Computer Physics Communications 284 (2023) 108608

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A novel MPI-based parallel smoothed particle hydrodynamics framework with dynamic load balancing for free surface flow ☆

Guixun Zhu, Jason Hughes, Siming Zheng *, Deborah Greaves

School of Engineering, Computing and Mathematics, University of Plymouth, Drake Circus, Plymouth PL4 8AA, UK

ARTICLE INFO

Article history:
Received 29 August 2022
Received in revised form 4 November 2022
Accepted 21 November 2022
Available online 23 November 2022

Keywords:
Smoothed particle hydrodynamics
Parallel computing
Load balance
Free surface flow

ABSTRACT

This paper presents a new Smoothed Particle Hydrodynamics (SPH) parallel framework, which is designed for free surface flows and is scalable on a High Performance Computer (HPC). The framework is accomplished by adopting a Message Passing Interface (MPI) approach with a domain partitioning strategy. A regular background grid is used to partition the entire computational domain and each subdomain is labelled using an index ordering method. Adjacent subdomains can be determined by the index list, and avoid global communications in the particle distribution process. Within the local grid, the grid is divided into an internal grid as well as an interactive grid to identify the particles for which information is to be transferred. The implementation of the dynamic loading balance strategy considers two different ways of determining loading: computation particle numbers and running time. The dynamic load balance strategy repositions neighbouring subdomains based on the local load imbalance between cores. To demonstrate the framework's capacity and distinctive properties, a variety of free surface flow benchmarks are studied. Intensive numerical experiments at various scales are used to assess the performance in detail.

1. Introduction

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian mesh-free method that was originally used to study problems of astrophysical gas dynamics formulated approximately forty years ago by Gingold and Monaghan [12] and Lucy [21]. Since Monaghan [28] first used SPH to study free-surface flows, SPH has been used to study various free surface flow problems [32,34,19,23,22]. The interactions between particles are built, based on the kernel function which represents the derivatives of continuous fields in a discrete form. SPH is deemed to be an ideal technique to simulate interfacial flows with large deformation [6,38,37]. The interface can be automatically detected without special interface detection treatment. The particles with physical properties can move arbitrarily, and complex interfaces can be reproduced well. Thus, sizeable free surface deformation, such as splashing [39,31], wave breaking [14], and so on, can be simulated naturally. However, the SPH model is known to be an expensive model with high computational costs. Each particle should be interpolated with tens or even hundreds of neighbouring particles. The number of neighbouring particles in SPH is far greater than the number of neighbouring nodes for mesh methods (finite volume method (FVM), finite element method (FEM), and so on) [4]. Secondly, modelling incompressible fluids requires a small time step (weakly compressible SPH model) or solving the pressure Poisson equation (Incompressible SPH model). For the scale of engineering problems, SPH models often require the simulation of millions or even tens of millions of particles, which leads to significant time consumption. The use of hardware acceleration and parallel programming is an option for achieving efficient computation of large particle numbers. Hardware acceleration is the specialist use of computing hardware to achieve more efficiency. Hardware acceleration is utilised through parallel programming which can be implemented through high performance computing on multiple CPU cores or through hardware architecture such as the graphics processing unit (GPU) [7,5].

Several parallel frameworks exist for the large-scale parallel simulation of SPH methods. Ihmsen et al. [16] introduced a parallel framework for simulating fluids using the SPH methods. Efficient parallel neighbourhood inquiries were suggested and evaluated for low computing costs per simulation step. To reduce the processing time for whole simulation sequences even further, solutions for optimising the time step and its implications for parallel implementations were examined. Cherfils et al. [3] developed JOSEPHINE, a parallel SPH code, for solving unstable free-surface flows. The dynamic loading balancing strategy in JOSEPHINE considered the

☆ The review of this paper was arranged by Prof. David W. Walker.

* Corresponding author.
E-mail address: siming.zheng@plymouth.ac.uk (S. Zheng).

https://doi.org/10.1016/j.cpc.2022.108608

dynamic loading differences between adjacent subdomains. The results showed that parallelism provided for a substantial decrease in processing time, especially when dealing with a high number of particles. DualSPHysics [8] is a hardware-accelerated SPH program designed to handle issues involving free-surface flow. DualSPHysics is an open-source software project that was created and distributed under the GNU General Public License (GPLv3). Along with the source code, a comprehensive set of instructions for compiling and running the source files is also included. DualSPHysics is accelerated by up to two orders of magnitude as compared to the serial version thanks to the parallel power processing of GPUs. DualSPHysics has been proven to be both efficient and trustworthy [9]. Various key points related to the specificities of the particle methods for their massive parallelisation on distributed memory were presented and discussed by Oger et al. [30]. Both the Orthogonal Recursive Bisection (ORB) and the particle domain updating procedures were presented and tested in terms of parallel efficiency for cases involving a large number of particles. The model was then tested on massive particle discretisation involving up to 3.2 billion particles and 32,768 cores, showing some good parallel efficiencies in both strong and weak scaling measurements. A new massively parallel scheme is developed by Guo et al. [13], to simulate free-surface flows with ISPH method for simulations involving more than 100 million particles. The new scheme uses a Hilbert space-filling curve with a cell-linked list to map the entire domain so that domain decomposition and load balancing can be achieved while taking advantage of the geometric locality to reduce cache access latency. Ji et al. [17] present a new multi-resolution parallel framework, which is designed for large-scale SPH simulations of fluid dynamics. An adaptive rebalancing criterion and monitoring system is developed to integrate the Centroidal Voronoi Particle (CVP) partitioning method as a rebalancer to achieve dynamic load balancing of the system. In addition, there is much work in the literature on hardware acceleration of the SPH model [11,26,10,33,35].

This paper focuses on distributed memory parallelisation of the SPH method by Message Passing Interface (MPI) libraries. Domain decomposition consists of geometrically subdividing the computational domain into as many smaller subdomains as desired, and performing solutions in a processor corresponding to a subdomain. Using MPI data communications among processes, the SPH numerical results can be reconstructed on the whole domain. We develop a new SPH parallel model for modelling the free surface problem. A regular background grid is used to partition the entire computational domain and each subdomain is labelled using an index ordering method. Within each subdomain, the classical regular grid is used to search for neighbouring particles. Within the local grid, the grid is divided into an internal grid as well as an interactive grid to identify the particles for which information is to be transferred. To consider the dynamic loading balance, two loading measures, computation particle number and running time, are discussed. The subdomains adjust the boundary positions according to the load distribution to meet the load balance requirements. An initial load balance strategy is set up to ensure that the initial phase of loading balance is achieved. The remainder of the paper is organised as follows. In §2, we first briefly review SPH model theory. The main contribution in this paper is presented in §3 and §4. The parallel framework is presented in §3. Dynamic load balance strategy is elaborated in §4. In §5, the free surface problems were simulated to validate the parallel framework. Loading strategies based on different criterions are compared. Parallel performance is also examined. Concluding remarks are given in §6.

2. SPH model

In this study, the flow is assumed to be viscous, weakly–compressible, and adiabatic. The adopted governing equations consist of the Navier–Stokes equations in the Lagrange framework:

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla p + F_\alpha + \boldsymbol{g}, \\ \frac{\mathrm{d}\rho}{\mathrm{d}t} = -\rho\nabla\cdot\boldsymbol{u}, \\ \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} = \boldsymbol{u}, \end{cases} \tag{1}$$

where ρ, $\boldsymbol{u}$, t, $\boldsymbol{r}$ and p denote the density, velocity vector, time, position vector and pressure, respectively. F_α is the viscosity term and $\boldsymbol{g}$ represents the gravitational acceleration. The governing equation can be discretized by an SPH approximation. According to the work in [20], the discrete pressure gradient can be written as:

$$-\frac{1}{\rho_i}\nabla p_i = -\frac{1}{\rho_i}\sum_j (p_j + p_i)\cdot\nabla_i W_{ij} V_j, \tag{2}$$

where $W_{ij} = W(\boldsymbol{r}_i - \boldsymbol{r}_j, h)$ is the kernel function, h is the smoothing length defining the influence area. Subscripts i and j denote the particle index. V_j is the volume of the particle ($V_j = m_j/\rho_j$, m denotes mass).

The artificial viscosity term can be added to the momentum equation to produce bulk and shear viscosity and also to stabilise the scheme as follows [29]

$$F_\alpha = \sum_j \alpha h c_s \frac{(\boldsymbol{u}_j - \boldsymbol{u}_i)\cdot(\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2}\cdot\nabla_i W_{ij} V_j, \tag{3}$$

where c_s is numerical sound speed, and $\alpha = 0.01$ is artificial viscosity factor. The velocity divergence can be discretized as [20]

$$-\rho_i\nabla\cdot\boldsymbol{u}_i = -\sum_j (\boldsymbol{u}_j - \boldsymbol{u}_i)\cdot\nabla_i W_{ij} V_j, \tag{4}$$

where $\boldsymbol{u}_i$ is the velocity of particle i. Spurious numerical oscillations generally exist in the pressure and density fields for traditional weakly compressible SPH. The δ–SPH model [1] is employed to avoid spurious high–frequency oscillations

$$\delta h c \sum_j \Psi_{ij}\cdot\nabla_i W_{ij} V_j, \tag{5}$$

where $\delta = 0.1$ [25,31] for all the following cases and

$$\begin{cases} \Psi_{ij} = 2(\rho_j - \rho_i)\frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} - (\langle\nabla\rho\rangle_i^L + \langle\nabla\rho\rangle_j^L), \\ \langle\nabla\rho\rangle_i^L = \sum_j (\rho_j - \rho_i) L_i \nabla_i W_{ij} V_j, \quad \text{where} \\ \quad L_i = \left[\sum_j (\boldsymbol{r}_j - \boldsymbol{r}_i)\otimes\nabla_i W_{ij} V_j\right]^{-1} \end{cases} \tag{6}$$

where $\otimes$ denotes tensor product. With the artificial diffusive δ term introduced, the continuity equation can be re-written as

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\sum_j (\boldsymbol{u}_j - \boldsymbol{u}_i)\cdot\nabla_i W_{ij} V_j + \delta h c \sum_j \Psi_{ij}\cdot\nabla_i W_{ij} V_j. \tag{7}$$

Meanwhile, the fluid pressure is related to the density explicitly according to the concept of artificial compressibility. Then, the pressure is obtained through the equation of state as

$$p = (\rho - \rho_0)c_s^2, \tag{8}$$

where ρ_0 is initial fluid density. In the present simulation, a prediction–correction time–stepping scheme is applied [27]. Following [25] and [2], the present model uses the regular fixed ghost particles that are created to represent the solid boundary. Particle shifting method [18] is also used to avoid errors due to grossly irregular distribution of particles.

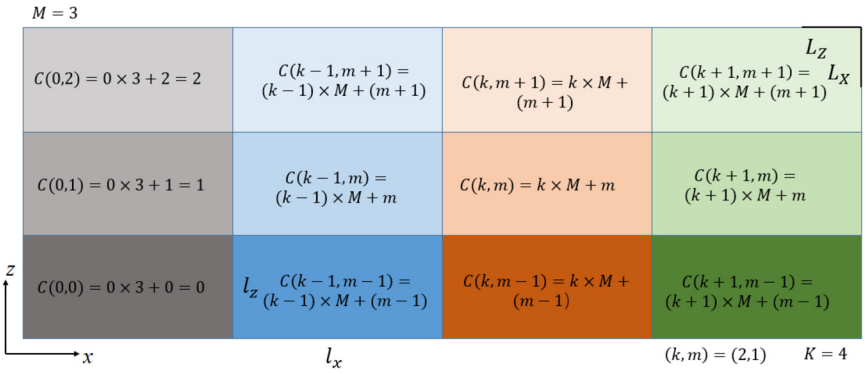

Fig. 1. Sketch of the 2-D domain decomposition and grid index list.

