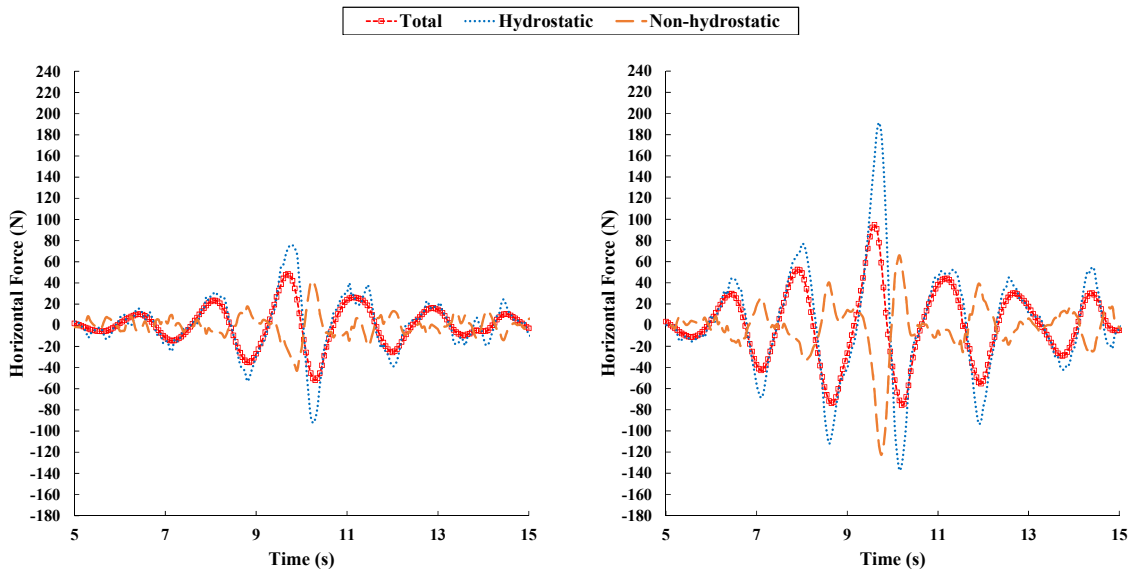


Fig. 6.15: The method for determining the free-surface elevation around the cylinder is validated by mapping nearby fluid particles (coloured by pressure value) over the cylinder-surface particles (coloured in black) with $P \neq 0$.

The total horizontal forces attributed to hydrostatic or non-hydrostatic pressures around the cylinder are computed via Eq. (6.15), but with the appropriate pressure component in place of P_i . Fig. 6.16 plots these forces for each case, the time axes are scaled as in Fig. 6.13 to highlight the main event including the peak total force on the cylinder.

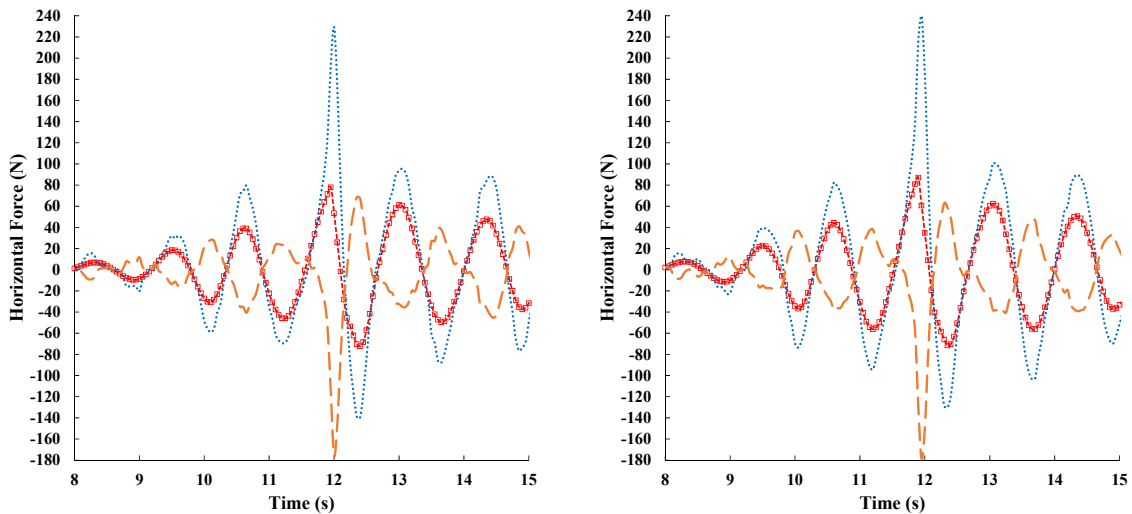
All plots possess similar characteristics. Hydrostatic pressure is the dominant component producing global loadings where magnitudes of the peaks and troughs are larger than, and in phase with, those of the overall total force. The force from the non-hydrostatic pressure component is out of phase with lower magnitudes. The interpretation of such data can be aided by observing snapshots of each simulation, as in Fig. 6.17, at the time of peak total force showing a slice through the centre of the cylinder and the fluid domain in the x - z plane.

There is clearly a difference in free-surface elevations on either side of the cylinder in the direction of the propagating wave. The plots in Fig. 6.16 therefore, shows that the force due to the changing free-surface elevation gradient across the cylinder, as described by Dean and Dalrymple [21], is responsible for the maximum loading in both cases of non-breaking and breaking waves when the difference in hydrostatic



(a) F3: $f_p = 0.61$ Hz, $H = 0.12$ m (non-breaking)

(b) F16: $f_p = 0.61$ Hz, $H = 0.23$ m (non-breaking)



(c) F14: $f_p = 0.82$ Hz, $H = 0.14$ m (breaking)

(d) F15: $f_p = 0.82$ Hz, $H = 0.22$ m (breaking)

Fig. 6.16: Incompressible-DualSPHysics results for the total, hydrostatic, and non-hydrostatic horizontal force on the cylinder subject to the various focused wave groups.

pressure is greatest.

Fig. 6.18 plots the free-surface profile around the cylinder, for all cases, at the times where the peak horizontal force is experienced. The values are taken as those computed from the process depicted in Fig. 6.14 which gives rise to the “step”-type nature of the plot. On comparison of the three steeper waves (F14, F15, and F16), the non-breaking case, F16, exhibits a lower free-surface height at the front of the cylinder to the two breaking cases, F14 and F15. Despite this, as found in Fig. 6.13, case F16 imposes a maximum total loading of approximately 95 N,

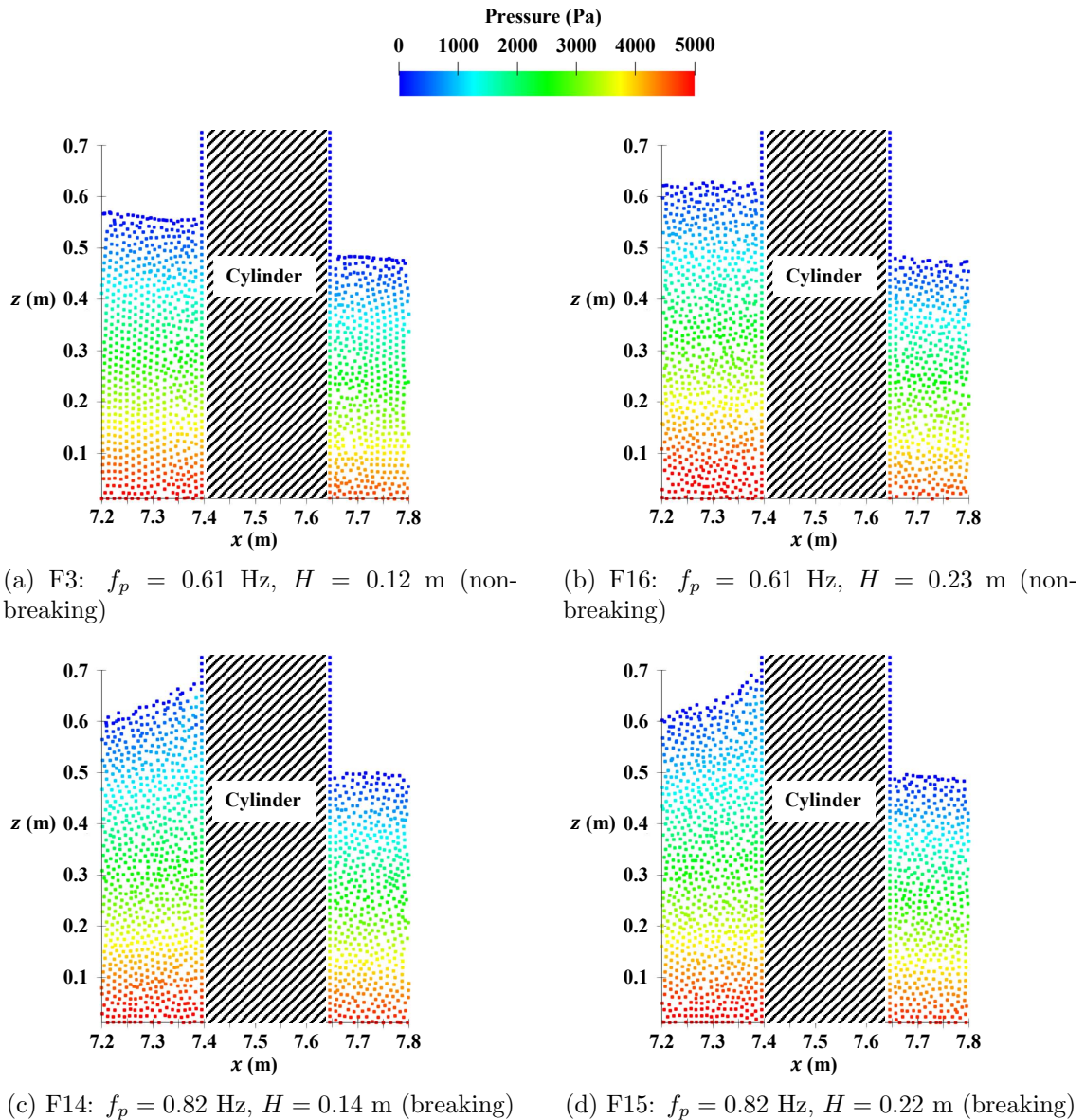


Fig. 6.17: Centreline vertical planes through the centre of the cylinder and fluid domain in the x - z plane for each case at the times, $t =$ (a) 9.7, (b) 9.6, (c) 11.95, and (d) 11.9 s, corresponding to peak total force experienced by the cylinder for F3, F16, F14, and F15 respectively. The wave direction is from left to right.

a force larger than those present in the two breaking-wave cases with larger wave heights, F14 and F15, which exhibit forces of about 78 N and 87 N respectively. This shows that the largest total forces originate from steep free-surface gradients and a globally dominant hydrostatic pressure difference around the cylinder. The three wave heights may be comparable, but the steepness of a wave is shown to be an important factor in considering the largest loads.

The findings so far however, only consider pressures forces on the cylinder as

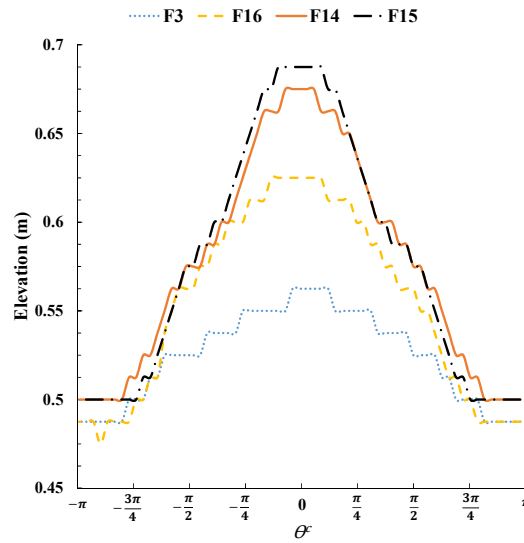


Fig. 6.18: Free-surface elevation around the column for all cases at their respective times of maximum total horizontal force experienced. $t = 9.7, 9.6, 11.95,$ and 11.9 s for cases F3, F16, F14, and F15 respectively.

an integrated global horizontal load. As hydrostatic pressure is dominant, the data says little about the fluid-structure interaction near the free surface. Experimental studies [192, 337, 338] have identified the effects of wave-breaking to be a local phenomenon and induce high magnitudes of force locally. This is investigated in Fig. 6.19, which plots the normalised cylinder particle pressures relative to the mean still-water-level pressure at the times of peak wave loading. Thus, the local effects of the free surface on the structure are isolated. The plots highlight the pressures in the wave crest for non-breaking and breaking cases. Despite the similarities in peak free-surface elevation and global cylinder loading between cases F14, F15, and F16, there exists significantly higher localised forces at the front of the cylinder by the free surface in the former two events. These larger pressures are due to the dynamics of breaking at the free-surface where particle velocities are relatively high compared to non-breaking waves, as investigated in Fig. 6.20. The plots show velocities of free-surface fluid particles within a distance h of the cylinder's surface for all cases. Fig. 6.20a shows the magnitude of the fluid particle's velocity component normal to the cylinder surface. Fig. 6.20b shows the tangential components, and 6.20c presents the z -velocity component, w .

Comparing the normal and tangential velocities, the former component is small

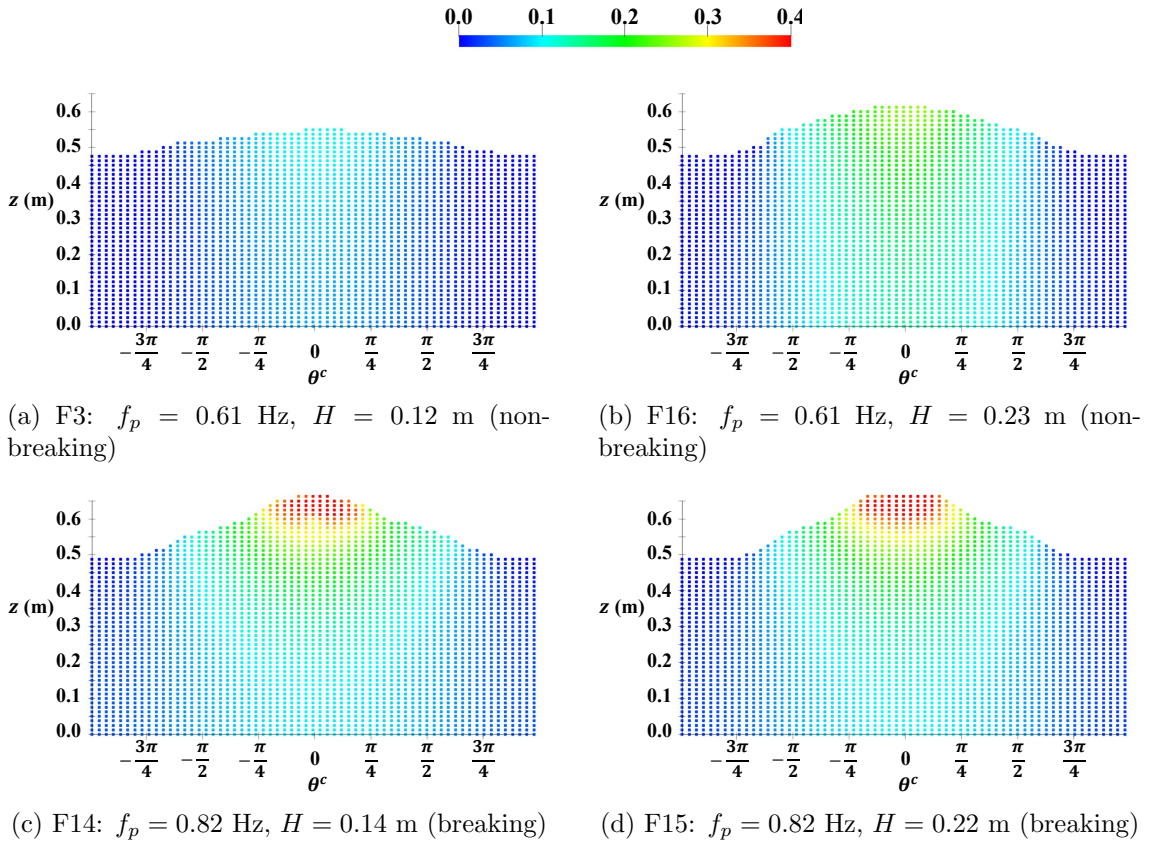


Fig. 6.19: Normalised cylinder particle pressures relative to the mean still-water-level for each case at times of peak total force experienced. Particles with zero pressure are omitted.

compared to the latter for all cases. The plot of tangential velocities however, displays much larger magnitudes near the front of the cylinder (θ^c) for the breaking cases (F14 and F15), than in the non-breaking cases (F3 and F16). Similarly, the z -velocity components shown in Fig. 6.20c are also much higher in the breaking cases (approximately 1.5-2 times) and greatest at the cylinder front.

6.3.4 Computation time and memory

All simulations in Section 6.3 were computed on an Nvidia Tesla K40c GPU (2880 CUDA cores at 0.88GHz, 12GB RAM). A total of 5,302,318 particles comprising 4,551,640 fluid and 750,678 boundary were used for $dp = 0.0125$ m. Table 6.2 shows the time taken to compute each case (20 s of physical time) and the number of time steps involved.

