# A distributed-memory MPI parallelization scheme for multi-domain incompressible SPH

Alessandra Monteleone[a,*] & Gaetano Burriesci[a,b] and Enrico Napoli[c]

[a] *Bioengineering Unit, Ri.MED Foundation, Palermo, Italy*
[b] *UCL Mechanical Engineering, University College London, London, United Kingdom*
[b] *Università degli Studi di Palermo, Dipartimento di Ingegneria, Viale delle Scienze, 90128 Palermo, Italy*

**Abstract**

A parallel scheme for a multi-domain truly incompressible smoothed particle hydrodynamics approach is presented. The proposed method is developed for distributed-memory architectures through the Message Passing Interface (MPI) paradigm as communication between partitions.

The proposal aims to overcome one of the main drawbacks of the SPH method, which is the high computational cost with respect to mesh-based methods, by coupling a multi-resolution approach with parallel computing techniques.

The multi-domain approach aims to employ different resolutions by subdividing the computational domain into non-overlapping blocks separated by *block interfaces*. The particles belonging to different blocks are efficiently distributed among processors ensuring well balanced loads.

The parallelization procedure handles particle exchanges both throughout the blocks and the competence domains of the processors. The matching of the velocity values between neighbouring blocks is obtained solving a system of interpolation equations at each *block interfaces* through a parallelized BiCGSTAB algorithm. Otherwise, a whole *pseudo-pressure* system is solved in parallel considering the Pressure Poisson equations of the fluid particles of all the blocks and the interpolation equations of all the *block interfaces*.

The employed test cases show the strong reduction of the computational efforts of the SPH method thanks to the interaction of the employed multi-resolution approach and the proposed parallel algorithms.

*Keywords:* Parallel distributed-memory computation; MPI; Load balancing; Smoothed particle hydrodynamics (SPH); ISPH; Multi-domain approach.

*amonteleone@fondazionerimed.com

## 1. Introduction

Smoothed particle hydrodynamics (SPH) is a mesh-less Lagrangian particle method originally developed by Gingold and Monaghan (1977) and Lucy (1977) to deal with astrophysical problems. Thanks to its remarkable flexibility, the SPH method is nowadays widely applied in many engineering applications to solve situations where grid-based methods have difficulties such as geometrically complex domains, moving boundaries, multi-phase flows (Liu and Liu, 2010; Monaghan, 2012) and fluid-structure interaction problems (Antoci et al., 2007; Liu and Zhang, 2019; Monteleone et al., 2022). However, it is commonly recognized that SPH is computationally more expensive than Eulerian methods. A very promising approach is to leverage the advantages of both methods by coupling the larger computational efficiency of grid-based approaches with the flexibility of SPH (Napoli et al., 2016; Chiron et al., 2018; Wang et al., 2021; Huang et al., 2022). To this aim, Eulerian methods (such as Finite Volumes or Finite Elements) can be applied in some portions of the computational domain, limiting the use SPH in regions where deeper weaknesses are found in grid-based methods.

Differently, this work aims to overcome the high computational cost of SPH by combining refinement strategies with parallel computing techniques on Central Processor Units (CPUs). In order to avoid prohibitive computational times, the use of multiple CPU and/or Graphical Processor Unit (GPU) architectures is a suitable and widespread strategy. In the field of parallel computing on CPUs, the Message Passing Interface (MPI) paradigm is widely used as communicator between partitions in distributed-memory architectures. The subdivision of the load-work among the CPUs (the so-called *domain distribution* process) is usually based on a spatial decomposition of the computational domain (De Marchis and Milici, 2016; Milici and De Marchis, 2016). Due to the Lagrangian mesh-less feature of the SPH method, with particles moving with the fluid velocity, the *domain distribution* is not trivial. Moreover, parallelization algorithms must dynamically handle particles leaving or entering the portion of the domain assigned to each processor. In order to obtain a well-balanced load, the total number of fluid particles should be thus distributed fairly among the processors and the CPU inter-communications should be minimized.

These difficulties in the parallelization of SPH codes are common for the two schemes employable to solve the governing equations for incompressible fluid flows with SPH: the weakly compressible (WCSPH) (Monaghan, 1992) and the truly incompressible (ISPH) (Lind et al., 2012) approaches.

