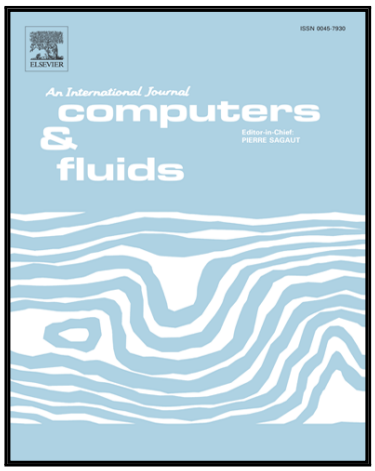


A GPU based compressible multiphase hydrocode for modelling violent hydrodynamic impact problems

Z.H. Ma, D.M. Causon, L. Qian, H.B. Gu, C.G. Mingham,
P. Martínez Ferrer

PII: S0045-7930(15)00236-4
DOI: 10.1016/j.compfluid.2015.07.010
Reference: CAF 2948

To appear in: *Computers and Fluids*

Received date: 17 April 2015
Revised date: 21 June 2015
Accepted date: 9 July 2015

Please cite this article as: Z.H. Ma, D.M. Causon, L. Qian, H.B. Gu, C.G. Mingham, P. Martínez Ferrer, A GPU based compressible multiphase hydrocode for modelling violent hydrodynamic impact problems, *Computers and Fluids* (2015), doi: 10.1016/j.compfluid.2015.07.010

Highlights

- A versatile compressible multiphase hydrocode for marine engineering impact problems.
- Non-physical pressure oscillation near the material interface is successfully avoided.
- It can accurately model violent hydrodynamic slamming problems by properly dealing the crucial compressibility effects of fluid.
- It can also model complex hull cavitation problems in addition to incipient cavitation.
- The hydrocode is successfully accelerated on the GPU by a speed up over 60x.

A GPU based compressible multiphase hydrocode for modelling violent hydrodynamic impact problems

Z. H. Ma[a,*], D. M. Causon[a], L. Qian[a], H. B. Gu[b], C. G. Mingham[a], P. Martínez Ferrer[a]

[a] *Centre for Mathematical Modelling and Flow Analysis, School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University, Manchester M1 5GD, United Kingdom*
[b] *School of Mechanical, Aerospace and Civil Engineering, University of Manchester, Manchester M13 9PL, United Kingdom*

Abstract

This paper presents a GPU based compressible multiphase hydrocode for modelling violent hydrodynamic impacts under harsh conditions such as slamming and underwater explosion. An effort is made to extend a one-dimensional five-equation reduced model (Kapila et al., Two-phase modeling of deflagration-to-detonation transition in granular materials: Reduced equations, Phys. Fluids, 2001, Vol.13, 3002 – 3024) to compute three-dimensional hydrodynamic impact problems on modern graphics hardware. In order to deal with free-surface problems such as water waves, gravitational terms, which are initially absent from the original model, are now considered and included in the governing equations. A third-order finite volume based MUSCL scheme is applied to discretise the integral form of the governing equations. The numerical flux across a mesh cell face is estimated by means of the HLLC approximate Riemann solver. The serial CPU program is firstly parallelised on multi-core CPUs with the OpenMP programming model and then further accelerated on many-core graphics processing units (GPUs) using the CUDA C programming language. To balance memory usage, computing efficiency and accuracy on multi- and many-core processors, a mixture of single and double precision floating-point operations is implemented. The most important data like conservative flow variables are handled with double-precision dynamic arrays, whilst all the other variables/arrays like fluxes, residual and source terms are treated in single precision. Several benchmark test cases including water-air shock tubes, one-dimensional liquid cavitation tube, dam break, 2D cylindrical underwater explosion near a planar rigid wall, 3D spherical explosion in a rigid cylindrical container and water entry of a 3D rigid flat plate have been calculated using the present approach. The obtained results agree well with experiments, exact solutions and other independent numerical computations. This demonstrates the capability of the present approach to deal with not only violent free-surface impact problems but also hull cavitation associated with underwater explosions. Performance analysis reveals that the running time cost of numerical simulations is dramatically reduced by use of GPUs with much less consumption of electrical energy than on the CPU.

Keywords: marine engineering, cavitation, slamming, underwater explosion, parallel computing

1. Introduction

Multiphase flow impact problems are frequently encountered in natural events and industrial applications. Typical examples include wave impacts on fixed structures (offshore Jacket and Jackup platforms, coastal seawalls) and floating bodies (FPSO vessels, LNG cargo ships, wave energy devices), water entry of space capsules and ditching of civil or military aircraft, etc. In these circumstances the structure will potentially experience large or even destructive primary shock loading [1, 2]. In addition, the liquid flow may undergo phase change to form vapour bubbles when the pressure reduces towards the limit of saturated vapour pressure. This phenomenon is usually named cavitation. In close proximity to structures, hull cavitation generated by underwater explosions or breaking waves can collapse very violently causing secondary re-loading on the structure [3, 4]. Hence, accurate prediction of the impact pressure and associated forces is fundamental in order to ensure the survivability of the structure [5].

In extreme events like a plunging breaker impact or the water entry of a blunt body, a pocket or layer of air may be trapped or enclosed by the water body and the water may also be aerated with many small air bubbles. The resulting increase in fluid compressibility will greatly affect the amplitude and duration of the impact loading on the structure [6, 7]. The classical assumption of incompressibility [8–10] will certainly lead to improper predictions and therefore the compressibility of the fluid mixture needs to be carefully considered [11–15].

In order properly to handle the compressibility of fluids for simulating violent hydrodynamic impact events, an appropriate compressible multiphase flow model is necessary to deal with complex physical features such as shock waves, bulk/hull cavitation and multi-valued free surfaces trapping air pockets that may occur in the domain of interest. One choice for these challenging problems is the Baer-Nunziato equations [16] that permit consideration of two materials/phases in non-equilibrium states with two velocity vectors and two pressures. This hyperbolic system consists of eleven equations in 3D and thus requests relatively high computer memory usage and intensive computing. Efforts to numerically solve these equations can be found in the work of e.g. Andrianov and Warnecke [17], Schwendeman et al. [18], Deledicque and Papalexandris [19], Tokareva and Toro [20] and Liang et al. [21] and references therein. Another option, which is of particular interest in the present work, is the reduced-equation model (also named Kapila et al. model) [22]. This model adopts a mechanical equilibrium assumption of one velocity and one pressure and has seven equations in 3D. As a matter of fact, the model can be derived from the Baer-Nunziato equations using an asymptotic analysis in the limit of zero relaxation time [23]. The reduced-equation model in 1D proves to be hyperbolic and complies with the fact that the material derivatives of the phase entropy are zero in the absence of shocks [23, 24]. Although this model does not explicitly include a mass transfer term facilitating the liquid-gas phase change,

*Corresponding author. Tel: +44 (0)161-247-1574
Email address: z.ma@mmu.ac.uk (Z. H. Ma)

cavitation can be mimicked as a mechanical relaxation process (see Section 4.5 and Appendix A of [25]). Since the reduced-equation model only deals with seven equations in 3D, it is computationally less expensive than the Baer-Nunziato equations. Meanwhile, a couple of important mathematical models for compressible multiphase flows have also been proposed by other researchers in the past several decades. Here we do not intend to give a comprehensive review of these models as this is beyond the scope of the present paper.

In the past, the reduced-equation model was analysed and solved in 1D and 2D for problems such as water-air shock tubes, liquid expansion tubes, instability of a material interface and shock bubble interactions [23–28]. It (and its variants) has also been utilised to compute detonation waves [29, 30], solid-fluid interface coupling [31] and powder compaction [32]. Based on past reported works, some researchers have argued that for cavitating flows the reduced-equation model (and its variants) has only been shown able to solve incipient cavitation problems such as a one-dimensional liquid expansion tube [33, 34]. However, the capability of this model (and its variants) to handle cavitating flows with mass transfer e.g. supercavitation has been demonstrated in the works of Saurel *et al.* [35] and Petitpas *et al.* [36]. Further demonstration of its capability to deal with underwater explosion related bulk/hull cavitation or other kinds of cavitating flows of the types as well as water entry of rigid/flexible objects is considered here. These challenging problems are extremely important in marine engineering [3, 5, 33].

It is also noted that previous numerical works carried out for the reduced-equation model were implemented on a single CPU core and not yet adapted to parallel computing on modern many-core graphics-based hardware which can deliver teraflop computing power in very recent years. Since 3D compressible multiphase flow simulations request very much more intensive computation than 1D problems, it is necessary to consider parallel computing implementations via many-core GPUs. The benefits of utilising GPU acceleration in CFD can be found in recent works pertaining to single-phase flows which successfully accelerated the flow solvers by several times and even orders of magnitude [37–41]. It is very likely that applying many core GPU techniques to dramatically reduce the computing time for simulating compressible multiphase flows will be attractive and beneficial to the academic and industrial communities.

When programming the flow algorithms care should be taken properly to handle the use of single- and double-precision data. Naively treating all data with double precision is not favourable to either the program execution speed or memory usage. Therefore, effort needs to be made to balance the memory usage, computing efficiency and accuracy. Special attention should also be paid to the parallelism of the code algorithms to avoid thread-racing conditions, under which no less than two threads attempt to access the same memory location concurrently and at least one access is a write operation. Race conditions may occur unexpectedly when a serial CPU code is directly converted to a GPU program.

In the present work, we make efforts to extend Kapila et al.'s one-dimensional reduced-equation model to simulate three-dimensional compressible multiphase flows on GPUs. In order to deal with free-surface problems like water waves, the gravitational terms, which are absent in the original model, are here considered and introduced into the governing equations. We utilise a third-order finite volume based MUSCL scheme to discretise the integral form of the governing equations. The numerical flux across a mesh cell face is estimated by means of the HLLC approximate Riemann solver. To achieve the objective of balancing memory usage against computing efficiency and accuracy on GPUs, we manage the computed data in a mixed precision style. The most important data like conservative flow variables is handled with double-precision dynamic arrays, whilst all the other variables/arrays like fluxes, residual and source terms are handled in single precision.

We will demonstrate through numerical examples that the present approach possesses the salient features necessary to deal with compressible multiphase flows. Firstly, it is capable of calculating violent hydrodynamic free-surface flow impacts such as slamming problems in ocean and offshore engineering where the fluid compressibility is crucial to the prediction of the duration and peak value of impact loads and which cannot properly be handled by classical numerical wave tanks based on a potential flow model or the incompressible Navier-Stokes equations that are routinely used in engineering design and analysis. Secondly, though it has been claimed by some researchers that for cavitating flows the so-called reduced-equation model (and its variants) has only been applied to solve incipient cavitation problems such as a 1D liquid expansion tube [33], here we will show that the proposed multi-dimensional extension of this model can also deal with complex hull cavitation associated with underwater explosions. These results have not been reported in the literature to the best knowledge of the authors. Last but not least, we will also show that the GPU accelerated flow solver not only provides an excellent performance speedup but also consumes much less electrical energy than its CPU counterpart.

The remainder of the paper is organised as follows. The multi-dimensional extension of the reduced-equation model for compressible multiphase flows is presented in Section 2.1.The numerical method, which utilises a finite volume approach, third-order MUSCL reconstruction and the HLLC approximate Riemann solver to compute the convective fluxes in the governing equations, is presented in Section 2.2. The strategy to parallelise the serial computer code on many-core processors is presented in Section 3. A series of test case benchmarks are carried out to verify the GPU accelerated multiphase flow solver in Section 4. Conclusions are drawn in Section 5.

2. Numerical model

2.1. Governing equations

The mathematical model used here for the flow of a compressible liquid-gas mixture consists of the momentum and energy conservation laws for the fluid. A conservation law of mass for each component is also included. In particular, gravitational effects should be considered and included for free-surface problems like water waves. These conservation laws are expressed by equation (1a). A non-conservative equation (1b) for the advection of gas-phase volume fraction function also needs to be included. The whole system can be expressed as

$$\frac{\partial}{\partial t}\begin{bmatrix}\alpha_1\rho_1\\ \alpha_2\rho_2\\ \rho u\\ \rho v\\ \rho w\\ \rho E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\alpha_1\rho_1 u\\ \alpha_2\rho_2 u\\ \rho u^2+p\\ \rho vu\\ \rho wu\\ \rho hu\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\alpha_1\rho_1 v\\ \alpha_2\rho_2 v\\ \rho uv\\ \rho v^2+p\\ \rho wv\\ \rho hv\end{bmatrix}+\frac{\partial}{\partial z}\begin{bmatrix}\alpha_1\rho_1 w\\ \alpha_2\rho_2 w\\ \rho uw\\ \rho vw\\ \rho w^2+p\\ \rho hw\end{bmatrix}=\begin{bmatrix}0\\ 0\\ 0\\ -\rho g\\ 0\\ -\rho gv\end{bmatrix} \tag{1a}$$

$$\frac{\partial \alpha_1}{\partial t}+\vec{V}\cdot\nabla\alpha_1 = K\nabla\cdot\vec{V} \tag{1b}$$

where α_1 and α_2 satisfying $\alpha_1+\alpha_2=1$ are volume fractions for the gas- and liquid-phases respectively; ρ_1, ρ_2 and $\rho=\alpha_1\rho_1+\alpha_2\rho_2$ are densities for the gas-component, liquid-component and mixture respectively; u, v and w are the three components of the velocity vector $\vec{V}$ along the x, y and z axes; g is the gravitational acceleration; p is the pressure; ρE is the total energy per unit volume; $h=(\rho E+p)/\rho$ is the enthalpy; K is a function given by

$$K=\alpha_1\alpha_2\left(\frac{1}{\rho_1 c_1^2}-\frac{1}{\rho_2 c_2^2}\right)\rho c^2 \tag{2}$$

For each fluid component, the stiffened gas equation of state (SG-EOS) is utilised to correlate the total energy with the pressure, density and velocity. The general form of the SG-EOS is given as

$$\rho e^{\mathrm{T}}=\frac{p+\gamma p_{\mathrm{c}}}{\gamma-1}+\frac{1}{2}\rho\vec{V}^2 \tag{3}$$

in which γ is a polytropic constant and p_{c} is a pressure constant. The fluid mixture has two components in mechanical

equilibrium (identical pressure and velocity); therefore, the total energy for the mixture is defined as

$$\begin{aligned}\rho E &= \alpha_1\rho_1 E_1 + \alpha_2\rho_2 E_2 \\ &= \alpha_1\left(\frac{p+\gamma_1 p_{c,1}}{\gamma_1-1}+\frac{1}{2}\rho_1\vec{V}^2\right)+\alpha_2\left(\frac{p+\gamma_2 p_{c,2}}{\gamma_2-1}+\frac{1}{2}\rho_2\vec{V}^2\right) \\ &= p\left(\frac{\alpha_1}{\gamma_1-1}+\frac{\alpha_2}{\gamma_2-1}\right)+\left(\frac{\alpha_1\gamma_1 p_{c,1}}{\gamma_1-1}+\frac{\alpha_2\gamma_2 p_{c,2}}{\gamma_2-1}\right)+\frac{1}{2}\rho\vec{V}^2\end{aligned} \tag{4}$$

The estimate of function K requests the volume fraction, density and speed of sound for each component as well as mixture density and speed of sound. The speed of sound for the two components can be calculated as

$$c_1 = \sqrt{\frac{\gamma_1}{\rho_1}(p+p_{c,1})}\ ,\quad c_2 = \sqrt{\frac{\gamma_2}{\rho_2}(p+p_{c,2})} \tag{5}$$

The speed of sound for the mixture can be estimated by Wood's formula[42]

$$\frac{1}{\rho c^2} = \frac{\alpha_1}{\rho_1 c_1^2}+\frac{\alpha_2}{\rho_2 c_2^2} \tag{6}$$

For a detailed derivation of the 3D system (1), please refer to the recent work of the authors [11].

If gravitational effects are excluded and only the x direction is considered, equation (1) reduces to a five-equation system which can be named the five-equation-reduced model [23] or Kapila et al. model [22, 25]. If we further neglect the right hand side of equation (1b), then the whole system will become a five-equation-transport model as pointed out by Murrone and Guillard [23]. These authors proved that the material derivatives of the phase entropy are not zero for the transport-model, but the reduced-equation model satisfies these conditions (see Section 2.4 of [23]).