3. Parallelisation SPH framework

Contrary to Eulerian grid-based methods, SPH is a Lagrange mesh-free method with particles. This specificity causes some problems: (i) The interpolation-based kernel support domain generally includes a lot of neighbouring particles. The size of the support domain depends on the smoothing length. For a particle near the subdomain boundary, there is an interpolation truncation due to domain decomposition, and so it requires adjacent subdomains to allow particle-to-particle interactions. The particle searching process for a particle in one subdomain should include particles in neighbouring subdomains. (ii) The particles in one subdomain may move into another one, and so some strategies should be introduced to transfer particles. (iii) Load balance strategy. The transfer of particles between different processors may result in various numbers of particles per processor. Simulating a complex free surface flow with the SPH model usually results in irregularly distributed particles across the computational domain. Different numbers of particles lead to different computational costs for each processor. Load-unbalance affects parallel efficiency. Therefore, in the development of a parallel SPH framework, we need to address several of these issues.

3.1. Domain decomposition

Firstly, we need to decompose the entire computational domain into several subdomains. Each subdomain corresponds to a core. Domain decomposition is achieved by using a fictitious background Cartesian grid. Here, we do not consider load balancing strategies, but rather focus on the completion of a parallel framework. Thus, the background grid is fixed in time during the entire simulation, as shown in Fig. 1. Each grid denotes a subdomain. K and M denote the number of grids in x and z directions of the whole simulation domain. The length and height of the whole domain are denoted by L_x and L_z, respectively. If the region decomposition is homogeneous over the entire computational domain, the size (l_x, l_z) of each grid can be determined by

$$(l_x, l_z) = \left(\frac{L_x}{K}, \frac{L_z}{M}\right). \tag{9}$$

The parallelisation of the background meshes was not easy. We need to determine each mesh to prevent race conditions for the particle input process, i.e., multiple threads inputting the same particle at the same time. To overcome this issue, the subdomains should be marked and each of them shall correspond to a unique core. Here, the index sort method is used to get grid index $C_i(k, m)$ ($i = 0, 1, \cdot, N_c - 1$, $k = 0, 1, \cdot, K - 1$, $m = 0, 1, \cdot, M - 1$, $N_c = K * M$ is the total core numbers). The grid index $C(k, m)$ is computed as

$$C(k, m) = k * M + m. \tag{10}$$

Note that the index $C(k, m)$ of each subdomain is unique. In the basic uniform grid, particle i with position $r_i = (x_i, z_i)$ is inserted into one spatial grid with coordinates $C(k, m)$. After determining the boundary for each subdomain, we can find a unique corresponding grid C for every particle according to the position r_i. The sorted particle array is processed in parallel. Thus, with the above strategy, we mark all particles and assign them to the corresponding subdomains.

3.2. Message passing

In the SPH method, the interpolation of a particle requires contributions from its neighbouring particles. Thus, a particle around the subdomain boundary requires message from particles in adjacent subdomains to ensure kernel support integrity. It needs to be identified: (i) lists of adjacent subdomains and (ii) the particles in each subdomain that need to be passed.

3.2.1. Lists of adjacent subdomains

According to the domain decomposition method, a fixed regularly distributed background grid is used, and each grid has a unique sort index $C(k, m)$. Adjacent subdomains for grid $C(k, m)$ represent all surrounding grids that are in contact with this grid, and their sort indexes are $C(k-1, m-1)$, $C(k-1, m)$, $C(k-1, m+1)$, $C(k, m-1)$, $C(k, m+1)$, $C(k+1, m-1)$, $C(k+1, m)$ and $C(k+1, m+1)$ for a grid $C(k, m)$. For some subdomains near the boundary of the entire computational domain, which are not fully surrounded by other subdomains, the rule should be adjusted accordingly. This strategy is entirely based on the grid index list. If the subdomain division does not change, then this list of adjacent grids applies to the entire calculation process.

3.2.2. Interaction particles

In the SPH method, the search of the neighbour particles is based on the mutual distance of the interpolation points [11]. With the help of the background mesh, the links between particles and their neighbouring particles are locally constructed, which are used in the calculation, as shown in Fig. 2. The minimum side length of a mesh must equal the size of the support domain. Thus, there is a local neighbouring mesh list in each subdomain, which has the same sort index $\mathbb{C}(\mathbb{K}, \mathbb{M})$ as introduced in the work of domain decomposition. $\mathbb{C}$ is the local index for neighbouring particle search. It is found that message passing is only required for particles in the meshes besides subdomain boundaries. The meshes that do not have any interaction with adjacent subdomains are named as 'interior meshes', and the meshes that influence adjacent subdomains are, called 'interactive meshes'. Each subdomain needs to be divided by a small grid in order to perform particle searches.

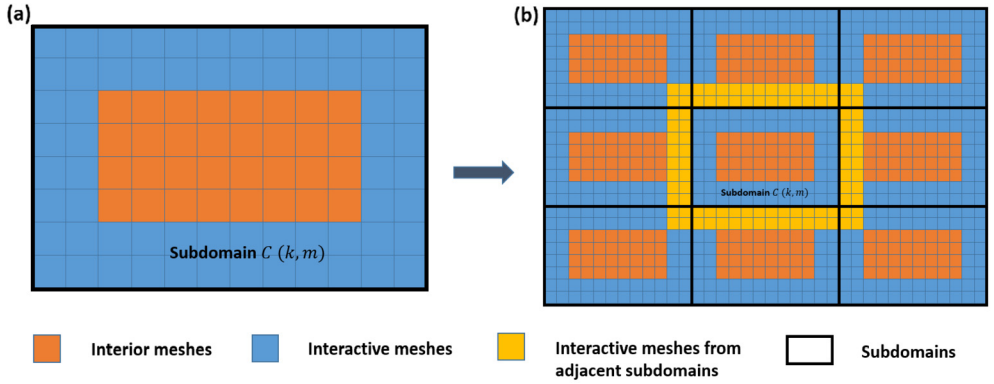

Fig. 2. Illustration of message passing. (a) Local neighbouring particle list, interior meshes and interactive meshes; (b) Interactive meshes message passing from adjacent subdomains. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

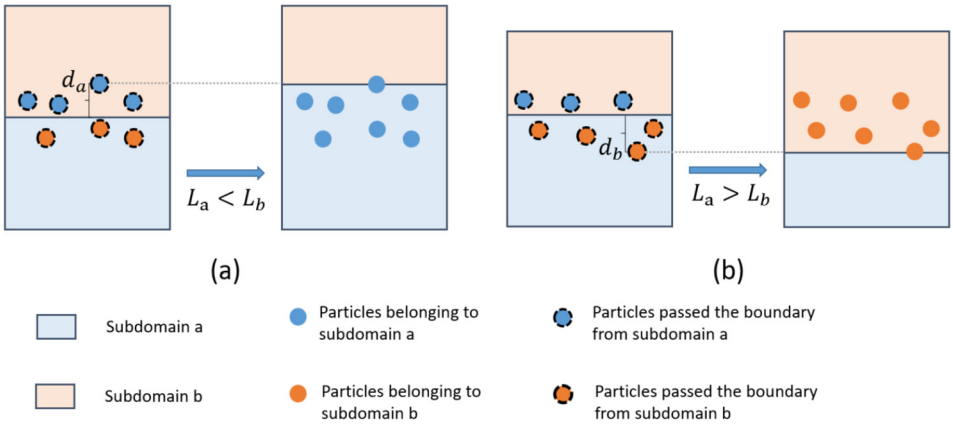

Fig. 3. Illustration of dynamic load balance strategy along y direction. (a): When $L_a < L_b$, subdomain boundary is shifted to subdomain b by d_a to reduce loading imbalance; (b) When $L_a > L_b$, subdomain boundary a is shifted to subdomain by d_b to reduce loading imbalance.

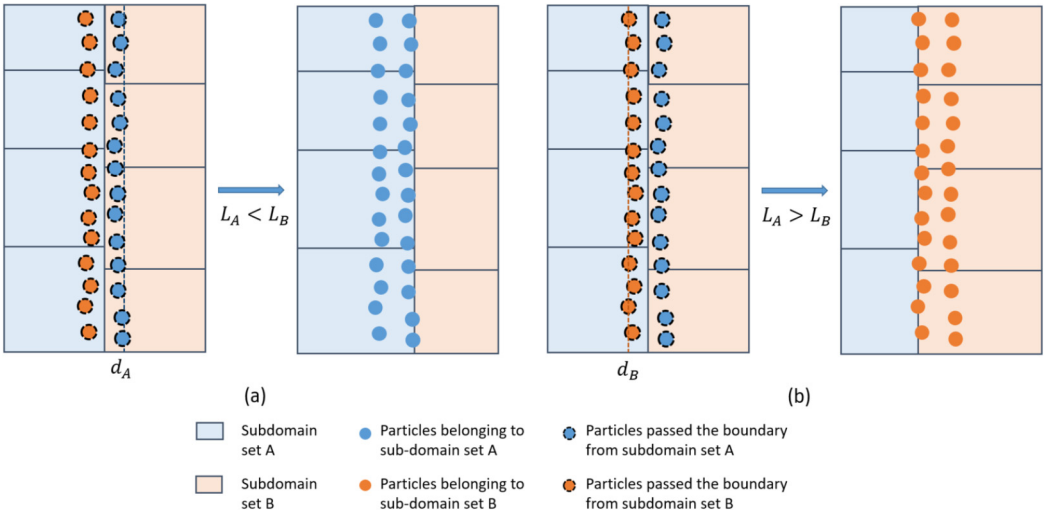

Fig. 4. Illustration of dynamic load balance strategy along x direction. (a): When $L_A < L_B$, subdomain boundary is shifted to subdomain set B by d_A to reduce loading imbalance; (b) When $L_A > L_B$, subdomain boundary is shifted to subdomain set A by d_B to reduce loading imbalance.

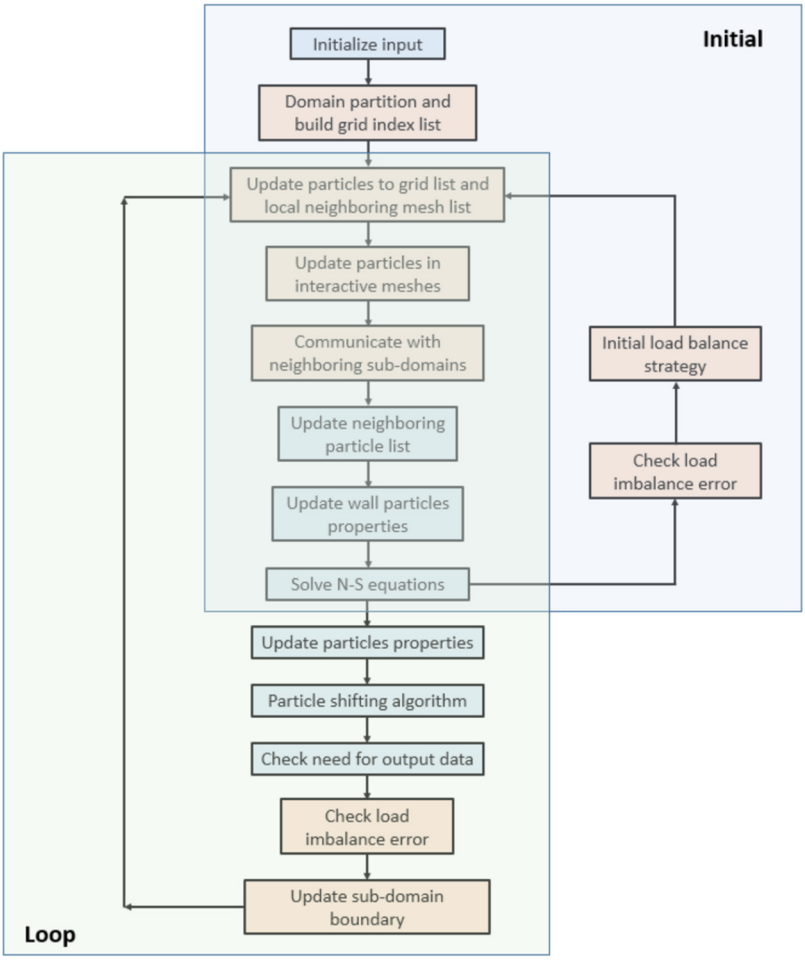

Fig. 5. Flowchart of SPH parallel framework.