The computation times here are on the order of hours, compared to weeks or

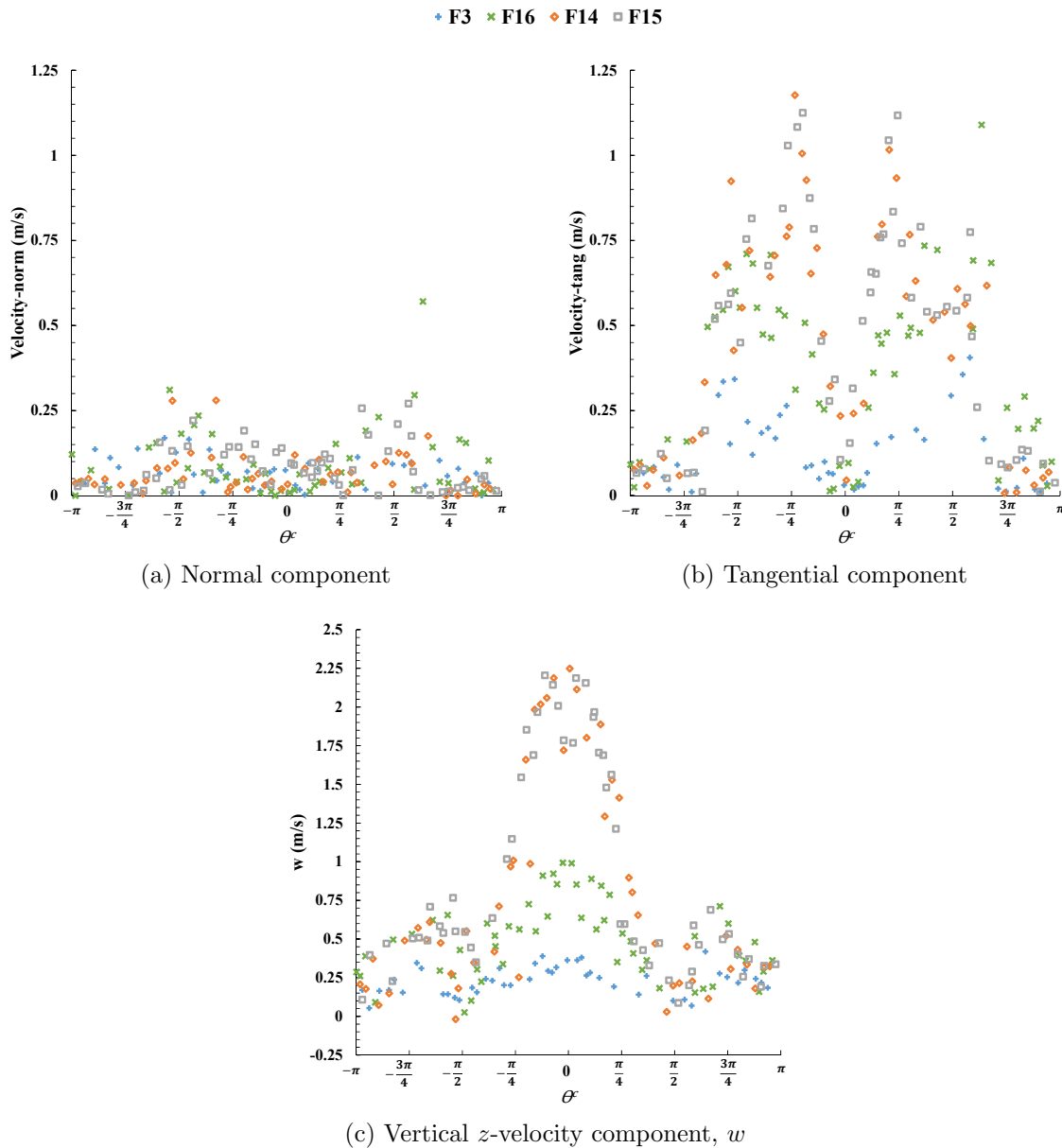


Fig. 6.20: Fluid velocities of free-surface particles within a distance h of the cylinder surface for all cases at instances of peak horizontal force experienced (as in Fig. 6.19). Normal and tangential components consider x - and y -velocity components, u and v .

Table 6.2: Incompressible-DualSPHysics computation times

Case	Computation time (h)	time steps
F3	11.5	2763
F16	20.8	4871
F14	21.3	5006
F15	22.4	5251

even months for previous single-core CPU ISPH codes [190]. The equivalent code for a single-thread CPU, based on the runtime comparisons in Sections 5.6.2 and 5.6.5, is estimated to take about 1 week.

Regular waves of wavelength $L = 2.2$ m, wave height $H = 0.14$ m, and period $T \approx 1.25$ s were simulated for 20s of physical time to compare runtimes with recent PICIN [38] and OpenFOAM [336] parallel-CPU simulations. The comparison between the three models are shown in Table 6.3.

Table 6.3: Number of particles for simulations in Section 6.3.2

	Incompressible-DualSPHysics	PICIN [38]	OpenFOAM [336]
Hardware	Tesla K40c GPU	80-core CPU	8-core CPU
Simulated time, t_s (h)	20.0	18.0	24.5
Computed wave periods, $n_w = t_s/T$	16	14.4	19.6
Computation time, t_c (h)	29.3	12.16	126.20
Time for 1 wave period, t_c/n_w (h)	1.83	0.84	6.44

The performance of each model for the application is evaluated as the average computational time required to simulate a single wave period. From a practical perspective, Incompressible-DualSPHysics compares well against the other two models. The PICIN solver is approximately 2.2 times faster, but at the cost of using 80 CPU cores, which requires greater expenditure and maintenance than a single GPU. A fairer comparison, in terms of cost, shows OpenFOAM with 8-cores taking about 3.5 times longer than Incompressible-DualSPHysics to compute a wave period. However, the OpenFOAM model simulates two-phases (water-air) and uses nearly twice as many cells as there are particles in ISPH. Upon examining a previous study [248], of a GPU-implemented multi-phase WCSPH model using DualSPHysics, it is estimated that a multi-phase Incompressible-DualSPHysics code would result in similar times to the OpenFOAM solver.

A total of about 6.13 GB of memory was assigned to the GPU from DualSPHysics where approximately 80% (4.84 GB) accounts for storing the PPE matrix.

6.4 Conclusions

This chapter has applied the new GPU-accelerated ISPH code developed with this study, Incompressible-DualSPHysics, to a real engineering application, modelling a

3-D numerical wave basin for simulating focused breaking wave-structure interaction. This is the first time that ISPH has been evaluated for such a complex application.

Extensions to the numerical methodology from Chapter 3 are made to reduce errors near the free surface and enable wave propagation. Correction terms in the Laplacian Morris operator are added for implementation of the Schwaiger operator [156]. In particle shifting, the vector normal to the free surface is improved with smoothing by kernel summation. Boundary particles above the free surface are excluded to prevent false summations. Additionally a variable time step size, based on the CFL condition, is implemented to improve runtimes and reduce numerical-error accumulation. Cylinder boundary particle positions are altered from a Cartesian arrangement, produced by the pre-processing software of DualSPHysics, to a radial configuration which minimizes noise during fluid-structure interaction.

Simulating non-breaking and breaking focused wave groups impacting a surface-piercing cylinder with the 3-D numerical model compares well with the peak cylinder forcings and focal point free-surface elevations provided in the experimental data of Zang et al. [20]. However, further experimental data is required to verify the discrepancies. It is speculated here whether dynamic structural response of the cylinder is a cause of higher frequency oscillations in the experimental force readings.

Post-processing analysis of the numerical data reveals new insights into the hydrodynamic characteristics of the cases studied here:

- In steep waves, the hydrostatic component of the global pressure field around the cylinder is dominant, and so the free-surface elevation and steepness of a wave across the cylinder is responsible for the maximum loading on the structure.
- Local cylinder pressure forces near the free-surface of a breaking-wave are large compared to those of a non-breaking wave.
- The change in jet momentum in the normal direction is directly responsible for local pressure increase. High vertical and tangential velocities then follow from conservation of energy.

Suggestions to improve the results of the numerical model for the application include implementing a multi-phase model [198, 331] to consider compressible air-effects, and increasing the resolution of simulations, where smaller amplitude waves and the still water depth can be resolved more accurately. Here, the mean still-water-level was 0.005 m lower than that of the experiments, and thus a mismatch of volume, due to the initial particle arrangement. Adopting a different arrangement or packing algorithm [165] could help to better represent the correct volume of fluid. For increased resolutions or problems requiring larger particle numbers, the memory usage of Incompressible-DualSPHysics requires optimisation. Alternatively, a multi-GPU implementation of ISPH would accommodate for 10s of millions of particles using 2-4 GPUs working in parallel on a single machine.

The following and final chapter of this thesis summarises and draws conclusions of the research conducted in this study, and recommendations for future research are made.

Chapter 7

Conclusions and Recommendations

7.1 General conclusions

This thesis developed an incompressible free-surface flow solver for violent hydrodynamic engineering applications by the novel methodology of accelerating the incompressible smoothed particle hydrodynamics (ISPH) method on a graphics processing unit (GPU). Smoothed particle hydrodynamics (SPH) is well-suited for violent free-surface flows because it implicitly handles highly non-linear phenomena, such as fragmentation. The method's independence of a mesh also allows for complex flows involving large deformations and discontinuities, which are present in many violent flows. ISPH in particular is ideal for such engineering applications, requiring accurate predictions of pressure, because of its ability to produce and maintain a near noise-free pressure field via the solution of a pressure Poisson equation (PPE).

For engineering applications however, very high numbers of particles are needed, and so the ISPH PPE subsequently requires the solution of extremely large sparse matrices which account for over 90% of the simulation time. This thesis has addressed the associated computational expense by means of hardware acceleration with a GPU. GPUs are relatively cheap, compact desktop devices with massively parallel architectures, highly suitable for the acceleration of SPH. The weakly-compressible SPH method has previously achieved speed ups of two orders of magnitude with the GPU. However, before now, there has been no rigorous attempt to implement ISPH on a GPU due to the increased complexity of the numerical algo-

rithm compared to WCSPH. Multiple challenges are presented and addressed within this work: (i) Constructing the Lagrangian ISPH PPE matrix on GPU streaming multiprocessors each time step due to moving computational points, (ii) Overcoming the GPU memory limitations for the inherently computationally expensive ISPH algorithm, (iii) Establishing a robust and accurate ISPH solid boundary condition suitable for parallel processing on the GPU, (iv) Exploiting fast linear algebra GPU libraries.

The open-source hybrid CPU-GPU WCSPH code DualSPHysics v4.0 [15] was modified considerably by converting the algorithm to execute the ISPH pressure projection step and combining the code with the open-source ViennaCL linear algebra library [16] for fast solutions of the PPE matrix on the GPU. The Marrone et al. [14] boundary condition was extended and applied to ISPH for the first time, requiring development for parallel execution on the GPU and application to the ISPH PPE. Addressing the GPU memory limitations were facilitated with mixed precision storage and computation. The ViennaCL library's GPU-based algebraic multigrid (AMG) preconditioner required modification for application to Lagrangian particle systems in ISPH.

The resulting GPU-accelerated code, Incompressible-DualSPHysics, enables simulations involving millions of particles to be computed in only a few hours. The target applications would be too impractical for other single-device CPU-based codes, taking several weeks or months to compute. The GPU was shown to provide simulation runtimes 18.2-25.3 times and 4.2-8.1 times faster than single and multi-threaded CPU-based codes respectively. In addition to determining speed ups, Incompressible-DualSPHysics has been rigorously evaluated. The code's accuracy and flexibility were validated through a series of demanding test cases. The practical use and performance of two kernels and two preconditioners for ISPH implemented on a GPU are also investigated. A profiling study showed Incompressible-DualSPHysics concentrates the majority of the GPU's processing power on solving the PPE.

Finally, the new Incompressible-DualSPHysics code was used to simulate a real-

engineering 3-D application of breaking focused-wave impacts on a surface-piercing cylinder for the first time. Simple extensions to the numerical methodology and code were made to enable wave propagation and impact with the cylinder. Four cases (two non-breaking and two breaking) of focused-wave groups impacting a cylinder were investigated, and the results compared against experimental data [20] and 2-D ISPH results [191]. Post-processing analysis, which cannot be conducted by means of experiments or 2-D simulations, revealed the similarities and differences between non-breaking and breaking waves during the fluid-structure impact events providing new insight into the physical processes.

7.2 Detailed conclusions

7.2.1 Implementing ISPH on the GPU

Addressing the challenges highlighted in Section 7.1 to implement ISPH on GPU required several new developments.

7.2.1.1 The pressure projection step

The ISPH pressure projection step was implemented as four stages: Stage 1 - Intermediate step, Stage 2 - Setup and solve PPE, Stage 3 - Corrector step, and Stage 4 - Particle shifting. Each stage featured a distinct particle sweep.

In Stage 2, a single-threaded GPU function, to setup the PPE matrix storage arrays, is detailed prior to presenting parallel GPU algorithms, for fluid and boundary particles, regarding the matrix population. Populating the matrix with separate algorithms for fluid and boundary particles reduces thread-latencies due to their different neighbourlist sizes. After population, the matrix is given to, and solved by, the ViennaCL linear algebra library [16]. The solution (pressure field) is then returned to Incompressible-DualSPHysics for the calculation of pressure gradients in Stage 3.

7.2.1.2 Applying the Marrone et al. boundary condition

The Marrone et al. [14] boundary condition, originally formulated for WCSPH, mirrors boundary particle positions across the physical boundary line into the fluid domain to obtain unique interpolation points (UIP). An SPH MLS interpolation kernel determines fluid properties at each UIP, which are used by corresponding boundary particles to impose desired boundary conditions. Herein, the boundary condition was applied to ISPH for the first time, requiring modification for use within the PPE by evaluating the pressure at boundary particles, and their respective UIPs, with use of the MLS interpolation kernel. This is an additional equation introduced to the PPE matrix system, in contrast to the Laplacian operator used for fluid particles. The use of the two equations to construct the linear system avoids the occurrence of an additional expensive nested for-loop and allows for parallel population of the matrix by explicitly including both boundary and fluid particles in the system.

The established boundary condition can be efficiently parallelised on the GPU for ISPH and is capable of representing complex geometries and rigid moving bodies. However, the current boundary conditions cannot deal with a situation where surface of different objects are close to each other. This is because the UIPs, which are positioned away from the object surface, would interpolate from within another object where there are no fluid particles. Consequently, non-physical quantities and instabilities may occur from inaccurate kernel interpolations.

7.2.1.3 Mixed precision storage and computation

Memory consumption is reduced by means of mixed precision storage where the majority of particle data is stored in single precision. Otherwise, double precision is used to store particle positions, maintain accuracy in the kernel and during particle advection, and the matrix storage arrays, required for the linear solver to converge with large particle numbers. The PPE matrix is represented by the compressed sparse row (CSR) format where only the non-zero elements of the sparse matrix are stored, reducing the number of retained elements from npm^2 to a value proportional

to npm , where npm is the number of particles in the matrix.

GPU computations are executed through a mixed precision computation model. Most computations are performed with single precision because Nvidia GPUs exhibit high single-precision FLOPs. Fast single precision register memory variables are used during intermediary computations. Double precision register variables are utilised when computations require a summation of values, followed by a division, such as within the inverse of matrices during the calculation of MLS kernel coefficients or the kernel gradient normalisation variables. Once computation is finished, the value of a double precision register is cast to single precision for global memory storage.

The mixed precision model achieves a balance between speed and accuracy, however it can be difficult to maintain consistency during implementation. The complete use of double precision computation and storage will require new methods to address the memory limitations, a rearrangement of the code structure, and new algorithms to improve double precision FLOPs performance with a GPU. A single precision only model would significantly reduce memory consumption for large numbers of particles but require computational methods to recover loss in accuracy. This particularly applies during the solution of the PPE matrix.

7.2.1.4 GPU-based algebraic multigrid preconditioner modification

Algebraic multigrid (AMG) preconditioners possess the property of mesh independence and therefore are highly scalable for increasing matrix sizes. This demands for the investigation of such a preconditioning technique for the solution of the ISPH PPE matrix when large numbers of particles are used.

To take advantage of the ViennaCL library's GPU-based maximum independent set (2) AMG preconditioner (MIS(2)AMG) for ISPH, the algorithm required modification to prevent an error in the preconditioner's setup phase. During the parallel computation of aggregates, if two particles share exactly the same neighbours (and including each other) there is the risk of coarse aggregate indices data being overwritten. This ultimately leads to errors in the next-level matrix system with divisions of zero. This was solved by including a conditional if-statement to

prevent the elimination of certain coarse aggregate indices data.

The need to modify the GPU-based MIS(2)AMG preconditioner for ISPH has exposed the complexity of solving a Lagrangian PPE matrix system. Further development of faster and more robust methods for the solution of such systems on a GPU is needed [271, 339].

7.2.2 Performance

The performance of the code was evaluated through CPU-GPU speed ups, memory consumption, a profiling study, and comparing options of SPH kernels and preconditioners for ISPH accelerated with a GPU. All simulations used a Bi-CGSTAB linear solver for the solution of the PPE matrix system.

The performance comparisons were conducted on two Nvidia GPU devices, a GTX 1070 and a Tesla K40c, and an Intel(R) Xeon(R) CPU E5-2640 v3 with 8 cores (16 threads). Two types of CPU runs were made, executing the CPU-equivalent code of the GPU algorithm, using either a single thread (serial computation) or 16 threads (OpenMP parallel computation). For all cases observed, the GTX 1070 was the fastest of the two GPUs and so speed-ups stated in this section are for that specific GPU.

The first 50 time steps of the 2-D incompressible flow around a moving square in a rectangular box were timed and repeated for various resolutions using the Jacobi preconditioner. For approximately 2 million fluid particles, GPU speed-ups of 27.5 times and 4.5 times were observed against the single-threaded and 16-threaded CPU times respectively.

Speed ups were also investigated using 2-D and 3-D dambreaks, timing the first 10 time steps of the simulation. Initialising the solver with an initial guess solution of the pressure field from the previous time step was found to reduce overall runtimes by up to 85% and 35% in 2D and 3D respectively. The GPU-CPU speed ups with solver initialisation were also greater. For 1.0-4.5 million fluid particles in 2D, the GPU was approximately 18.3-25.3 times faster than the single-threaded CPU runs, and about 4.6-8.1 times faster than the 16-threaded CPU timings. Similarly in

3D for up to 2 million fluid particles, the GPU observed runtimes 18.2-21.0 times faster than the serial CPU runs, and speed ups of 4.2-4.7 times over the parallel multi-threaded CPU timings.

Upon examination of the CUDA kernels in the new Incompressible-DualSPHysics code, it was found that the theoretical and achieved occupancy, and branch divergence of the functions were similar to those within the original DualSPHysics code. Therefore, it was determined the performance of Incompressible-DualSPHysics GPU code was maintained during conversion from WCSPH to ISPH. The main computational expense of ISPH remains to be the solution of the PPE after acceleration on a GPU. The greatest improvements in overall speed can be achieved with relatively little effort by investigating current linear solvers and preconditioners from various open-source GPU libraries.

The performance of the Jacobi and MIS(2) AMG preconditioners during a 2-D dambreak simulation were compared. Although the MIS(2) AMG preconditioner proved significantly faster in the initial stages of the flow, fragmentation of the fluid from impact on the far wall caused a significant increase of PPE solution time. The Jacobi preconditioner on the other hand, appeared to be more robust when dealing with discontinuities. This indicates the need for future investigations of the behaviour of preconditioning techniques during the solution of systems arising from meshless methods whereby the connectivity of computational points change with time.