With reference to the WCSPH scheme, where the governing equations are solved introducing a thermodynamic equation of state to relate pressure and density, some CPU-based parallelizing strategies have been proposed in the past. Ferrari et al. (2009) presented a parallelized SPH scheme using the MPI paradigm with a dynamic load balancing procedure for simulating three-dimensional non-hydrostatic free-surface flows. Marrone et al. (2012) developed

a hybrid MPI-OpenMP programming SPH model to analyse $3D$ wave patterns generated by a ship. Specifically, they adopted a parallelization scheme which combines a domain decomposition, performed on distributed-memory architectures through MPI, and a data decomposition, implemented on shared-memory architectures through OpenMP directives. Oger et al. (2016) proposed a dynamic load balancing algorithm helpful to simulate fragmented fluid domains (*e.g.* drops and jet breakup), following a modified Orthogonal Recursive Bisection (ORB) method based on a sampled distribution of the particles in each direction. Moreover, there have been successful parallelization schemes of WCSPH on the emerging GPU-based parallelization, such as the open-source *DualSPHysics* code (Crespo et al., 2015; Domínguez et al., 2013). In this context, unconventional algorithms were proposed and integrated in the parallelized version of the code, related to the treatment of open boundary conditions (Tafuni et al., 2018) and neighbour list searching (Winkler et al., 2018). Zhan et al. (2019) developed a GPU-accelerated coupled total Lagrangian and WCSPH approach for the modelling of fluid-structure interaction problems.

When the ISPH approach is considered, the parallel computing implementation is even more challenging, due to the difficulty in parallelizing the Pressure Poisson Equations (PPEs). The parallelization procedure of PPE should be carefully and efficiently implemented, as the PPE system solution absorbs most of the computational time. Furthermore, since the particle connectivity is constantly changing with the evolution of the flow, the sparse coefficient matrix of the PPE system changes at each time step, increasing the computational complexity and execution time. In the framework of the ISPH approach, Guo et al. (2018) proposed a massively parallel scheme to simulate free-surface flows, involving more than 100 million particles, based on MPI libraries. A very promising ISPH parallelization using GPU threads was proposed by Chow et al. (2018) within the *DualSPHysics* code. Recently, O'connor et al. (2021) proposed a parallelized strategy of the incompressible Eulerian SPH on multi-GPU. In this approach, the PPE is treated implicitly and, as a results of the Eulerian formulation, the particles are fixed within the domain.

The complexity in the parallelization of SPH is increased when variable refinement strategies are employed to save computational costs. These strategies are mostly based on the use of variable particle mass in different regions of the domain, adapting their dimension (and, accordingly, their mutual distance and number) to the local requirements. In this way, large numbers of smaller particles are employed where a higher refinement is required, while maintaining an acceptably lower number of particles elsewhere. In these approaches, splitting and coalescing techniques are required for the particles, due to their dimensional changes as they move towards regions with different refinement (Vacondio et al., 2013; Spreng et al., 2014; Barcarolo et al., 2014; Vacondio et al., 2016; Hu et al., 2017). A preliminary multi-GPU implementation of adaptive particle splitting and merging to adjust local resolution was proposed by Xiong et al. (2013).

A very different refinement procedure, based on a multi-domain approach, was proposed by Monteleone et al. (2018). In this approach, the domain is partitioned into non-overlapping blocks (or subdomains), each characterised by

a different refinement. The blocks are separated by interfaces (named *block interfaces*), where suitable procedures allow to obtain the matching of the solution between neighbouring blocks, thus satisfying the continuity constraint while particles transfer from one block to another. The classical SPH method is thus maintained in each block, without the need of splitting and coalescing methods to take into account the different mass of the particles going through the *interfaces*. The multi-domain technique, developed in the framework of the ISPH scheme, has been implemented in the open-source *PANORMUS* (PArallel Numerical Open-souRce Model for Unsteady flow Simulations) package, distributed under the terms of the *GNU General Public License*. In such multi-domain approach, the implementation of parallel computing techniques becomes extremely complex, due to the difficulty to balance among the processors the distribution of particles belonging to different blocks, and to handle the relative data exchanges. Further difficulties arise when the *block interfaces* are shared among different partitions.

The afore-mentioned issues are addressed in this paper, where a parallel multi-domain ISPH scheme on multiple CPUs is proposed by exploiting of the MPI libraries. The proposed parallelization procedure is implemented in the *PANORMUS* code. Although the technique is implemented in the framework of the multi-domain SPH approach of Monteleone et al. (2018), it can be directly applied with no modification for the "classical" ISPH method with constant resolution, where only a single-domain is considered.