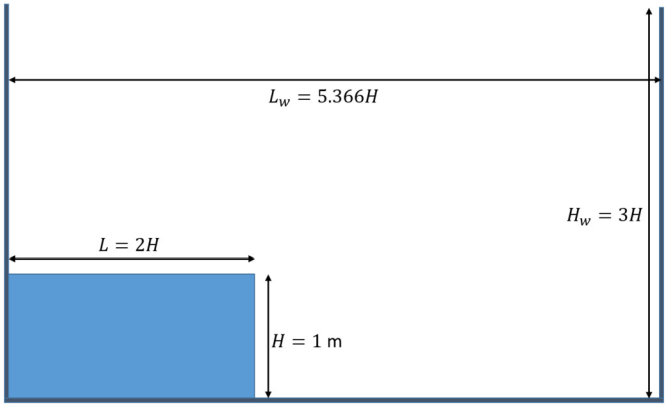

Fig. 6. Sketch for two-dimensional dam break problem.

However, the subdomain size is sometimes not exactly divided by the grid. This means that the local meshes of neighbouring subdomains are not perfectly matched. Therefore, the two-layer mesh near the subdomain boundary needs to be considered. As the local neighbouring particle list, the interactive meshes can be determined by

$$\forall\, \mathbb{C} \in [\mathbb{K} < 2 \vee \mathbb{K} > (\mathfrak{K} - 3) \vee \mathbb{M} < 2 \vee \mathbb{M} > (\mathfrak{M} - 3)] \\ \Longrightarrow \mathbb{C} \in \text{Interactive meshes}, \tag{11}$$

where $\mathfrak{K}$ and $\mathfrak{M}$ denote the number of grids in the x and z directions of the local neighbouring search mesh. As a result, this procedure dedicated to finding the interactive meshes should be completed just before the message passing. Then particles in the interactive meshes are interaction particles.

According to the region decomposition strategy, any subdomain is a regular quadrilateral. All subdomain boundaries are uniquely determined. Particle information can be reloaded into a new core based on the particle's position. At same time, the particle information is deleted from the previous core. Thus, the transfer of particles among cores is carried out by the deletion and addition of particle information. Since the displacement of the particle at each time step is small relative to the subdomain size, the transfer of particles only takes place in adjacent subdomains, which further narrows the choice of new subdomains for particles.

4. Dynamic load balance

As a Lagrangian particle-based method, SPH particles could move from one processor to another processor. In some particular cases, particles will be present in large numbers in one processor and in small numbers in other processors. In the numerical calculation of SPH, the number of SPH particles is related to the calculation load. At the same time, the computational time spent on the whole calculation process depends on the processor with the longest computational time. The uneven distribution of computational effort between processors is called 'load imbalance'. One of the biggest problems with load imbalance is that it reduces the speed of computation. This is because when a processor with a small number of particles has completed calculation, it needs to wait for a processor with a larger load to complete the compu-

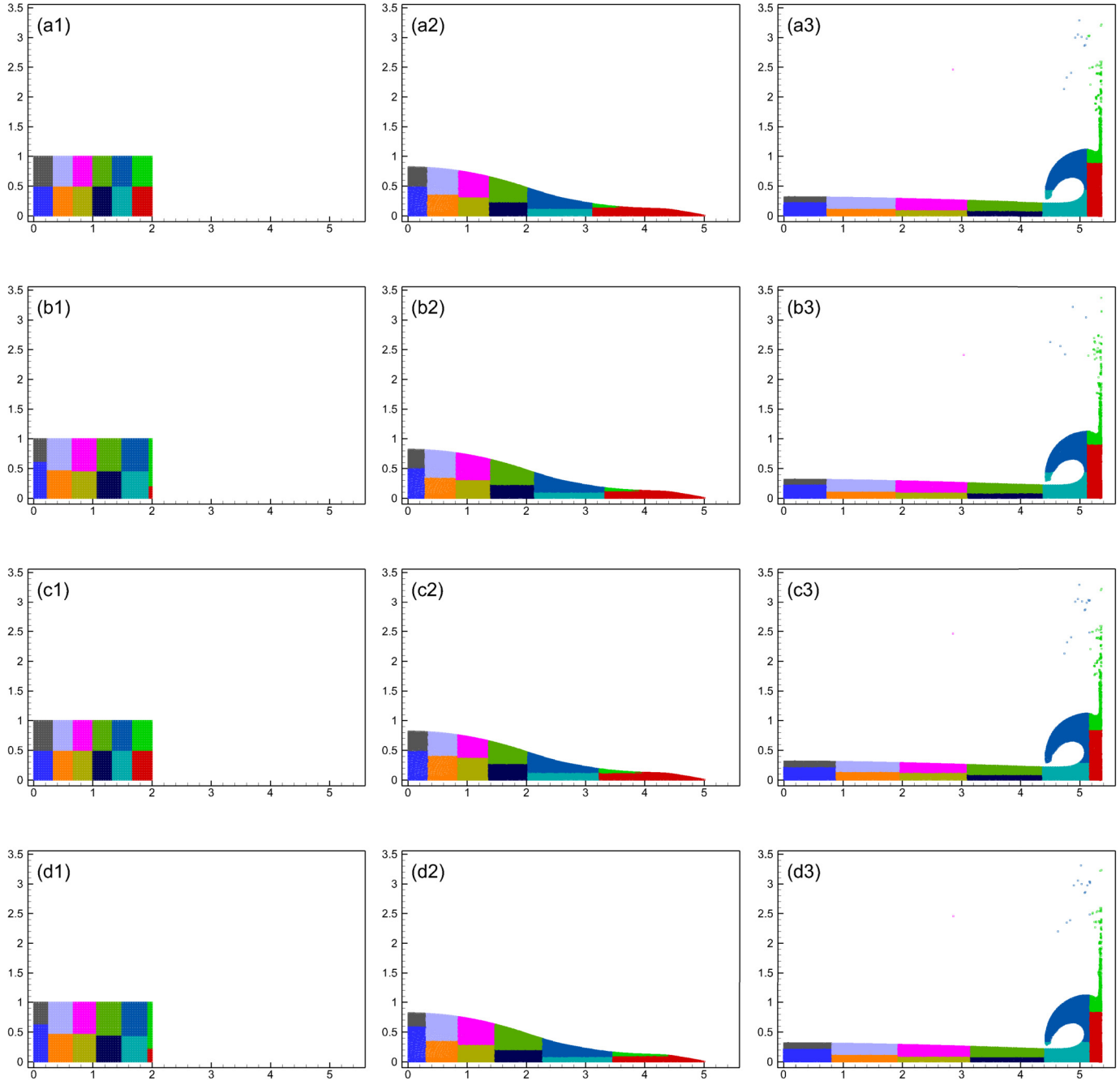

Fig. 7. Domain decomposition for test (a), (b), (c), and (d) at t = 0 s, 1.0 s, and 1.9 s. (Label '1', '2', and '3' denote t = 0 s, 1.0 s, and 1.9 s, respectively.) Each subdomain is shown in a different colour.

tation before it can start the next time step. Therefore, we need to specify special methods to ensure that each processor is loaded equally at each time step in order to maximise the efficiency of parallelism.

4.1. Computation load

4.1.1. Computation particle numbers

The distribution of the number of particles has a direct relationship to the computational loading. Thus, the measurement of the calculation load in each core is translated into a measurement of the number of particles. The load balancing problem is converted into a geometric problem, i.e. the number of particles in each region is guaranteed to be balanced by geometric division. Computational load balance strategies based on the number of particles are widely considered in published works [11,16,3,30,13]. In these works, the entire computational domain is discretized into regular meshes. The meshes are grouped into subdomains by means of space-filling curves or division methods. The load balance of the subdomains is balanced taking into account the number of particles in these meshes.

In the present SPH model, the calculation particle numbers N_{cn} can be calculated as

$$L = N_{cn} = N_f + N_w + N_{in} + N_{ip}, \tag{12}$$

where N_f, N_w, N_{in}, and N_{ip} are the numbers of fluid particles, wall particles, interpolation nodes, and interaction particles from other processors. L denotes computation load.

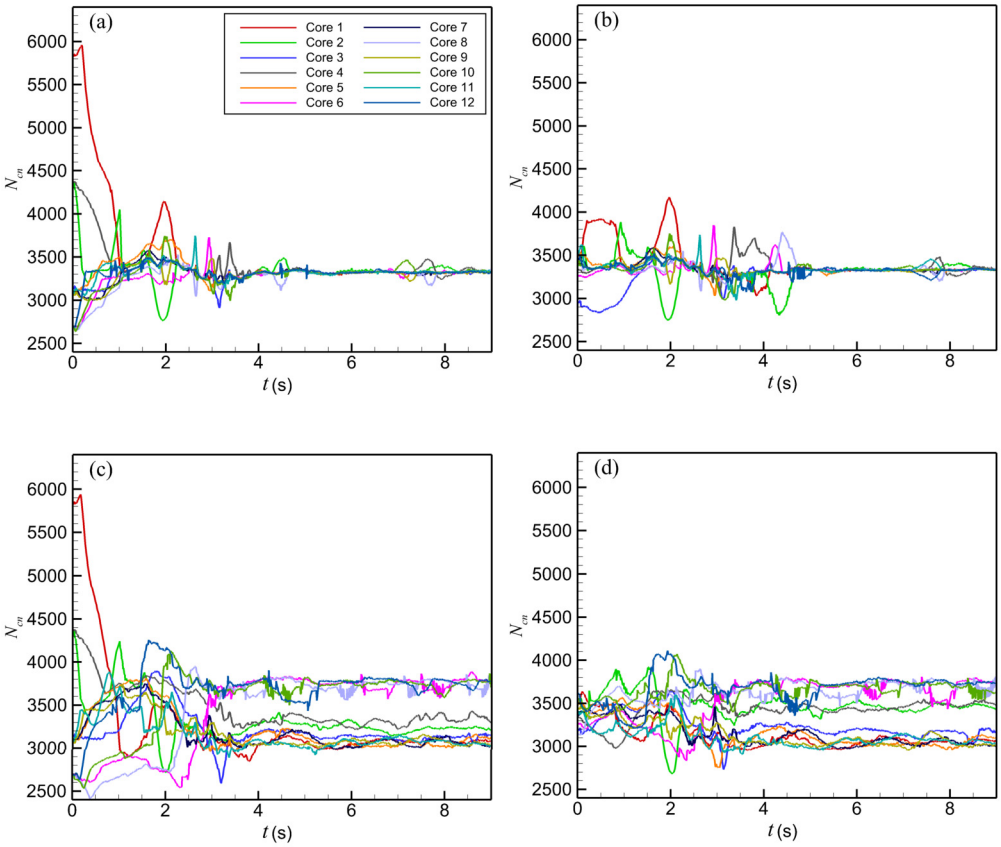

Fig. 8. History of calculation particle numbers N_{cn}. (Label 'a', 'b', 'c' and 'd' denote tests (a), (b), (c), and (d), respectively.)