7.2.3 Numerical wave basin

The new ISPH-DualSPHysics code was used as a numerical wave basin to simulate the application of focused breaking-wave impact on a vertical cylinder.

Extensions to the methodology are made for the application of the numerical wave basin, these include:

- Additional terms to the Laplacian Morris operator for implementation of the Schwaiger operator [156] for improved accuracy near the free-surface.

- Defining the normal direction to the free-surface with a kernel-weighted summation to reduce error propagation with particle shifting near the interface.
- Excluding boundary particles above the free surface from particle interactions to minimise false summations.
- Adopting a radial arrangement of particles to represent a cylindrical structure for improved accuracy and prevention of spurious pressures. This is contrary to the Cartesian-based configuration produced by DualSPHysics' pre-processing software, GenCase.

A convergence study of the wave generation model shows a convergence rate of 1.78 for the L^2 error norm of the peak free-surface elevation at the focal point, compared to linear theory for a low amplitude JONSWAP spectrum focused wave group. However, this is achieved only when the particle spacing is sufficiently fine to resolve accurately the still-water depth and propagating wave.

For four focused wave groups (two non-breaking and two breaking), measurements of the free-surface elevation and cylinder forcing show acceptable agreement with the experimental data of Zang et al. [20]. The sources of error are unclear due to insufficient information about the physical experiments. Potential solutions are the use of a finer resolution, or modelling the cylinder as a non-rigid structure, as high frequency oscillations in the forcings were observed in the experimental data. Snapshots of a simulation featuring the impact of a breaking-wave with the cylinder show high levels of fragmentation at the free surface which cannot be easily achieved with a mesh-based method.

Post-processing analysis reveals the variation of the free-surface elevation around the cylinder is significant particularly in steep waves with a correspondingly large force contribution from hydrostatic pressure. While breaking does not have a marked effect on total force, the associated high particle velocities amplify local pressures considerably on the cylinder near the water surface.

For a particle spacing, $dp = 0.0125$ m, over five million particles were used to model the numerical wave basin and a total of 6.13 GB of memory was assigned to

the GPU of which approximately 80% was required for the PPE matrix. To improve the accuracy of the wave propagation model, the inclusion of an air phase or high resolutions are suggested, although both would require work to reduce the memory consumption of the code, or a multi-GPU implementation to remove the memory limitations of a single GPU device as recommended later the Section 7.3.4.

7.3 Recommendations for future research

The work presented in this thesis has been shown to provide a robust GPU-accelerated ISPH solver demonstrating significant speed ups over CPU-based codes and facilitated the practicality of the ISPH method for engineering applications. However, this is the first time ISPH on the GPU has been researched and there are numerous possibilities for improving the numerical model, developing the algorithm, and applying the model to challenging cases.

7.3.1 Algorithmic developments

Regarding the algorithmic implementation of ISPH on the GPU, there are several areas that can be explored to reduce the total memory consumption of the current model and/or speed of the code.

Due to the memory limitations of the GPU, the original intention of this study was to use single precision. However, it was found that for systems of more than 500,000 particles, double precision computation accuracy is required for convergence of the linear solver. This required double precision storage of the PPE matrix elements by the ViennaCL library. As seen in Chapter 6, PPE matrix memory requirements for a 3-D simulation of approximately 5 million particles used about 60% of the total GPU memory assigned by Incompressible-DualSPHysics. Research into reducing the memory storage requirements of the PPE matrix is therefore recommended, including algorithms that enable linear solver convergence with single precision storage for large particle numbers.

Leroy [190] did not store the PPE matrix, and instead computed matrix-vector

products during the execution of the linear solver. However, this can be time consuming (particularly in 3D) requiring the computation of particle-particle interactions multiple times within each solver iteration. This problem could potentially benefit from the use of “CUDA dynamic parallelism”, where threads within a `_global_` kernel function can launch their own `_global_` kernels.

However, memory management and latencies, and thread synchronisation pose further challenges. Nevertheless, dynamic parallelism could at least be explored for accelerating the linear solver as shown by Aliaga et al. [340].

Memory consumption can also be reduced through means of adaptivity. The numerical wave basin in Chapter 6 required millions of particles due to the large domain size, yet the actual area of application interest which requires a fine resolution is relatively small. This may include variable resolution [163, 341] or higher order Eulerian-Lagrangian ISPH schemes [203, 206]

Murotani et al. [261] developed algorithms for MPS on the GPU to improve the efficiency and speed of neighbour searching and particle interactions during the computation of differential operators. With a focus on the arrangement and locality of particle-data, the same principles in their work can be applied to SPH. Further considerations should follow recent works [342–344] focused on analysing and improving the accuracy of the Laplacian operators for the PPE, which can have an impact on the conditioning of the resultant linear systems.

7.3.2 Linear solvers and preconditioners for ISPH

The work conducted in this study for implementing ISPH on the GPU has drawn attention to the complexities of solving the ISPH PPE efficiently. Consequently, new areas of research concerning the solution of the ISPH PPE have been identified.

The ViennaCL linear algebra library was selected here to solve the PPE for its ease of coupling with DualSPHysics and functionality. CPU and GPU algorithms within the library allowed for direct speed-up comparisons, and the range of linear solvers and preconditioners. Within the library there are many more algorithms, besides those tested here, which can be investigated for solving the PPE matrix.

Moreover, there are numerous other GPU-based linear algebra libraries [269, 270, 315, 345, 346] with varying ranges of functionality. Investigating all of the options is a simple exercise which could lead to faster execution times with the current Incompressible-DualSPHysics code.

In Section 5.6.2, the 2-D runtimes of the MIS(2)AMG preconditioner showed linear scalability and PPE solution times much faster than the Jacobi (without initialisation). Unfortunately, convergence slowed down considerably in a fragmented flow and also showed poor scalability in 3D. AMG methods usually require specialist manipulation and knowledge of a problem for the best performance. Development of a parallel AMG preconditioner tailored for ISPH applications would be a significant advancement for the numerical method.

7.3.3 Improving the physical model

The proposed ISPH methodology has been proven robust and applicable for a range of free-surface flows, but there are still numerous developments that could enhance the accuracy and enable the wider range of applications for the model.

Multi-phase SPH models [248, 331] have shown the presence of an air phase in violent hydrodynamic applications has a marked effect on the accuracy of the flow mechanics and pressure field.

Incorporating a multi-phase model would improve predictions of flows such as wave propagation and impact, tank sloshing, and channel flows.

The presented boundary condition is suitable for rigid solid boundaries of various geometries, with the possibility of linear motion. To extend the capabilities towards the wider range of engineering applications, floating bodies and rotational motion can be simply included by using the code from the original DualSPHysics. However, further development of the the Marrone et al. [14] boundary condition for situations involving interactions between multiple independent boundaries is required. Ultimately, one would be able to simulate applications involving multiple bodies, floating and non-floating.

High frequency oscillations observed from the forcing data of Zang et al. [20],

in Chapter 6, has been speculated to arise from structural response. Including a flexible boundary model would confirm this [204].

7.3.4 Multi-GPU implementation

Whilst this work has achieved the computation of millions of particles on a single GPU, industrial applications still require many more on the order of 10^7 to 10^9 . The physical memory of a GPU is limited and non-expandable, affecting scalability and the maximum allowable number of particles. This can be addressed with the development of a multi-GPU ISPH code. WCSPH accelerated by multiple interconnected GPUs has already been achieved through use of MPI processes for communication between devices [222, 251].

Modern day motherboards can support four to six GPU devices, and so simulations of 100 million particles can be achieved for ISPH on a single desktop if the memory requirements of the code are also reduced using the suggestions from Section 7.3.1.

7.3.5 Suggestions of future applications

The proposed methodology has been successfully applied to the dambreak case and a numerical wave basin involving breaking-waves. Incompressible-DualSPHysics opens up a vast potential of new 3-D applications previously too impractical:

- In addition to wave impacts with structures (cylinder arrays, floating devices etc.) other than that in Chapter 6, the numerical wave basin can be used to simulate flow around moving vessels. Computational models investigate the complex hydrodynamic phenomena created by a moving ship hull in water in the presence, or absence, of waves.
- In coastal engineering there is the need to assess fluid-structure impact loading and flow behaviour for breakwaters, bridges, and buildings etc.
- In earthquake engineering, the code can be used to investigate the behaviour of the fluid inside water towers or buildings with liquid tuned dampers. Sim-

ulation of forces exerted on the structure by the violent motion of the fluid during an earthquake event can help optimise design. Similar applications include anti-roll tanks and liquid transportation vessels.

- In open-channel flows, weirs, fish passes, spillways etc. can be modelled with high resolutions in 3D to give full detail of flow behaviour for engineering design. Additionally, impact loading on the infrastructure during violent flash-flood events can be predicted.

Incompressible-DualSPHysics provides the means to simulate a vast amount of complex and violent hydrodynamic engineering applications in 3D enabled by acceleration on the GPU. The list is non-exhaustive, and the presented methodology can be applied to many areas of industrial/research applications. The methodology also provides a basis for expansion into other areas such as multi-phase flows.

References

- [1] H. G. Weller, G. Tabor, H. Jasak, C. Fureby, A tensorial approach to computational continuum mechanics using object-oriented techniques, *Comput. Phys.* 12 (6) (1998) 620–631.
- [2] ANSYS, ANSYS - Fluids, <http://www.ansys.com/en-gb/products/fluids> (2018) [Online accessed: 28/03/2018].
- [3] Siemens, STAR-CCM+, <https://mdx.plm.automation.siemens.com/star-ccm-plus> (2018) [Online accessed: 07/01/2018].
- [4] R. A. Gingold, J. J. Monaghan, Smoothed particle hydrodynamics: Theory and application to non-spherical stars, *Mon. Not. R. Astron. Soc.* 181 (1977) 375–389.
- [5] L. Lucy, A numerical approach to the testing of the fission hypothesis, *Astron. J.* 82 (1977) 1013–1024.
- [6] J. J. Monaghan, Simulating free surface flows with SPH, *J. Comput. Phys.* 110 (1994) 399–406.
- [7] S. Shao, Simulation of breaking wave by SPH method coupled with $k-\epsilon$ model, *J. Hydraul. Res.* 44 (3) (2006) 338–349.
- [8] S. M. Longshaw, B. D. Rogers, Automotive fuel cell sloshing under temporally and spatially varying high acceleration using GPU-based Smoothed Particle Hydrodynamics (SPH), *Adv. Eng. Softw.* 83 (2015) 31–44.
- [9] A. Skillen, S. J. Lind, P. K. Stansby, 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. Method Appl. Mech. Eng.* 265 (2013) 163–173.
- [10] A. J. C. Crespo, C. Altomare, J. M. Domínguez, J. González-Cao, M. Gómez-Gesteira, Towards simulating floating offshore oscillating water column converters with Smoothed Particle Hydrodynamics, *Coast. Eng.* 126 (2017) 11–26.
- [11] S. J. Lind, R. Xu, P. K. Stansby, B. D. Rogers, Incompressible smoothed particle hydrodynamics for free-surface flows: A generalised diffusion-based algorithm for stability and validations for impulsive flows and propagating waves, *J. Comput. Phys.* 231 (2012) 192–205.
- [12] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable Parallel Programming with CUDA, *Queue - GPU Computing* 6 (2) (2008) 40–53.
- [13] J. E. Stone, D. Gohara, G. Shi, OpenCL: A Parallel Programming Standard For Heterogeneous Computing Systems, *Comput. Sci. Eng.* 12 (3) (2010) 66–73.
- [14] S. Marrone, M. Antuono, A. Colagrossi, G. Colicchio, A. Le Touzé, G. Graziani, δ -SPH model for simulating violent impact flows, *Comput. Methods Appl. Mech. Eng.* 200 (2011) 1526–1542.
- [15] A. J. C. Crespo, J. M. Domínguez, B. D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, O. García-Feal, DualSPHysics: open-source parallel CFD solver on Smoothed Particle Hydrodynamics (SPH), *Comput. Phys. Commun.* 187 (2015) 204–216.
- [16] K. Rupp, J. Weinbub, 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*, 2010, pp. 1–6.
- [17] D. H. Peregrine, Flow due to a vertical plate moving in a channel, *Personal Communication*.

- [18] A. Colagrossi, 6th SPHERIC benchmark test case, <http://spheric-sph.org/tests/test-6> (2011) [Online accessed: 20/07/2017].
- [19] S. Koshizuka, Y. Oka, Moving-Particle Semi-Implicit Method for Fragmentation of Incompressible Fluid, *Nucl. Sci. Eng.* 123 (1996) 421–434.
- [20] J. Zang, P. H. Taylor, G. Morgan, R. Stringer, J. Orszaghova, J. Grice, M. Tello, Steep wave and breaking wave impact on offshore wind turbine foundations - ringing revisited, in: 25th International Workshop on Water Waves and Floating Bodies, Harbin, 2010.
- [21] R. G. Dean, R. A. Dalrymple, *Advanced Series on Ocean Engineering - Volume 2: Water Wave Mechanics for Engineers and Scientists*, World Scientific, 1991.
- [22] A. Schmidt, Breaking Wave, <https://www.publicdomainpictures.net/en/view-image.php?image=2766&picture=breaking-wave> (nd) [Online accessed: 08/08/2018].
- [23] R. G. Dean, R. A. Dalrymple, *Coastal Processes with Engineering Applications*, Cambridge University Press, 2004.
- [24] S. Chandrasekaran, *Dynamic Analysis and Design of Offshore Structures*, Second Edition, Springer, 2018.
- [25] R. A. Ibrahim, *Liquid Sloshing Dynamics: Theory and Applications*, Cambridge University Press, UK, 2005.
- [26] A. Souto-Iglesias, L. Delorme, L. Pérez-Rojas, S. Abril-Pérez, Liquid moment amplitude assessment in sloshing type problems with smoothed particle hydrodynamics, *Ocean Eng.* 33 (2006) 1462–1484.
- [27] T. S. Katko, J. J. Hukka, Social and Economic Importance of Water Services in the Built Environment: Need for more structured Thinking, *Proc. Econ. Financ.* 21 (2015) 217–223.

- [28] S. M. Husain, J. R. Muhammed, H. U. Karunaratha, D. E. Reeve, Investigation of pressure variations over stepped spillways using smooth particle hydrodynamics, *Adv. Water Resour.* 66 (2014) 52–69.
- [29] I. Federico, S. Marrone, A. Colagrossi, F. Aristodemo, M. Antuono, Simulating 2D open-channel flows through an SPH model, *Eur. J. Mech. B-Fluid.* 34 (2012) 35–46.
- [30] J. D. Anderson, Jr, *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill Book Co., 1995.
- [31] A. Mokos, Multi-phase modelling of violent hydrodynamics using smoothed particle hydrodynamics (SPH) on graphics processing units (GPUs), Ph.D. thesis, The University of Manchester (2013).
- [32] J. Tu, G.-H. Yeoh, C. Liu, *Computational Fluid Dynamics: A Practical Approach*, Third Edition, Butterworth-Heinemann, 2018.
- [33] M. W. Evans, F. H. Harlow, E. Bromberg, *The Particle-In-Cell Method for Hydrodynamic Calculations*, Tech. rep., Los Alamos National Lab (1957).
- [34] F. H. Harlow, Hydrodynamic Problems Involving Large Fluid Distortions, *J. ACM* 4 (2) (1957) 137–142.
- [35] S. Mckee, M. F. Tomé, V. G. Ferreira, J. A. Cuminato, A. Castelo, F. S. Sousa, N. Mangiavacchi, The MAC method, *Comput. Fluids* 37 (2008) 907–930.
- [36] J. U. Brackbill, H. M. Ruppel, FLIP: A Method for Adaptively Zoned, Particle-in-Cell Calculations of Fluid Flows in Two Dimensions, *J. Comput. Phys.* 65 (1986) 314–343.
- [37] J. U. Brackbill, D. B. Kothe, H. M. Ruppel, FLIP: A Low-Dissipation, Particle-In-Cell Method for Fluid Flow, *Comput. Phys. Commun.* 48 (1988) 25–38.

- [38] Q. Chen, J. Zang, D. M. Kelly, A. S. Dimakopoulos, A 3D parallel Particle-In-Cell solver for wave interaction with vertical cylinders, *Ocean Engineering* 147 (2018) 165–180.
- [39] F. H. Harlow, J. E. Welch, Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface, *Phys. Fluids* 8 (12) (1965) 2182–2189.
- [40] J. E. Welch, F. H. Harlow, S. J. P., B. J. Daly, The MAC method, Tech. rep., Los Alamos National Lab (1965).
- [41] F. S. de Sousa, N. Mangiavacchi, L. G. Nonato, A. Castelo, M. F. Tomé, V. G. Ferreira, J. A. Cuminato, S. McKee, A front-tracking/front-capturing method for the simulation of 3D multi-fluid flows with free surfaces, *J. Comput. Phys.* 198 (2004) 469–499.
- [42] C. W. Hirt, B. D. Nichols, Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries, *J. Comput. Phys.* 39 (1981) 201–225.
- [43] N. Ashgriz, J. Y. Poo, FLAIR: Flux Line-Segment Model for Advection and Interface Reconstruction, *J. Comput. Phys.* 93 (1991) 449–468.
- [44] M. Rudman, A Volume-Tracking Method for Incompressible Multifluid Flows with Large Density Variations, *Int. J. Numer. Meth. Fluids* 28 (1998) 357–378.
- [45] D. Gueyffier, J. Li, A. Nadim, R. Scardovelli, S. Zaleski, Volume-of-Fluid Interface Tracking with Smoothed Surface Stress Methods for Three-Dimensional Flows, *J. Comput. Phys.* 152 (1999) 423–456.
- [46] J. E. Pilliod Jr., E. G. Puckett, Second-order accurate volume-of-fluid algorithms for tracking material interfaces, *J. Comput. Phys.* 199 (2004) 465–502.
- [47] S. Osher, J. A. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *J. Comput. Phys.* 79 (1) (1988) 12–49.