2.2. *Finite volume discretisation*

Since equation (1b) is not conservative, directly using it will present difficulties when dealing with a material interface owing to an inconsistency between the wave speeds (shock wave, rarefaction and contact discontinuity) and the velocity vector $\vec{V}$ [43]. To overcome this difficulty, we adopt Johnsen and Colonius's strategy [43] to transform the advection equation into a conservative formulation

$$\frac{\partial\alpha_1}{\partial t}+\nabla\cdot(\alpha_1\vec{V}) = (\alpha_1+K)\nabla\cdot\vec{V} \tag{7}$$

For a more sophisticated method to deal with the non-conservative volume fraction equation, readers may refer to the work of Saurel *et al.* [25]. Replacing the non-conservative equation (1b) by equation (7), the integral formulation of

the overall quasi-conservative system can now be expressed as

$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{U}\,\mathrm{d}\,\Omega+\oint_{\partial\Omega}\mathbf{F}(\mathbf{U})\,\mathrm{d}\,S=\int_{\Omega}\mathbf{G}\,\mathrm{d}\,\Omega \tag{8}$$

in which Ω represents the flow domain, $\partial\Omega$ is its boundary; the vectors $\mathbf{U}$, $\mathbf{F}$ and $\mathbf{G}$ are given by

$$\mathbf{U}=\begin{bmatrix}\alpha_1\\ \alpha_1\rho_1\\ \alpha_2\rho_2\\ \rho u\\ \rho v\\ \rho w\\ \rho e^{\mathrm{T}}\end{bmatrix},\quad \mathbf{F}=\begin{bmatrix}\alpha_1 q\\ \alpha_1\rho_1 q\\ \alpha_2\rho_2 q\\ \rho uq+pn_x\\ \rho vq+pn_y\\ \rho wq+pn_z\\ \rho e^{\mathrm{T}}q+pq\end{bmatrix}=q\,\mathbf{U}+p\,\mathbf{N}_q,\quad \mathbf{G}=\begin{bmatrix}(\alpha_1+K)\nabla\cdot\vec{V}\\ 0\\ 0\\ 0\\ -\rho g\\ 0\\ -\rho g v\end{bmatrix} \tag{9}$$

where (n_x, n_y, n_z) is a unit normal vector across the boundary $\partial\Omega$, $q=\vec{V}\cdot\vec{n}=un_x+vn_y+wn_z$ and $\mathbf{N}_q=[0,0,0,n_x,n_y,n_z,q]^{\mathrm{T}}$.

Supposing the mesh does not vary with time, the discrete form of equation (8) for a mesh cell m can be written as

$$\Omega_m\frac{\mathbf{U}_m^{n+1}-\mathbf{U}_m^{n}}{\Delta t}+\sum_{f=1}^{N_m}\mathbf{F}_f\,S_f=\Omega_m\,\mathbf{G}_m \tag{10}$$

where the subscript m stands for the mesh cell itself, the subscript f represents a mesh cell face and N_m is the total number of faces for the mesh cell. In the present work, the surface flux term is treated as a discontinuity and computed by an approximate Riemann solver

$$\mathbf{F}_f=\mathbf{F}(\mathbf{U}_f^{+},\mathbf{U}_f^{-}) \tag{11}$$

where the symbols $^{+}$ and $^{-}$ indicate the left and right sides of the face. A third order MUSCL scheme [44] with a slope limiter is utilised to reconstruct the primitive solution variables $\mathbf{W}=(\alpha_1,\rho_1,\rho_2,u,v,w,p)^{\mathrm{T}}$

$$\mathbf{W}_f^{+}=\mathbf{W}_{\mathrm{L}}+\frac{\phi_{\mathrm{L}}}{4}\left[(1-\kappa\phi_{\mathrm{L}})\,\mathbf{\Delta}_{\mathrm{L}}+(1+\kappa\phi_{\mathrm{L}})\,\mathbf{\Delta}\right] \tag{12a}$$

$$\mathbf{W}_f^{-}=\mathbf{W}_{\mathrm{R}}+\frac{\phi_{\mathrm{R}}}{4}\left[(1-\kappa\phi_{\mathrm{R}})\,\mathbf{\Delta}_{\mathrm{R}}+(1+\kappa\phi_{\mathrm{R}})\,\mathbf{\Delta}\right] \tag{12b}$$

where the subscripts L and R represent the left and right neighbouring mesh cells respectively,

$$\begin{cases} \boldsymbol{\Delta}_{\mathrm{L}} & = \mathbf{W}_{\mathrm{L}} - \mathbf{W}_{\mathrm{LL}} \\ \boldsymbol{\Delta}_{\mathrm{R}} & = \mathbf{W}_{\mathrm{RR}} - \mathbf{W}_{\mathrm{R}} \\ \boldsymbol{\Delta} & = \mathbf{W}_{\mathrm{R}} - \mathbf{W}_{\mathrm{L}} \end{cases} \tag{13}$$

in which the subscript LL indicates the left neighbour of the left cell, and RR represents the right neighbour of the right cell. The parameter κ provides different options of upwind or centred schemes

$$\kappa = \begin{cases} -1 & \text{second order upwind} \\ 0 & \text{second order centred} \\ 1/3 & \text{third order semi-upwind} \end{cases} \tag{14}$$

In the present work, $\kappa = 1/3$ is adopted. To prohibit spurious oscillations introduced by the high order interpolation, we apply van Albada's limiter defined as

$$\phi_{\mathrm{L,R}} = \max(0, \frac{2r_{\mathrm{L,R}}}{r_{\mathrm{L,R}}^2 + 1}) \tag{15}$$

where

$$r_{\mathrm{L}} = \frac{\boldsymbol{\Delta}_{\mathrm{L}}}{\boldsymbol{\Delta}}, \quad r_{\mathrm{R}} = \frac{\boldsymbol{\Delta}}{\boldsymbol{\Delta}_{\mathrm{R}}} \tag{16}$$

The reconstructed conservative variables at the left and right sides of a mesh cell face can be easily obtained as

$$\mathbf{U}_f^+ = \mathbf{U}(\mathbf{W}_f^+), \quad \mathbf{U}_f^- = \mathbf{U}(\mathbf{W}_f^-) \tag{17}$$

The numerical flux term represented by equation (11) is calculated by the HLLC approximate Riemann solver defined as

$$\mathbf{F}_f = \begin{cases} \mathbf{F}_{\mathrm{L}} & 0 \le S_{\mathrm{L}} \\ \mathbf{F}_{\mathrm{L}} + S_{\mathrm{L}}(\mathbf{U}_{\mathrm{L}}^* - \mathbf{U}_{\mathrm{L}}) & S_{\mathrm{L}} < 0 \le S_{\mathrm{M}} \\ \mathbf{F}_{\mathrm{R}} + S_{\mathrm{R}}(\mathbf{U}_{\mathrm{R}}^* - \mathbf{U}_{\mathrm{R}}) & S_{\mathrm{M}} < 0 \le S_{\mathrm{R}} \\ \mathbf{F}_{\mathrm{R}} & 0 \ge S_{\mathrm{R}} \end{cases} \tag{18}$$

in which the middle left and right states are evaluated by

$$\mathbf{U}_{\mathrm{L,R}}^* = \frac{(q_{\mathrm{L,R}} - S_{\mathrm{L,R}})\mathbf{U}_{\mathrm{L,R}} + (p_{\mathrm{L,R}}\mathbf{N}_{q_{\mathrm{L,R}}} - p_{\mathrm{L,R}}^*\mathbf{N}_{q_{\mathrm{L,R}}^*})}{q_{\mathrm{L,R}}^* - S_{\mathrm{L,R}}} \tag{19}$$

the intermediate wave speed can be calculated by

$$S_{\mathrm{M}} = q_{\mathrm{L}}^{*} = q_{\mathrm{R}}^{*} = \frac{\rho_{\mathrm{R}} q_{\mathrm{R}}(S_{\mathrm{R}} - q_{\mathrm{R}}) - \rho_{\mathrm{L}} q_{\mathrm{L}}(S_{\mathrm{L}} - q_{\mathrm{L}}) + p_{\mathrm{L}} - p_{\mathrm{R}}}{\rho_{\mathrm{r}}(S_{\mathrm{R}} - q_{\mathrm{R}}) - \rho_{\mathrm{L}}(S_{\mathrm{L}} - q_{\mathrm{L}})} \tag{20}$$

and the intermediate pressure may be estimated as

$$p_{\mathrm{M}} = \rho_{\mathrm{L}}(q_{\mathrm{L}} - S_{\mathrm{L}})(q_{\mathrm{L}} - S_{\mathrm{M}}) + p_{\mathrm{L}} = \rho_{\mathrm{R}}(q_{\mathrm{R}} - S_{\mathrm{R}})(q_{\mathrm{R}} - S_{\mathrm{M}}) + p_{\mathrm{R}} \tag{21}$$

The left and right state wave speeds are computed by

$$S_{\mathrm{L}} = \min(q_{\mathrm{L}} - c_{\mathrm{L}}, \tilde{q} - \tilde{c}) \tag{22a}$$

$$S_{\mathrm{R}} = \max(q_{\mathrm{R}} + c_{\mathrm{R}}, \tilde{q} + \tilde{c}) \tag{22b}$$

where $\tilde{q}$ is an averaged velocity component evaluated by the component Roe-averages and the averaged speed of sound $\tilde{c}$ is calculated by the formulation proposed by Hu et al. [45] as

$$\tilde{c}^2 = \widetilde{\Psi} + \widetilde{\Gamma}\widetilde{\left(\frac{p}{\rho}\right)} \tag{23}$$

More details concerning the use of equation (23) can be found in reference [45].

The first component in the source function $\mathbf{G}$ contains a divergence of the velocity vector, therefore its evaluation is less straightforward than other components in $\mathbf{G}$. To handle this problem for a mesh cell m, we assume that

$$\alpha_1(\Omega_m) = \frac{1}{\Omega_m}\int_{\Omega_m} \alpha_m \,\mathrm{d}\,\Omega = \overline{\alpha_{1,m}} = \alpha_{1,m} \tag{24a}$$

$$K(\Omega_m) = \frac{1}{\Omega_m}\int_{\Omega_m} K_m \,\mathrm{d}\,\Omega = \overline{K_m} = K_m \tag{24b}$$

in which α_1 and K_i are stored at the mesh cell centre. Then we compute the volume integral of the first component in the source function as

$$\int_{\Omega_m} G_1 \,\mathrm{d}\,\Omega = \int_{\Omega_m} (\alpha_1 + K)\nabla\cdot\vec{V}\,\mathrm{d}\,\Omega = (\alpha_{1,m} + K_m)\oint_{\partial\Omega_m} \vec{V}\cdot\vec{n}\,\mathrm{d}\,S = (\alpha_{1,m} + K_m)\sum_{f=1}^{N_m} q_f S_f \tag{25}$$

Similarly, we use the HLLC Riemann solver to compute q_f as

$$q_f = \begin{cases} q_{\mathrm{L}} & 0 \le S_{\mathrm{L}} \\ S_{\mathrm{M}}(q_{\mathrm{L}} - S_{\mathrm{L}})/(S_{\mathrm{M}} - S_{\mathrm{L}}) & S_{\mathrm{L}} < 0 \le S_{\mathrm{M}} \\ S_{\mathrm{M}}(q_{\mathrm{R}} - S_{\mathrm{R}})/(S_{\mathrm{M}} - S_{\mathrm{R}}) & S_{\mathrm{M}} < 0 \le S_{\mathrm{R}} \\ q_{\mathrm{R}} & 0 \ge S_{\mathrm{R}} \end{cases} \tag{26}$$

2.3. TVD Runge-Kutta time advancement

In the present work, a third order TVD (total variation diminishing) Runge-Kutta scheme is applied to update the numerical solution from time level n to $n + 1$

$$\mathbf{U}^{(1)} = \mathbf{U}^n + \Delta t \mathbf{L}(\mathbf{U}^n) \tag{27a}$$

$$\mathbf{U}^{(2)} = \frac{3}{4}\mathbf{U}^n + \frac{1}{4}\mathbf{U}^{(1)} + \frac{1}{4}\Delta t \mathbf{L}(\mathbf{U}^{(1)}) \tag{27b}$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\mathbf{U}^n + \frac{2}{3}\mathbf{U}^{(1)} + \frac{2}{3}\Delta t \mathbf{L}(\mathbf{U}^{(2)}) \tag{27c}$$

For a mesh cell m, the partial differential operator $\mathbf{L}$ is defined as

$$\mathbf{L}(\mathbf{U}_m) = -\left(\sum_{f=1}^{N_m} \mathbf{F}_f S_f - \Omega_m \mathbf{G}_m \right) / \Omega_m \tag{28}$$

3. Implementation on multi-core and many-core processors

Traditionally, computer software has been written for serial computation. To solve a problem in scientific computing, or other areas, an appropriate algorithm is constructed and implemented as a serial stream of instructions. These instructions are executed one by one on a CPU. With increases of CPU clock frequency, the software execution speed can easily be improved. Although the CPU clock frequency has become faster and faster it has now hit a power wall potentially consuming exponentially increasing power with each factorial increase in operating frequency. This clearly poses manufacturing, system design and deployment problems that cannot be justified in the face of the diminished gains in performance. Therefore processor vendors like Intel, AMD and NVIDIA now manufacture multi-core CPUs or many-core GPUs in order to continue delivering regular performance improvements. This in turn requires software developers employing parallel programming models to (re-)design their code in order to harvest the powerful computing capability of multi- and many-core processors.

In the present work, the computer program for solving compressible multiphase flow problems is developed in three continuous stages. We firstly code the program in a single-CPU-thread manner. Then we add OpenMP directives

to the program to utilise multi-core CPUs. Finally we use CUDA C language to port the code from CPU to GPU. The whole computer program is coded in C++ language. A sketch of the main program, which prepares the grid, initialises the flow field and selects the solver, is illustrated in Listing 1.

Listing 1: The main program.

```
1  int main()
2  {
3    flowSystem freeFall;
4
5    freeFall.gridSet();
6    freeFall.flowSet();
7
8    if(CPU_Compute==true) //slove the flow on the CPU
9      freeFall.flowSolver_CPU();
10   else //run simulation on the GPU
11     freeFall.flowSolver_GPU();
12
13   return 0;
14 }
```

3.1. Mixed precision data

We use C++ classes to encapsulate data and functions. Key data including flow variables **U**, fluxes **F**, source terms **G** and residuals **R** are handled within the structure of arrays. As is well known, single-precision computer programs are usually faster and request less memory storage than their double-precision counterparts. Naively treating all data with double precision is not beneficial to the program execution speed. Therefore, effort needs to be made to balance memory usage against computing efficiency and accuracy. In due consideration of these important factors, we code our computer program in a mixed precision style. In the program, we declare the conservative flow variables **U** with double precision dynamic arrays, whilst all the other variables/arrays can be treated as single precision.

For robustness, the numerical code should be able flexibly to choose double precision or mixed precision to manage the computed data. For this purpose we create a C++ type alias *REAL* to indicate double-precision or single-precision in a header file as shown in Listing 2. This header file can be included in all of the C++ and CUDA C source files. As shown in Listing 3, except the conservative flow variables that are explicitly declared with double precision, all the other variables are declared with *REAL* type. When line 4 in Listing 2 is not commented out, all the flux, residual and source terms in List 3 will actually be treated with single precision. Otherwise, all the data will be handled by double precision. An analysis will be conducted in Section 4.7 to explicitly show the advantages of mixed precision data treatment for reducing memory usage and improving computing performance.

Listing 2: Header file "header.hh" .

```
#ifndef HEADER_HH
#define HEADER_HH

#define SINGLE_PRECISION

#ifdef SINGLE_PRECISION
typedef float REAL;
#else
typedef double REAL;
#endif

#endif
```

Listing 3: Mix-precision data in the program.

```
#include "header.hh"

class flowSystem{
protected:
  double *U1, *U2, ..., *U7; //
  REAL *F1, *F2, ..., *F7; //flux
  REAL *R1, *R2, ..., *R7; //residual
  REAL *G1, *G2, ..., *G7; //source
  ...
public:
  //to declare the functions
}
```

3.2. *Multi-core CPU with OpenMP*

Once the serial CPU program is finished and verified, OpenMP directives can be utilised to parallelise the code. An example is given in Listing 4 to demonstrate the procedure. The code from line 8 to 17 represents an update of the solution data $\mathbf{U}$ from the time step n to $n+1$. To distribute this part of work to several CPU cores, OpenMP directives are introduced from line 5 to 7. When executing the program, the OpenMP library will automatically divide the task among cooperative CPU cores. OpenMP is relatively easy and fast to implement as programmers only need to add simple directives to the code in most cases.