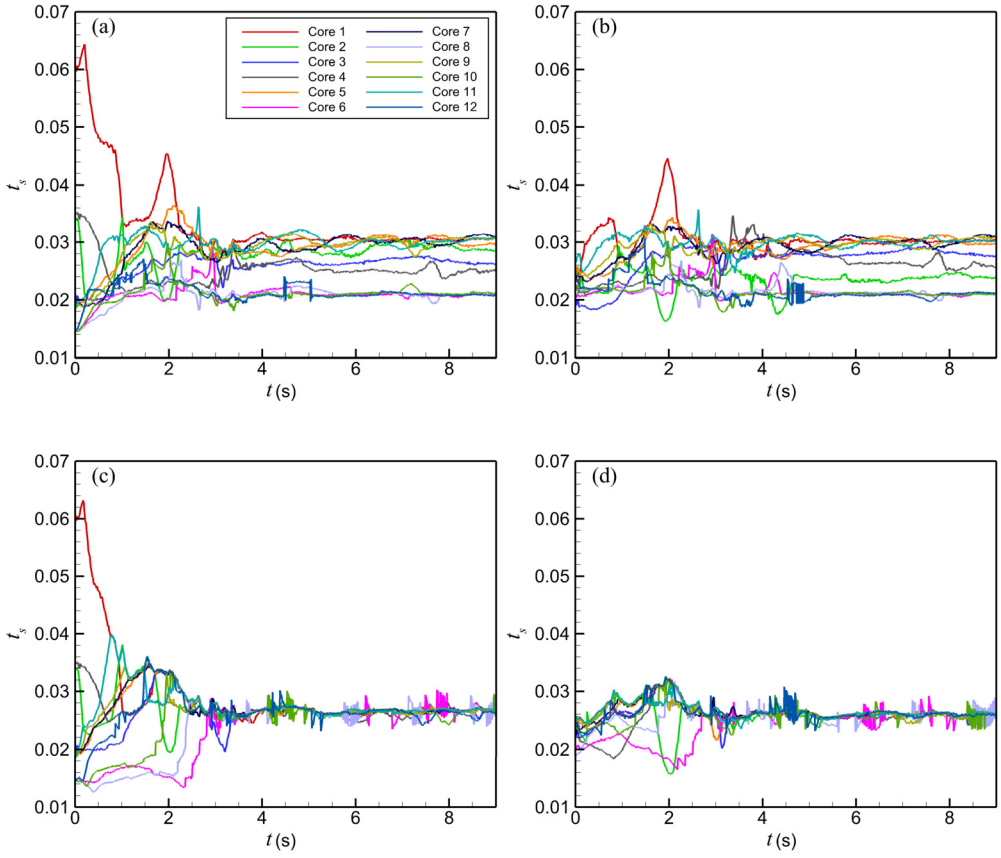

Fig. 9. History of running time t_s in each core. (Label 'a', 'b', 'c' and 'd' denote tests (a), (b), (c), and (d), respectively.)

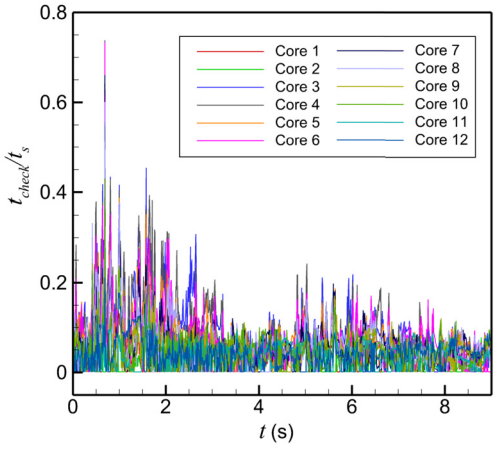

Fig. 10. History of ratio of check time t_{check} to running time t_s in each core for case (d).

However, there are several problems with the division based on calculating the number of particles. Firstly, the computational cost of various nodes/particles is inconsistent. For example, fixed ghost particles need an additional cost to determine their physical quantities. Thus, fluid particles near walls and fixed ghost particles require more computational effort. In addition, the correlation between the calculation load and the number of particles decreases if there are severe inhomogeneous distributions in the subdomains such as fragmentation, splashes, complex interface, etc. In the case of irregular distribution, the support domain of a particle is not fully filled with neighbouring particles. Even if the number of particles is the same, it does not mean that the number of particle pairs is the same. A parallel strategy based on computation particle numbers cannot guarantee a consistent computational cost in each core.

4.1.2. Running time

At each time step, the parallel SPH code is divided into two main parts: MPI communication and SPH solver. MPI communication represents the transfer of particle data between different cores. SPH solver represents the numerical solution of the SPH model, including neighbouring particle search, boundary conditions and solution of the Navier-Stokes equation. The load balance needs to take into account all of these components to achieve perfect results. However, in the current SPH code, we use $MPI-Send$ and $MPI-Recv$ for point-to-point communication. Communication load is not only affected by the number of transfer particles, but also communication blocks, which make it difficult to take into account the load balance of MPI communication for large-scale MPI parallel problems.

A more simple and reliable way to evaluate computational loading is to use running time. The main purpose of dynamic load balance is to improve parallel efficiency, i.e., to ensure that the computation time cost is consistent in each core. Running time based load balance ignores the complexity of the SPH process (code) and directly considers the primary purpose of load balance strategy. Moreover, the variability of core performance is already implicitly included. The calculation load L can be calculated as

$$L = t_s, \tag{13}$$

where t_s denotes the time cost for SPH process (not including communication cost) in each core. The greater the time consumption, the more the computational load. However, it seems like it requires us to run the code at the beginning of each time step to obtain the time cost of each core. It means that we need to run the code at least twice at each time step. The update of particle position is small due to small SPH time step. Thus, the information about the time cost of each core at the previous time step can be used as a computational load distribution to optimise subdomain partitioning

$$L^n = t_s^{n-1}, \tag{14}$$

where L^n is the computational load at n^{th} time step. t_s^{n-1} is the time cost at $n^{th}-1$ time step. The message of subdomain division at previous time is also recorded as a reference.

4.2. Subdomain update

In most previously published studies, the entire computational domain is divided into small grids. These small grids are then searched using space-filling curves. The combination of small meshes along the space-filling curve can achieve load balance. However, this approach would result in a completely new subdomain division, which cannot refer to the SPH time of the previous time step when considering the SPH time balance. In the present parallel strategy, the computational domain has been delineated as regular subdomains and marked by grid index list. Therefore, to achieve dynamic loading balance, subdomains are updated by changing the subdomain boundaries (size). This leads to the adjustment of its neighbouring subdomain boundaries, to ensure subdomains have no overlapping parts.

At each time step, many particles flow into and out of the subdomain. The inflow and outflow of particles at the boundary of a subdomain are shown in Fig. 3. The calculation loads are L_a and L_b for subdomain a and b, respectively. The change of boundary requires the determination of a reference position. For this purpose, we record the position of the particle furthest from the boundary as the reference position. The positions of the inflow and outflow particles furthest from the subdomain boundary are denoted as d_a and d_b, respectively. However, there is no inflow and outflow particle along subdomain boundary sometimes, so then a particle spacing is used as a reference position. When $L_a > L_b$, the boundary along the y-axis direction should be shifted inside the subdomain a to d_a in order to reduce the size of the subdomain A to ensure there is a load balance between subdomains a and b. When $L_a < L_b$, the boundary should be moved inside the subdomain b to d_b in order to increase calculation loads in subdomain b. The change in boundaries causes the division of subdomains to become irregular. To be able to continue using the grid index list, we set the subdomains of the same column to have the same boundaries along the x-axis. When considering changes in the boundaries of subdomains along the x-axis, we need to consider all subdomains in the same column as a whole, called set A and B as shown in Fig. 4. The rules of subdomain boundaries along x-axis direction are the same as the those along y-axis direction, as shown in Fig. 4. Total calculation particle numbers in set A and B are considered as the computation load for the balance strategy based on calculation particle numbers. While the maximum SPH time in set A and B is used as computation load for load balance strategy based on running time.

To achieve load balance of the system, an accurate evaluation of load is important. Evaluation of the load situation allows the determination of whether to implement the dynamic load balancing strategy. We define an imbalance monitoring tag as

$$E_{i,max} = \frac{L_S^{max} - L_S^{min}}{L_S^{avg}} < e_d, \tag{15}$$

where $E_{i,max}$ is max error for all cores, e_d is user defined error tolerance. L_S^{max}, L_S^{min}, and L_S^{avg} denote max, min, and average calculation load for all cores. Once max error $E_{i,max}$ is greater than