- [48] M. Sussman, E. G. Puckett, A Coupled Level Set and Volume-of-Fluid Method for Computing 3D and Axisymmetric Incompressible Two-Phase Flows, *J. Comput. Phys.* 162 (2) (2000) 301–337.
- [49] X. Yang, A. J. James, J. Lowengrub, X. Zheng, V. Cristini, An adaptive coupled level set and volume-of-fluid interface capturing method for unstructured triangular grids, *J. Comput. Phys.* 217 (2) (2006) 364–394.
- [50] D. L. Sun, W. Q. Tao, A coupled volume-of-fluid and level set (VOSET) method for computing incompressible two-phase flows, *J. Comput. Phys.* 53 (4) (2010) 645–655.
- [51] J.-C. Marongiu, F. Leboeuf, J. Caro, E. Parkinson, Free surface flows simulations in Pelton turbines using an hybrid SPH-ALE method, *J. Hydraul. Res.* 48 (2010) 40–49.
- [52] C. W. Hirt, J. Amsden, J. L. Cook, An arbitrary lagrangian-eulerian computing method for all flow speeds, *J. Comput. Phys.* 14 (1974) 227–253.
- [53] J. Donea, A. Huerta, J.-P. Ponthot, A. Rodriguez-Ferran, Arbitrary Lagrangian-Eulerian Methods, in: *Encyclopaedia of Computational Mechanics*, John Wiley & Sons, Ltd, 2004, Ch. 14.
- [54] J. Baiges, R. Codina, A. Pont, E. Castillo, An adaptive Fixed-Mesh ALE method for free-surface flows, *Comput. Meth. Appl. Mech. Eng.* 313 (2017) 159–188.
- [55] G. R. McNamara, G. Zanetti, Use of the Boltzmann Equation to Simulate Lattice-Gas Automata, *Phys. Rev. Lett.* 61 (1988) 2332.
- [56] F. J. Higuera, J. Jiménez, Boltzmann Approach to Lattice Gas Simulations, *Europhys. Lett.* 9 (1989) 663.
- [57] J. Hardy, Y. Pomeau, O. d. Pazzis, Time evolution of a two-dimensional model system. I. Invariant states and time correlation functions, *J. Math. Phys.* 14 (1973) 1746–1759.

- [58] J. G. Zhou, A lattice Boltzmann model for the shallow water equations, *Comput. Meth. Appl. Mech. Eng.* 191 (32) (2002) 3527–3539.
- [59] C. Janssen, M. Krafczyk, A lattice Boltzmann approach for free-surface-flow simulations on non-uniform block-structured grids, *Comput. Math. Appl.* 59 (7) (2010) 2215–2235.
- [60] D. Anderi, S. Bogner, C. Rauh, U. Rüde, A. Delgado, Free surface lattice Boltzmann with enhanced bubble model, *Comput. Math. Appl.* 67 (2) (2014) 331–339.
- [61] D. A. Perumal, A. K. Dass, A Review on the development of lattice Boltzmann computation of macro fluid flows and heat transfer, *Alexandria Eng. J.* 54 (4) (2015) 955–971.
- [62] H. Amirshaghghi, M. H. Rahimian, H. Safari, Application of a two phase lattice Boltzmann model simulation of free surface jet impingement heat transfer, *Int. Commun. Heat Mass Transf.* 75 (2016) 282–294.
- [63] W. Wang, Development and Application of Lattice Boltzmann Method for Complex Axisymmetric Flows, Ph.D. thesis, The University of Liverpool (2015).
- [64] K. Mattila, J. Hyväluoma, J. Timonen, T. Rossi, Comparison of implementations of the lattice-Boltzmann method, *Comput. Math. Appl.* 55 (7) (2008) 1514–1524.
- [65] M. Sheida, M. Taeibi-Rahni, V. Esfahanian, A New Approach to Reduce Memory Consumption in Lattice Boltzmann Method on GPU, *J. Appl. Fluid Mech.* 10 (1) (2017) 55–67.
- [66] E. Group, OpenFOAM - The open source CFD toolbox, <https://www.openfoam.com/> (2018) [Online accessed: 28/03/2018].

- [67] T. Belytschko, Y. Krongauz, D. Organ, P. Fleming, M. Krysl, Meshless methods: An overview and recent developments, *Comput. Meth. Appl. Mech. Eng.* 139 (1996) 3–47.
- [68] J. J. Monaghan, Smoothed Particle Hydrodynamics, *Annu. Rev. Astron. Astrophys.* 30 (1992) 543–574.
- [69] S. N. Atluri, T. Zhu, A new Meshless Local Petrov-Galerkin (MLPG) approach in computational mechanics, *Comput. Mech.* 22 (1998) 117–127.
- [70] G. R. Liu, Y. T. Gu, Meshless Local Petrov-Galerkin (MLPG) method in combination with finite element and boundary element approaches, *Comput. Mech.* 26 (6) (2000) 536–546.
- [71] Q. W. Ma, Meshless Local Petrov-Galerkin method for two-dimensional nonlinear water wave problems, *J. Comput. Phys.* 205 (2005) 611–625.
- [72] V. Sriram, Q. W. Ma, Improved MLPG_R method for simulating 2D interaction between violent waves and elastic structures, *J. Comput. Phys.* 231 (2012) 7650–7670.
- [73] M. Najafi, A. Aremanesh, V. Enjilela, Meshless local Petrov-Galerkin method - higher Reynolds numbers fluid flow applications, *Eng. Anal. Bound. Elem.* 36 (2012) 1671–1685.
- [74] H. Lin, S. N. Atluri, The Meshless Local Petrov-Galerkin (MLPG) Method for Solving Incompressible Navier-Stokes Equations, *Tech Sci. Press* 2 (2) (2001) 117–142.
- [75] G. R. Liu, Y. T. Gu, A Local Radial Point Interpolation Method (LRPIM) For Free Vibration Analyses Of 2-D Solids, *J. Sound Vib.* 246 (1) (2001) 29–46.
- [76] L. Y. Wu, G. R. Liu, A meshfree formulation of local radial point interpolation method (LRPIM) for incompressible flow simulation, *Comput. Mech.* 30 (2003) 355–365.

- [77] J. G. Wang, L. Yan, G. R. Liu, A local radial point interpolation method for dissipation process of excess pore water pressure, *Int. J. Numer. Meth. Heat & Fluid Flow* 15 (6) (2004) 567–587.
- [78] B. Nayroles, G. Touzot, P. Villon, Generalizing the finite element method: Diffuse approximation and diffuse elements, *Comput. Mech.* 10 (1992) 307–318.
- [79] Y. Kronguaz, T. Belytschko, Consistent pseudo-derivatives in meshless methods, *Comput. Meth. Appl. Mech. Eng.* 146 (1997) 371–386.
- [80] A. Arefmanesh, M. Najafi, H. Abdi, A Meshless Local Petrov-Galerkin Method for Fluid Dynamics and heat Transfer Applications, *J. Fluids Eng.* 127 (2005) 647–655.
- [81] Y. Kronguaz, T. Belytschko, A Petrov-Galerkin Diffuse Element Method (PGDEM) and its comparison to EFG, *Computational Mechanics* 19 (1997) 327–333.
- [82] P. J. Hoogerbrugge, J. M. V. A. Koelman, Simulating Microscopic Hydrodynamic Phenomena with Dissipative Particle Dynamics, *EPL* 19 (3) (1992) 155–160.
- [83] B. J. Alder, T. E. Wainwright, Studies in Molecular Dynamics. I. General Method, *J. Chem. Phys.* 31 (2) (1959) 459–466.
- [84] P. Español, P. Warren, Statistical Mechanics of Dissipative Particle Dynamics, *EPL* 30 (4) (1995) 191–196.
- [85] A. J. Chorin, Numerical study of slightly viscous flow, in: *J. Fluid Mech.*, Vol. 57, 1973, pp. 785–796.
- [86] M. Perlman, On the Accuracy of Vortex Methods, *J. Comput. Phys.* 59 (1985) 200–223.
- [87] G.-H. Cottet, P. Koumoutsakos, M. L. O. Salihi, Vortex Methods with Spatially Varying Cores, *J. Comput. Phys.* 162 (2000) 164–185.

- [88] A. Ojima, K. Kamemoto, Vortex method simulation of 3D and unsteady vortices in a swirling flow apparatus experimented in "Politechnica University" of Timisoara, in: IOP Conference Series: Earth and Environmental Science, Vol. 12, 2010.
- [89] P. Español, M. Revenga, Smoothed dissipative particle dynamics, *Phys. Rev. E* 67 (2003) 026705.
- [90] J. Kuhnert, General Smoothed Particle Hydrodynamics, Ph.D. thesis, Department of Mathematics, University of Kaiserslautern (1999).
- [91] S. Tiwari, J. Kuhnert, A Meshfree Method For incompressible Fluid Flows with Incorporated Surface Tension, *J. Comput. Appl. Math.* 11 (2002) 965–987.
- [92] C. Drumm, S. Tiwari, J. Kuhnert, H.-J. Bart, Finite pointset method for simulation of the liquid-liquid flow field in an extractor, *Comput. Chem. Eng.* 32 (2008) 2946–2957.
- [93] B. Seibold, Multigrid and M-Matrices in the Finite Pointset Method for Incompressible Flows, in: Griebel, M. and Schweitzer, M. A. (Ed.), *Meshfree Methods for Partial Differential Equations III*, Vol. 57, Springer, Bonn, Germany, 2007, pp. 219–234.
- [94] A. Souto-Iglesias, F. Marcià, L. M. González, J. L. Cercos-Pita, On the consistency of MPS, *Comput. Phys. Commun.* 184 (3) (2013) 732–745.
- [95] A. Souto-Iglesias, F. Marcià, L. M. González, J. L. Cercos-Pita, Addendum to "On the consistency of MPS" [*Comput. Phys. Comm.* 184 (3) (2013) 732-745, *Comput. Phys. Commun.* 185 (2) (2014) 595–598.
- [96] S. Natsui, H. Takai, T. Kumagai, T. Kikuchi, R. O. Suzuki, Stable mesh-free moving particle semi-implicit method for direct analysis of gas-liquid two-phase flow, *Nuclear Science and Engineering* 123 (2014) 421–434.

- [97] H. Imanian, M. Kolahdoozan, A. R. Zarrati, Waves Simulation in Viscous Waters Using MPS, in: *Communication Software and Networks (ICCSN)*, 2011 IEEE 3rd International Conference on, 2011, pp. 264–268.
- [98] A. Khayyer, H. Gotoh, Modified Moving Particle Semi-implicit methods for the prediction of 2D wave impact pressure, *Coastal Engineering* 56 (2009) 419–440.
- [99] V. Springel, Smoothed Particle Hydrodynamics in Astrophysics, *Annu. Rev. Astron. Astrophys.* 48 (2010) 391–430.
- [100] L. D. Libersky, A. G. Petschek, T. C. Carney, J. R. Hipp, F. A. Allahdadi, High Strain Lagrangian Hydrodynamics: A Three-Dimensional SPH Code for Dynamic Material Response, *J. Comput. Phys.* 109 (1993) 67–75.
- [101] J. P. Gray, J. J. Monaghan, R. P. Swift, SPH elastic dynamics, *Comput. Methods Appl. Mech. Eng.* 190 (2001) 6641–6662.
- [102] K. Sugiura, S.-i. Inutska, An extension of Godunov SPH II: Application to elastic dynamics, *J. Comput. Phys.* 333 (2017) 78–103.
- [103] P. W. Cleary, J. J. Monaghan, Conduction Modelling Using Smoothed Particle Hydrodynamics, *J. Comput. Phys.* 148 (1999) 227–264.
- [104] A. W. AlShaer, B. D. Rogers, L. Li, Smoothed Particle Hydrodynamics (SPH) modelling of transient heat transfer in pulsed laser ablation of Al and associated free-surface problems, *Comput. Mater. Sci.* 127 (2017) 161–179.
- [105] M. Farrokhpanah, A. Bussmann, J. Mostaghimi, New smoothed particle hydrodynamics (SPH) formulation for modeling heat conduction with solidification and melting, *Numer. Heat Tr. B-Fund.* 71 (2017) 299–312.
- [106] G. R. Johnson, R. A. Stryk, S. R. Beissel, SPH for high velocity impact computations, *Comput. Meth. Appl. Mech. Eng.* 139 (1996) 347–373.

- [107] H. Asadi Kalameh, A. Karamali, C. Anitescu, T. Rabczuk, High velocity impact of metal sphere on thin metallic plate using smooth particle hydrodynamics (SPH) method, *Front. Struct. Civ. Eng.* 6 (2) (2012) 101–110.
- [108] A. Grimaldi, A. Sollo, M. Guida, F. Marulo, Parametric study of a SPH high velocity impact analysis - A birdstrike windshield application, *Compos, Struct.* 96 (2013) 616–630.
- [109] G. K. Batchelor, *Introduction to Fluid Dynamics*, Cambridge University Press, 1974.
- [110] J. J. Monaghan, Gravity currents and solitary waves, *Physica D* 98 (1996) 523–533.
- [111] J. J. Monaghan, A. Kos, Solitary waves on a cretan beach, *J. Waterw. Port Coast. Ocean Eng.* 125 (1999) 145–154.
- [112] M. P. Tulin, M. Landrini, Breaking waves in the ocean and around Ships, in: *Proceedings of the 23rd ONR Symposium on Naval Hydrodynamics*, 2000, pp. 713–745.
- [113] R. A. Dalrymple, O. Knio, SPH modelling of water waves, in: *Fourth Conference on Coastal Dynamics*, 2001, pp. 779–787.
- [114] Y. Li, F. Raichlen, Energy Balance Model for Breaking Solitary Wave Runup, *J. Waterw. Port Coast. Ocean Eng.* 129 (2001) 47–59.
- [115] B. D. Rogers, R. A. Dalrymple, SPH modeling of breaking waves, in: *Proceedings of the 29th International Conference on Coastal Engineering*, 2004, pp. 415–427.
- [116] M. H. Dao, H. Xu, E. S. Chan, P. Tkalich, Numerical modelling of extreme waves by Smoothed Particle Hydrodynamics, *Nat. Hazards Earth Syst. Sci.* 11 (2011) 419–429.

- [117] R. A. Dalrymple, O. Knio, D. T. Cox, M. Gesteira, S. Zou, Using a Lagrangian Particle Method for Deck Overtopping, in: Proceedings of the Fourth International Symposium on Ocean Wave Measurement and Analysis, 2001, pp. 1082–1091.
- [118] M. Gómez-Gesteira, D. Cerqueiro, C. Crespo, R. A. Dalrymple, Green water overtopping analyzed with a SPH model, *Ocean Eng.* 32 (2005) 223–238.
- [119] A. Panizzo, R. A. Dalrymple, SPH modelling of Underwater Landslide Generated Waves, in: Proceedings of the 29th International Conference on Coastal Engineering, 2004, pp. 1147–1159.
- [120] B. D. Rogers, R. A. Dalrymple, SPH modeling of Tsunami Waves, in: Advances in Coastal and Ocean Engineering, 2008, pp. 75–100.
- [121] T. Capone, A. Panizzo, J. Monaghan, SPH modelling of water waves generated by submarine landslides, *J. Hydraul. Res.* 48 (2010) 80–84.
- [122] M. Aghili, P. Ghadimi, Y. F. Maghrebi, H. Nowruzi, Simulating the interaction of solitary wave and submerged horizontal plate using SPH method, *Int. J. Phys. Res.* 2 (2014) 16–26.
- [123] M. Hayatdavoodi, R. C. Ertekin, Nonlinear Forces on a Submerged, Horizontal Plate: The G-N Theory, in: Proceedings of the 27th International Workshop on Water Waves and Floating Bodies, 2012.
- [124] A. Iturrioz, R. Guanche, J. A. Armesto, M. A. Alves, C. Vidal, I. J. Losada, Time-domain modeling of a fixed detached oscillating water column towards a floating multi-chamber device, *Ocean Eng.* 76 (2014) 65–74.
- [125] A. Iturrioz, R. Guanche, J. L. Lara, C. Vidal, I. J. Losada, Validation of OpenFOAM for oscillating water column three-dimensional modelling, *Ocean Eng.* 107 (2015) 222–236.
- [126] C. Altomare, A. J. C. Crespo, J. M. Domínguez, M. Gómez-Gesteira, T. Suzuki, T. Verwaest, Applicability of Smoothed Particle Hydrodynamics

- for estimation of sea wave impact on coastal structures, *Coast. Eng.* 96 (2015) 1–12.
- [127] C. Altomare, J. M. Domínguez, A. J. C. Crespo, J. González-Cao, T. Suzuki, M. Gómez-Gesteira, P. Trocuj, Long-crested wave generation and absorption for SPH-based DualSPHysics model, *Coast. Eng.* 127 (2017) 37–54.
- [128] A. J. C. Crespo, M. Gómez-Gesteira, R. A. Dalrymple, Boundary Conditions Generated by Dynamic Particles in SPH Methods, *Comput., Mater., Con.* 5 (3) (2007) 173–184.
- [129] D. Violeau, R. Issa, Numerical modelling of complex turbulent free-surface flows with the SPH method: an overview, *Int. J. Numer. Meth. Fluids* 53 (2007) 277–304.
- [130] M. Gomez-Gesteira, B. D. Rogers, R. A. Dalrymple, A. J. C. Crespo, State-of-the-art of classical SPH for free-surface flows, *J. Hydraul. Res.* 48 (2010) 6–27.
- [131] A. J. C. Crespo, M. Gómez-Gesteira, R. A. Dalrymple, Modeling Dam Break Behaviour over a Wet Bed by a SPH Technique, *J. Waterw. Port Coast. Ocean Eng.* 134 (6) (2008) 313–320.
- [132] I. M. Jánosi, D. Jan, K. G. Szabó, T. Tél, Turbulent drag reduction in dam-break flows, *Exp. Fluids* 37 (2004) 219–229.
- [133] M. Gómez-Gesteira, R. A. Dalrymple, Using a Three-Dimensional Smoothed Particle Hydrodynamics Method for Wave Impact on a Tall Structure, *J. Waterw. Port Coast. Ocean Eng.* 130 (2004) 63–69.
- [134] K. Pan, R. H. A. IJzermans, B. D. Jones, A. Thyagarajan, B. W. H. van Beest, J. R. Williams, Application of the SPH method to solitary wave impact on an offshore platform, *Comput. Part. Mech* 196 (2015) 304–316.