Listing 4: Updating the flow on the CPU.

```
1  #include "header.hh"
2  void flowSystem::flowUpdate_CPU(REAL dt)
3  {
4    int idx;
5  #pragma omp parallel for default(none) \
6    shared(dt) \
7    private(idx)
8    for(idx=0;idx<nCell;idx++)
9      {
10       U1[idx] = U1[idx] - dt*R1[idx]/Volume[idx];
11       U2[idx] = U2[idx] - dt*R2[idx]/Volume[idx];
12       U3[idx] = U3[idx] - dt*R3[idx]/Volume[idx];
13       U4[idx] = U4[idx] - dt*R4[idx]/Volume[idx];
14       U5[idx] = U5[idx] - dt*R5[idx]/Volume[idx];
15       U6[idx] = U6[idx] - dt*R6[idx]/Volume[idx];
16       U7[idx] = U7[idx] - dt*R7[idx]/Volume[idx];
17     }
18 }
```

3.3. Many-core GPU with CUDA C

Developing CUDA code is much more demanding because the programmer needs to understand the hardware architecture and to explicitly handle the memory objects on the GPU. Hence, the development cycle is much longer and more error-prone than OpenMP. However, it is worth spending time and effort on GPU programming to obtain orders-of-magnitude speedup. A GPU computing program usually needs the CPU to input the information from the hard drive (or elsewhere) and pre-process the data. The data is then sent from the CPU to the GPU. The complete or partial computing task is off-loaded to the GPU. Once the task is finished, the result is transferred back to the CPU. This general procedure for a GPU based CFD program is illustrated in Figure 1.

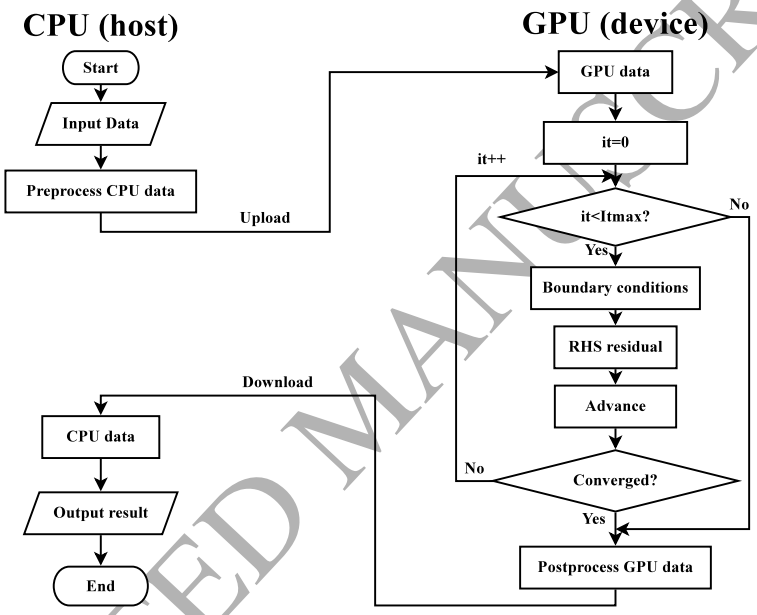


Figure 1: A general procedure of GPU computing program. The CPU is responsible to input/output the data, the GPU is recruited to tackle the computing intensive task.

3.3.1. CUDA thread and memory hierarchy

Compared to MPI or OpenMP, CUDA provides much finer-grain parallelism. It can launch thousands (or even many thousands) of lightweight threads on the graphical hardware. Each thread can be properly used to deal with a computational stencil. It can access the data stored in the memory and carry out algebraic operations on the data, etc. CUDA uses grid and blocks to manage these threads, and every thread has a unique index.

Figure 2 presents a simple layout of the CUDA thread and memory hierarchy. The data stored in the CPU memory is firstly copied to the global memory of the GPU device. On the device, every thread can access the global memory. All the threads in the same block can access the shared memory belonging to this block and each thread can have its own private registers. Proper use of shared memory will greatly benefit the program performance especially

when there is a lot of data reuse [46]. Using constant memory provided on GPUs may reduce the required memory bandwidth [46]. In the present work, we use constant memory to store important parameters such as the pressure constants, polytropic parameters and gravitational acceleration. Adequate threads can be created on the device to accomplish a one-to-one mapping of the CFD grid. A CUDA grid can have up to three dimensions to manage the thread blocks and map the corresponding CFD grid.

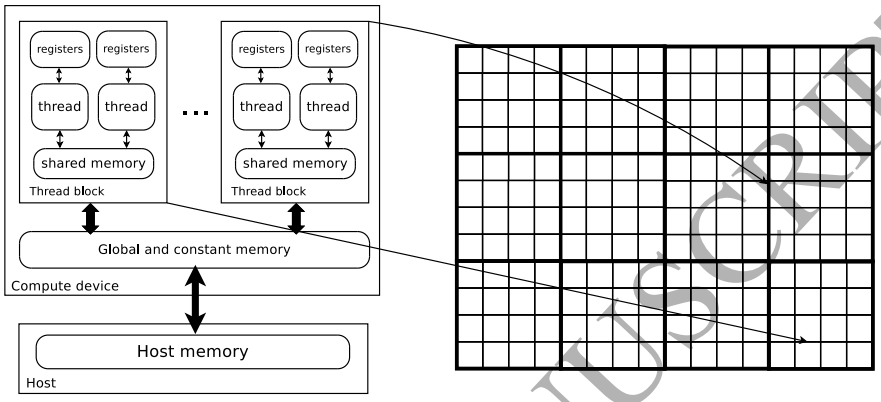


Figure 2: CUDA memory and thread hierarchy. A one-to-one mapping of the CFD grid can be established if adequate threads are created on the CUDA device.

3.3.2. CUDA kernel and its arguments

An example of a GPU kernel function, which updates the flow solution from time step n to $n + 1$ is presented in Listings 5. We use three-dimensional thread blocks to manage each individual CUDA thread. Local 3D thread indices are firstly converted to the global 3D indices (see line 6 to 8 in Listing 5). And the global indices are converted to the actual position index of the CFD mesh cell stored in the linear global memory (see line 12 in Listing 5). Then the flow variables are advanced to a new time step (see line 14 to 20 in Listing 5).

When converting a CPU function to a GPU kernel, attention should be paid to the arguments of the function. The whole size of the arguments of a kernel function is limited to 256 bytes on the CUDA device. This may cause some problems if the function has too many arguments and their size exceed the upper limit. One way to work around this issue is to declare on the device a structure of device pointers directing to the device memory, and pass that device structure to the kernel instead of a long list of device pointers themselves. More details of this strategy can be found in Section 3.5 of [47]. Another solution is to split the big kernel into several smaller ones, and only a portion of the arguments are passed to each small kernel. For example, face flux terms for 3D Cartesian grid can be split into three directions, therefore we can code three kernels to compute the flux along x, y and z directions respectively as presented in Listing 6 and 7. If necessary, three different CUDA streams can be created to run the three split kernel functions concurrently. One should bear in mind that whether independent streams can be concurrently executed relies on the

graphical hardware itself and the resources requested by the GPU kernel functions.

Listing 5: Updating the flow on the GPU.

```
#include "header.hh"

__global__ void flowUpdate_CUDA(double *U1, double *U2,
  double *U3, double *U4, double *U5, double *U6, double *U7,
  REAL *R1, REAL *R2, REAL *R3, REAL *R4, REAL *R5, REAL *R6,
  REAL *R7, REAL *Volume, REAL dt)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int k = blockIdx.z * blockDim.z + threadIdx.z;

  if(k<d_cellNK && j<d_cellNJ && i<d_cellNI)
    {
      int idx=k*d_cellNI*d_cellNJ+j*d_cellNI+i;

      U1[idx] = U1[idx] - dt*R1[idx]/Volume[idx];
      U2[idx] = U2[idx] - dt*R2[idx]/Volume[idx];
      U3[idx] = U3[idx] - dt*R3[idx]/Volume[idx];
      U4[idx] = U4[idx] - dt*R4[idx]/Volume[idx];
      U5[idx] = U5[idx] - dt*R5[idx]/Volume[idx];
      U6[idx] = U6[idx] - dt*R6[idx]/Volume[idx];
      U7[idx] = U7[idx] - dt*R7[idx]/Volume[idx];
    }
}
```

Listing 6: A big CUDA kernel with potential to exceed 256 bytes limit by its arguments.

```
#include "header.hh"

__global__ void flowFaceFlux(
  double *U1, double *U2, double *U3, double *U4, double *U5, double *U6, double *U7,
  REAL *F1x, REAL *F2x, REAL *F3x, REAL *F4x, REAL *F5x, REAL *F6x, REAL *F7x,
  REAL *F1y, REAL *F2y, REAL *F3y, REAL *F4y, REAL *F5y, REAL *F6y, REAL *F7y,
  REAL *F1z, REAL *F2z, REAL *F3z, REAL *F4z, REAL *F5z, REAL *F6z, REAL *F7z);
```

Listing 7: Three split CUDA kernels for face flux term computation.

```
#include "header.hh"

__global__ void flowFaceFlux_x(
  double *U1, double *U2, double *U3, double *U4, double *U5, double *U6, double *U7,
  REAL *F1x, REAL *F2x, REAL *F3x, REAL *F4x, REAL *F5x, REAL *F6x, REAL *F7x);

__global__ void flowFaceFlux_y(
```

```
    double *U1, double *U2, double *U3, double *U4, double *U5, double *U6, double *U7,
    REAL *F1y, REAL *F2y, REAL *F3y, REAL *F4y, REAL *F5y, REAL *F6y, REAL *F7y);

__global__ void flowFaceFlux_z(
    double *U1, double *U2, double *U3, double *U4, double *U5, double *U6, double *U7,
    REAL *F1z, REAL *F2z, REAL *F3z, REAL *F4z, REAL *F5z, REAL *F6z, REAL *F7z);
```

3.3.3. Thread race conditions

Many edge-/face-based finite volume programs loop over every mesh edge/face, compute the flux and then add/-subtract the flux to/from its two neighbour mesh cells. This algorithm is well suited to serial computing. However it does not fit parallel computing since it may encounter thread race conditions under which a mesh cell may be accessed by conflicting read and write operations at the same time.

To prevent race conditions, we use the store-and-gather strategy in the present work. When computing the residual for each mesh cell, we first calculate the fluxes across each mesh face and store this information in the GPU global memory. Then we gather the face fluxes and source terms for every mesh cell to obtain the residual. This approach can successfully avoid thread race conditions that a serial pattern edge-based finite volume method program inherently encounters. Other approaches like the colour edge technique (no two edges of the same colour update the same mesh cell) and redundant computing [38] can also be used to circumvent race conditions.

3.4. Hardware and software platform

The following numerical simulations presented in this paper are performed on two platforms, for which the configurations are listed in Table 1. Platform 1 has one Intel Core i7-2600 CPU with a NVIDIA Quadro 2000 graphics card, while platform 2 has two Intel Xeon E5-2650 CPUs with a NVIDIA Tesla K20X card. Specifications of the CPUs and GPUs are listed in tables 2 and 3 respectively. For both platforms, the optimisation level for GCC and NVCC compilers is set to `-O3` without debugging and profiling options.

Table 1: Description of the platforms.

Platform	CPU	RAM	GPU	Operating System	GCC	CUDA
1	1× Intel Core i7-2600	16 GB	NVIDIA Quadro 2000	Debian Linux	4.6.3	5.0
2	2× Intel Xeon E5-2650	64 GB	NVIDIA Tesla K20X	CentOS Linux	4.8.1	6.0

4. Numerical results and discussions

A series of benchmark test cases including water-air shock tubes, a liquid expansion tube, dam break, underwater explosions in the proximity of a rigid planar or curved wall and water entry of a rigid flat plate are computed to verify

the flow algorithm on many-core hardware and demonstrate the potential performance enhancements. Here, we first present the computed results on the NVIDIA Tesla K20X graphics card (with mixed-precision data) to verify the numerical model in sections 4.1 ~ 4.6. Then a performance analysis will be conducted in Section 4.7 to investigate the memory usage, energy consumption and computing efficiency of CPU and GPU.

4.1. Water-air shock tubes

Here, we consider a group of shock tube problems involving water-air mixtures. The shock tube initially has an imaginary membrane in the middle of the tube, or at other specified locations, which separates it into left and right parts filled with fluids at different thermodynamic states. In total, we report here on four of these shock tube tests, for which the initial conditions, stopping times, membrane positions and number of mesh cells used are listed in Table 4 (case 1~4). Gravitational effects are not included for these problems.

Computed results for the four cases are shown in figures 3 ~ 6. Computations of water-air shock tube problems are known to be challenging for numerical methods, particularly around the material interface where spurious nonphysical oscillations may occur. A close examination of these results shows generally good agreement with exact solutions and published independent numerical predictions with other flow codes. This verifies that the present method resolves wave speeds correctly and does not produce spurious nonphysical oscillations near shocks or at material interfaces.

In particular in Figure 6, it can be seen, in contrast, that Plumerault et al.'s solution exhibits strong nonphysical numerical oscillations around the shock and material interface and the present method has superior performance. Agreement with the exact solution is good apart from a slight underprediction of the discontinuity in density in the present results at the material interface in the region around $x = 0.5$.

4.2. Cavitation test

This problem is used to test the capability of a numerical method to deal with liquid flow cavitation. As shown in Figure 7, a tube of 1 m length is filled with water of density $\rho_2 = 1000$ kg/m^3 and volume fraction $\alpha_2 = 0.99$. A small volume of air ($\alpha_1 = 0.01$, $\rho_1 = 1$ kg/m^3) is uniformly present in the tube. An imaginary barrier placed in the middle separates the tube into two parts. On the left side, the velocity is set to $u = -100$ m/s. On the right side, the velocity is set to $u = 100$ m/s. The initial pressure is everywhere set to $p = 1$ bar. Gravitational effects are not included for this case. The corresponding polytropic indices and pressure constants for water and air are listed in Table 4 (case 5). We discretise the domain with 1000 uniform mesh cells and compute the flow to time $t = 1.85$ ms. The obtained solution is depicted in Figure 8.

Rarefaction waves are observed to propagate outwards from the centre of the tube and the pressure decreases symmetrically from the middle in both left and right directions. Responding to the reduction of pressure, air entrained

Table 2: Specifications of the Intel Core i7-2600 and Xeon E5-2650 CPUs.

CPU	Clock rate	Cores	Cache	Power consumption
Intel Core i7-2600	3.4 GHz	4	8 MB	95 W
Intel Xeon E5-2650	2.6 GHz	8	20 MB	95 W

Table 3: Specifications of the NVIDIA Quadro 2000 and Tesla K20X GPUs.

	Quadro 2000	Tesla K20X
Clock rate	1.25 GHz	732 MHz
Global memory	1 GB	6 GB
Shared memory	48 KB	64 KB
Registers per block	32768	49152
Number of multiprocessors	4	14
Cores per multiprocessor	48	192
Total number of cores	192	2688
Compute capability	2.1	3.5
Power consumption	62 W	235 W

Table 4: Initial conditions for water-air shock tubes (case 1~4) and 1D cavitation test (case 5).