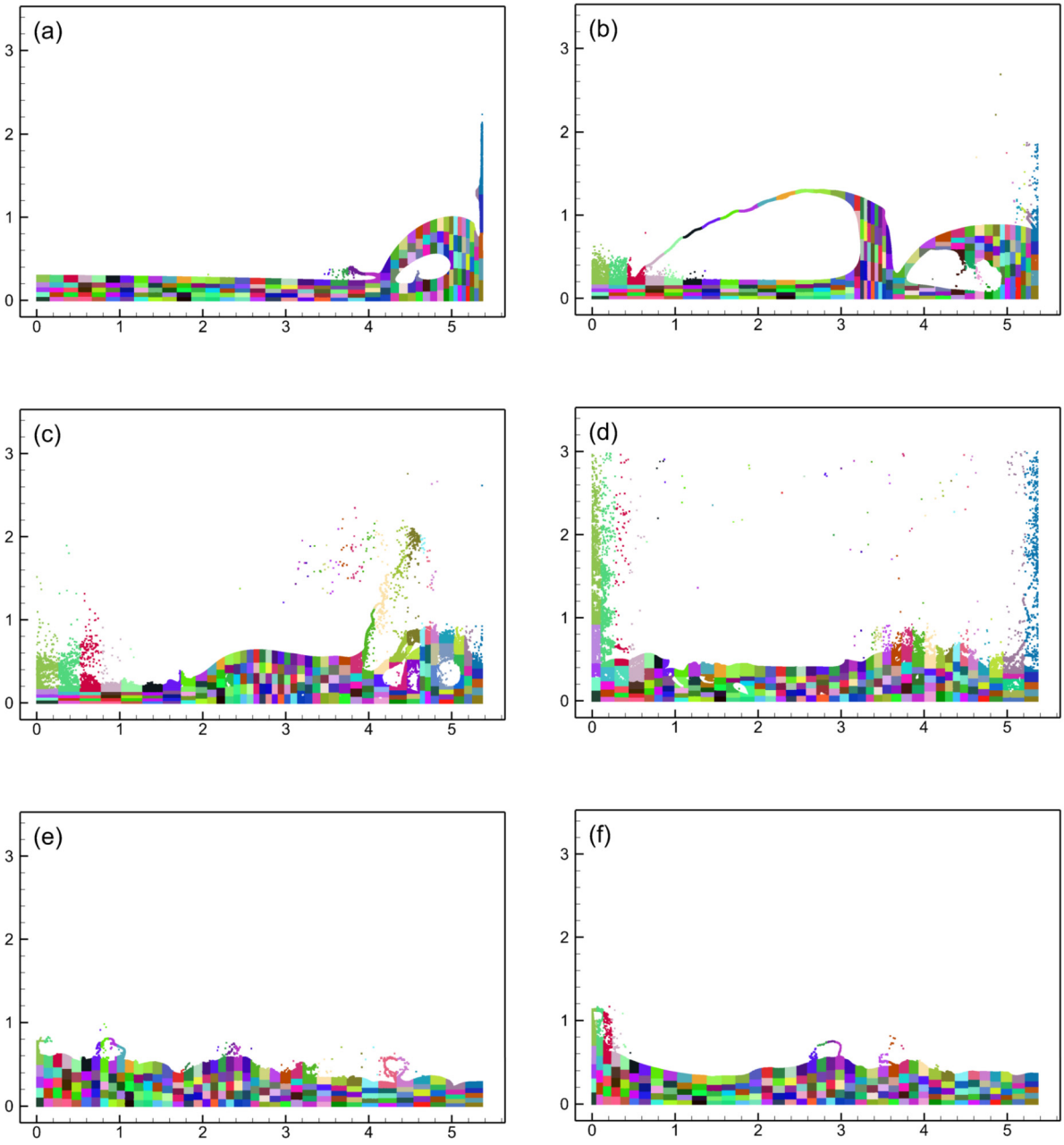

Fig. 11. Subdomain distribution at t = 2.0 s (a), 2.5 s (b), 3.0 s (c), 3.5 s (d), 5.0 s (e) and 9.0 s (f).

the set value, the dynamic load balancing strategy can be implemented. In addition, this criterion also applies to the local load balance distribution E_i as follows

$$E_i = \frac{L_a - L_b}{L_a} < e_d. \tag{16}$$

This means that the load balance between local cores is not simultaneous, but depends on the local load balance distribution. The check of dynamic loading balance can be implemented for every few tens or hundreds of time steps. This depends on the definition of the user.

It takes some time to obtain load balance for initially unstable fluid problems. Thus, we develop an initial load balance strategy to maintain the initial homogeneous load balance. Initially, the entire computational domain is divided evenly. The subdomain boundary positions are then updated according to calculation loads. This approach is the same as the dynamic loading balance strategy described above. The subdomain boundaries are continuously adjusted until the conditions of Eq. (15) are met. The maximum number of iterations (Number of adjustments) is set to 1000, and the physical information of the particles is not updated in the initial homogeneous load balance.

A detailed flowchart of the developed framework is presented in Fig. 5 to summarise all the algorithms. The code is written in FORTRAN using open source libraries OpenMPI. If not specifically stated, studies in this paper are conducted on the Fotcluster2 in High Performance Computer Centre in University of Plymouth. Fotcluster2 is a 752 core distributed-memory cluster, which is comprised of: a 3U combined head & storage node, plus 56 compute nodes. The tests are conducted on the phase2 consisting of 36 Viglen $HX425T^2i$ HPC 2U Compute Nodes, equipped with Dual In-

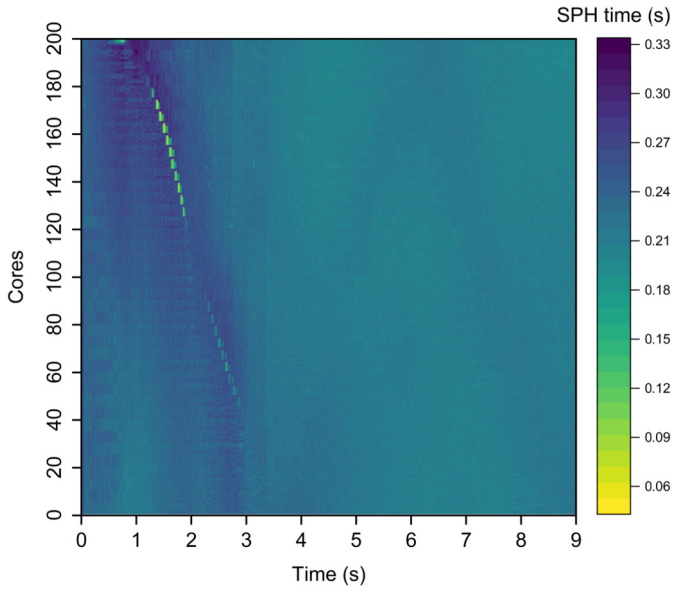

Fig. 12. The heat map of SPH time for the dam break case with 1,000,000 fluid particles and 200 cores.

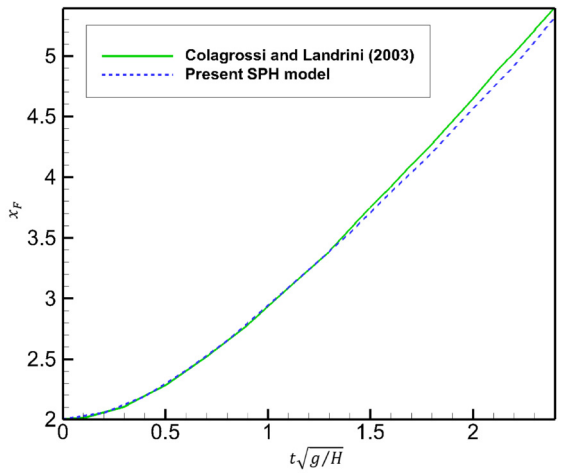

Fig. 13. Time series of water front.

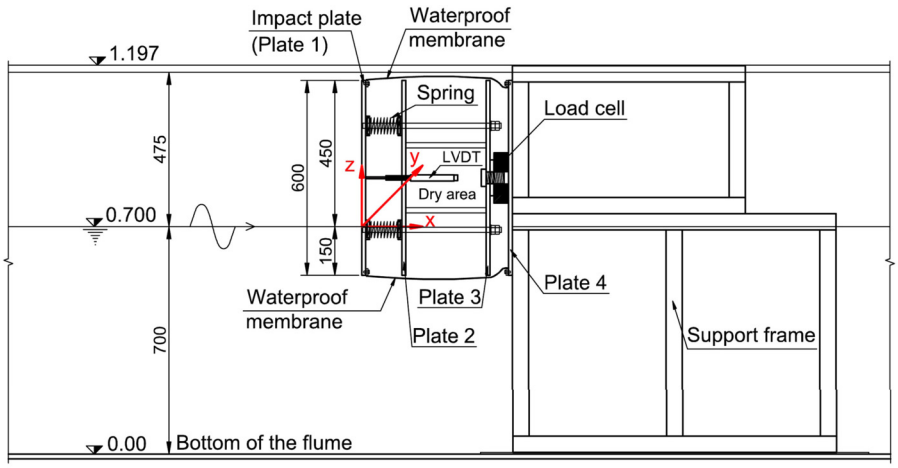

Fig. 14. Side view sketch of physical model [15].

tel Xeon E5650 (Westmere) Six Core 2.66 GHz processors and 12 GB of memory per motherboard.

5. Numerical validation

5.1. Dam breaking

Dam breaking is widely investigated in the SPH literature, since this case shows the ability of SPH models to deal with large deformation problems. The deformation of the water phase can be used to validate the dynamic load balance strategy. Fig. 6 shows a sketch of the initial setup, where the reservoir height is $H = 1.0$ m, length and height of the tank are $d = 5.366H$ and $D = 3.0H$. Initial particle spacing is 0.01 m, and 20,000 fluid particles are used. Time step is 0.0005 s, and numerical sound speed is $c_0 = 10\sqrt{gH}$. A total of 12 cores, 6 cores along the x-direction and 2 cores along the y-direction, are used. Four tests are employed here for a comprehensive assessment of the performance: (a) dynamic load balance strategy based on calculation particle number, fluid domain uniformly divided at the beginning; (b) dynamic load balance strategy based on calculation particle number with initial homogeneous load balance; (c) dynamic load balance strategy based on running time, fluid domain uniformly divided at the beginning; (d) dynamic load balance strategy based on running time with initial homogeneous load balance. e_d is 0 for all these four cases. The check of dynamic loading balance can be implemented at each time step.

Fig. 7 compares the division of the computational domain for test (a), (b), (c), and (d) at $t = 0$ s, 1 s, and 1.9 s. Although the initial divisions are the same for tests (a) and (c), the division of subdomains showed some differences at 1.0 s and 1.9 s because of different load balance strategies. Although tests (a) and (b) ((c) and (d)) use the same load balance strategy, there are still some differences between the divisions at 1.0 s and 1.9 s due to the different divisions initially.

Fig. 8 shows the time series of calculation particle numbers $Ntol$ in each core for tests (a), (b), (c), and (d). Tests (a) and (b) show small oscillations around the mean value generally after $t = 4.0$ s. Although tests (c) and (d) show relatively stable changes after $t = 4.0$ s, the stability interval for N_{cn} in each core is not consistent. The changes of N_{cn} in tests (b) and (d) are relatively stable compared to tests (a) and (c) in the beginning, respectively. Note that, the dynamic load balance strategy based on the calculation

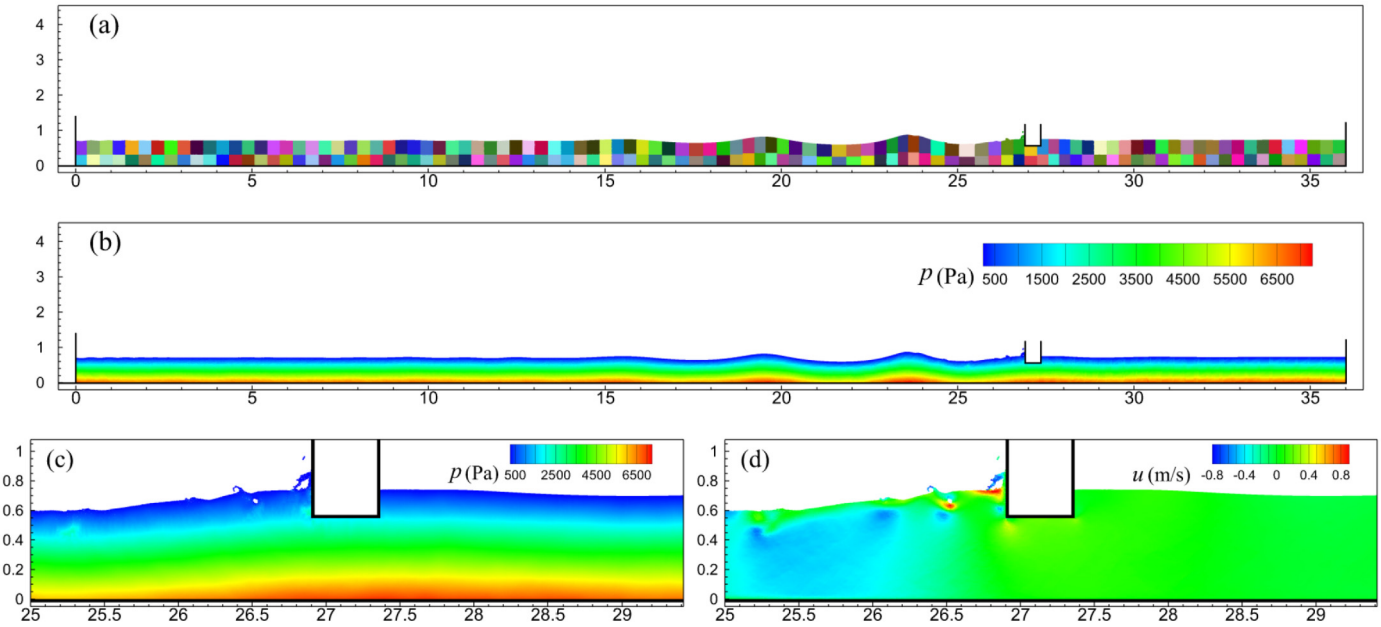

Fig. 15. Subdomain distribution at 35.1 s (a); Pressure field in the whole flume at 35.1 s (b); Local pressure field at 35.1 s (c); Local horizontal velocity field at 35.1 s (d).