- [135] P. E. Raad, R. Bidoae, The three-dimensional Eulerian-Lagrangian marker and micro cell method for the simulation of free surface flows, *J. Comput Phys.* 203 (2005) 668–699.
- [136] SPHERIC, SPHERIC: SPH European Research Interest Community, <http://spheric-sph.org/index> (2017) [Online accessed: 25/01/2018].
- [137] K. M. T. Kleefsman, G. Fekken, A. E. P. Veldman, B. Iwanowski, B. Buchner, A Volume-of-Fluid based simulation method for wave impact problems, *J. Comput Phys.* 206 (2005) 363–393.
- [138] A. C. Crespo, J. M. Dominguez, A. Barreiro, M. Gómez-Gesteira, B. D. Rogers, GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods, *PLoS ONE* 6 (6) (2011) e20685. doi:10.1371/journal.pone.0020685.
- [139] A. Mayrhofer, M. Ferrand, C. Kassiotis, D. Violeau, F.-X. Morel, Unified semi-analytical wall boundary conditions in SPH: analytical extension to 3-D, *Numer. Algor.* 68 (2015) 15–34.
- [140] A. Souto-Iglesias, L. Pérez-Rojas, R. R. Z., Simulation of anti-roll tanks and sloshing type problems with smoothed particle hydrodynamics, *Ocean Eng.* 31 (2004) 1169–1192.
- [141] H. G. Hornung, C. Willert, S. Turner, The flow field downstream of a hydraulic jump, *J. Fluid Mech.* 287 (1995) 299–316.
- [142] D. López, R. Marivela, L. Garrote, Smoothed particle hydrodynamics model applied to hydraulic structures: a hydraulic jump test case, *J. Hydraul. Res.* 48 (2010) 142–158.
- [143] W. H. Hager, M. Schwalt, Broad-Crested Weir, *J. Irrig. Drain Eng.* 120 (1994) 13–26.
- [144] I. Meireles, J. Matos, Skimming Flow in the Non-aerated Region of Stepped Spillways over Embankment Dams, *J. Hydraul. Eng.* 135 (2009) 685–689.

- [145] K. H. Frizsel, B. W. Melford, Designing Spillways to Prevent Cavitation Damage, *Concr. Int.* 13 (1991) 58–64.
- [146] A. Amador, M. Sánchez-Juny, J. Dolz, Developing Flow Region and Pressure Fluctuations on Steeply Sloping Stepped Spillways, *J. Hydraul. Res.* 135 (2009) 1092–1100.
- [147] D. Violeau, B. D. Rogers, Smoothed particle hydrodynamics (SPH) for free-surface flows: past, present and future, *J. Hydraul. Res.* 54 (1) (2016) 1–26.
- [148] R. Courant, K. Friedrichs, H. Lewy, Über die partiellen Differenzgleichungen der mathematischen Physik, *Math. Ann.* 100 (1) (1928) 32–74.
- [149] D. Violeau, *Fluid Mechanics and the SPH Method: Theory and Applications*, Oxford University Press, UK, 2012.
- [150] A. Colagrossi, M. Landrini, Numerical simulation of interfacial flows by smoothed particle hydrodynamics, *J. Comput. Phys.* 191 (2003) 448–475.
- [151] R. A. Dalrymple, B. D. Rogers, Numerical modeling of water waves with the SPH method, *Coast. Eng.* 53 (2006) 141–147.
- [152] D. Molteni, A. Colagrossi, A simple procedure to improve the pressure evaluation in hydrodynamics using the SPH, *Comput. Phys. Commun.* 180 (2009) 861–872.
- [153] M. Antuono, A. Colagrossi, S. Marrone, D. Molteni, Free-surface flows solved by means of SPH schemes with numerical diffusive terms, *Comput. Phys. Commun.* 181 (2010) 532–549.
- [154] J. Bonet, T.-S. L. Lok, Variational and momentum preservation aspects of Smoothed Particle Hydrodynamic formulations, *Comput. Methods Appl. Mech. Eng.* 180 (1999) 97–115.
- [155] A. improved SPH method: Towards higher order convergence.

- [156] H. F. Schwaiger, An implicit corrected SPH formulation for thermal diffusion with linear free surface boundary conditions, *Int. J. Numer. Meth. Eng.* 75 (2008) 647–671.
- [157] J. P. Morris, P. J. Fox, Y. Zhu, Modeling Low Reynolds Number Incompressible Flows Using SPH, *J. Comput. Phys.* 136 (1997) 214–226.
- [158] S. Adami, X. Y. Hu, N. A. Adams, A generalized wall boundary condition for smoothed particle hydrodynamics, *J. Comput. Phys.* 231 (2012) 7057–7075.
- [159] M. Ferrand, D. R. Laurence, B. D. Rogers, D. Violeau, C. Kassiotis, Unified semi-analytical wall boundary conditions for inviscid, laminar or turbulent flows in the meshless SPH method, *Int. J. Numer. Meth. Fluids* 71 (2013) 446–472.
- [160] J. J. Monaghan, SPH without a Tensile Instability, *J. Comput. Phys.* 159 (2000) 290–311.
- [161] R. Xu, P. Stansby, D. Laurence, Accuracy and stability in incompressible SPH (ISPH) based on the projection method and a new approach, *J. Comput. Phys.* 228 (2009) 6703–6725.
- [162] M. S. Shadloo, A. Zainali, M. Yildiz, A. Suleman, A robust and weakly compressible SPH method and its comparison with and incompressible SPH, *Int. J. Numer. Meth. Eng.* 89 (2012) 939–956.
- [163] R. Vacondio, B. D. Rogers, P. K. Stansby, P. Mignosa, Variable resolution for SPH in three dimensions: Towards optimal splitting and coalescing for dynamic adaptivity, *Comput. Methods Appl. Mech. Eng.* 300 (2016) 442–460.
- [164] D. Le Touzé, A. Colagrossi, G. Colicchio, M. Greco, A critical investigation of smoothed particle hydrodynamics applied to problems with free-surfaces, *Int. J. Numer. Meth. Fluids* 73 (2013) 660–691.
- [165] A. Colagrossi, B. Bouscasse, M. Antuono, S. Marrone, Particle packing algorithm for SPH schemes, *Comput. Phys. Commun.* 183 (2012) 1641–1653.

- [166] J. P. Vila, On Particle Weighted Methods and Smoothed Particle Hydrodynamics, *Math. Models Meth. Appl. Sci.* 13 (1999) 1097–1112.
- [167] A. N. Parshikov, S. A. Medin, I. I. Loukashenko, V. A. Milekhin, Improvements in SPH method by means of interparticle contact algorithm and analysis of perforation tests at moderate projectile velocities, *Int. J. Impact. Eng.* 24 (2000) 779–796.
- [168] A. N. Parshikov, S. A. Medin, Smoothed Particle Hydrodynamics Using Interparticle Contact Algorithms, *J. Comput. Phys.* 180 (2002) 358–382.
- [169] B. D. Rogers, R. A. Dalrymple, P. K. Stansby, Simulation of caisson breakwater movement using 2-d sph, *J. Hydraul. Res.* 48 (2010) 135–141.
- [170] R. Xu, An Improved Incompressible Smoothed Particle Hydrodynamics Method and Its Application in Free-Surface Simulations, Ph.D. thesis, The University of Manchester (2009).
- [171] M. Ellero, M. Serrano, P. Español, Incompressible smoothed particle hydrodynamics, *J. Comput. Phys.* 226 (2007) 1731–1752.
- [172] S. J. Cummins, M. Rudman, An SPH Projection Method, *J. Comput. Phys.* 152 (1999) 584–607.
- [173] A. J. Chorin, Numerical Solution of the Navier-Stokes Equations, *Math. Comput.* 22 (104) (1968) 745–762.
- [174] O. A. Ladyžhenskaya, *The Mathematical Theory of Viscous Incompressible Flow*, Gordon and Breach, 1963.
- [175] E.-S. Lee, C. Moulinec, R. Xu, D. Violeau, D. Laurence, P. Stansby, Comparisons of weakly compressible and truly incompressible algorithms for the SPH mesh free particle method, *J. Comput. Phys.* 227 (2008) 8417–8436.
- [176] E. Y. M. Lo, S. Shao, Simulation of near-shore solitary wave mechanics by an incompressible SPH method, *Appl. Ocean. Res* 24 (2002) 275–286.

- [177] S. Shao, E. Y. M. Lo, Incompressible SPH method for simulating Newtonian and non-Newtonian flows with a free surface, *Adv. Water Resour.* 25 (2003) 787–800.
- [178] S. Koshizuka, A. Nobe, Y. Oka, Numerical Analysis of Breaking Waves Using The Moving Particle Semi-Implicit Method, *Int. J. Numer. Meth. Fluids* 26 (1998) 751–769.
- [179] H. Gotoh, A. Khayyer, H. Ikari, T. Arikawa, K. Shimosako, On enhancement of Incompressible SPH method for simulation of violent sloshing flows, *Appl. Ocean Res.* 46 (2014) 104–115.
- [180] A. Khayyer, H. Gotoh, Y. Shimizu, K. Gotoh, On enhancement of energy conservation properties of projection-based particle methods, *Eur. J. Mech. B Fluids* 66 (2017) 20–37.
- [181] X. Y. Hu, N. A. Adams, An incompressible multi-phase SPH method, *J. Comput. Phys.* 227 (2007) 264–278.
- [182] R. Nestor, M. Basa, N. Quinlan, Moving Boundary Problems in Finite Volume Particle Method, in: *Proceedings of the 3rd International SPHERIC Workshop*, 2008.
- [183] A. Mokos, B. D. Rogers, P. K. Stansby, A multi-phase particle shifting algorithm for SPH simulations of violent hydrodynamics with a large number of particles, *J. Hydraul. Res.* 55 (2017) 143–162.
- [184] A. Khayyer, H. Gotoh, Y. Shimizu, Comparative study on accuracy and conservation properties of two particle regularization schemes and proposal of an optimized particle shifting scheme in ISPH context, *J. Comput. Phys.* 332 (2017) 236–256.
- [185] A. Khayyer, H. Gotoh, S. D. Shao, Corrected Incompressible SPH method for accurate water-surface tracking in breaking waves, *Coast. Eng.* 55 (3) (2008) 236–250.

- [186] A. Khayyer, H. Gotoh, S. Shao, Enhanced predictions of wave impact pressure by improved incompressible SPH methods, *Appl. Ocean Res.* 31 (2009) 111–131.
- [187] A. Khayyer, H. Gotoh, Wave Impact Pressure Calculations by Improved SPH Methods, *Int. J. Offshore Polar Eng.* 19 (4) (2009) 300–307.
- [188] S. T. Grilli, I. A. Svendsen, R. Subramanya, Breaking criterion and characteristics for solitary waves on slopes, *J. Waterw. Port Coast. Ocean Eng.* 123 (3) (1997) 102–112.
- [189] C. Lachaume, B. Biaisser, S. T. Grilli, P. Fraunie, S. Guignard, Modeling of breaking and post-breaking waves on slopes by coupling of BEM and VOF methods, in: *Proceedings of the 13th Offshore and Polar Engineering Conference, ISOPE 2003, Honolulu, HI, USA, 2003*, pp. 353–359.
- [190] A. Leroy, A new incompressible SPH model: towards industrial applications, Ph.D. thesis, L'Université Paris (2014).
- [191] S. J. Lind, P. K. Stansby, B. D. Rogers, Fixed and moored bodies in steep and breaking waves using SPH with Froude-Krylov approximation, *J. Ocean Eng. Mar. Energy* 2 (2016) 331–354.
- [192] M. Luck, M. Benoit, Wave Loading on Monopile Foundation for Offshore Wind Turbines in Shallow-Water Areas, in: *Coastal Engineering 2004, 2005*, pp. 3992–4004.
- [193] M. Hann, D. Greaves, A. Raby, Snatch loading of a single taut moored floating wave energy converter due to focussed wave groups, *Ocean Eng.* 96 (2015) 258–271.
- [194] D. Liang, J. Zhang, H. Liu, SPH Simulation of Solitary Wave Generation and Impact on a Vertical Wall, in: *Proceedings of the Eleventh Pacific/Asia Offshore Mechanics Symposium, 2014*, pp. 299–306.

-
- [195] M. J. Cooker, P. D. Weidman, D. S. Bale, Reflection of a high-amplitude solitary wave at a vertical wall, *J. Fluid Mech.* 342 (2007) 141–158.
- [196] J. C. Marongiu, F. Leboeuf, E. Parkinson, Numerical simulation of the flow in a Pelton turbine using the meshless method smoothed particle hydrodynamics: a new simple solid boundary treatment, *Proc. Inst. Mech. Eng. A* 221 (2007) 849–856.
- [197] E.-S. Lee, D. Violeau, R. Issa, S. Ploix, Application of weakly compressible and truly incompressible SPH to 3-D watercollapse in waterworks, *J. Hydraul. Res.* 48 (2010) 50–60.
- [198] S. J. Lind, P. K. Stansby, B. D. Rogers, P. M. Lloyd, Numerical predictions of water-air wave slam using incompressible-compressible smoothed particle hydrodynamics, *Appl. Ocean Res.* 71 (2015) 57–71.
- [199] J. J. Monaghan, A. Rafiee, A simple SPH algorithm for multi-fluid flow with high density ratios, *Int. J. Numer. Meth. Fluids* 71 (2013) 537–561.
- [200] G. Fourtakas, P. K. Stansby, B. D. Rogers, S. J. Lind, S. Yan, Q. W. Ma, On the coupling of Incompressible SPH with a Finite Element potential flow solver for nonlinear free surface flows, in: *Proceedings of the Twenty-seventh International Ocean and Polar Engineering Conference, 2017*, pp. 570–577.
- [201] Q. W. Ma, S. Yan, Quasi ALE finite element method for nonlinear water waves, *J. Comput. Phys.* 212 (1) (2006) 52–72.
- [202] S. Yan, Q. W. Ma, Numerical simulation of fully nonlinear interaction between steep waves and 2D floating bodies using the QALE-FEM method, *J. Comput. Phys.* 221 (2) (2007) 666–692.
- [203] S. J. Lind, P. K. Stansby, High-order Eulerian incompressible smoothed particle hydrodynamics with transition to Lagrangian free-surface motion, *J. Comput. Phys.* 326 (2016) 290–311.

- [204] A. M. Nasar, Eulerian and Lagrangian Smoothed Particle Hydrodynamics as Models for the Interaction of Fluids and Flexible Structures in Biomedical Flows, Ph.D. thesis, The University of Manchester (2016).
- [205] M. Ordoubadi, A. Farhadi, F. Yeganehdoust, H. Emdad, M. Yaghoubi, E. Goltasebi Rad, Eulerian ISPH Method for Simulating Internal Flows, *J. Appl. Fluid Mech.* 9 (3) (2016) 1477–1490.
- [206] G. Fourtakas, P. K. Stansby, B. D. Rogers, S. J. Lind, An Eulerian-Lagrangian incompressible SPH formulation (ELI-SPH) connected with a sharp interface, *Comput. Meth. Appl. Mech. Eng.* 329 (2018) 532–552.
- [207] G. Y. Buss, P. K. Stansby, SAWW - A Computer Program to Calculate the Properties of Steady Water Waves, Tech. rep., Simon Engineering Laboratories, The University of Manchester (1982).
- [208] Q. Gui, S. Shao, P. Dong, Wave Impact Simulations by an Improved ISPH Model, *J. Waterw. Port Coast. Ocean Eng.* 140 (3) (2014) 04014005.
- [209] H. Xiao, W. Huang, Numerical modeling of wave runup and forces on an idealized beachfront house, *Ocean Eng.* 35 (1) (2008) 106–116.
- [210] A. Ferrari, M. Dumbser, E. F. Toro, A. Armanini, A New Parallel SPH Method for 3D Free Surface Flows, in: *High Performance Computing on Vector Systems 2009*, 2009, pp. 179–188.
- [211] M. Yildiz, R. A. Rook, A. Suleman, SPH with multiple boundary tangent method, *Int. J. Numer. Methods Eng.* 77 (2009) 1416–1438.
- [212] G. Fourtakas, R. Vacondio, B. D. Rogers, On the approximate zeroth and first-order consistency in the presence of 2-D irregular boundaries in SPH obtained by the virtual boundary particle methods, *Int. J. Numer. Meth. Fluids* 78 (2015) 475–501.

- [213] S. Kulasegaram, J. Bonet, R. W. Lewis, M. Profit, A variational formulation based contact algorithm for rigid boundaries in two-dimensional SPH applications, *Comput. Mech.* 33 (2004) 316–325.
- [214] J. Feldman, J. Bonet, Dynamic refinement and boundary contact forces in SPH with applications in fluid flow problems, *Int. J. Numer. Meth. Eng.* 72 (2007) 295–324.
- [215] F. Maciá, M. Antuono, L. M. González, A. Colagrossi, Theoretical Analysis of the No-Slip Boundary Condition Enforcement in SPH Methods, *Prog. Theor. Phys.* 125 (6) (2011) 1091–1121.
- [216] R. Vacondio, B. D. Rogers, P. K. Stansby, Smoothed Particle Hydrodynamics: Approximate zero-consistent 2-D boundary conditions and still shallow-water tests, *Int. J. Numer. Meth. Fluids* 69 (2012) 226–253.
- [217] H. Gotoh, A. Khayyer, Current achievements and future perspectives for projection-based particle methods with applications in ocean engineering, *J. Ocean Eng. Mar. Energy* 2 (2016) 251–278.
- [218] H. Takeda, S. M. Miyama, M. Sekiya, Numerical Simulation of Viscous Flow by Smoothed Particle Hydrodynamics, *Prog. Theor. Phys.* 92 (5) (1994) 939–960.
- [219] A. Amicarelli, J.-C. Marongiu, F. Leboeuf, J. Leduc, J. Caro, SPH truncation error in estimating a 3D function, *Comput. Fluids* 44 (2011) 279–296.
- [220] P. Nair, G. Tomar, Volume conservation issues in incompressible smoothed particle hydrodynamics, *J. Comput. Phys.* 297 (2015) 689–699.
- [221] A. Khayyer, H. Gotoh, Y. Shimizu, Comparative study on accuracy and conservation properties of two particle regularization schemes and proposal of an optimized particle shifting scheme in ISPH context, *J. Comput. Phys.* 332 (2017) 236–256.