	Case	1	2	3	4	5
	Membrane (x)	0.5	0.5	0.7	0.5	0.5
	Stop time(μs)	237.44	200	200	551.37	1850
	Mesh cells	1000	1000	1000	800	1000
Left	α_1	10^{-8}	0.5	0.2	0.00195	0.01
	ρ_1(kg/m^3)	50	50	1	6.908	1
	γ_1	1.4	1.4	1.4	1.4	1.4
	P_{c1}(MPa)	0	0	0	0	0
	ρ_2(kg/m^3)	1000	1000	1000	1027	1000
	γ_2	4.4	4.4	4.4	4.4	4.4
	P_{c2}(MPa)	600	600	600	600	600
	u	0	0	0	0	-100
	p(bar)	10000	10000	10000	10	1
Right	α_1	$1-10^{-8}$	0.5	0.8	0.01	0.01
	ρ_1(kg/m^3)	50	50	1	1.33	1
	γ_1	1.4	1.4	1.4	1.4	1.4
	P_{c1}(MPa)	0	0	0	0	0
	ρ_2(kg/m^3)	1000	1000	1000	1027	1000
	γ_2	4.4	4.4	4.4	4.4	4.4
	P_{c2}(MPa)	600	600	600	600	600
	u	0	0	0	0	100
	p (bar)	1	1	1	1	1

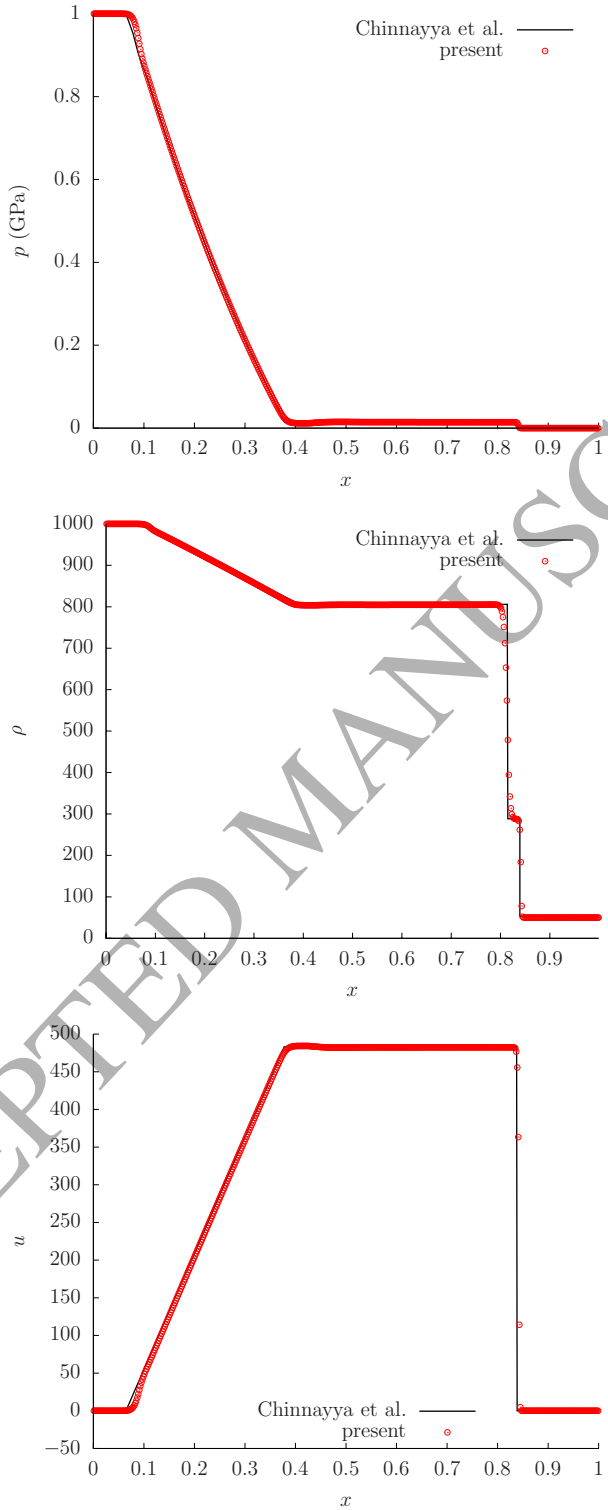
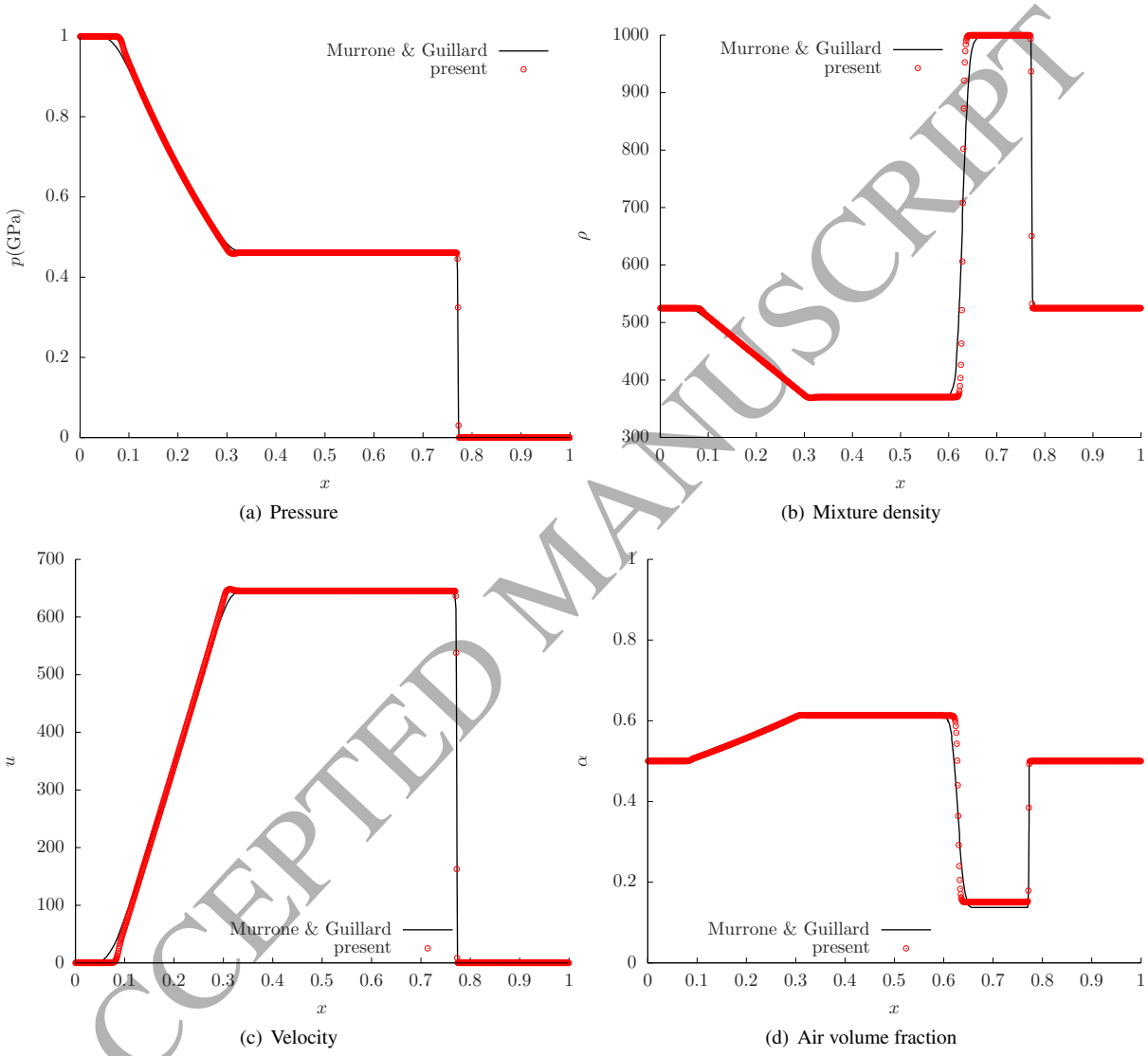


Figure 3: Comparison of the exact solution (Chinnayya et al. [48]) and the present results (red dots) for water-air shock tube 1 (Top: pressure; Middle: mixture density; Bottom: velocity).



(a) Pressure (b) Mixture density

(c) Velocity (d) Air volume fraction

Figure 4: Comparison of the present computation (red curve) and Murrone and Guillard's results [23] (black line) for water-air shock tube 2.

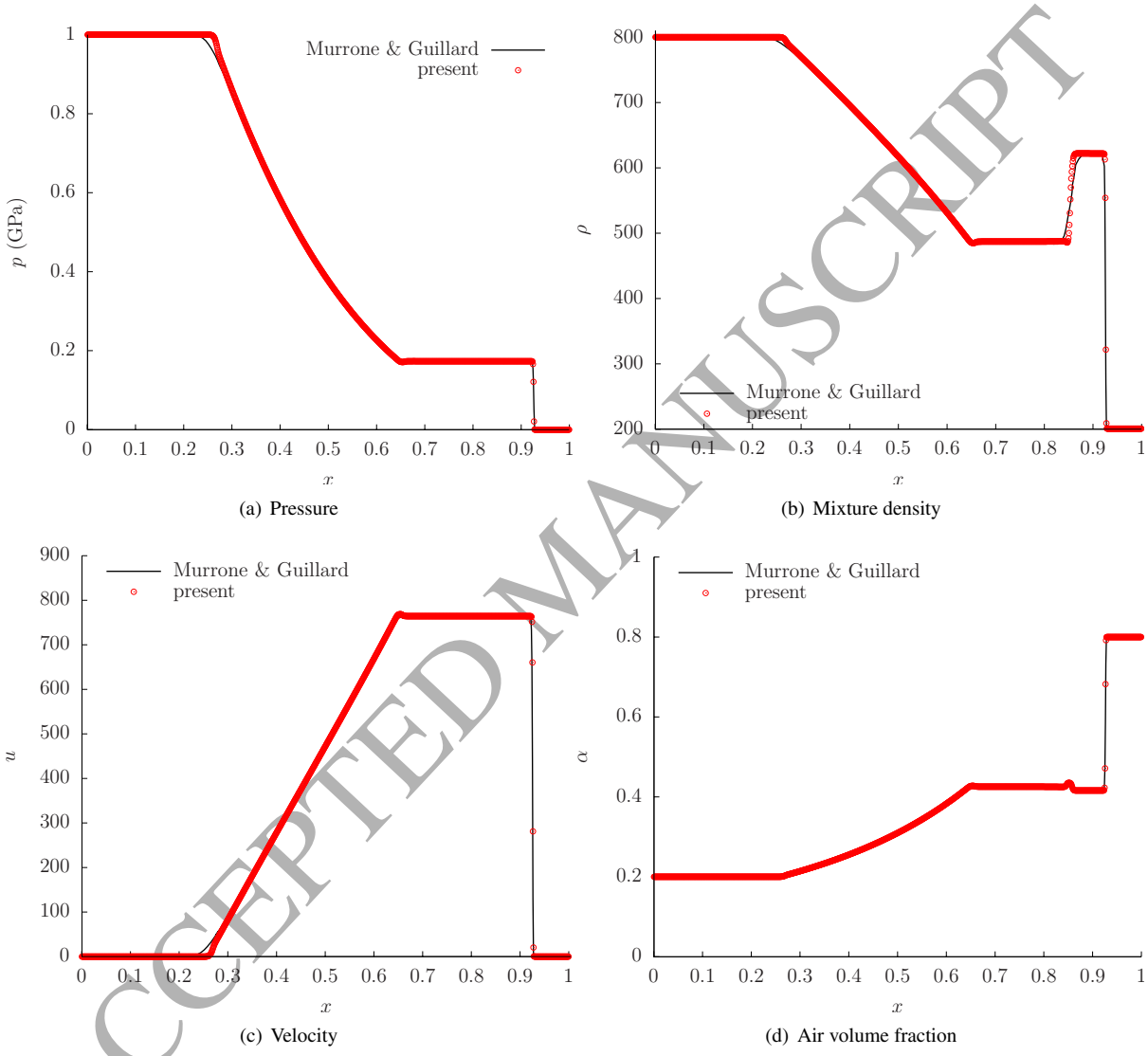
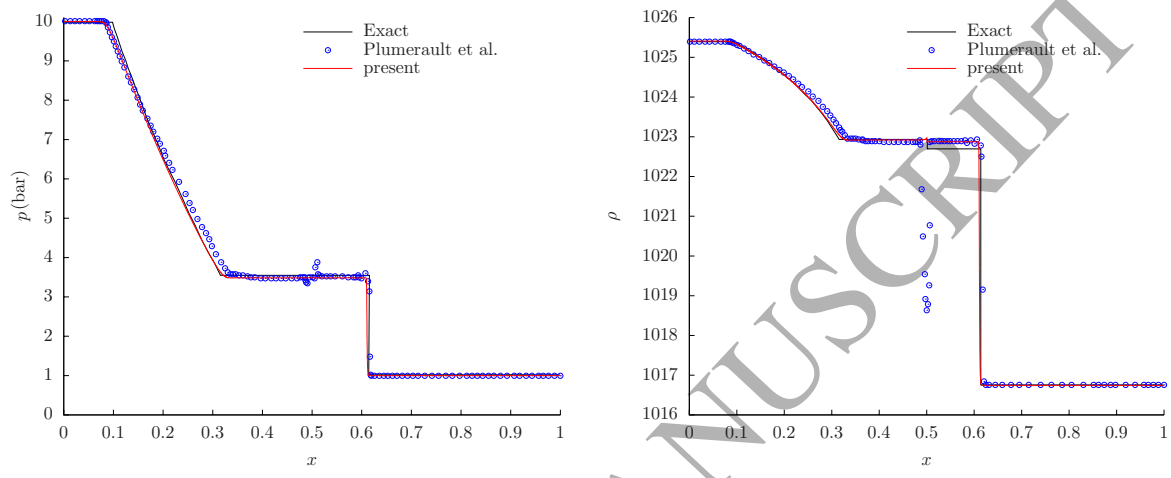
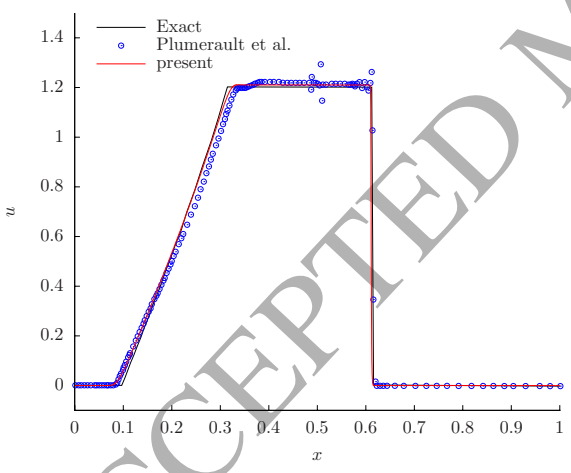


Figure 5: Comparison of the present computation (red curve) and Murrone and Guillard's results [23] (black line) for water-air shock tube 3.

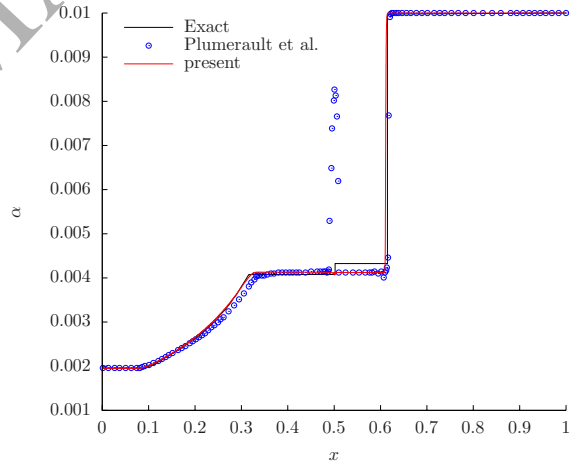


(a) Pressure

(b) Mixture density



(c) Velocity



(d) Air volume fraction

Figure 6: Comparison of the present computation (red curve), exact solution and Plumerault et al.'s results [14] (blue dots) for water-air shock tube 4.

in the water begins to expand and its volume fraction increases. Accordingly, the volume fraction of the liquid phase falls. This creates a cavitation pocket in the middle of the tube and results in the dynamic appearance of two interfaces that were not present initially [25]. These results indicate that the present method can deal with low pressures quite well neither violating the physics nor producing a nonphysical negative value of absolute pressure that would cause the solver to diverge numerically. The model can also handle the creation, during the computation, of interfaces not present initially. Excellent agreement with the exact solution and Saurel et al.'s calculated results is observed in Figure 8.

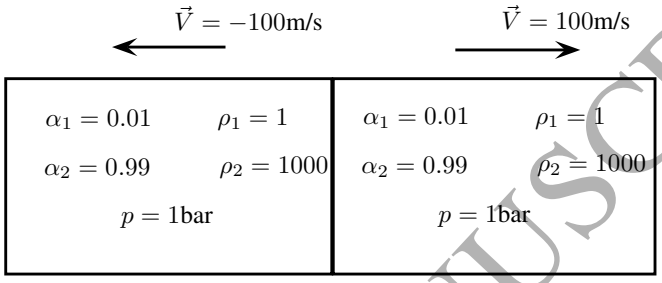


Figure 7: Setup of the liquid expansion tube

4.3. Dam break

This is a classical problem in hydrodynamics studied using both laboratory experiments and numerical simulations. The geometrical configuration is shown in Figure 9. The length of the tank is 0.5 m and its height is 0.15 m. The height of the water column is 0.12 m and its width is a = 0.06 m. The water column between the left wall of the tank and a barrier is initially still. Once the barrier is removed instantaneously from the tank, the water will collapse under gravity. The free surface will continuously deform with time as shown in Figure 10. Two important parameters are gauged in the computations. One is the position of the water front along the bottom of the tank and the other is the height of the water column along the left boundary of the tank. Their dimensionless forms against dimensionless time are shown in Figure 11. Please note that the water front position is divided by the initial width of the water column a and the time is multiplied by $\sqrt{2g/a}$. The water height is divided by the initial height of the water column $2a$ and time is multiplied by $\sqrt{g/a}$. The present computed results agree well with Martin and Moyce's experiments [49] and are slightly superior to Murrone and Guillard's predictions [23].