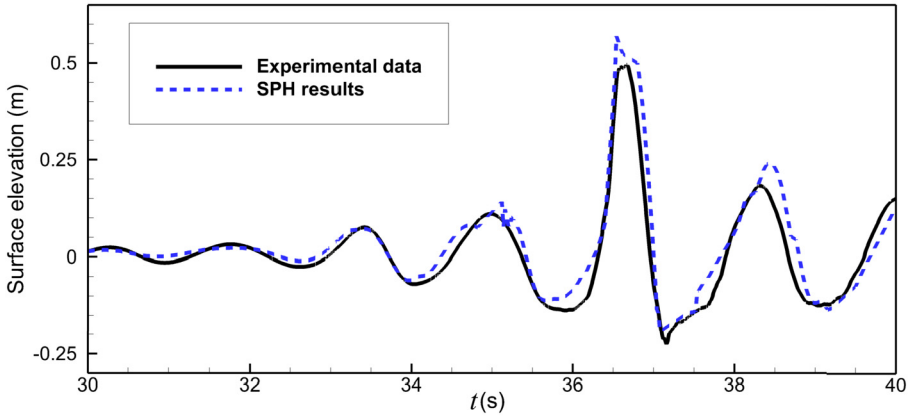

Fig. 16. Comparison of wave elevation at a position 26.885 m.

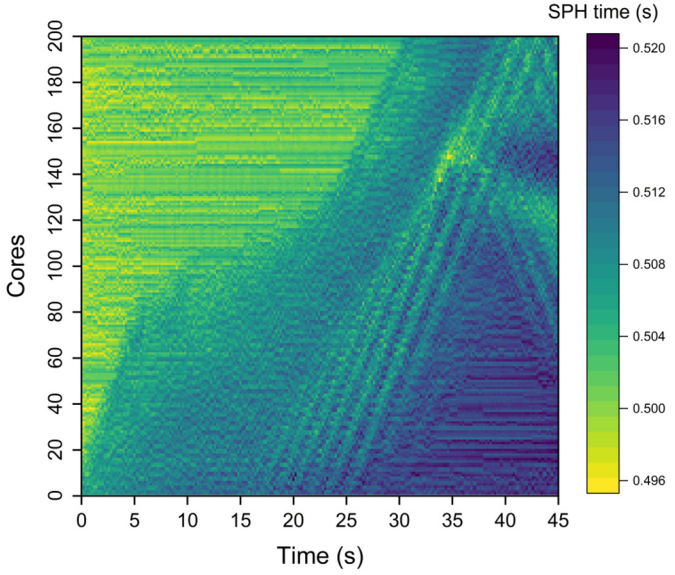

Fig. 17. The heat map of SPH time for the wave structure case with 1,575,000 fluid particles and 200 cores.

particle numbers N_{cn} ensures that calculation particle number in each core is balanced.

Fig. 9 shows the time series of SPH time T_s in each core for tests (a), (b), (c), and (d). Although N_{cn} in tests (a) and (b) reach the steady state after $t = 4.0$ s, the stability interval for T_s varies in each core. Moreover, the difference between the maximum (about 0.031 s) and minimum (about 0.021 s) running times is approximately 0.01 s. In comparison, the running time for tests (c) and (d) are stable around 0.025 s after $t = 4.0$ s. The time spent for a time step depends on the core that consumes the longest amount of time. Obviously, the time consumption of a time step for a loaded balancing strategy based on the number of particles computed is greater than that of a balancing strategy based on the running time after $t = 4.0$ s. Furthermore, the initial load balancing strategy ensures that the running time is relatively balanced at the start of the calculation.

Sharp changes in calculation particle numbers N_{cn} and running time T_s are observed before $t = 4.0$ s. It can be seen that the use of load balancing strategies does not instantly result in perfect load balancing when loading is most unbalanced. The update of the subdomain boundaries is based on the gradual adjustment of the particle position changes. Nevertheless, the load balance strategy avoids an exacerbation of the load unbalance condition. The

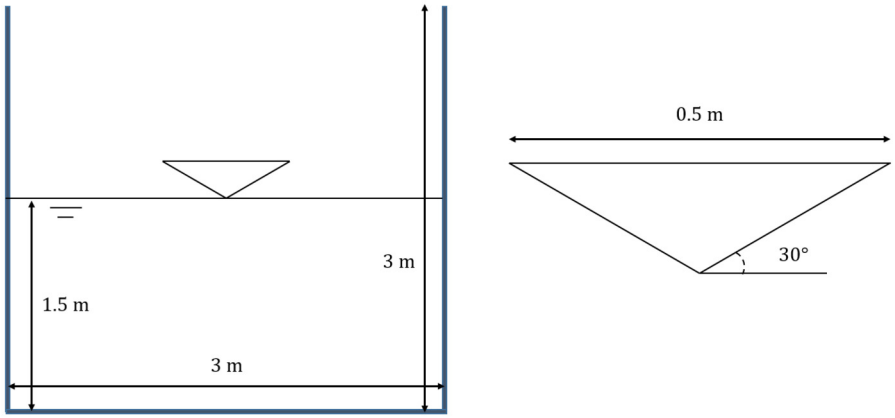

Fig. 18. Geometry of water tank and wedge.

Table 1
SPH setup and computational time.

Test	Fluid particle number	Time step (s)	Number of cores	Physical time (s)	Wall time (s)
(a)	20000	0.0005	12	9	1836.33
(b)	20000	0.0005	12	9	1784.83
(c)	20000	0.0005	12	9	1646.83
(d)	20000	0.0005	12	9	1549.16

load balance strategy guarantees almost perfect balance at each time step, in terms of either the running time or the number of calculated particles after $t = 4.0$ s. Table 1 shows the time cost in these four cases. The load balance strategy based on running time can improve computational efficiency more, compared to the one based on calculation particle numbers. For example, test (d) is 15.2% more efficient than test (b). Although the initial load balancing strategy requires extra computational effort at the initial moment, the overall computational cost is effectively reduced.

Fig. 10 presents the ratio of check time t_{check} to SPH time t_s at each time step. It can be found that the ratio of check time spent is greater than 0.2 when time (t) is less than 3 s. At this point, the small SPH time resulting from loading imbalance leads to a large ratio. At t = 3 s to 5 s and 8 s to 9 s, the ratio varies around 0.05. In addition, the total check time is 15.64 s, which is 0.01 of the total time spent. Therefore, the detection time spent is small in the overall calculation.

The dam break then is extended to be simulated with 1 million fluid particles (Initial particle spacing is 0.002 m) to test the performance of present model at a scale of the hundreds of cores and millions of particles. A total of 200 cores, 40 cores along the x-direction, 5 cores along the y-direction, are used. The time step is 0.0001 s. This case uses the proposed dynamic load balance strategy based on running time with initial homogeneous load balance. The physical time was 9 s, taking a total of 9.5 hours to complete the calculation. Fig. 11 shows the subdomain snapshots at six moments. The complex free surface can be observed at 2.5 s, where the upwardly deflected fluid then falls violently onto the wet deck under the recovering action of gravity producing a large splash at 3.0 s. Eventually, it reaches a relatively stable state at 9 s, i.e., no strong slamming with large splashes occurring. The complex free surface is divided into 200 non-overlapping subdomains by regular small rectangles. Fig. 12 shows the heat map, where The y-axis represents the number of cores and the x-axis represents physical time, colour indicates SPH time. An imbalance in running time can be observed until 3 s. After 3 s, the SPH time is fairly even across the 200 cores. In particular, the balancing running time at 3.5 s corresponds to Fig. 11, showing the performance of the current parallel model with dynamic load balancing. Fig. 13 shows the time series of the water front. The agreement with the reference solution demonstrates the accuracy of the parallel model.

5.2. Wave structure simulation

Physical experiments of a focused wave group interacting with a truncated vertical wall are modelled in this section [24,15]. The wave flume is 35 m long, 0.6 m wide in cross-section, 1.2 m high and operates at a still water depth of 0.7 m. The plate is 26.9 m from the wave maker and submerged to a depth of 0.15 m. An incident wave set (focused wave) was used in the experiment. A theoretical focus time is 42 s and a theoretical focus position is at 31.90 m. The values of still water level, position of the plate and wave condition were set in general agreement with the experiment as shown in Fig. 14. In particular, the plates are fixed during the experiment. The numerical flume length is 36 m. A sponge layer located between 31 m and 36 m is used to absorb the wave to prevent wave reflection. The free surface level is measured at 26.885 m from the wave maker for comparison with the experimental data. The initial particle spacing was 0.004 m with a total of 1,575,000 fluid particles. e_d is set to 0.05, and the time step was set to 0.0002 s. We have chosen a different $E_{i,max}$ to show that a wide range of options of $e_{i,max}$ does not affect the parallel implementation. The physical time was 45 seconds, taking a total of 31.5 hours to complete the calculations.

As shown in Fig. 15(a), a total of 200 cores are used in this example, with 2 cores in the vertical direction and 100 cores distributed along the horizontal direction. The smooth pressure distribution in the whole flume at 35.1 s can be observed in Fig. 15(b). Fig. 15(c) and Fig. 15(d) show the locally smooth pressure distribution as well as the horizontal velocity distribution. Splashing shows the ability of SPH to reconstruct the non-linear free liquid surface. Fig. 16 shows the free surface elevation in front of the plate in comparison with the experiment. The peak free surface predicted by SPH near 36.7 s is overestimated by 0.05 m. After 38 s, there is a phase shift of the free surface, which may be due to the effect of the sponge layer not absorbing waves well.