- [222] D. Valdez-Balderas, J. M. Domínguez, B. D. Rogers, A. J. C. Crespo, Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters, *J. Parallel Distrib. Comput.* 73 (2013) 1483–1493.
- [223] J. L. Cercos-Pita, AQUAplusph, a new free 3D SPH solver accelerated with OpenCL, *Comput. Phys. Commun.* 192 (2015) 295–312.
- [224] G. Viccione, V. Bovolin, E. P. Carratelli, Defining and optimizing algorithms for neighbouring particle identification in SPH fluid simulations, *Int. J. Numer. Methods Fluids* 58 (2008) 625–638.
- [225] J. M. Domínguez, A. J. C. Crespo, G.-G. M., J. C. Marongiu, Neighbour lists in smoothed particle hydrodynamics, *Int. J. Numer. Methods Fluids* 67 (12) (2010) 2026–2042.
- [226] M. Forum, MPI: A Message Passing Interface, in: *Proceedings of Supercomputing*, 1993, pp. 878–883.
- [227] L. Dagum, R. Menon, OpenMP: An Industry Standard API for Shared-Memory Programming, *IEEE Comput. Sci. Eng.* 5 (1) (1998) 46–55.
- [228] TOP500.org©, Top500 List - November 2017, <https://www.top500.org/lists/2017/11/> (2017) [Online accessed: 31/01/2018].
- [229] P. Maruzewski, D. Le Touzé, G. Oger, F. Avellan, SPH high-performance computing simulations of rigid solids impacting the free-surface of water, *J. Hydraul. Res.* 47 (2009) 126–134.
- [230] G. Oger, D. Le Touzé, D. Guibert, M. de Lefte, J. Biddiscombe, J. Soumagne, On distributed memory MPI-based parallelization of SPH codes in massive HPC context, *Comput. Phys. Commun.* 200 (2016) 1–14.
- [231] S. Braun, L. Wieth, R. Koch, C. Höfler, T. Dauch, M. C. Keller, HPC Predictions of Primary Atomization with SPH: Challenges and Lessons Learned, in: *Proceedings of the 11th International SPHERIC Workshop*, 2016.

- [232] X. Guo, S. J. Lind, B. D. Rogers, P. K. Stansby, M. Ashworth, Efficient Massive Parallelisation for Incompressible Smoothed Particle Hydrodynamics with 10^8 Particles, in: Proceedings of the 8th International SPHERIC Workshop, 2013.
- [233] S. Yeylaghi, B. Moa, P. Oshkai, B. Buckham, C. Crawford, ISPH modelling of an oscillating wave surge converter using an OpenMP-based parallel approach, *J. Ocean Eng. Mar. Energy* 2 (2016) 301–312.
- [234] S. Yeylaghi, B. Moa, P. Oshkai, B. Buckham, C. Crawford, ISPH modelling for hydrodynamic applications using a new MPI-based parallel approach, *J. Ocean Eng. Mar. Energy* 3 (2016) 35–50.
- [235] S. McIntosh-Smith, T. Wilson, A. A. Ibarra, J. Crisp, R. B. Sessions, Benchmarking Energy Efficiency, Power Costs and Carbon Emissions on Heterogeneous Systems, *Comput. J.* 5 (2) (2012) 1499–1523.
- [236] Nvidia, CUDA C Programming Guide, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (2017) [Online accessed: 01/02/2018].
- [237] Khronos, OpenCL 2.1 Reference Pages, <https://www.khronos.org/registry/OpenCL/sdk/2.1/docs/man/xhtml/> (2017) [Online accessed: 02/02/2018].
- [238] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, S. Kato, Power and Performance Analysis of GPU-Accelerated Systems, in: Hot Power '12: Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems, 2012, pp. 75–100.
- [239] T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, K. Chihara, Particle-Based Fluid Simulation on GPU, in: Proceedings of the ACM Workshop on General-purpose Computing on Graphics Processors, 2004.
- [240] A. Kolb, N. Cuntz, Dynamic particle coupling for GPU-based fluid simulation, in: Proceedings of the 18th Symposium on Simulation Technique, 2005, pp. 722–727.

- [241] T. Harada, S. Koshizuka, Y. Kawaguchi, Smoothed Particle Hydrodynamics on GPUs, *Comput. Graph. Int.* 42 (4) (2007) 63–70.
- [242] A. Héroult, G. Bilotta, R. A. Dalrymple, SPH on GPU with CUDA, *J. Hydraul. Res.* 48 (2010) 74–79.
- [243] R. A. Dalrymple, A. Héroult, G. Bilotta, R. J. Farahani, GPU-Accelerated SPH model for water waves and free surface flows, in: *Proceedings of the 32nd International Conference on Coastal Engineering*, 2010, p. waves.9.
- [244] R. Weiss, A. Munoz, R. A. Dalrymple, A. Héroult, G. Bilotta, Three-Dimensional Modeling of Long-Wave Runup: Simulation of Tsunami Inundation With GPU-SPHysics, in: *Proceedings of the 32nd International Conference on Coastal Engineering*, 2010, p. currents.8.
- [245] G. Pringgana, L. S. Cunningham, B. D. Rogers, Modelling of tsunami-induced bore and structure interaction, *Proc. ICE - Eng. Comput. Mech.* 169 (3) (2016) 109–125.
- [246] M. D. Green, Sloshing simulations with the smoothed particle hydrodynamics (SPH) method, Ph.D. thesis, Imperial College London (2017).
- [247] J. L. Cercos-Pita, G. Bulian, L. Pérez-Rojas, A. Francescutto, Coupled simulation of nonlinear ship motions and free surface tank, *Ocean Eng.* 120 (2016) 281–288.
- [248] A. Mokos, B. D. Rogers, P. K. Stansby, J. M. Domínguez, Multi-phase SPH modelling of violent hydrodynamics on GPUs, *Comput. Phys. Commun.* 196 (2015) 304–316.
- [249] G. Fourtakas, B. D. Rogers, Modelling multi-phase liquid-sediment scour and resuspension induced by rapid flows using Smoothed Particle Hydrodynamics (SPH) accelerated with a Graphics Processing Unit (GPU), *Adv. Water Res.* 92 (2016) 186–199.

- [250] A. Barreiro, J. M. Domínguez, A. J. C. Crespo, H. Ganzález-Jorge, D. Roca, M. Gómez-Gesteira, Integration of UAV Photogrammetry and SPH Modelling of Fluids to Study Runoff on Real Terrains, *PLoS ONE* 9 (11) (2014) e111031. doi:10.1371/journal.pone.0111031.
- [251] J. M. Domínguez, A. J. C. Crespo, D. Valdez-Balderas, B. D. Rogers, M. Gómez-Gesteira, New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters, *Comput. Phys. Commun.* 184 (2013) 1848–1860.
- [252] E. Rustico, G. Bilotta, A. Hérault, C. Del Negro, G. Gallo, Advances in Multi-GPU Smoothed Particle Hydrodynamics Simulations, *IEEE Trans. Parallel Distrib. Syst.* 25 (1) (2014) 43–52.
- [253] P. Berczik, N. Nakasato, I. Berentzen, R. Spurzem, G. M. Martinez, G. Lienhart, A. Kugel, R. Maenner, A. Burkert, M. Wetzstein, T. Naab, H. Vasquez, S. B. Vinogradov, Special, hardware accelerated, parallel SPH code for galaxy evolution, in: *SPHERIC Second International Workshop, 2007*, pp. 5–8.
- [254] R. Spurzem, P. Berczik, G. Marcus, A. Kugel, G. Lienhart, I. Berentzen, R. Männer, R. Klessen, R. Banerjee, Accelerating astrophysical particle simulations with programmable hardware (FPGA and GPU), *CSR D* 23 (2009) 231–239.
- [255] C.-L. Su, P.-Y. Chen, C.-C. Lan, L.-S. Huang, K.-H. Wu, Overview and comparison of OpenCL and CUDA technology for GPGPU, in: *IEEE Asia Pacific Conference on Circuits and Systems, 2012*.
- [256] J. Fang, A. L. Varbanescu, H. Sips, A Comprehensive Performance Comparison of CUDA and OpenCL, in: *International Conference on Parallel Processing, 2011*.
- [257] C. Hori, H. Gotoh, H. Ikari, A. Khayyer, GPU-acceleration for Moving Particle Semi-Implicit method, *Comput. Fluids* 51 (2011) 174–183.

- [258] X. Zhu, L. Cheng, L. Lu, B. Teng, Implementation of the moving particle semi-implicit method on GPU, *Sci. China* 54 (3) (2011) 523–532.
- [259] W. Gou, S. Zhang, Y. Zheng, Simulation of isothermal multi-phase fuel-coolant interaction using MPS method with GPU acceleration, *Kerntechnik* 81 (3) (2016) 330–336.
- [260] H. Li, Y. Zhang, D. Wan, GPU Based Acceleration of MPS for 3D Free Surface Flows, in: *Proceedings of the 9th International Workshop on Ship and Marine Hydrodynamics*, 2015.
- [261] K. Murotani, I. Masaie, T. Matsunaga, S. Koshizuka, R. Shioya, M. Ogino, T. Fujisawa, Performance improvements of differential operators code for MPS method on GPU, *Comput. Part. Mechanics* 3 (2015) 155–166.
- [262] G. Duan, B. Chen, Comparison of parallel solvers for Moving Particle Semi-Implicit method, *Eng. Comput.* 32 (3) (2015) 834–862.
- [263] GPUSPH, GPUSPH, <http://www.gpusph.org/> (2017) [Online accessed: 28/03/2018].
- [264] AQUAopusph, AQUAopusph, <http://canal.etsin.upm.es/aquagpusph/> (2016) [Online accessed: 02/02/2018].
- [265] DualSPHysics, DualSPHysics, <http://dual.sphysics.org/> (2018) [Online accessed: 25/04/2018].
- [266] R. J. Farahani, R. A. Dalrymple, A. Hérault, G. Bilotta, Three-Dimensional SPH Modeling of Bar/Rip Channel System, *J. Waterw. Port Coast. Ocean Eng.* 140 (1) (2014) 05016001.
- [267] Z. Wei, R. A. Dalrymple, E. Rustico, A. Hérault, G. Bilotta, Simulation of Nearshore Tsunami Breaking by Smoothed Particle Hydrodynamics Method, *J. Waterw. Port Coast. Ocean Eng.* 142 (4) (2016) 05016001.

- [268] B. Serván-Camas, J. L. Cercós-Pita, J. Colom-Cobb, J. García-Espinosa, A. Souto-Iglesias, Time domain simulation of coupled sloshing-seakeeping problems by SPH-FEM coupling, *Ocean Eng.* 123 (2016) 383–396.
- [269] S. Dalton, N. Bell, M. Olson, L. and Garland, Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations, <http://cusplibrary.github.io/> (2015) [Online accessed: 20/07/2017].
- [270] Nvidia, CUSPARSE LIBRARY v9.2, http://docs.nvidia.com/cuda/pdf/CUSPARSE_Library.pdf (2018) [Online accessed: 24/05/2018].
- [271] B. Seibold, M-Matrices in Meshless Finite Difference Methods, Ph.D. thesis, Technische Universität Kaiserslautern (2006).
- [272] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd Edition, Soc. Ind. Appl. Math., Phila., PA, USA, 2003.
- [273] T. Tomczak, K. Zadarnowska, Z. Koza, M. Matyka, L. Mirosław, Acceleration of iterative Navier-Stokes solvers on graphics processing units, *Int. J. Comput. Fluid Dyn.* 27 (2013) 201–209.
- [274] H. Elman, D. Silvester, A. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford Science Publications, 2014.
- [275] P. Dirac, *The Principles of Quantum Mechanics* (4th Ed.), Oxford University Press, 1958.
- [276] D. S. Balsara, von Neumann Stability Analysis of Smoothed Particle Hydrodynamics - Suggestions for Optimal Algorithms, *J. Comput. Phys.* 121 (1995) 357–372.
- [277] H. Wendland, Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree, *Adv. Comput. Math.* 4 (1995) 389–396.
- [278] M. B. Liu, G. R. Liu, Smoothed Particle Hydrodynamics (SPH): an Overview and recent Developments, *Arch. Comput. Methods Eng.* 17 (2010) 25–76.

- [279] L. Brookshaw, A Method of Calculating Radiative Heat Diffusion in Particle Simulations, in: Proceedings of the Astronomical Society of Australia, 1985, pp. 207–210.
- [280] S. Adami, X. Y. Hu, N. Adams, A transport-velocity formulation for smoothed particle hydrodynamics, *J. Comput. Phys.* 241 (2013) 292–307.
- [281] G. Oger, S. Marrone, D. Le Touzé, M. de Leffe, SPH accuracy and improvement through the combination of a quasi-Lagrangian shifting transport velocity and consistent ALE formalisms, *J. Comput. Phys.* 313 (2016) 76–98.
- [282] J. Swegle, D. L. Hicks, S. W. Attaway, Smoothed Particle Hydrodynamics Stability, *J. Comput. Phys.* 116 (1995) 123–134.
- [283] W. Dehnen, H. Aly, Improving convergence in smoothed particle hydrodynamics simulations without pairing instability, *Mon. Not. R. Astron. Soc.* 425 (2012) 1068–1082.
- [284] E.-S. Lee, Truly incompressible approach for computing incompressible flow in SPH and comparisons with the traditionally weakly compressible approach, Ph.D. thesis, The University of Manchester (2007).
- [285] A. Leroy, D. Violeau, M. Ferrand, C. Kassiotis, Unified semi-analytical wall boundary conditions applied to 2-D incompressible SPH, *J. Comput. Phys.* 261 (2013) 106–129.
- [286] B. Bouscasse, A. Colagrossi, S. Marrone, M. Antuono, Nonlinear water wave interaction with floating bodies in SPH, *J. Fluids Struct.* 42 (2013) 112–129.
- [287] H. Nguyen, *GPU Gems 3*, Addison-Wesley Professional, 2007.
- [288] H. Anzt, S. Tomov, J. Dongarra, On the performance and energy efficiency of sparse linear algebra on GPUs, *Int. J. High Perform. Comput. Appl.* 31 (5) (2017) 375–390.
- [289] NVidia, *NVIDIA GeForce GTX 1080*, Tech. rep. (2016).

- [290] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Second Edition, Pearson Education, 2003.
- [291] J. M. Domínguez, A. J. C. Crespo, M. Gómez-Gesteira, Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method, *Comput. Phys. Commun.* 184 (2013) 617–627.
- [292] M. Gomez-Gesteira, B. D. Rogers, A. J. C. Crespo, R. A. Dalrymple, M. Narayanaswamy, J. M. Domínguez, SPHysics - development of a free-surface fluid solver - Part 1: Theory and formulations, *Comput. Geosci.* 48 (2012) 289–299.
- [293] M. Gomez-Gesteira, A. J. C. Crespo, B. D. Rogers, R. A. Dalrymple, J. M. Domínguez, A. Barreiro, SPHysics - development of a free-surface fluid solver - Part 2: Efficiency and test cases, *Comput. Geosci.* 48 (2012) 300–307.
- [294] DualSPHysics, References, <http://dual.sphysics.org/index.php/references/> (2018) [Online accessed: 25/04/2018].
- [295] Kitware, Paraview, <https://www.paraview.org/> (2018) [Online accessed: 20/01/2018].
- [296] D. Winkler, M. Rezavand, W. Rauch, Neighbour lists for smoothed particle hydrodynamics on GPUs, *Comput. Phys. Commun.* In Press.
- [297] I. S. Duff, A. M. Erisman, J. K. Reid, Direct Methods for Sparse Matrices, Oxford University Press, 1986.
- [298] H. A. van der Vorst, Iterative Krylov Methods for Large Linear Systems, Cambridge University Press, 2003.
- [299] M. Manguoglu, A domain-decomposing parallel sparse linear system solver, *J. Comput. Appl. Math.* 236 (2011) 319–325.
- [300] M. Ferronato, Preconditioning for Sparse Linear Systems at the Dawn of the 21st Century: History, Current Development, and Future Perspectives, *ISRN Appl. Math.* 2012 (2012) Article ID 127647.

- [301] M. Benzi, Preconditioning Techniques for Large Linear Systems: A Survey, *J. Comput. Phys.* 182 (2002) 418–477.
- [302] F. H. Pereira, S. L. L. Verardi, S. I. Nabeta, A fast algebraic multigrid preconditioned conjugate gradient solver, *Appl. Math. Comput.* 179 (2006) 344–351.
- [303] K. Rupp, J. Weinbub, F. Rudolf, A. Morhammer, T. Grasser, A. Jünger, A Performance Comparison of Algebraic Multigrid Preconditioners on CPUs, GPUs, and Xeon Phi, *Numer. Linear Algebra Appl.* 00 (2015) 1–14.
- [304] G. Brussino, V. Sonnad, A Comparison of Direct and Preconditioner Iterative Techniques for Sparse, Unsymmetric Systems of Linear Equations, *Int. J. Numer. Meth. Eng.* 28 (1989) 801–815.
- [305] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Inc., 1980.
- [306] J. Liesen, P. Tichý, Convergence analysis of Krylov subspace methods, *GAMM-Mitt.* 27 (2) (2004) 153–173.
- [307] Y. Saad, M. H. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.* 7 (3) (1986) 856–869.
- [308] C. C. Paige, M. A. Saunders, Solution of sparse indefinite systems of linear equations, *SIAM J. Numer. Anal.* 12 (4) (1975) 617–629.
- [309] W. E. Arnoldi, The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem, *Quart. Appl. Math.* 9 (1) (1951) 17–29.
- [310] H. A. van der Vorst, Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for Solution of Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.* 13 (2) (1992) 631–644.
- [311] P. Sonneveld, CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear Solvers, *SIAM J. Sci. Stat. Comput.* 10 (1) (1989) 36–52.