4.4. 2D cylindrical underwater explosion near a planar rigid wall

This case is chosen to investigate a shock wave interaction with a planar rigid wall in water with possible hull cavitation loading/re-loading on the wall. Figure 12 depicts the set up for this problem. At the initial state, an explosive

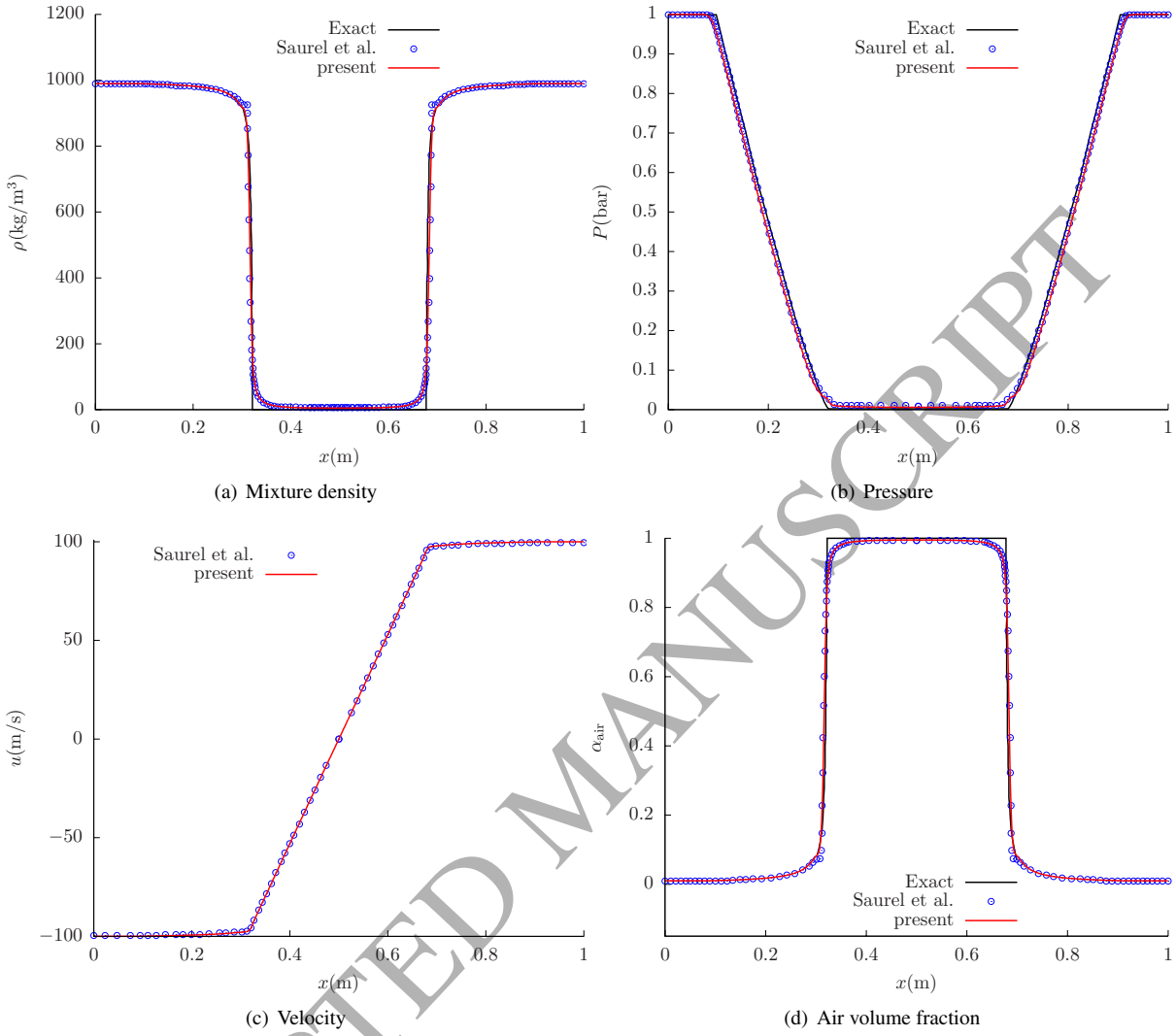


Figure 8: Liquid expansion tube with a cavitation pocket. The present results (dashed line) are compared to the exact solution (black line) and Saurel et al.'s independent computation [25] (green dots). The domain is uniformly partitioned with 1000 mesh cells.

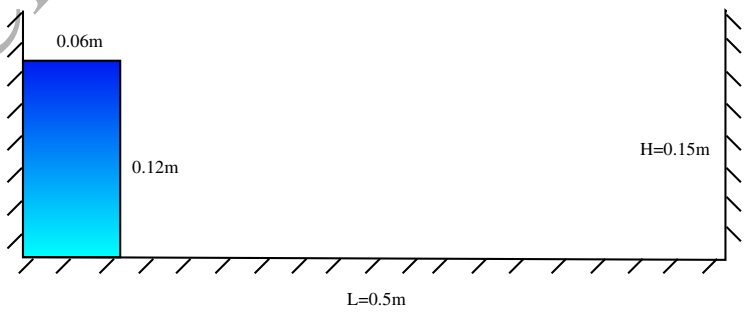


Figure 9: Configuration of the dam break problem. The density of air is $\rho_1 = 1$ kg/m^3 and the density of water is $\rho_2 = 1000$ kg/m^3. The domain is uniformly divided into 200×60 mesh cells. The top of the tank is an open boundary; the other three boundaries are solid walls.

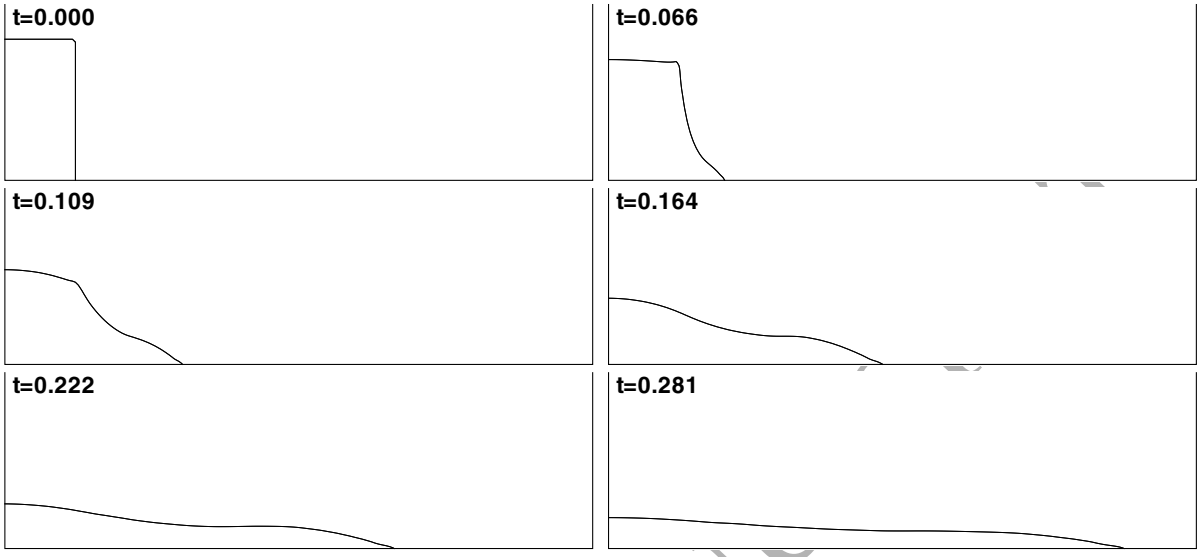


Figure 10: Snapshots of the free surface for the dam break problem.

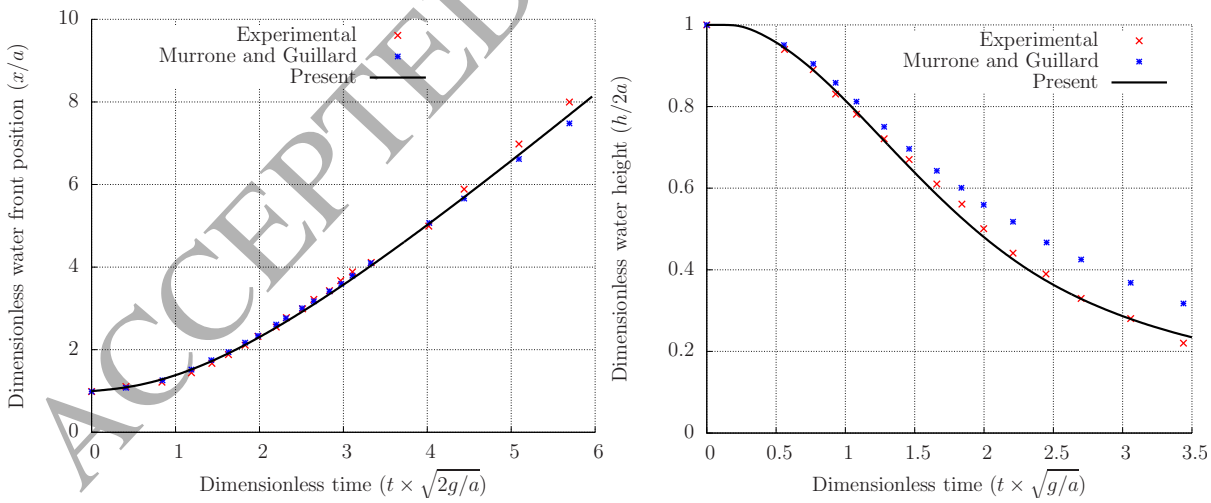


Figure 11: Comparison with Martin and Moyce's experiments [49] and Murrone and Guillard's computation [23]. Left: water front position; Right: water column height.

gas bubble of unit radius is located at the origin $(0, 0)$ in water. The density of the air bubble is $\rho_g = 1270$ kg/m^3 , the pressure is $p_g = 8290$ bar and the ratio of specific heats is $\gamma_g = 2$. For the water, the density is $\rho_l = 1000$ kg/m^3, the pressure is $p_l = 1$ bar, the polytropic constant is $\gamma_l = 7.15$ and the pressure constant is $p_c = 3 \times 10^8$. For this case, the present two-fluid method needs a very small volume of air to be uniformly distributed in the water and so we set $\alpha_1 = 0.5\%$ with the density $\rho_1 = 1$.

The computational domain is a rectangular region of $[-6, 6] \times [-6, 3]$. The horizontal rigid planar wall is located at $y = 3$. The domain is uniformly covered by 360×270 mesh cells which is the same to the grid used in Xie's work [34]. The problem is also solved on a fine mesh with 720×540 cells to convince the mesh convergence of solutions. A numerical pressure gauge is placed at the centre of the planar wall (point P) to monitor the impact pressure. Around this gauge point, we also integrate the pressure in a square region of 4×4 to obtain the impact force or loading. The length of this region is along the planar wall, and the width is perpendicular to the $x - y$ plane. The boundary conditions for this problem correspond a rigid upper wall and all other boundaries are transmissive.

Figure 15 shows several snapshots of the pressure contours in the domain computed on the coarse mesh. At $t = 1.5$ ms, the explosion generated main shock has been reflected from the planar wall. The reflected shock interacts with the expanding gas bubble undergoing a refraction process that results in a strong rarefaction wave propagating back towards the planar wall at $t = 2$ ms and a weaker reflected shock traversing the gas bubble. After the rarefaction wave impacts the wall and is reflected at $t = 3$ ms, a low pressure region forms near the planar wall causing a cavitation region to appear in this low pressure region at $t = 4$ ms. Simultaneously, the main shock continues to propagate through the computational domain. At $t = 5.5$ ms, the cavitation pocket collapses from the centre of the cavitation zone, resulting in a water-jet forming directed towards the planar wall and a relatively strong secondary compression wave propagating towards the gas bubble. The secondary compression wave impacts the gas bubble and leads to a second rarefaction wave propagating back towards the planar wall at $t = 7$ ms in a cyclic process that subsequently weakens with time.

Figure 13 shows the pressure (at point P) and force (covering the blue region around point P) acting upon the planar rigid wall. Numerical solutions computed on the coarse and fine meshes are almost identical and this confirms the convergence of the results. It can clearly be seen from this figure that the structural loading consists of shock loading and subsequent cavitation collapse reloading. The peak pressure of the shock is much higher than the peak pressure of cavitation collapse, whilst the duration of cavitation collapse reloading is longer than that of shock.

The present work gives about 7% higher prediction of the shock wave peak pressure (7000 bar) compared to Xie's solution (6500 bar) [34]. The start time of the cavitation collapse predicted by the present work is $t = 5.05$ ms, while Xie's one-fluid solver prediction is $t = 5.25$ ms [34]. The amplitude of the pressure at cavitation collapse computed by

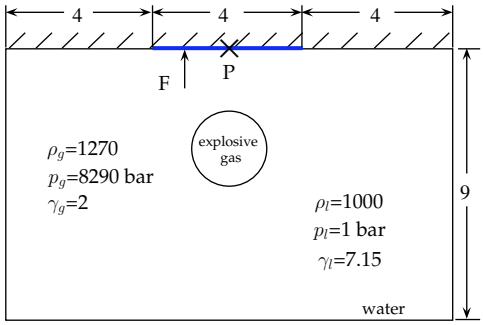


Figure 12: Setup of the 2D cylindrical underwater explosion near a planar rigid wall problem. The top boundary is a solid wall, all the others are open boundaries. The radius of the gas bubble is 1 m. The domain is discretised by a uniform mesh with 360×270 cells. At the initial stage, a small amount of gas $\alpha_1 = 0.5\%$ is mixed in the water.

the present method is very close to Xie's solution. Considering the impact force, the present work and Xie's solution give almost the same amplitude of the peak force at about 9.8×10^{10} N. The rise time of the peak force calculated by Xie is slightly higher than our prediction. This can be attributed to small differences in the numerical dissipation present in the respective flow solvers. In general, the presented results agree reasonably well with the work of Xie [34].

Additionally, time series of the gas-phase volume fraction and sound speed at point P are illustrated in Figure 14. It is clearly shown that the gas volume fraction is dramatically reduced at $t = 1$ ms when the primary shock reaching point P and strongly compressing the fluid. The gas volume fraction then has a rapid rise starting at $t = 3.7$ ms corresponding to the inception of cavitation. Alternatively, it has another a quick drop at $t = 5.05$ ms due to the collapse of cavitation. The speed of sound at point P is initially 200 m/s and rises to 2400 m/s after the fluid being compressed by the primary shock. It then gradually reduces to 1500 m/s with the gas volume fraction slowly growing from $t = 1$ ms to $t = 3.7$ ms. It then has a sudden drop corresponding to cavitation inception starting at $t = 3.7$ ms and a second quick rise to 1500 m/s at $t = 5.05$ ms when the cavitation has a collapse. Results computed on the two different meshes are very close to each other apart from the higher gas phase expansions predicted on the fine mesh at around $t = 5$ and 8.5 ms of which the second higher expansion produces a lower speed of sound.

4.5. 3D spherical underwater explosion in a rigid cylindrical container

Figure 16 illustrates the setup of this problem. The diameter of the rigid cylinder is 0.0889 m and its height is 0.2286 m. A spherical explosive gas bubble is placed at the centre of the cylindrical container. The diameter of the bubble is 0.03 m, the density of the gas is 1770 kg/m^3, the pressure is 20000 bar and its polytropic constant is $\gamma = 2$. Fresh water at ambient pressure 1 bar fills the remaining part of the container.

Although this problem can usually be solved by 2D codes in the axisymmetric coordinate system [33, 50], here we directly use our program to compute it on a 3D Cartesian grid without transforming the governing equations or

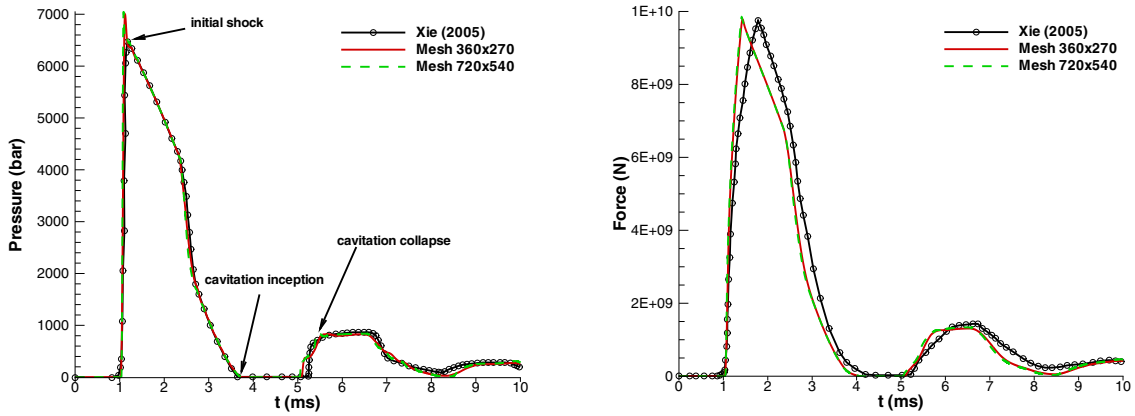


Figure 13: Comparison of the impact loading on the solid wall. Left: pressure time series at point P; Right: force acting on a 4m ×4m area of the solid wall. Red and green lines are the present results computed on the coarse (360×270) and fine mesh (720×540) respectively, black lines are Xie's computation on a mesh with 360×270 cells [34].