In Fig. 17, all cores spend a similar amount of time at the beginning, due to the implementation of the initial load balancing strategy. Before 30 seconds, the time spent increases for an in-

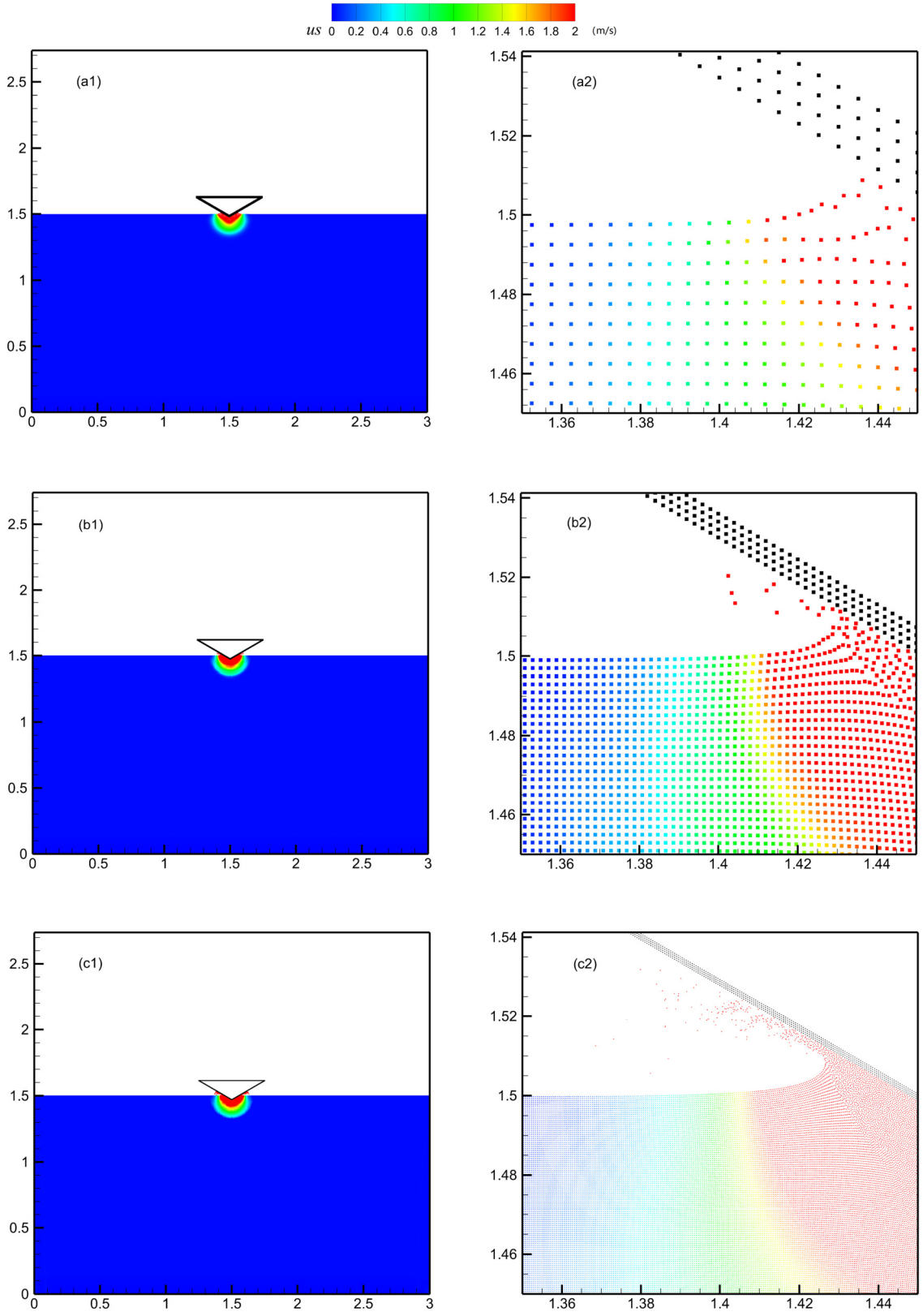

Fig. 19. The velocity of wedge impact simulation involving various particle resolution at $t = 0.005$ s. (Label 'a', 'b', and 'c' denote tests 45 thousand, 1.125 million, and 18 million, respectively; '1' and '2' denote whole water wank and local domain, respectively.)

Table 2
CPU data of water entry of a wedge.

Test	Resolution (m)	Fluid particle number	Timestep (s)	Number of timestep	Cores	Serial CPU time (s/particle/time-step)
(a)	0.01	4.5×10^4	0.00005	100	12	1.7×10^{-4}
(b)	0.002	1.125×10^6	0.00001	500	288	2.4×10^{-4}
(c)	0.0005	1.8×10^7	0.0000025	2000	1920	4.6×10^{-4}
(d)	0.002	1.125×10^6	0.00001	500	144	2.0×10^{-4}

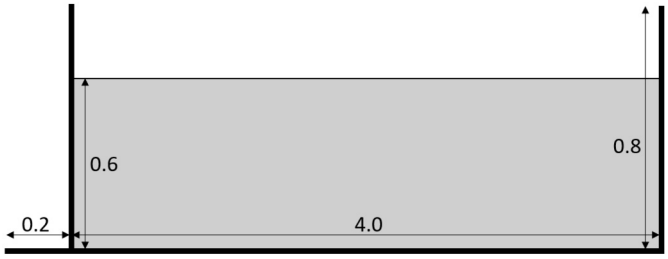

Fig. 20. Simulation setup for scaling tests.

creasing serial number of cores. The larger the core serial number, the later the increase in SPH time occurs. This may be due to the fact that the wave propagates gradually from the wave maker (on the left side) to the right side. Thus, when the wave propagates on the left side of the tank, the complications of the flow condition (the motion of free surface, particles transfer between adjacent cores, etc.) make the corresponding processors slower compared to those of the still water related ones. In spite of this, the maximum difference in time consumption for the 200 cores over the entire 45 seconds is approximately 0.0255 seconds (5.1% relative to the minimum run time), which is a very slight loading imbalance. The current parallel model still achieves relatively good load balance.

5.3. Water entry of a wedge

This case simulates the free fall of a wedge in initially calm water [36]. The width of the wedge is 0.5 m with an angle 30° and mass 72.5 kg, as shown in Fig. 18. The depth and the width of water tank are 1.5 m and 3 m, respectively. Initial vertical velocity of the wedge is 6.15 m/s. The convergence of the present SPH model is checked via three different particle resolutions in this part: 0.01 m, 0.002 m, and 0.0005 m, resulting in 45 thousand, 1.125 million, and 18 million fluid particles, correspondingly. e_d is 0.05 for this case. The tests are conducted on the China Science and Technology Cloud (CSTC), which consists of 823688 cores on 858 compute nodes. The tests are conducted on 96 core CPU nodes each using Intel Xeon Platinum 9242@2.3 GHz.

A snapshot at $t = 0.005$ s is illustrated in Fig. 19, where the effect of particle resolutions on the free surface jet capture can be observed. As particle resolution increases, splashes are better captured. Table 2 presents the numerical set up and time cost for three particle resolutions. The number of time steps varies from resolution to resolution because to the Courant-Friedrichs-Levy (CFL) condition; it is inversely proportional to the particle resolution. Table 2 shows the comparable serial CPU time for each resolution. A complete parallelisation should provide an equivalent serial CPU time. This value rises with spatial resolution, as seen in Table 2. This difference may mostly be attributed to communication blockage. The additional time caused by blocking communication becomes more obvious as the number of cores rises.

5.4. Performance analysis

Weak and strong scaling tests are conducted to evaluate the overall parallel performance of our current implementation. The test case examined is the 2D 2^{nd} Stokes wave (wave height 0.1 m and wave period 2.0 s) in a numerical wave flume. Each simulation is run for 1000 time steps. The initial set-up is as shown in Fig. 20. The regular numerical tank facilitates the setting up of numerical cases to meet the needs of strong and weak scale tests. The tests are conducted on CSTC, and e_d is 0.05.

The weak scale test is a test to ensure that the number of particles in each core is consistent while increasing the number of cores. Ideally, as the number of core increases, the computation time should remain constant. However, due to communication load, as the number of cores increases, the computation time also increases. We define the computational cost t_r for 60 cores as a reference and calculate the weak scale efficiency e_w for different numbers of cores

$$e_w = \frac{t_i}{t_r}, \tag{17}$$

where t_i is the time cost for the number i of cores. Three different groups of single core particle numbers of 30,000, 100,000, and 200,000 (written as 30T, 100T, and 200T) were tested at 60, 120, 240, 480, 960, and 1920 cores, respectively. Fig. 21 shows the time spent and the efficiency of the weak scale tests. As the number of cores increases, the calculation time increases, whereas the efficiency decreases as expected. Ultimately, for the cases run in 1920 cores, the efficiencies are only 0.73, 0.86, and 0.92 for 30, 100, and 200 thousand particles per core, respectively. Although the number of particles in each core remains the same, the increase in communication time due to the increased number of cores affects the overall computational efficiency. For the case of more particles in each core, the communication time takes up less of the overall time, leading to a higher computational efficiency.

In the case of strong scaling, the number of cores is increased while the problem size remains constant, resulting in a reduced workload per core. Speedup S_s and efficiency e_s for strong scaling studies are calculated from

$$S_s = \frac{N_r \times T_r}{T_s}, \tag{18}$$

$$e_s = \frac{N_r \times T_r}{N_s \times T_s}. \tag{19}$$

Three different sets of total particle numbers of 2.4, 15 and 38.4 million (written as 2.4M, 15M, and 38.4M) were tested at 60, 120, 240, 480, 960, and 1920 cores, respectively. Fig. 22 shows the computational speedups as well as the efficiencies under the strong scale tests. Increasing the number of cores can reduce the number of particles per cores. This leads to a reduction in the cost per core calculation. However, the cost for overhead computation and communication of information also increases. For the cases run in 1920 cores, the efficiencies are 0.72, 0.89 and 0.93 for the cases with 2.4, 15 and 38.4 million particles, respectively. It is expected that the efficiency could be improved. Meanwhile, higher efficiency can be obtained by introducing a data decomposition of particle interaction loops through a shared memory parallel framework.

5.5. Three-dimensional (3D) dam breaking

To demonstrate the capabilities of the model for 3D problems, the dam breaking case in §5.1 is extended to three dimensions

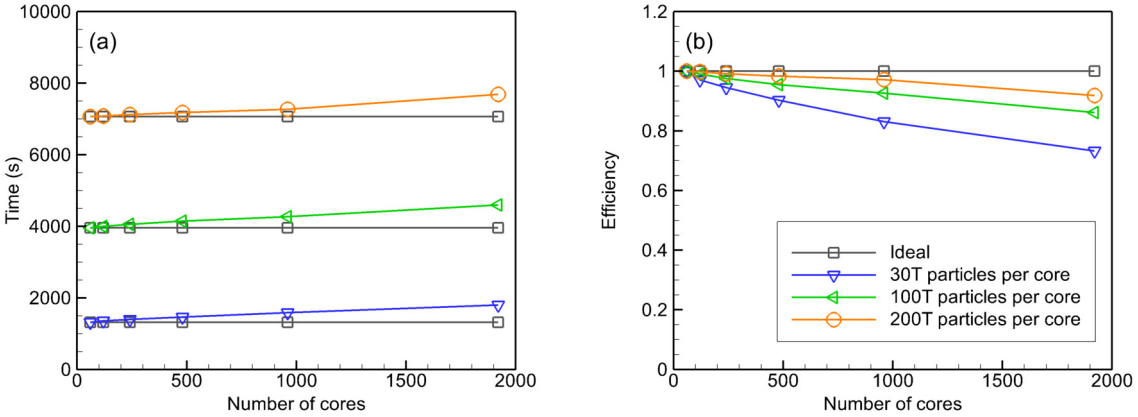

Fig. 21. Time cost (a) and efficiency (b) of weak scaling tests.