- [312] D. GÖddeke, Fast and Accurate Finite-Element Multigrid Solvers for PDE Simulations on GPU Clusters, Ph.D. thesis, Der Technischen Universität Dortmund (2010).
- [313] Y. Li, R. Saad, GPU-accelerated preconditioned iterative linear solvers, *J. Supercomput.* 63 (2013) 443–466.
- [314] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Web page, <http://www.mcs.anl.gov/petsc> (2017) [Online accessed: 02/02/2018].
- [315] P. labs, Paralution v1.0.0, <https://www.paralution.com/> (2015) [Online accessed: 02/02/2018].
- [316] K. Rupp, P. Tillet, F. Rudolf, J. Weinbub, A. Morhammer, T. Grasser, A. Jüngel, S. Selberherr, ViennaCL - Linear Algebra Library for Multi- and Many-Core Architectures, *SIAM J. Sci. Comput.* 38 (5) (2016) S412–S439.
- [317] H. Anzt, M. Gates, J. Dongarra, M. Kreutzer, G. Wellein, M. Köhler, Preconditioned Krylov solvers on GPUs, *Parallel Comput.* 68 (2017) 32–44.
- [318] K. Stüben, A review of algebraic multigrid, *J. Comput. Appl. Math.* 128 (2001) 281–309.
- [319] W. Hackbusch, *Iterative Solution of Large Sparse Linear Systems of Equations*, Springer-Verlag, 2004.
- [320] S. Lew, C. H. Wolters, T. Dierkes, C. Röer, R. S. MacLeod, Accuracy and run-time comparison for different potential approaches and iterative solvers in finite element method based EEG source analysis, *Appl. Numer. Math.* 59 (2009) 1970–1988.
- [321] O. Axelsson, P. S. Vassilevski, Algebraic Multilevel Preconditioning Methods. I, *Numer. Math.* 56 (1989) 157–177.

- [322] O. Axelsson, P. S. Vassilevski, Algebraic Multilevel Preconditioning Methods. II, *SIAM J. Numer. Anal.* 27 (6) (1989) 1569–1590.
- [323] N. Bell, S. Dalton, L. Olsen, Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods, *SIAM J. Sci. Comput.* 34 (4) (2012) 123–152.
- [324] M. Luby, A Simple Parallel Algorithm for the Maximal Independent Set Problem, *SIAM J. Comput.* 15 (4) (1986) 1036–1053.
- [325] A. J. Roberts, Transient free-surface flows generated by a moving vertical plate, *Quart. J. Mech. Appl. Math.* 40 (1987) 129–158.
- [326] P. N. Brown, H. F. Walker, GMRES on (nearly) singular systems, *SIAM J. Matrix Anal. Appl.* 18 (1) (1997) 37–51.
- [327] K. Hayami, M. Sugihara, A geometric view of Krylov subspace methods on singular systems, *Numer. Linear Algebra Appl.* 18 (2011) 449–469.
- [328] S. Murakami, S. Iizuka, CFD Analysis of Turbulent Flow Past Square Cylinder Using Dynamic LES, *J. Fluids Struct.* 13 (1999) 1097–1112.
- [329] Y. Cao, T. Tamura, Large-eddy simulations of flow past a square cylinder using structured and unstructured grids, *Comput. Fluids* 137 (2016) 36–54.
- [330] A. H. Baker, M. Schulz, U. M. Yang, On the Performance of an Algebraic Multigrid Solver on Multicore Clusters, in: *High Performance Computing for Computational Science VECPAR 2010*, 2010.
- [331] S. J. Lind, Q. Fang, P. K. Stansby, B. D. Rogers, G. Fourtakas, A Two-Phase Incompressible-Compressible (Water-Air) Smoothed Particle Hydrodynamics (ICSPH) Method Applied to Focused Wave Slam on Decks, in: *Proceedings of the Twenty-seventh International Ocean and Polar Engineering Conference*, 2017, pp. 686–692.
- [332] D. M. Kelly, Q. Chen, J. Zang, PICIN: A Particle-in-Cell Solver for Incompressible Free Surface Flows with Two-Way Fluid-Solid Coupling, *SIAM J. Sci. Comput.* 37 (3) (2015) B403–B424.

- [333] E. Napoli, M. De Marchis, C. Gianguzzi, B. Milici, A. Monteleone, A coupled Finite Volume-Smoothed Particle Hydrodynamics method for incompressible flows, *Comput. Meth. Appl. Mech. Eng.* 310 (2016) 674–693.
- [334] S. Marrone, A. Di Mascio, D. Le Touzé, Coupling of Smoothed Particle Hydrodynamics with Finite Volume method for free-surface flows, *J. Comput. Phys.* 310 (2016) 161–180.
- [335] P. S. Tromans, A. R. Anaturk, P. Hagemeyer, A new model for the kinematics of large ocean waves-application as a design wave., in: *The First International Offshore and Polar Engineering Conference*, International Society of Offshore and Polar Engineers, Edinburgh, UK, 1991, pp. 64–71.
- [336] L. F. Chen, J. Zang, A. J. Hillis, G. C. J. Morgan, A. R. Plummer, Numerical investigation of wave-structure interaction using OpenFOAM, *Ocean Eng.* 88 (2014) 91–109.
- [337] J. Wienke, H. Oumeraci, Breaking wave impact force on a vertical and inclined slender pile - theoretical and large-scale model investigations, *Coast. Eng.* 52 (2005) 435–462.
- [338] P. K. Stansby, L. C. Devaney, T. J. Stallard, Breaking wave loads on monopiles for offshore wind turbines and estimation of extreme overturning moment, *IET Renew. Power Gen.* 7 (5) (2013) 514–520.
- [339] B. Seibold, Performance of algebraic multigrid methods for non-symmetric matrices arising in particle methods, *Numer. Linear Algebra Appl.* 17 (2010) 433–451.
- [340] J. Aliaga, D. Davidović, J. Pérez, E. S. Quintana-Ortí, Harnessing CUDA Dynamic Parallelism for the Solution of Sparse Linear Systems, in: G. R. Joubert, H. Leather, M. Parsons, F. Peters, M. Sawyer (Eds.), *Parallel Computing: On the Road to Exascale*, IOS Press, Amsterdam, Netherlands, 2016, pp. 217–226.

-
- [341] S. Khorasanizade, J. M. M. Sousa, Dynamic flow-based particle splitting in smoothed particle hydrodynamics, *Int. J. Numer. Meth. Eng.* 106 (2016) 397–410.
- [342] Q. W. Ma, Y. Zhou, S. Yan, A review on approaches to solving Poissons equation in projection-based meshless methods for modelling strongly nonlinear water waves, *J. Ocean Eng. Mar. Energy* 2 (2016) 279–299.
- [343] T. Tamai, K. Murotani, S. Koshizuka, On the consistency and convergence of particle-based meshfree discretization schemes for the Laplace operator, *Comput. Fluids* 142 (2017) 79–85.
- [344] D. Violeau, A. Leroy, A. Joly, A. Hérault, Spectral properties of the SPH Laplacian operator, In Press.
- [345] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, R. Strzodka, AmgX: A Library for GPU Accelerated Algebraic Multigrid and Preconditioned Iterative Methods, *SIAM J. Sci. Comput.* 37 (5) (2015) S602–S626.
- [346] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, *Parallel Comput.* 36 (2010) 232–240.

Appendix A

Kernel gradient normalisation matrix inverse

The normalised kernel gradient (as described in Section 3.2.2.1):

$$\nabla_i W_{ij} = \mathbf{L}(\mathbf{r}) \nabla_i \omega_{ij}, \quad (\text{A.1})$$

requires the computation of the inverse of a 3×3 matrix for the term $\mathbf{L}(\mathbf{r})$. The inverse is computed using the method of minors, cofactors, and adjugate. however the number of computations have been reduced by using a shorthand version as shown in the following:

$$\mathbf{L}(\mathbf{r}) = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}^{-1} = \frac{1}{\det} \begin{pmatrix} A_{I,11} & A_{I,12} & A_{I,13} \\ A_{I,21} & A_{I,22} & A_{I,23} \\ A_{I,31} & A_{I,32} & A_{I,33} \end{pmatrix}, \quad (\text{A.2})$$

where

$$\det = (A_{11}A_{22}A_{33} + A_{12}A_{23}A_{31} + A_{21}A_{32}A_{13}) - (A_{31}A_{22}A_{13} + A_{21}A_{12}A_{33} + A_{23}A_{32}A_{11}), \quad (\text{A.3})$$

$$\begin{aligned} A_{I,11} &= A_{22}A_{33} - A_{23}A_{32}, & A_{I,12} &= A_{13}A_{32} - A_{12}A_{33}, & A_{I,13} &= A_{12}A_{23} - A_{13}A_{22}, \\ A_{I,21} &= A_{23}A_{31} - A_{21}A_{33}, & A_{I,22} &= A_{11}A_{33} - A_{13}A_{31}, & A_{I,23} &= A_{13}A_{21} - A_{11}A_{23}, \\ A_{I,31} &= A_{21}A_{32} - A_{22}A_{31}, & A_{I,32} &= A_{12}A_{31} - A_{11}A_{32}, & A_{I,33} &= A_{11}A_{22} - A_{12}A_{21} \end{aligned} \quad (\text{A.4})$$

Appendix B

Derivation of the pressure

Poisson equation (PPE)

The pressure Poisson equation (PPE), as described in Section 3.3.1, is a key part of the ISPH methodology for enforcing incompressibility and obtaining an accurate pressure field by implicit solution of the equation via a linear matrix system. In WC-SPH, where density is permitted to vary, velocity and pressure are coupled through the continuity equation and equation of state. However, in ISPH representing truly incompressible flow, where $d\rho/dt = 0$, the coupling of pressure and velocity is not as straightforward and instead requires a Poisson equation. In this appendix, the derivation of a Poisson equation within the (2D) continuous domain is presented in the next section. Then the particular form of the PPE, used here for ISPH, is derived from Chorin's pressure projection method [173], where a divergence-free velocity field is maintained.

B.1 A Poisson equation for a continuous domain

The Lagrangian form of the Navier-Stokes equations for incompressible flow are the equations for conservation of mass (where $d\rho/dt = 0$):

$$\nabla \cdot \mathbf{u} = 0, \tag{B.1}$$

and momentum:

$$\frac{d\mathbf{u}}{dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{u} \quad (\text{B.2})$$

For simplicity, Eqs (B.1) and (B.2) are expressed in their Eulerian form, i.e. Eq. (B.1) for the conservation of mass can also be written as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \quad (\text{B.3})$$

and similarly, Eq. (B.2) for the conservation of momentum can also be expressed in x and y-component directions in Eqs (B.4) and (B.4) respectively:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \quad (\text{B.4})$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right), \quad (\text{B.5})$$

where the LHS has been obtained from the equivalence:

$$\frac{d}{dt} \equiv \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \quad (\text{B.6})$$

A Poisson equation can be obtained by first taking the divergence of the momentum equations (Eqs (B.4) and (B.5)):

$$\frac{\partial}{\partial x}\frac{\partial u}{\partial t} + \left(\frac{\partial u}{\partial x}\right)^2 + u\frac{\partial^2 u}{\partial x^2} + \frac{\partial v}{\partial x}\frac{\partial u}{\partial y} + v\frac{\partial^2 u}{\partial x\partial y} = -\frac{1}{\rho}\frac{\partial^2 P}{\partial x^2} + \nu\left(\frac{\partial^3 u}{\partial x^3} + \frac{\partial^3 u}{\partial x\partial y^2}\right), \quad (\text{B.7})$$

$$\frac{\partial}{\partial y}\frac{\partial v}{\partial t} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + u\frac{\partial^2 v}{\partial y\partial x} + \left(\frac{\partial v}{\partial y}\right)^2 + v\frac{\partial^2 v}{\partial y^2} = -\frac{1}{\rho}\frac{\partial^2 P}{\partial y^2} + \nu\left(\frac{\partial^3 v}{\partial y\partial x^2} + \frac{\partial^3 v}{\partial y^3}\right) \quad (\text{B.8})$$

Eqs (B.7) and (B.8) are combined to give:

$$\begin{aligned} & \frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) + \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} \\ & \quad + u\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) + v\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \\ & = -\frac{1}{\rho}\left(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2}\right) + \nu\left(\frac{\partial^2}{\partial x^2}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\frac{\partial^2}{\partial y^2}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right) \end{aligned} \quad (\text{B.9})$$

The divergence-free velocity field condition is met by applying Eq. (B.3), which reduces Eq. (B.9) to:

$$\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} = -\frac{1}{\rho}\left(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2}\right), \quad (\text{B.10})$$

which is a Poisson equation for coupling velocity with pressure, whilst satisfying the conservation of mass equation (Eq. (B.1)). The Poisson equation is usually written as:

$$\nabla^2 P = -s, \quad (\text{B.11})$$

where s is a source term. Here, the source term comes from the velocity field. The next section explains how to obtain the PPE with a divergence-free velocity field numerically.

B.2 The PPE with divergence-free velocity field in the projection method

In a numerical scheme, the divergence-free velocity condition is desired at timestep $n + 1$:

$$\nabla \cdot \mathbf{u}^{n+1} = 0 \quad (\text{B.12})$$

In Chorin's pressure projection method [173], this problem is solved with fractional timestepping where the momentum (Eq. (B.2)) is solved in two parts considering the accelerations due to the pressure gradient and viscous forces separately. Ultimately, one would solve:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = -\frac{1}{\rho}\nabla P^{n+1} + \nu\nabla^2\mathbf{u}^n - \mathbf{u}^n \cdot \nabla\mathbf{u}^n, \quad (\text{B.13})$$

with the condition of Eq. (B.12).

In the projection method, an intermediate velocity field, \mathbf{u}^* , is introduced which is obtained by considering the acceleration due to viscous terms and the advection

term:

$$\mathbf{u}^* = \mathbf{u}^n + (\nu \nabla^2 \mathbf{u}^n) \Delta t - \mathbf{u}^n \cdot \nabla \mathbf{u}^n \quad (\text{B.14})$$

The momentum equation would be completed by including the pressure gradient term:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \left(\frac{1}{\rho} \nabla P^{n+1} \right) \Delta t \quad (\text{B.15})$$

However, the pressure field is unknown at this instant, therefore the divergence of Eq. (B.16) is taken:

$$\nabla \cdot \mathbf{u}^{n+1} = \nabla \cdot \mathbf{u}^* - \nabla \cdot \left(\frac{1}{\rho} \nabla P^{n+1} \right) \Delta t, \quad (\text{B.16})$$

where $\nabla \cdot \mathbf{u}^*$ the expanded is:

$$\nabla \cdot \mathbf{u}^* = \nabla \cdot \mathbf{u}^n + \nabla \cdot (\nu \nabla^2 \mathbf{u}^n) \Delta t - \nabla \cdot (\mathbf{u}^n \cdot \nabla \mathbf{u}^n) \quad (\text{B.17})$$

Assuming incompressibility at all times, $\nabla \cdot \mathbf{u}^{n+1} = 0$ in Eq. (B.16) and the first two terms on the RHS of Eq. (B.17) all become 0, leaving only the time discretised version of Eq. (B.10) in Eulerian form:

$$\nabla \cdot \left(\frac{1}{\rho} \nabla P^{n+1} \right) = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^* \quad (\text{B.18})$$

In Lagrangian form, the calculation of the advection term, $\mathbf{u}^n \cdot \nabla \mathbf{u}^n$, is not explicit, but is implicitly included following the change in particle position over timestep Δt .

Appendix C

Effect of boundary conditions on the Lagrangian PPE matrix

In ISPH, because of the kernel symmetry between particle pairs, when it comes to construction of the PPE matrix for fluid-fluid particle interactions, the form of the matrix is symmetric. However, when SPH boundary conditions are involved, the fluid-boundary particle interactions differ to that of the boundary-fluid particle interactions resulting in a non-symmetric matrix system defined as $[\mathbf{A}] \neq [\mathbf{A}]^T$. For instance, in this study, the matrix coefficients of the i -fluid particles are governed by the Laplacian operator using a standard SPH kernel, whereas the coefficients i -boundary particles use an MLS kernel summation due to the use of the Marrone et al. [14] boundary condition. Moreover, boundary particle summations are made about unique interpolation points (UIPs) (see Section 3.4.3) where the UIP-fluid interactions are different to the respective boundary-fluid particle interactions. One can simply see this effect visually, without calculation, from Fig C.1, where the non-zero elements are highlighted in the PPE matrix for a simple 1-D system of regularly distributed particles. The system contains 6 fluid particles (particles 2, 3, 4, 5, 6, and 7) and 2 boundary particles (particles 0 and 1). Boundary particles 0 and 1 are at the LHS of the domain with their respective UIPs positioned at the same locations of particles 2 and 3 respectively. For simplicity, the kernel support radius of a particle spans no more than 2 adjacent particles (4 interactions in total). The matrix elements within the red box, indicate the fluid-fluid particle interactions. If these elements were to be a separate matrix, that system would be symmetric.

However, with the presence of Marrone et al. boundary particles here, the matrix is non-symmetric because the summations are taken about each boundary particle's respective UIP and not at the position of the particle itself.

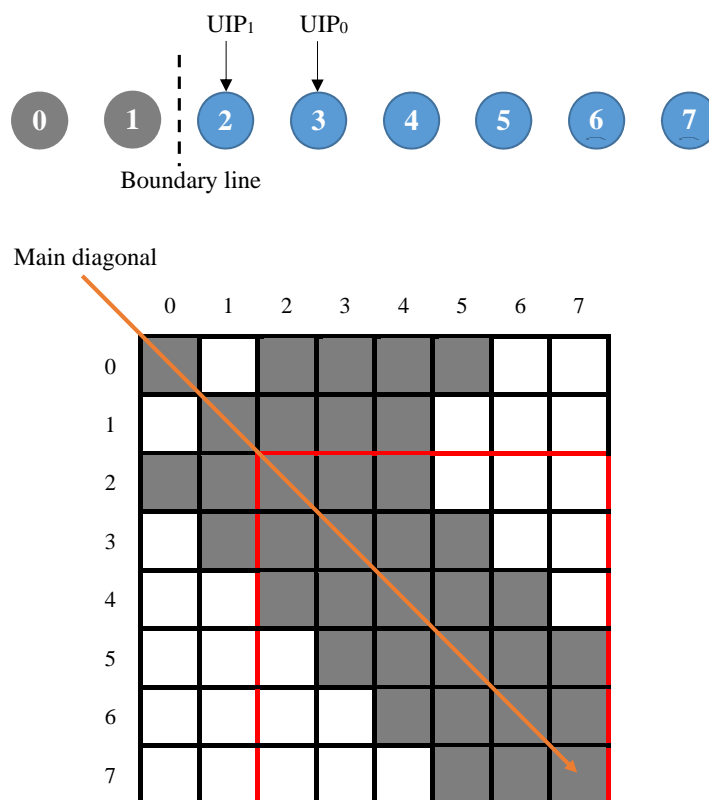


Fig. C.1: An example 1-D case of uniformly distributed fluid particles (blue) and fixed dummy particles (grey) with unique interpolation points. Arrows labelled “UIP” point to the location of a UIP belonging to the boundary particle indicated by the subscript. The associated matrix is shown where non-zero elements are shaded in grey. Elements within the red box indicate fluid-fluid particle interaction values.