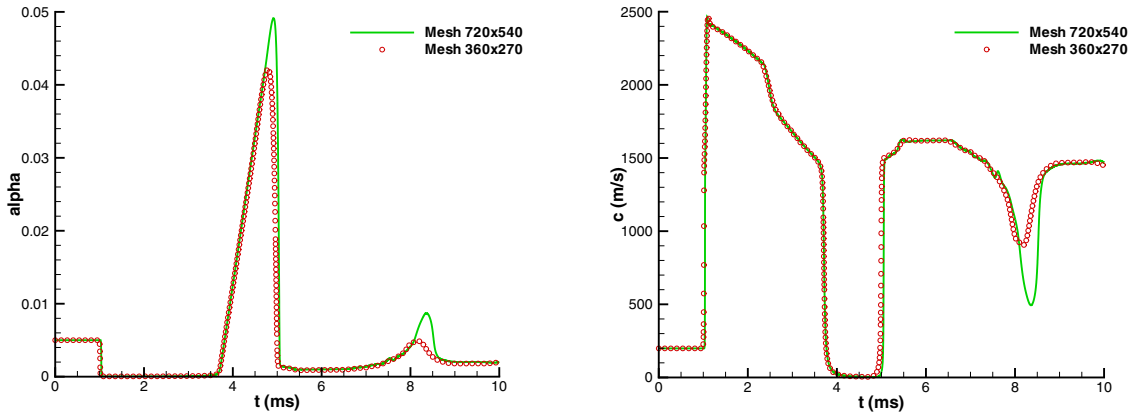


Figure 14: Time series of the gas-phase volume fraction (left) and speed of sound (right) at point P.

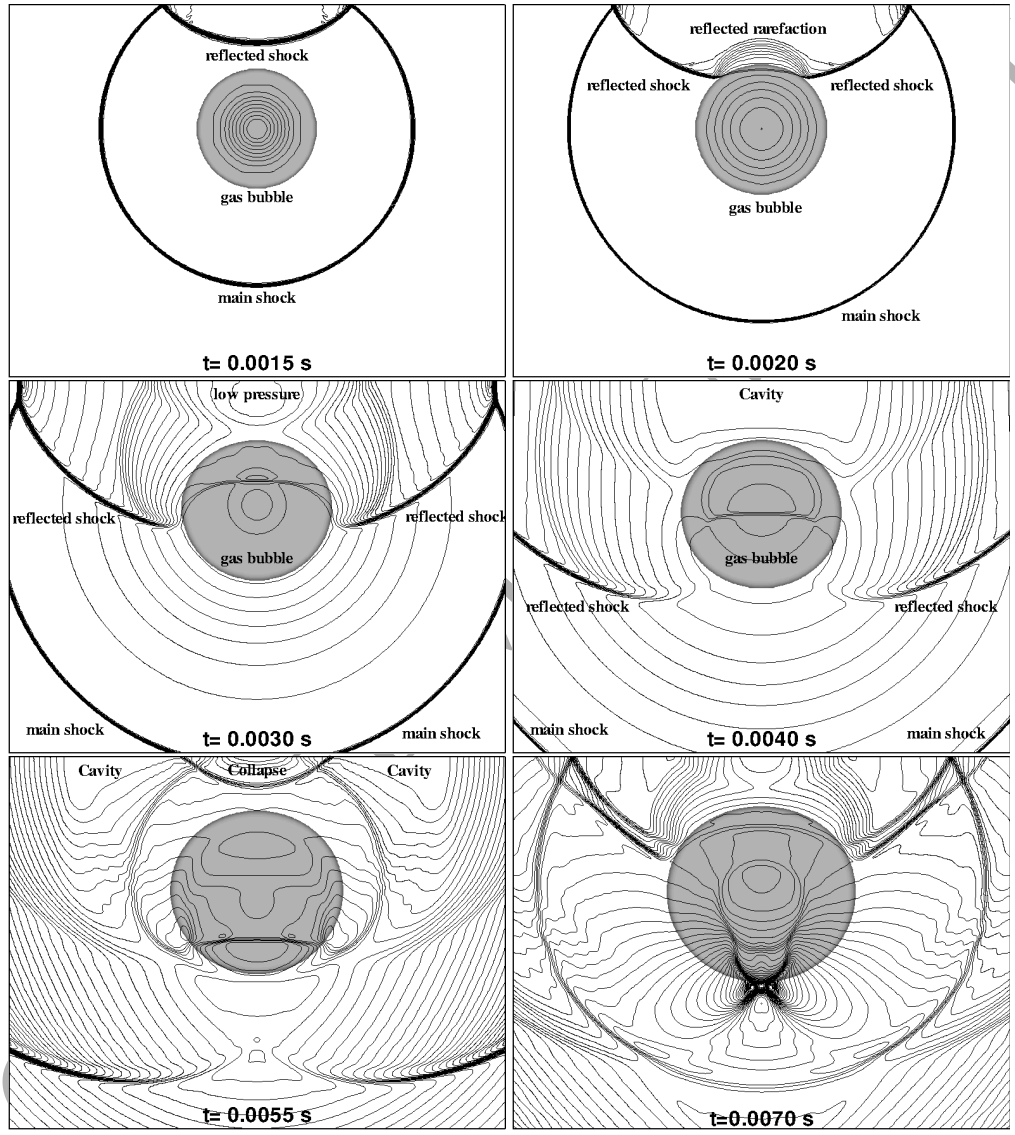


Figure 15: Line contours of pressure and the exploding gas bubble (grey colour) computed on the mesh with 360×270 cells.

modifying the code. A Cartesian mesh with $100 \times 300 \times 100$ elements is used to cover the domain. Any elements outside the container walls are blanked out. This simple strategy is only utilised here to demonstrate the capability of our code to handle 3D problems; a cut-cell method or body-fitted mesh will be implemented in future work to improve the boundary condition treatment. Similar to the underwater 2D cylindrical explosion case, dynamic interfaces might be expected to appear during the numerical computation due to cavitation formation. For this case, a small amount of air corresponding to a volume fraction $\alpha_1 = 1.5\%$ is uniformly introduced into the fresh water in the present work. The boundary conditions in this case correspond to all walls rigid and reflecting.

Figure 17 shows line contours of pressure and the expanding gas bubble in the central section of the cylinder ($x - y$ plane) at t=30, 60, 90 and 120 μs. At t=30μs, the main shock reflected at the cylinder side wall is partially reflected as a rarefaction wave by the gas bubble. At t=60μs, very low pressure region appears and this leads to cavitation formation near the side wall. As the surrounding fluid pressure is much higher than the cavitation region, the pressure in the cavity starts to rise and the cavity contracts and results in a violent collapse. Consequently, the side wall experiences a second hydrodynamic impact loading as shown in Figure 18. The first impact pressure peak at the side wall predicted by the present work is about 7000 bar. This is close to Zhu et al.'s computation [50] but a bit higher than Liu et al.'s results [33]. Our approach estimates the cavity starts to collapse at t=95μs, which is slightly earlier compared to other independent numerical simulations. However, slight differences would be expected owing to differences in the numerical dissipation implicit in the respective flow solvers. Considering the amplitude of second pressure peak, our result is close to Liu et al.'s isentropic solution of 4400 bar. Zhu et al. give a much higher prediction near 6600 bar. For this case, the cavitation collapse reloading is also potentially dangerous to the structure since its amplitude is comparable to the primary shock loading. This case was inspired by the experiments of Wardlaw et al. [3] for an explosion in a single wall containment vessel. Owing to insufficient information being made available in the published work about the explosion yield of the point charge used in the experiments, the numerical codes have each attempted to reconstruct this case approximately. The laboratory results, however, show a similar trend of primary shock loading and secondary cavitation reloading.

4.6. Water entry of a 3D rigid flat plate

Studies of slamming problems are very important in offshore and ocean engineering. In laboratories, the test object is firstly placed above a calm water surface. Then, it is dropped freely or with a prescribed velocity to impact the water surface. Impact pressure and force are recorded throughout the test. High speed cameras may also be used to capture the transient evolution of the whole process.

For this type of problem, the simplest test object may appear to be a rigid flat plate. However, despite the simplicity of the geometry, the physics describing the fluid flow during the impact process is rather complex. When the plate

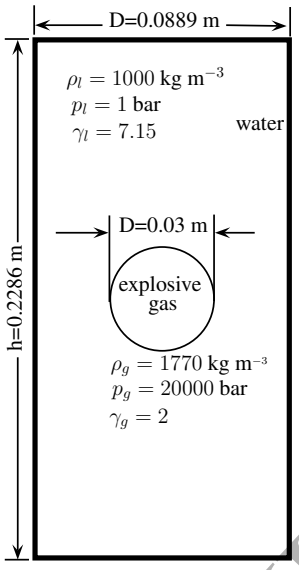


Figure 16: Setup of the spherical underwater explosion in a cylindrical container.

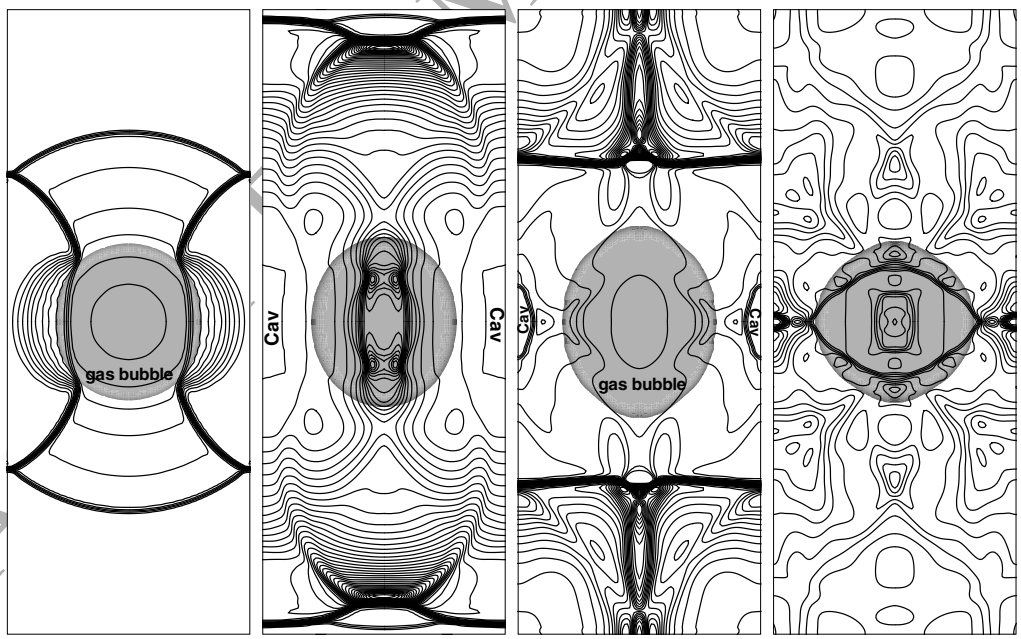


Figure 17: The line contours of pressure and gas bubble (grey colour). From left to right: t=30, 60, 90 and 120 μs. "cav" indicates the cavitation area.

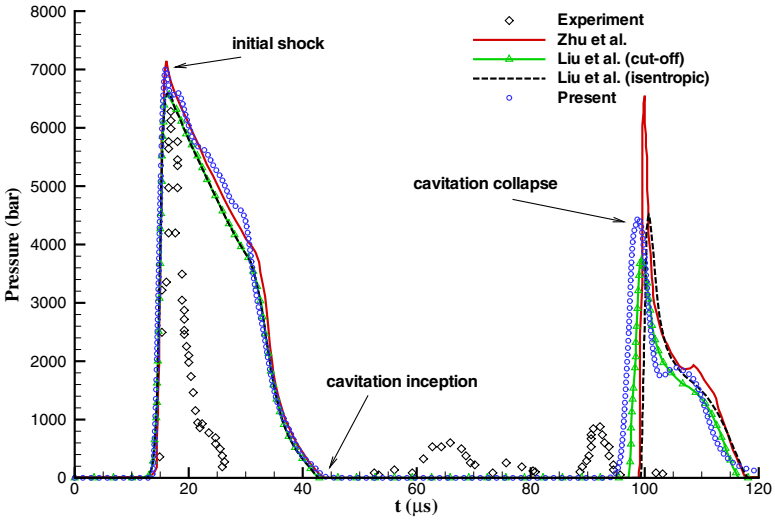


Figure 18: Comparison of the impact pressures at the centre of the side wall. The diamond symbols are the experiments of Wardlaw et al. for a single wall cylinder [3]. The red line is Zhu et al.'s WENO5-RKDG3 computation [50]. The black dashed line and blue line with triangle symbols are Liu et al.'s isentropic solution and cut-off method results respectively [33]. The blue dots are the present results.

approaches the water surface, the air layer between the plate and water will be compressed with a consequent rise of the pressure. The compressed air will then accelerate with a horizontal velocity outwards and deform the initial flat water surface that becomes slightly curved. When the plate impacts the water surface, a very thin layer of highly compressed air is driven into the water body. At the same time, the impact pressure rises rapidly to its peak value. After the pressure peak, the air layer will expand and the pressure starts to drop. The air layer may expand to such an extent that the pressure quickly descends below atmospheric pressure and even to a very small value around the vapour pressure of water. At this point, the surrounding liquid water is likely to be in tension. After a short time of decline, the fluid pressure will again increase. The plate will then experience a second hydrodynamic impact loading due to the rapid recovery of pressure in the trapped air layer. The fluid compressibility in the air/water mixture is thus very important and cannot be ignored by numerical simulations. For engineers, the characteristics of the impact pressure and force are of particular interest, because they are key parameters for the safety and survivability of structures slamming into water. Therefore, we focus on the suitability of the present flow model for the computation of impact loadings on structures in the current work.

Here, we consider a rigid square plate slamming into pure still water in the vertical direction. The width of the plate is 0.25 m; this dimension is the same as the plate used in the experimental work of Mai et al. [51]. The initial distance of the plate to the water surface is set to 0.1 m in the present work. Figure 19 shows a sketch of the setup for this problem. The present method is constructed under the Eulerian frame with a fixed finite volume method. To solve this slamming problem, we fix the flat plate in the mesh and cause the water to move upward with a prescribed

Table 5: Peak values for gauge pressure, force and slamming coefficient.

Case	P1 (bar)	P2 (bar)	F (kN)	C_s
Plate 32 kg	22.1	8.5	68.4	36.2
Plate 52 kg	23.6	10.7	69.6	36.8
Present	21.6	10.4	68.4	36.2

velocity. This strategy is appropriate for the short-period impact process [52].

In order quantitatively to analyse the accuracy of the numerical computation, here we compute a case corresponding to the experimental conditions with an impact velocity of 5.5 m/s. This impact problem is inertia dominant and lasts for a very short period. Therefore the flow can be appropriately assumed to be symmetric about the central section planes ($x-y$ and $y-z$ planes) of the plate. Under these considerations, we only compute a quarter of the domain and a Cartesian mesh with $40\times1600\times40$ elements is adopted to cover a [0,0.4]×[0,0.4]×[0,0.4] region enclosing a quarter of the flat plate. In Figure 20, we compare the numerical computation to the experimental measurement. The upper row shows the gauge pressures at P1 and P2. The lower row shows the overall impact force and slamming coefficient. The black curve represents the numerical computation, the green line is the measurement for a plate of mass 32 Kg and the red line is the measurement for a 52 Kg plate. Here we would like to point out that the experimental measurements depicted in figures 20 are not filtered but are the original signals obtained from the pressure transducers. The peak values for pressure, force and slamming coefficient are listed in Table 5. In general, the numerical computation and experimental measurements are in a reasonable agreement considering the peak values and the evolution process of the impact loading.