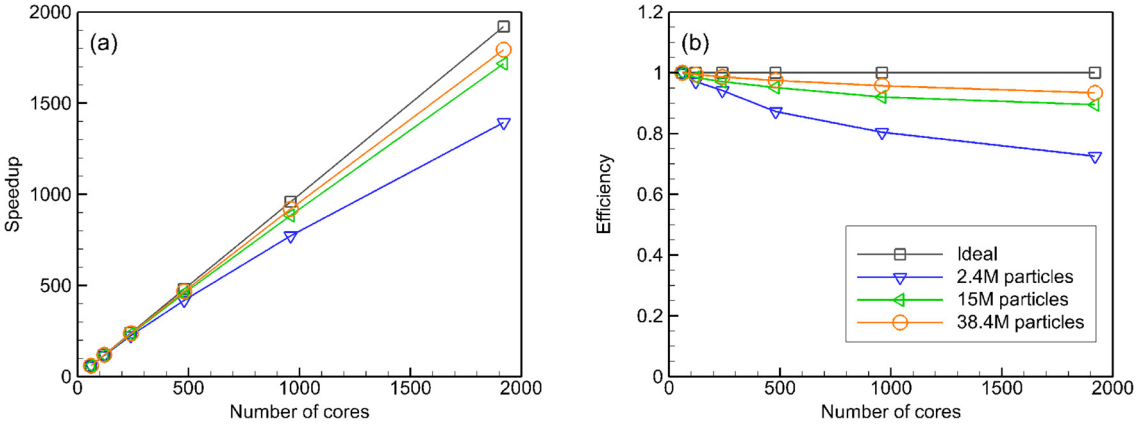

Fig. 22. Speedup (a) and efficiency (b) of strong scaling tests.

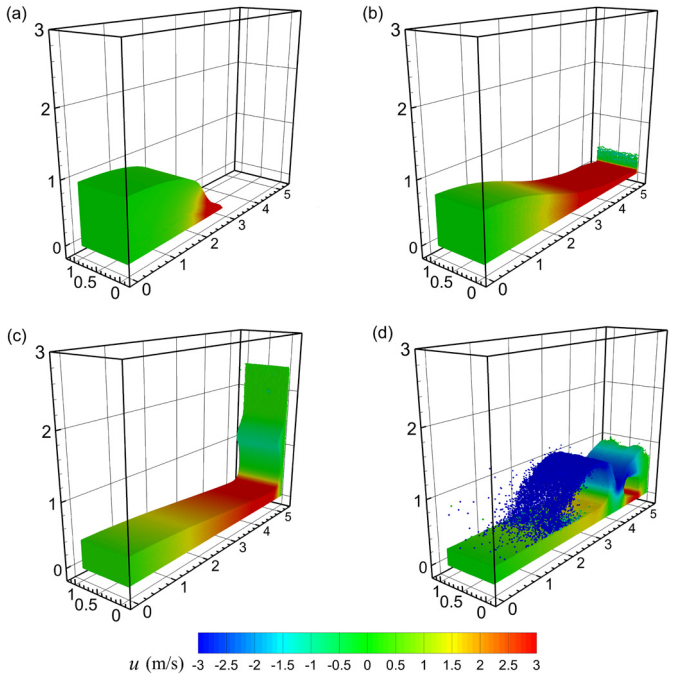

Fig. 23. Snapshot of the horizontal velocity field of 3D dam breaking case at t = 0.3 s (a), 0.8 s (b), 1.5 s (c) and 2.3 s (d).

with a width of 1.0 m. The initial particle spacing is 0.01 m, and a total of 2,000,000 fluid particles are simulated. A total of 96 cores, 4 cores along the x-direction, 24 cores along the y-direction, are used. The time step is 0.0001 s. This case uses the proposed dynamic load balance strategy based on running time with initial homogeneous load balance. The physical time was 2.5 s, taking a total of 15.6 hours to complete the calculation. Fig. 23 shows the horizontal velocity at four different moments. It can be noticed that the parallel model simulates the 3D problem and still achieves smooth and stable results, although the current model only considers dynamic loading balance in the direction of two degrees of freedom (x and z). When dealing with complex 3D problems, loading balance in three directions (x, y and z) of freedom needs to be considered in the future.

6. Conclusions

This paper has presented a new MPI-based parallel SPH framework with a dynamic load balance strategy for free-surface flow. The new framework uses a background Cartesian grid to decompose the domain, and a grid list to map the entire domain. Meanwhile, within each subdomain there exists a local mesh for neighbouring particle search as well as determining the particles that need to be transferred. In the dynamic load balance strategy, two evaluation criteria, i.e., computation particle numbers and running time, are considered. The update of the subdomain divisions is achieved by updating the subdomain boundary according to the workload/subdomain division from the previous time step. An initial load balance strategy is developed to maintain the initial homogeneous load balance. Results show that the present parallel SPH framework can simulate free surface flow with load balancing, even for free surface conditions with large deformation.

In the dam breaking cases, the load balance strategy based on the calculation particle number can achieve the balance of the

calculation particle number. The load balancing strategy based on running time ensures that the running times in each core are almost uniform, regardless of different particle numbers. In terms of overall time cost, the load balancing strategy based on running time achieves better parallel efficiency. The complex free surface in the dam breaking cases can be captured, demonstrating that the current parallel framework can guarantee dynamic load balance even in the face of large deformations. The wave-structure case demonstrates the ability of the present SPH model to simulate the numerical wave tank with millions of particles for tens of seconds. The water entry of wedge is simulated with various particle resolutions, which involved up to 18 million particles as well as 1920 cores. The results show that higher resolutions allow for better capture of the non-linear free-surface condition. However, the difference in equivalent serial CPU time at different resolutions and cores shows the extra cost of blocking communication. This parallel SPH model was then tested on a large scale uniform particle distribution of up to 1920 cores. Nevertheless, it still shows a decrease in efficiency in the case of 1920 cores. Finally, the parallel SPH model is used to simulate a 3D dam breaking case. Future work should consider the extension of the model to 3D complex flows. Dynamic loading balancing requires consideration of loading imbalances in the three degrees of freedom (x, y and z) directions. Meanwhile the indexed list method needs to be extended to three degrees of freedom. The development of non-blocking MPI parallel strategies to overlap communication and calculation cost has the potential to further improve parallel efficiency for 3D problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

The research was supported by the financial support from China Scholarship Council (Grant No. 201806060137). The corresponding author gratefully acknowledges the support from Open Research Fund Program of State Key Laboratory of Ocean Engineering, Shanghai Jiao Tong University (Grant No. 1916). The authors acknowledge the data of wave structure simulation from the CCP-WSI Working Group.

References

[1] M. Antuono, A. Colagrossi, S. Marrone, Comput. Phys. Commun. 183 (2012) 2570–2580.
[2] B. Bouscasse, A. Colagrossi, S. Marrone, M. Antuono, J. Fluids Struct. 42 (2013) 112–129.
[3] J.M. Cherfils, G. Pinon, E. Rivoalen, Comput. Phys. Commun. 183 (2012) 1468–1480.
[4] A.D. Chow, Incompressible SPH (ISPH) on the GPU, The University of Manchester, United Kingdom, 2018.
[5] A.D. Chow, B.D. Rogers, S.J. Lind, P.K. Stansby, Comput. Phys. Commun. 226 (2018) 81–103.
[6] A. Colagrossi, M. Landrini, J. Comput. Phys. 191 (2003) 448–475.
[7] A.C. Crespo, J.M. Dominguez, A. Barreiro, M. Gómez-Gesteira, B.D. Rogers, PLoS ONE 6 (2011) e20685.
[8] A.J. Crespo, J.M. Domínguez, B.D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, O. García-Feal, Comput. Phys. Commun. 187 (2015) 204–216.
[9] J.M. Domínguez, G. Fourtakas, C. Altomare, R.B. Canelas, A. Tafuni, O. García-Feal, I. Martínez-Estévez, A. Mokos, R. Vacondio, A.J. Crespo, et al., Comput. Part. Mech. (2021) 1–29.
[10] M.S. Egorova, S.A. Dyachkov, A.N. Parshikov, V. Zhakhovsky, Comput. Phys. Commun. 234 (2019) 112–125.
[11] A. Ferrari, M. Dumbser, E.F. Toro, A. Armanini, Comput. Fluids 38 (2009) 1203–1217.
[12] R.A. Gingold, J.J. Monaghan, Mon. Not. R. Astron. Soc. 181 (1977) 375–389.
[13] X. Guo, B.D. Rogers, S. Lind, P.K. Stansby, Comput. Phys. Commun. 233 (2018) 16–28.
[14] F. He, H. Zhang, C. Huang, M. Liu, Coast. Eng. 156 (2020) 103617.
[15] Z.Z. Hu, T. Mai, D. Greaves, A. Raby, J. Fluids Struct. 75 (2017) 99–116.
[16] M. Ihmsen, N. Akinci, M. Becker, M. Teschner, in: Computer Graphics Forum, Wiley Online Library, 2011, pp. 99–112.
[17] Z. Ji, L. Fu, X.Y. Hu, N.A. Adams, Comput. Methods Appl. Mech. Eng. 346 (2019) 1156–1178.
[18] A. Khayyer, H. Gotoh, Y. Shimizu, J. Comput. Phys. 332 (2017) 236–256.
[19] M. Liu, Z. Zhang, Sci. China, Phys. Mech. Astron. 62 (2019) 984701.
[20] M.B. Liu, G.R. Liu, Arch. Comput. Methods Eng. 17 (2010) 25–76.
[21] L.B. Lucy, Astron. J. 82 (1977) 1013–1024.
[22] M. Luo, A. Khayyer, P. Lin, Appl. Ocean Res. 114 (2021) 102734.
[23] H.G. Lyu, P.N. Sun, X.T. Huang, S.Y. Zhong, Y.X. Peng, T. Jiang, C.N. Ji, Energies 15 (2022) 502.
[24] T. Mai, Z.Z. Hu, D. Greaves, A. Raby, in: The Twenty-Fifth International Ocean and Polar Engineering Conference, OnePetro, 2015.
[25] S. Marrone, M. Antuono, A. Colagrossi, G. Colicchio, D.L. Touzé, G. Graziani, Comput. Methods Appl. Mech. Eng. 200 (2011) 1526–1542.
[26] S. Marrone, B. Bouscasse, A. Colagrossi, M. Antuono, Comput. Fluids 69 (2012) 54–66.
[27] J.J. Monaghan, J. Comput. Phys. 82 (1989) 1–15.
[28] J.J. Monaghan, J. Comput. Phys. 110 (1994) 399–406.
[29] J.J. Monaghan, Rep. Prog. Phys. 68 (2005) 1703–1759.
[30] G. Oger, D. Le Touzé, D. Guibert, M. De Leffe, J. Biddiscombe, J. Soumagne, J.G. Piccinali, Comput. Phys. Commun. 200 (2016) 1–14.
[31] P.N. Sun, A.M. Zhang, S. Marrone, F. Ming, Appl. Ocean Res. 72 (2018) 60–75.
[32] D. Violeau, B.D. Rogers, J. Hydraul. Res. 54 (2016) 1–26.
[33] E. Yang, H.H. Bui, H. De Sterck, G.D. Nguyen, A. Bouazza, Comput. Geotech. 121 (2020) 103474.
[34] T. Ye, D. Pan, C. Huang, M. Liu, Phys. Fluids 31 (2019) 011301.
[35] H. Zhang, Z. Zhang, F. He, M. Liu, Eur. J. Mech. B, Fluids 94 (2022) 1–16.
[36] R. Zhao, O. Faltinsen, J. Aarsnes, in: Proceedings of the 21st Symposium on Naval Hydrodynamics, 1996, pp. 408–423.
[37] G. Zhu, L. Zou, Int. J. Numer. Methods Fluids 92 (2020) 950–975.
[38] G. Zhu, L. Zou, Z. Chen, A. Wang, M. Liu, Int. J. Numer. Methods Fluids 86 (2018) 167–184.
[39] L. Zou, G. Zhu, Z. Chen, Y. Pei, Z. Zong, Int. J. Comput. Methods 15 (2018) 1850008.