Appendix D

Paraview python scripts

The results for the numerical wave basin experiments in Chapter 6 were obtained with the post-processing data visualisation software *Paraview* [295]. This appendix shows python script codes used in Paraview for automation of processing large quantities of data obtained from the 5 million particle simulations. Scripts were made for:

- Extracting fluid free-surface particles at the location of the cylinder-front and plotting their elevation. The corresponding code, as in Section D.1), was used to obtain results for Fig. 6.9.
- Displaying the cylinder’s surface in a 2-D θ - z plane and obtaining forcing and free-surface elevation data about the cylinder. The corresponding code, as in Section D.2), was used to obtain results for Figs 6.13, 6.16, 6.18, and 6.19.
- Extracting fluid free-surface particles around the cylinder for plotting of particle velocities in a 2-D θ - z plane. The corresponding code, as in Section D.3), was used to obtain results for Fig 6.20.

The variable “Rhop” in the scripts stands for pressure, not density. In the original WCSPH DualSPHysics code, the variable “velrhop” is a tuple of velocity components and density, however this has been replaced with “velPressp” for the new Incompressible-DualSPHysics code. The data output functions were not changed so the pressure component has been labelled as Rhop.

D.1 Free-surface elevation at a focal point

```
1 from paraview.simple import *
2
3 fluid_000 = FindSource('Fluid_0*')
4
5 focalPointX=7.395
6 focalPointY=0
7 dp=0.0125
8 depth=0.5
9
10 #Filter out free-surface particles
11 thresholdnew = Threshold(Input=fluid_000)
12 thresholdnew.Scalars = ['POINTS', 'Rhop']
13 thresholdnew.ThresholdRange = [0,0]
14 renderView1 = GetActiveViewOrCreate('RenderView')
15 thresholdnewDisplay = Show(thresholdnew, renderView1)
16 thresholdnew.Scalars = ['POINTS', 'Idp']
17 thresholdnew.Scalars = ['POINTS', 'Rhop']
18 Hide(thresholdnew, renderView1)
19
20 #Get x-coordinate of particles
21 calculatornew = Calculator(Input=thresholdnew)
22 calculatornew.ResultArrayName = 'X'
23 calculatornew.Function = 'coordsX'
24
25 #Get y-coordinate of particles
26 calculatornew = Calculator(Input=calculatornew)
27 calculatornew.ResultArrayName = 'Y'
28 calculatornew.Function = 'coordsY'
29
30 #Filter out particles near focal point
31 thresholdnew = Threshold(Input=calculatornew)
32 thresholdnew.Scalars = ['POINTS', 'X']
33 Lower=focalPointX-dp*2
34 Upper=focalPointX+dp*2
```

```

35 thresholdnew.ThresholdRange = [Lower,Upper]
36 renderView1 = GetActiveViewOrCreate('RenderView')
37 thresholdnewDisplay = Show(thresholdnew, renderView1)
38 thresholdnew.Scalars = ['POINTS', 'Idp']
39 thresholdnew.Scalars = ['POINTS', 'X']
40 Hide(thresholdnew, renderView1)
41
42 thresholdnew = Threshold(Input=thresholdnew)
43 thresholdnew.Scalars = ['POINTS', 'Y']
44 Lower=focalPointY-dp*2
45 Upper=focalPointY+dp*2
46 thresholdnew.ThresholdRange = [Lower,Upper]
47 renderView1 = GetActiveViewOrCreate('RenderView')
48 thresholdnewDisplay = Show(thresholdnew, renderView1)
49 thresholdnew.Scalars = ['POINTS', 'Idp']
50 thresholdnew.Scalars = ['POINTS', 'Y']
51 Hide(thresholdnew, renderView1)
52
53 #Calculate elevation
54 calculatornew = Calculator(Input=thresholdnew)
55 calculatornew.ResultArrayName = 'FreeSurfaceElevation'
56 calculatornew.Function = 'coordsZ-depth'
57 renderView1 = GetActiveViewOrCreate('RenderView')
58 Show(calculatornew, renderView1)
59 Hide(fluid_000, renderView1)
60
61 #plot data
62 query = 'Id >= 0'
63 s = SelectPoints(query)
64 plotSelectionOverTime = PlotSelectionOverTime(Input=calculatornew,
        Selection=s)
65 view = CreateView('QuartileChartView')
66 Show(plotSelectionOverTime, view)
67
68 # get display properties
69 plotSelectionOverTimeDisplay = GetDisplayProperties(

```

```

        plotSelectionOverTime , view=quartileChartView1)
70
71 # Properties modified on plotSelectionOverTime1Display
72 plotSelectionOverTimeDisplay.ShowQuartiles = 0
73 plotSelectionOverTimeDisplay.ShowRanges = 0
74 plotSelectionOverTimeDisplay.ShowAverage = 1
75 plotSelectionOverTimeDisplay.ShowMedian = 0

```

D.2 Extracting cylinder data

```

1 from paraview.simple import *
2 import math
3
4 viewPointColumns=63
5
6 column_000 = FindSource('Column-0*')
7
8 calculator1 = Calculator(Input=column_000)
9 calculator1.ResultArrayName = 'Z'
10 calculator1.Function = 'coordsZ'
11
12 #Calculate angle phi of particle around cylinder
13 calculator4 = Calculator(Input=calculator1)
14 calculator4.ResultArrayName = 'phi'
15 calculator4.Function = '((coordsY+1e-15)/abs(coordsY+1e-15))*acos
        ((0.125^2+(7.52-coordsX)^2-(7.395-coordsX)^2)/(0.25*sqrt((7.52-
        coordsX)^2+(coordsY+1e-15)^2)))'
16
17 #exclude particles below tank bottom
18 threshold1 = Threshold(Input=calculator4)
19 threshold1.Scalars = ['POINTS', 'Z']
20 ZLower=-0.0125/2
21 ZUpper=1.2
22 threshold1.ThresholdRange = [ZLower, ZUpper]
23 renderView1 = GetActiveViewOrCreate('RenderView')
24 threshold1Display = Show(threshold1 , renderView1)

```

```

25 threshold1.Scalars = ['POINTS', 'Idp']
26 threshold1.Scalars = ['POINTS', 'Z']
27 Hide(threshold1, renderView1)
28
29 #Pressure magnitude
30 calculatornew = Calculator(Input=threshold1)
31 calculatornew.ResultArrayName = 'MagPressure'
32 calculatornew.Function = 'abs(Rhop)'
33
34 #exclude P=0 particles
35 threshold1 = Threshold(Input=calculatornew)
36 threshold1.Scalars = ['POINTS', 'MagPressure']
37 threshold1.ThresholdRange = [0.00001,1000000]
38 renderView1 = GetActiveViewOrCreate('RenderView')
39 threshold1Display = Show(threshold1, renderView1)
40 threshold1.Scalars = ['POINTS', 'Idp']
41 threshold1.Scalars = ['POINTS', 'MagPressure']
42 Hide(threshold1, renderView1)
43
44 #Calculate free-surface elevation around column
45 pi=math.pi
46 phiLower=pi/2+pi/viewPointColumns+14*2*pi/viewPointColumns
47 phiUpper=pi/2+pi/viewPointColumns+15*2*pi/viewPointColumns
48
49 for x in range(0,viewPointColumns):
50     thresholdnew = Threshold(Input=threshold1)
51     thresholdnew.Scalars = ['POINTS', 'phi']
52     thresholdnew.ThresholdRange = [phiLower,phiUpper]
53     renderView1 = GetActiveViewOrCreate('RenderView')
54     thresholdnewDisplay = Show(thresholdnew, renderView1)
55     thresholdnew.Scalars = ['POINTS', 'Idp']
56     thresholdnew.Scalars = ['POINTS', 'phi']
57     Hide(thresholdnew, renderView1)
58
59     pythonCalculatornew = PythonCalculator(Input=thresholdnew)
60     pythonCalculatornew.ArrayName = 'FS'

```

```

61     pythonCalculatornew.Expression = 'max(Z)+0.0125'
62
63     phiUpper=phiLower
64     phiLower=phiLower-2*pi/63
65
66     #Group data together for easy processing
67     data=[FindSource('PythonCalculator1')]
68
69     for x in range(2,viewPointColumns+1):
70         data=data+[FindSource('PythonCalculator' + str(x))]
71
72     groupDatasets1 = AppendDatasets(Input=data)
73
74     #Calculate hydrostatic pressure
75     calculatornew = Calculator(Input=groupDatasets1)
76     calculatornew.ResultArrayName = 'HydrostaticPressure'
77     calculatornew.Function = '(FS-Z)*9810*abs(Rhop)/abs(Rhop+1e-15)'
78
79     #Calculate non-hydrostatic pressure
80     calculatornew = Calculator(Input=calculatornew)
81     calculatornew.ResultArrayName = 'NonHydrostaticPressure'
82     calculatornew.Function = 'Rhop-HydrostaticPressure'
83
84     #Calculate hydrostatic force
85     calculatornew = Calculator(Input=calculatornew)
86     calculatornew.ResultArrayName = 'HydrostaticForce'
87     calculatornew.Function = 'HydrostaticPressure*0.0125*0.0125*(7.52 -
        coordsX)/sqrt((7.52 - coordsX)^2+(coordsY+1e-15)^2)'
88
89     #Calculate non-hydrostatic force
90     calculatornew = Calculator(Input=calculatornew)
91     calculatornew.ResultArrayName = 'NonHydrostaticForce'
92     calculatornew.Function = 'NonHydrostaticPressure
        *0.0125*0.0125*(7.52 - coordsX)/sqrt((7.52 - coordsX)^2+(coordsY+1e
        -15)^2)'
93

```

```
94 #Calculate global total force
95 calculatornew = Calculator(Input=calculatornew)
96 calculatornew.ResultArrayName = 'Force'
97 calculatornew.Function = 'Rhop*0.0125*0.0125*(7.52-coordsX)/sqrt
    ((7.52-coordsX)^2+(coordsY+1e-15)^2)'
98
99 #Calculate global hydrostatic force
100 pythonCalculatornew = PythonCalculator(Input=calculatornew)
101 pythonCalculatornew.ArrayName = 'TotalHydrostaticForce'
102 pythonCalculatornew.Expression = 'sum(HydrostaticForce)'
103
104 #Calculate global non-hydrostatic force
105 pythonCalculatornew = PythonCalculator(Input=pythonCalculatornew)
106 pythonCalculatornew.ArrayName = 'TotalNonHydrostaticForce'
107 pythonCalculatornew.Expression = 'sum(NonHydrostaticForce)'
108
109 #Calculate global total force
110 pythonCalculatornew = PythonCalculator(Input=pythonCalculatornew)
111 pythonCalculatornew.ArrayName = 'TotalForce'
112 pythonCalculatornew.Expression = 'sum(Force)'
113
114 #Calculate pressure relative to hydrostatic pressure for mean
    still-water level
115 calculatornew = Calculator(Input=pythonCalculatornew)
116 calculatornew.ResultArrayName = 'MinusAvHP'
117 calculatornew.Function = '(Rhop-(0.5-Z)*9810)/4905'
118
119 #Visualise in theta-z plane
120 calculatornew = Calculator(Input=calculatornew)
121 calculatornew.ResultArrayName = 'MoveX'
122 calculatornew.Function = '7.52-coordsX'
123
124 calculatornew = Calculator(Input=calculatornew)
125 calculatornew.ResultArrayName = 'MoveY'
126 calculatornew.Function = 'phi*0.2-coordsY'
127
```



```

128 warpByScalar1 = WarpByScalar(Input=calculatornew)
129 warpByScalar1.Scalars = ['POINTS', 'MoveY']
130 warpByScalar1.Normal = [0.0, 1.0, 0.0]
131
132 warpByScalar1 = WarpByScalar(Input=warpByScalar1)
133 warpByScalar1.Scalars = ['POINTS', 'MoveX']
134 warpByScalar1.Normal = [1.0, 0.0, 0.0]
135 renderView1 = GetActiveViewOrCreate('RenderView')
136 Show(warpByScalar1, renderView1)

```

D.3 Fluid particles around cylinder

```

1 from paraview.simple import *
2
3 fluid_000 = FindSource('Fluid_0*')
4
5 #Get x-coordinate of particles
6 calculatornew = Calculator(Input=fluid_000)
7 calculatornew.ResultArrayName = 'X'
8 calculatornew.Function = 'coordsX'
9
10 #Get y-coordinate of particles
11 calculatornew = Calculator(Input=calculatornew)
12 calculatornew.ResultArrayName = 'Y'
13 calculatornew.Function = 'coordsY'
14
15 #Calculate distance from cylinder centre
16 calculatornew = Calculator(Input=calculatornew)
17 calculatornew.ResultArrayName = 'CylinderDistance'
18 calculatornew.Function = 'sqrt((7.52-coordsX)^2+(0-coordsY)^2)'
19
20 #Include only particles close to cylinder
21 thresholdnew = Threshold(Input=calculatornew)
22 thresholdnew.Scalars = ['POINTS', 'CylinderDistance']
23 Lower=0.125
24 Upper=0.125+0.0125*1.3

```

```

25 thresholdnew.ThresholdRange = [Lower,Upper]
26 renderView1 = GetActiveViewOrCreate('RenderView')
27 thresholdnewDisplay = Show(thresholdnew, renderView1)
28 thresholdnew.Scalars = ['POINTS', 'Idp']
29 thresholdnew.Scalars = ['POINTS', 'CylinderDistance']
30 Hide(thresholdnew, renderView1)
31
32 #Calculate phi for each particle
33 calculatornew = Calculator(Input=thresholdnew)
34 calculatornew.ResultArrayName = 'phi'
35 calculatornew.Function = '((coordsY+1e-15)/abs(coordsY+1e-15))*
    acos((0.125^2+(7.52-coordsX)^2-(7.395-coordsX)^2)/(0.25*sqrt
    ((7.52-coordsX)^2+(coordsY+1e-15)^2)))'
36
37 #Visualise in a theta-z plane
38 calculatornew = Calculator(Input=calculatornew)
39 calculatornew.ResultArrayName = 'MoveX'
40 calculatornew.Function = '7.52-coordsX'
41
42 calculatornew = Calculator(Input=calculatornew)
43 calculatornew.ResultArrayName = 'MoveY'
44 calculatornew.Function = 'phi*0.2-coordsY'
45
46 thresholdnew = Threshold(Input=calculatornew)
47 thresholdnew.Scalars = ['POINTS', 'Rhop']
48 thresholdnew.ThresholdRange = [0,0]
49 renderView1 = GetActiveViewOrCreate('RenderView')
50 thresholdnewDisplay = Show(thresholdnew, renderView1)
51 thresholdnew.Scalars = ['POINTS', 'Idp']
52 thresholdnew.Scalars = ['POINTS', 'Rhop']
53 Hide(thresholdnew, renderView1)
54
55 #Calculate normal vector x-direction to cylinder surface at
    particle position
56 calculatornew = Calculator(Input=thresholdnew)
57 calculatornew.ResultArrayName = 'NormDirX'

```

```
58 calculatornew.Function = '(coordsX-7.52)/sqrt((coordsX-7.52)^2+(
    coordsY-0)^2)'
59
60 #Calculate normal vector y-direction to cylinder surface at
    particle position
61 calculatornew = Calculator(Input=calculatornew)
62 calculatornew.ResultArrayName = 'NormDirY'
63 calculatornew.Function = '(coordsY-0)/sqrt((coordsX-7.52)^2+(
    coordsY-0)^2)'
64
65 #Calculate particle normal x-velocity
66 calculatornew = Calculator(Input=calculatornew)
67 calculatornew.ResultArrayName = 'NormVelX'
68 calculatornew.Function = 'NormDirX*(Vel_X*NormDirX+Vel_Y*NormDirY)
    ,
69
70 #Calculate particle normal y-velocity
71 calculatornew = Calculator(Input=calculatornew)
72 calculatornew.ResultArrayName = 'NormVelY'
73 calculatornew.Function = 'NormDirY*(Vel_X*NormDirX+Vel_Y*NormDirY)
    ,
74
75 #Calculate particle normal velocity magnitude
76 calculatornew = Calculator(Input=calculatornew)
77 calculatornew.ResultArrayName = 'NormVelMag'
78 calculatornew.Function = 'sqrt(NormVelX*NormVelX+NormVelY*NormVelY
    )'
79
80 #Calculate particle tangential x-velocity
81 calculatornew = Calculator(Input=calculatornew)
82 calculatornew.ResultArrayName = 'TangVelX'
83 calculatornew.Function = 'Vel_X-NormVelX'
84
85 #Calculate particle tangential y-velocity
86 calculatornew = Calculator(Input=calculatornew)
87 calculatornew.ResultArrayName = 'TangVelY'
```

```
88 calculatornew.Function = 'Vel_Y-NormVelY'
89
90 #Calculate particle tangential velocity magnitude
91 calculatornew = Calculator(Input=calculatornew)
92 calculatornew.ResultArrayName = 'TangVelMag'
93 calculatornew.Function = 'sqrt(TangVelX*TangVelX+TangVelY*TangVelY
    )'
94
95 #Calculate particle velocity magnitude
96 calculatornew = Calculator(Input=calculatornew)
97 calculatornew.ResultArrayName = 'VelMag'
98 calculatornew.Function = 'sqrt(Vel_X*Vel_X+Vel_Y*Vel_Y+Vel_Z*Vel_Z
    )'
99
100 warpByScalar1 = WarpByScalar(Input=calculatornew)
101 warpByScalar1.Scalars = ['POINTS', 'MoveY']
102 warpByScalar1.Normal = [0.0, 1.0, 0.0]
103 warpByScalar1.Scalars = ['POINTS', 'CylinderDistance']
104 warpByScalar1.Scalars = ['POINTS', 'MoveY']
105
106 warpByScalar1 = WarpByScalar(Input=warpByScalar1)
107 warpByScalar1.Scalars = ['POINTS', 'MoveX']
108 warpByScalar1.Normal = [1.0, 0.0, 0.0]
109 warpByScalar1.Scalars = ['POINTS', 'CylinderDistance']
110 warpByScalar1.Scalars = ['POINTS', 'MoveX']
111
112 renderView1 = GetActiveViewOrCreate('RenderView')
113 Show(warpByScalar1, renderView1)
```