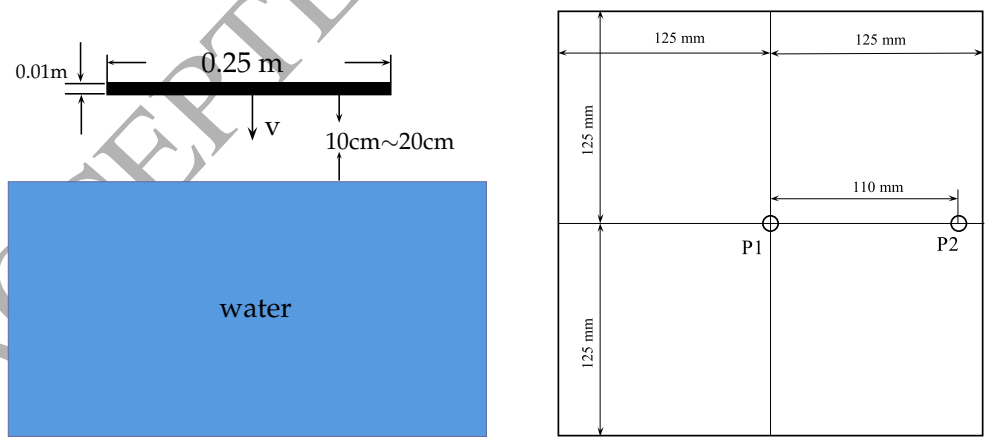


Figure 19: Setup of the water entry problem. Left: front view of the plate. Right: top view of the plate. Pressure sensor P1 is at the centre of the plate. Pressure sensor P2 is on the central section line with a distance of 0.11 m away from the centre.

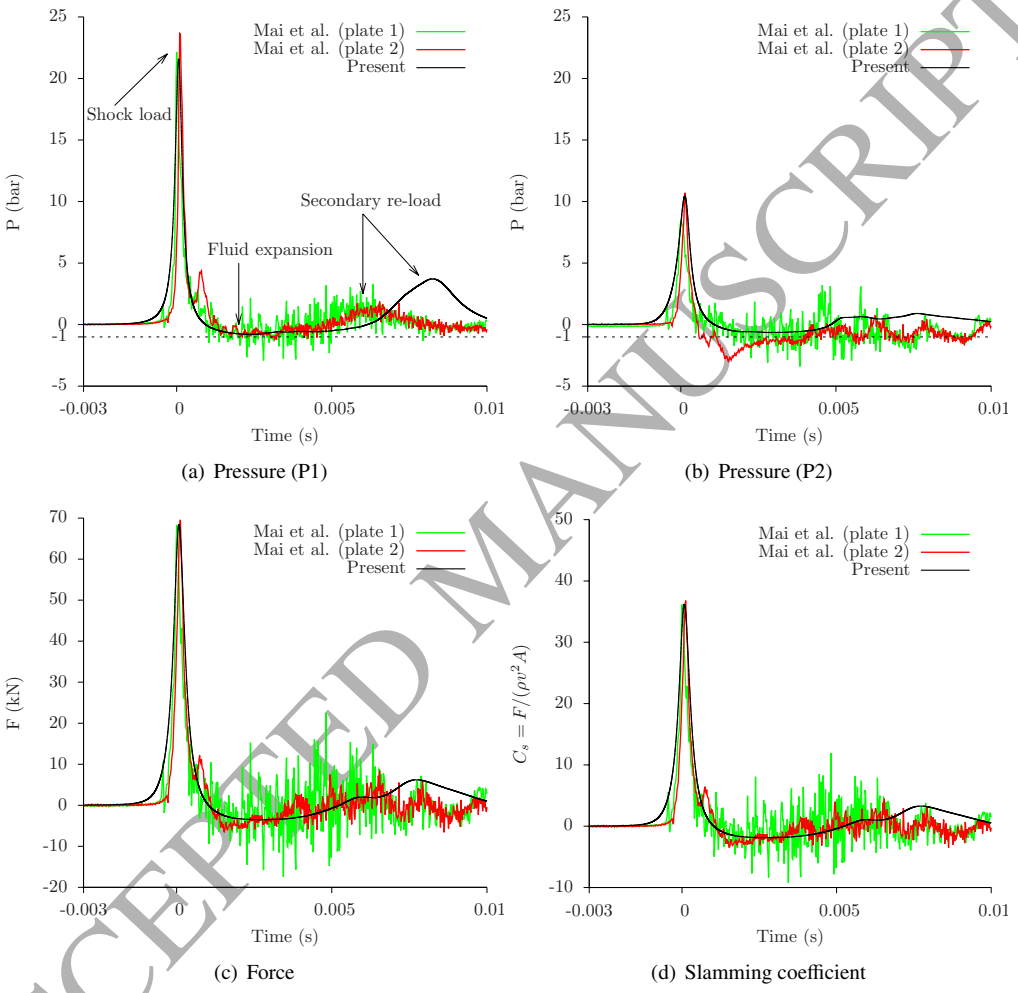


Figure 20: Impact loadings on the plate for v=5.5 m/s. The horizontal dashed line represents a perfect vacuum. The phase of all the results has been adjusted to correlate the first pressure peak at time zero. The side length of the square plate is 0.25 m. In Mai et al.'s experiments, the first plate's mass is 32 kg and the second plate's mass is 52 kg [51]. The laboratory measurements presented here are not filtered.

4.7. Performance analysis

In order quantitatively to evaluate the efficiency of the flow solver on multi-core CPUs and many-core GPUs, we here select the 3D underwater explosion problem as a benchmark case and fix the TVD Runge-Kutta time step iteration number to 50000. This number is fixed here only to facilitate direct comparisons of runs. In practice, it is of course determined by the required physical time duration of a particular test case taking into account the time step according to the usual Courant condition for numerically stable computations. On each platform, the flow solver is tested under six conditions as listed in Table 6. The wall clock time is recorded and the consumed electrical energy is calculated as $Energy = Power \times Time$. It should be noted that if the flow solver is only running on a single CPU core, then the energy consumption is divided by the number of cores residing in the CPU.

Figures 21, 22, 23 and 24 show the running time cost, electrical energy consumption, GPU performance speedup and energy efficiency, respectively. The solid line indicates that the flow solver throughout uses double-precision data (DP); the dashed line indicates that the solver adopts mixed-precision data (MP). A first glance at these four figures impresses upon us that MP arithmetic can indeed help to reduce the running time cost especially for the GPUs. It can also reduce the electrical energy consumption.

Focusing on Figure 23 for performance speedup, we find that on platform 1 the GPU is around 7~8 times faster than a single CPU core and about 2 times faster than four CPU cores if DP is used. When MP is adopted, the GPU is around 16 times faster than a single CPU core and about 7~8 times faster than four CPU cores. When DP is used on platform 2, the GPU is 40 times faster than a single CPU core and 7 times faster than 16 CPU cores. When MP is used on platform 2, the GPU is 60 times faster than a single CPU core and 15 times faster than 16 CPU cores. In this figure, it can also be noticed that the scalability of OpenMP on multi-core CPUs is not satisfactory since two eight-core CPUs only give a 4~5× speedup compared to a single core on platform 2. This phenomenon is very similar to Ortega et al.'s observation that attainable speedups on multi-core CPUs will drop once the number of processor cores is over 4 due to the high cache miss rate and limited memory bandwidth of CPU (see figures 13 and 14 of [53]).

Looking at Figure 24, we find that on each platform the GPU consumes much less electrical energy compared to the CPU. When DP is used on platform 1, the ratio of energy consumption of the GPU compared to a single CPU decreases from 0.6 to 0.32 with the increase of problem size. Considering the four CPU cores, this parameter drops from 0.3 to 0.2. If DP is used, the ratio of energy consumption decreases from 0.25 to 0.17 for the GPU compared to a single CPU core. The ratio drops from 0.14 to 0.08 for the GPU compared to the four CPU cores. Similar trends can be found for the energy consumption on platform 2.

In order to assess the positive effect of MP on the program execution efficiency, we compare the running time cost of DP to MP and plot the results in Figure 25. It is clearly shown that MP can boost the performance by 10%~30% for

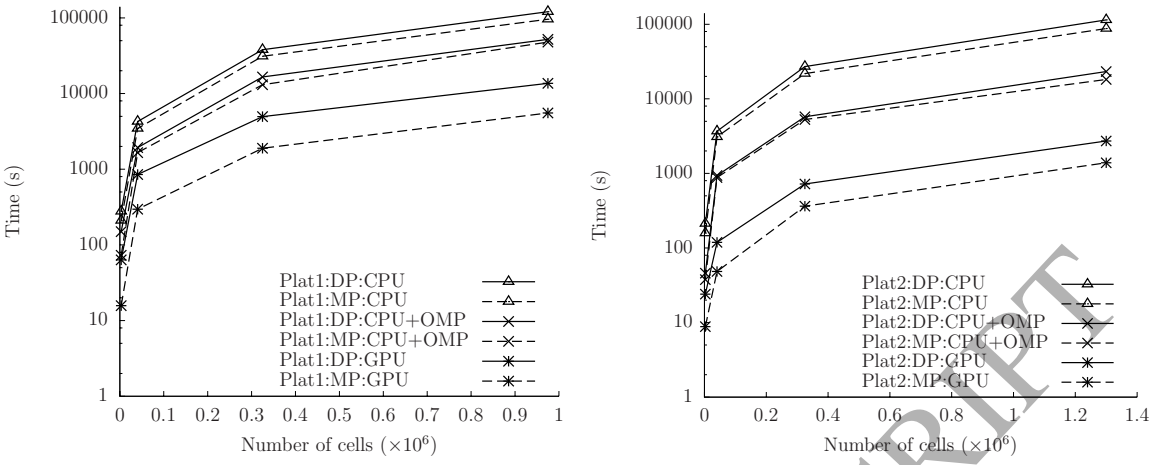


Figure 21: Running time cost of CPUs and GPUs. Left: platform 1; Right: platform 2. The iteration number is fixed to 50000 for comparison purposes. Please refer to Table 6 for interpretation of the graph notation.

CPUs while it is even more interesting to find that MP can boost the performance by 100% (or even more) for GPUs. Figure 26 exhibits the memory usages of DP and MP on both platforms. It is clearly shown that MP can almost reduce by half the memory usage.

Table 6: Specification of the six running conditions of the flow solver.

Condition	Denotation	Description
1	DP:CPU	Double-precision data, a single CPU core
2	MP:CPU	Mixed-precision data, a single CPU core
3	DP:CPU+OMP	Double-precision data, all available CPU cores
4	MP:CPU+OMP	Mixed-precision data, all available CPU cores
5	DP:GPU	Double-precision data, GPU
6	MP:GPU	Mixed-precision data, GPU

5. Conclusions

A one-dimensional reduced-equation model has been successfully extended to solve three-dimensional compressible multiphase flow impact problems on graphics processor units. The integral form of the governing equations are discretised by a high-order finite volume scheme based on MUSCL reconstruction and an HLLC approximate Riemann solver for compressible liquid-gas mixtures. The numerical method is assessed through a series of test cases including water-air shock tubes, 1D liquid expansion tube, a dam break, 2D cylindrical underwater explosion near a planar rigid wall, 3D spherical explosion in a rigid cylindrical container and water entry of a rigid square flat plate. The obtained results agree well with experiments, exact solutions and other independent numerical computations. Systematic analysis reveals that GPUs can not only improve compute performance by orders-of-magnitude but also

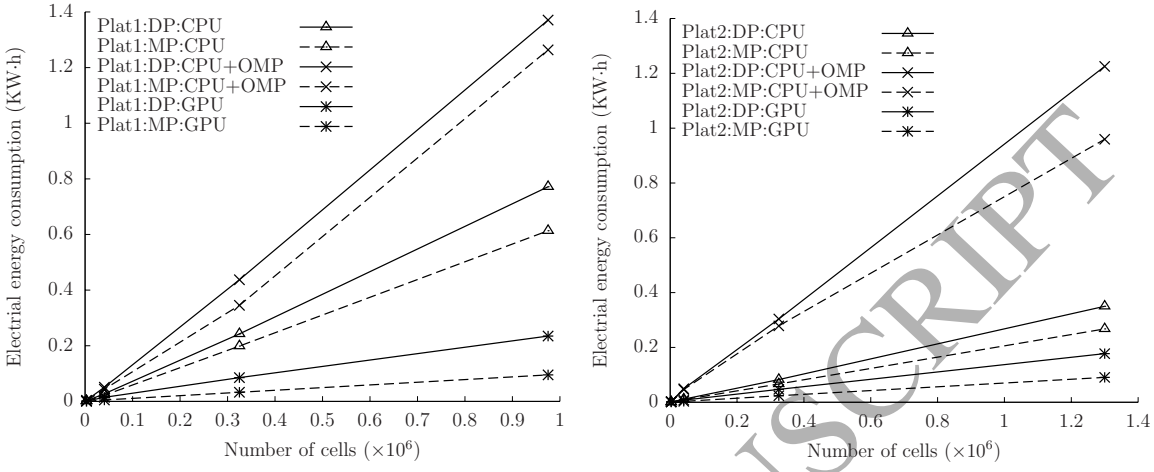


Figure 22: Energy consumption of CPUs and GPUs. Left: platform 1; Right: platform 2. The time step iteration number is fixed to 50000 for comparison purpose. Please refer to Table 6 for interpretation of the graph notation.

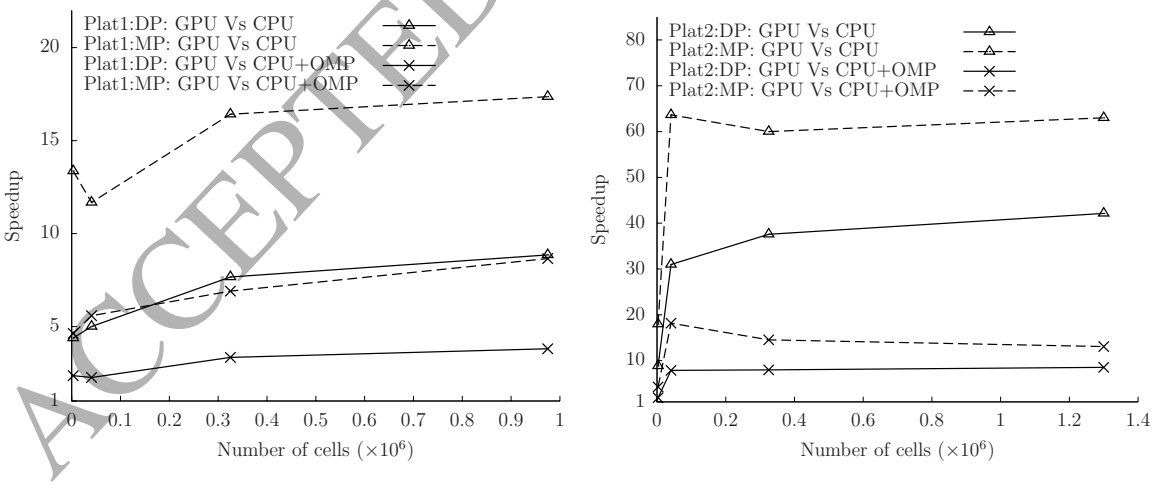


Figure 23: Performance speedup of GPUs. Left: platform 1; Right: platform 2. The time step iteration number is fixed to 50000 for comparison purposes. Please refer to Table 6 for interpretation of the graph notation.

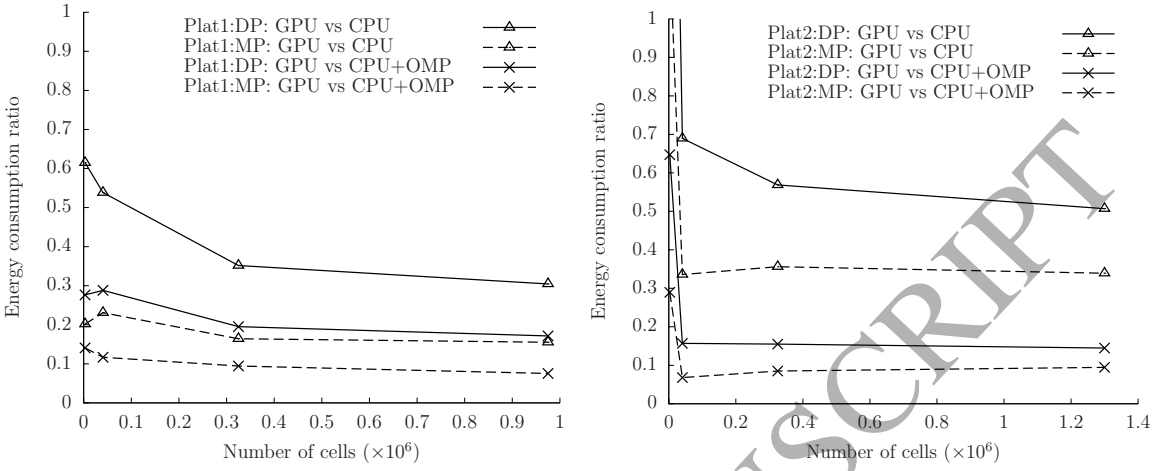


Figure 24: Energy efficiency of GPUs. The energy consumption ratio is calculated by dividing the GPU energy consumption with the CPU energy consumption. Note that a single CPU core's energy consumption should be computed as *Power×Time/NumberOfCores*. Left: platform 1; Right: platform 2. The time step iteration number is fixed to 50000 for comparison purpose. Please refer to Table 6 for interpretation of the graph notation.

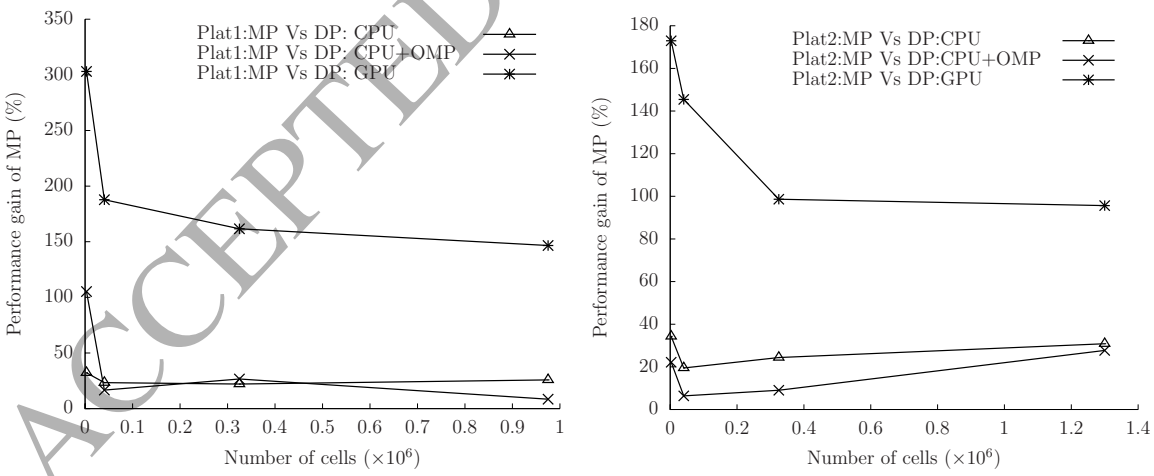


Figure 25: The effect of mixed precision treatment on performance. The performance gain of MP is calculated as $gain = (T_{DP} - T_{MP})/T_{MP} \times 100\%$, where T means the wall clock time. Left: platform 1; Right: platform 2. On both platforms the mixed precision (MP) program runs faster than the double precision (DP) program. Especially for GPUs, MP provides 100% or even more performance gain compared to DP.

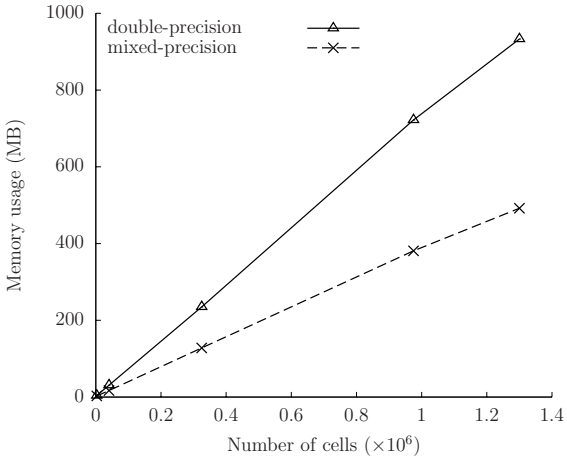


Figure 26: A comparison of the memory usage for double-precision and mixed-precision data. The mixed precision approach almost saves 50% memory usage compared to pure double precision treatment.

reduce the consumption of electrical energy dramatically by 50% or even more. A mixed precision treatment in the parallel implementation can save almost half of the memory usage compared to a pure double precision strategy and helps to improve program execution efficiency. Especially for GPUs, this has been shown to provide a 100% or even more performance speedup. In future we will extend the current multiphase compressible flow method to investigate further hydrodynamic impact loads on rigid and flexible fixed and floating structures taking advantage of the compute performance gains obtained through GPU acceleration.

Acknowledgements

The authors gratefully acknowledge financial support from the Engineering and Physical Sciences Research Council (EPSRC), U.K. under grant numbers EP/J010197/1, EP/J012793/1 and EP/K037889/1. The authors would also like to thank Dr. Tri Mai, Prof. Deborah Greaves and Dr. Alison Raby of Plymouth University for providing the experimental data for the rigid flat plate slamming problem.

References

[1] H. Mitsuyasu, Shock pressure of breaking wave, Coastal Engineering Proceedings 1 (10) (1966) 268–283.

[2] M. S. Kirkgoz, Shock pressure of breaking waves on vertical walls, Journal of the Waterway Port Coastal and Ocean Division 108 (1) (1982) 81–95.

[3] A. B. Wardlaw Jr., J. A. Luton, Fluid-structure interaction mechanisms for close-in explosions, Shock and Vibration 7 (5) (2000) 265–275. doi:10.1155/2000/141934.

[4] S. U. Galiev, R. G. Flay, Interaction of breaking waves with plates: The effect of hull cavitation, Ocean Engineering 88 (2014) 2733. doi:10.1016/j.oceaneng.2014.04.024.

[5] G.-d. Xu, W.-y. Duan, Review of prediction techniques on hydrodynamic impact of ships, Journal of Marine Science and Application 8 (3) (2009) 204–210. doi:10.1007/s11804-009-8039-7.

[6] C. Lugni, M. Brocchini, O. M. Faltinsen, Evolution of the air cavity during a depressurized wave impact. II. The dynamic field, Phys. Fluids 22 (5) (2010) 056102. doi:10.1063/1.3409491.

[7] G. Bullock, C. Obhrai, D. Peregrine, H. Bredmose, Violent breaking wave impacts. part 1: Results from large-scale regular wave tests on vertical and sloping walls, Coastal Eng. 54 (8) (2007) 602 – 617. doi:10.1016/j.coastaleng.2006.12.002.

[8] H. Gu, L. Qian, D. Causon, C. Mingham, P. Lin, Numerical simulation of water impact of solid bodies with vertical and oblique entries, Ocean Engineering 75 (2014) 128–137. doi:10.1016/j.oceaneng.2013.11.021.

[9] Z. Hu, D. Causon, C. Mingham, L. Qian, A Cartesian cut cell free surface capturing method for 3d water impact problems, International Journal for Numerical Methods in Fluids 71 (10) (2013) 1238–1259. doi:10.1002/fld.3708.

[10] L. Qian, D. M. Causon, C. G. Mingham, D. M. Ingram, A free-surface capturing method for two fluid flows with moving bodies, Proc. R. Soc. Lond. A 462 (2006) 21–42. doi:10.1098/rspa.2005.1528.

[11] Z. H. Ma, D. M. Causon, L. Qian, C. G. Mingham, H. B. Gu, P. M. Ferrer, A compressible multiphase flow model for violent aerated wave impact problems, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science 470 (2172) (2014) 0542. doi:10.1098/rspa.2014.0542.

[12] H. Bredmose, D. Peregrine, G. Bullock, Violent breaking wave impacts. part 2: modelling the effect of air, J. Fluid Mech. 641 (1) (2009) 389–430. doi:10.1017/S0022112009991571.

[13] S. Wilson, A mathematical model for the initial stages of fluid impact in the presence of a cushioning fluid layer, Journal of Engineering Mathematics 25 (3) (1991) 265–285. doi:10.1007/BF00044334.

[14] L.-R. Plumerault, D. Astruc, P. Villedieu, P. Maron, A numerical model for aerated-water wave breaking, Int. J. Numer. Methods Fluids 69 (12) (2012) 1851–1871. doi:10.1002/fld.2667.

[15] D. Peregrine, Water-wave impact on walls, Annu. Rev. Fluid Mech. 35 (1) (2003) 23–43. doi:10.1146/annurev.fluid.35.101101.161153.

[16] M. Baer, J. Nunziato, A two-phase mixture theory for the deflagration-to-detonation transition (DDT) in reactive granular materials, Journal of Multiphase Flow 12 (1986) 86–889. doi:10.1016/0301-9322(86)90033-9.

[17] N. Andrianov, G. Warnecke, The Riemann problem for the Baer–Nunziato model of two-phase flows, Journal of Computational Physics 195 (2004) 434–464. doi:10.1016/j.jcp.2003.10.006.

[18] D. Schwendeman, C. Wahle, A. Kapila, The Riemann problem and a high-resolution Godunov method for a model of compressible two-phase flow, Journal of Computational Physics 212 (2) (2006) 490 – 526. doi:10.1016/j.jcp.2005.07.012.

[19] V. Deledicque, M. Papalexandris, An exact Riemann solver for compressible two-phase flow models containing non-conservative products, Journal of Computational Physics 222 (2007) 217–245. doi:10.1016/j.jcp.2006.07.025.

[20] S. Tokareva, E. Toro, HLLC-type Riemann solver for the Baer-Nunziato equations of compressible two-phase flow, J. Comput. Phys. 229 (10) (2010) 3573 – 3604. doi:10.1016/j.jcp.2010.01.016.

[21] S. Liang, W. Liu, L. Yuan, Solving seven-equation model for compressible two-phase flow using multiple GPUs, Computers & Fluids 99 (2014) 156–171. doi:10.1016/j.compfluid.2014.04.021.

[22] A. Kapila, R. Menikoff, J. Bdzil, S. Son, D. Stewart, Two-phase modeling of deflagration-to-detonation transition in granular materials: Reduced equations, Phys. Fluids 13 (2001) 3002–3024. doi:http://link.aip.org/link/doi/10.1063/1.1398042.

[23] A. Murrone, H. Guillard, A five equation reduced model for compressible two phase flow problems, J. Comput. Phys. 202 (2) (2005) 664 – 698. doi:10.1016/j.jcp.2004.07.019.

[24] R. Saurel, O. Le Mtayer, J. Massoni, S. Gavrilyuk, Shock jump relations for multiphase mixtures with stiff mechanical relaxation, Shock Waves 16 (2007) 209–232. doi:10.1007/s00193-006-0065-7.

[25] R. Saurel, F. Petitpas, R. A. Berry, Simple and efficient relaxation methods for interfaces separating compressible fluids, cavitating flows and shocks in multiphase mixtures, Journal of Computational Physics 228 (5) (2009) 1678–1712. doi:10.1016/j.jcp.2008.11.002.

[26] R. Abgrall, V. Perrier, Asymptotic expansion of a multiscale numerical scheme for compressible multiphase flows, SIAM Journal of Multiscale and Modeling and Simulation 5 (2006) 84–115. doi:10.1137/050623851.

[27] F. Petitpas, E. Franquet, R. Saurel, O. L. Metayer, A relaxation-projection method for compressible flows. Part II: Artificial heat exchanges for multiphase shocks, J. Comput. Phys. 225 (2) (2007) 2214 – 2248. doi:10.1016/j.jcp.2007.03.014.

[28] J. J. Kreeft, B. Koren, A new formulation of Kapila's five-equation model for compressible two-fluid flow, and its numerical treatment, Journal of Computational Physics 229 (18) (2010) 6220–6242. doi:10.1016/j.jcp.2010.04.025.

[29] F. Petitpas, R. Saurel, E. Franquet, A. Chinnayya, Modelling detonation waves in condensed energetic materials: multiphase CJ conditions and multidimensional computations, Shock Waves 19 (2009) 377–401. doi:10.1007/s00193-009-0217-7.

[30] S. Schoch, N. Nikiforakis, B. J. Lee, R. Saurel, Multi-phase simulation of ammonium nitrate emulsion detonations, Combustion and Flame 160 (9) (2013) 18831899. doi:10.1016/j.combustflame.2013.03.033.

[31] N. Favrie, S. Gavrilyuk, R. Saurel, Solid–fluid diffuse interface model in cases of extreme deformations, Journal of Computational Physics 228 (16) (2009) 60376077. doi:10.1016/j.jcp.2009.05.015.

[32] R. Saurel, N. Favrie, F. Petitpas, M.-h. Lallemand, S. L. Gavrilyuk, Modelling dynamic and irreversible powder compaction, J. Fluid Mech. 664 (2010) 348396. doi:10.1017/s0022112010003794.

[33] T. Liu, B. Khoo, W. Xie, Isentropic one-fluid modelling of unsteady cavitating flow, Journal of Computational Physics 201 (1) (2004) 80–108. doi:10.1016/j.jcp.2004.05.010.

[34] W. Xie, A numerical simulation of underwater shock-cavitation-structure interaction, Ph.D. thesis, National University Of Singapore (2005).

[35] R. Saurel, F. Petitpas, R. Abgrall, Modelling phase transition in metastable liquids: application to cavitating and flashing flows, J. Fluid Mech. 607 (2008) 313–350. doi:10.1017/s0022112008002061.

[36] F. Petitpas, J. Massoni, R. Saurel, E. Lapebie, L. Munier, Diffuse interface model for high speed cavitating underwater systems, International Journal of Multiphase Flow 35 (8) (2009) 747759. doi:10.1016/j.ijmultiphaseflow.2009.03.011.

[37] V. G. Asouti, X. S. Trompoukis, I. C. Kampolis, K. C. Giannakoglou, Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units, Int. J. Numer. Methods Fluids 67 (2) (2011) 232–246. doi:10.1002/fld.2352.

[38] A. Corrigan, F. F. Camelli, R. Lhner, J. Wallin, Running unstructured grid-based CFD solvers on modern graphics hardware, Int. J. Numer. Methods Fluids 66 (2) (2011) 221–229. doi:10.1002/fld.2254.

[39] E. Elsen, P. LeGresley, E. Darve, Large calculation of the flow over a hypersonic vehicle using a GPU, J. Comput. Phys. 227 (24) (2008) 10148 – 10161. doi:10.1016/j.jcp.2008.08.023.

[40] Z. H. Ma, H. Wang, S. H. Pu, GPU computing of compressible flow problems by a meshless method with space-filling curves, Journal of Computational Physics 263 (2014) 113–135. doi:10.1016/j.jcp.2014.01.023.

[41] Z. H. Ma, H. Wang, S. H. Pu, A parallel meshless dynamic cloud method on graphic processing units for unsteady compressible flows past moving boundaries, Computer Methods in Applied Mechanics and Engineering 285 (2015) 146–165. doi:10.1016/j.cma.2014.11.010.

[42] A. B. Wood, A textbook of sound, G. Bell and Sons, London, 1941.

[43] E. Johnsen, T. Colonius, Implementation of WENO schemes in compressible multicomponent flow problems, J. Comput. Phys. 219 (2) (2006) 715 – 732. doi:10.1016/j.jcp.2006.04.018.

[44] B. van Leer, Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method , J. Comput. Phys. 32 (1)

(1979) 101 – 136. doi:10.1016/0021-9991(79)90145-1.

[45] X. Hu, N. Adams, G. Iaccarino, On the HLLC Riemann solver for interface interaction in compressible multi-fluid flow, J. Comput. Phys. 228 (17) (2009) 6572 – 6589. doi:10.1016/j.jcp.2009.06.002.

[46] NVIDIA, CUDA C programming guide (2012).
URL http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[47] C. Mccabe, Smoothed particle hydrodynamics on graphics processing units, Ph.D. thesis, Manchester Metropolitan University (2012).

[48] A. Chinnayya, E. Daniel, R. Saurel, Modelling detonation waves in heterogeneous energetic materials, J. Comput. Phys. 196 (2) (2004) 490 – 538. doi:10.1016/j.jcp.2003.11.015.

[49] J. C. Martin, W. J. Moyce, Part IV. An experimental study of the collapse of liquid columns on a rigid horizontal plane, Proc. R. Soc. Lond. A 244 (882) (1952) 312–324. doi:10.1098/rsta.1952.0006.

[50] J. Zhu, T. Liu, J. Qiu, B. C. Khoo, RKDG methods with WENO limiters for unsteady cavitating flow, Computers & Fluids 57 (2012) 52 – 65. doi:10.1016/j.compfluid.2011.12.004.

[51] T. Mai, D. Greaves, A. Raby, Physical Experiments of the Drop Test (Unplished results), School of Marine Science and Engineering, Plymouth University, Plymouth PL4 8AA, UK.

[52] C. O. Ng, S. C. Kot, Computations of a water impact on a two-dimensional flat-bottomed body with a volume-of-fluid method, Ocean Engineering 19 (1992) 377–393.

[53] E. Ortega, E. Oate, S. Idelsohn, R. Flores, Comparative accuracy and performance assessment of the finite point method in compressible flow problems, Computers & Fluids 89 (2014) 53 – 65. doi:10.1016/j.compfluid.2013.10.024.