## 2. The numerical method

### *2.1. Fundamentals of the SPH method*

The SPH method is based on the approximation of a function $f$ at point $\mathbf{x}$ using the convolution product of $f$ with a *kernel function* $W$

$$\langle f(\mathbf{x})\rangle = \int_D f(\mathbf{x}')\, W(\mathbf{x}-\mathbf{x}', h) d\mathbf{x}' \tag{1}$$

where $D$ is the computational domain and $h$ is a characteristic width of $W$, called *smoothing length*. The employed *kernel function*s have compact support, assuming null values for distances $|\mathbf{x}-\mathbf{x}'|$ larger than $kh$, where $k$ is a constant depending on the shape of the specified *kernel function*. In this paper the *Wendland* function (Wendland, 1995) is used, where the proportionality constant $k$ is equal to 2.

In the particle approximation, the fluid domain is represented by a finite number of *effective particles* which, while moving in compliance with the Navier-Stokes equations, carry fluid properties such as position, mass, velocity, pressure, etc.. Each particle $i$ is associated with a *support domain* $\Omega_i$, defined by a sphere of radius $kh$ with center at the particle position $\mathbf{x}_i$. In the discretized form, the convolution integral in eqn. (1) for particle $i$ can be approximated as the

summation over the $N_i$ particles belonging to the domain $\Omega_i$

$$f_i = \sum_{j=1}^{N_i} \frac{m_j}{\rho_j} f(\mathbf{x}_j)\ W_{ij} \tag{2}$$

where $f_i \approx \langle f(\mathbf{x}_i) \rangle$, $m_j$ and $\rho_j$ are the mass and density, respectively, of the neighbouring particle $j$, and $W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j, h)$ depends on the distance $d_{ij}$ between particles $i$ and $j$.

A reference isotropic initial particle distance $\Delta x$ is commonly assigned as proportional to $kh$; in this paper $\Delta x$ is set equal to $kh/2$.

The first derivatives of $f$ can be obtained with the derivative of the *kernel function*, and the Laplacian operator can be expressed by the Morris' formula (Morris et al., 1997).

In order to identify the list of neighbouring particles, a Cartesian *virtual grid* made of cubic cells of side length equal to $kh$ is employed, which simplifies the particle search.

Several modifications of the above formulae are commonly used to improve the accuracy of computation, in order to overcome problems related to irregularity in the particles distribution (see among others Oger et al. (2007); Xu et al. (2009); Napoli et al. (2015)).

### *2.2. Boundary treatment*

In order to account for the truncation of the *support domain* occurring at the solid walls and to impose suitable boundary conditions, the *mirror particle* technique was used. This procedure, which is described in detail among others by Napoli et al. (2015), relies on the generation of virtual particles through the mirroring, with respect to the solid walls, of the *effective particles* located at distances not larger than $kh$ from the boundaries.

A similar procedure is employed at the inlet and outlet sections, where additional particles are introduced to impose the required pressure and velocity boundary conditions. The procedure is described in detail in Monteleone et al. (2017).

### *2.3. Truly incompressible SPH (ISPH) methodology*

In order to solve the momentum and continuity equations for incompressible flows using an ISPH algorithm, a *fractional-step* procedure (Kim and Moin, 1985; Chorin, 1968) is used, which estimates the updated velocity as the sum of the non-solenoidal intermediate velocity and the corrective velocity. In the predictor-step, the intermediate velocity is calculated by removing the pressure gradient term from the momentum equations. This velocity is not divergence-free, since the continuity equation is not solved. In order to obtain the corrective velocity field, which is irrotational (with potential $\psi \Delta t$), the Pressure Poisson equations (PPE) are then solved obtaining the *pseudo-pressure* $\psi$. In the corrector-step, the corrective velocity can be obtained as $\nabla \psi \Delta t$. Particles are thus moved at the end of each time step, using the calculated velocity field.

For a detailed description of the ISPH procedure used in the PANORMUS code, see (Napoli et al., 2015; Monteleone et al., 2017).

In this paper, in order to overcome the well-known *tensile instability* problem (Swegle et al., 1995), the algorithm proposed by Xu et al. (2009) is adopted. This involves shifting slightly the particles across streamlines, in order to avoid their stretching and bunching.

### *2.4. The Multi-domain procedure*

In this section, a brief description about the multi-domain procedure is provided. For more details see Monteleone et al. (2018).

#### *2.4.1. Domain decomposition*

In the multi-domain approach proposed by Monteleone et al. (2018), the computational domain is partitioned into a set of confining blocks which are adjacent to each other, without overlapping regions. The surfaces of separation, named *block interfaces*, can be plane or curved and are discretized into triangles, in order to easily identify the interface normal directions.

Each block $B_n$ has a *smoothing length* $h_n$ which is maintained constant, thus allowing to employ the classical SPH formulation, whilst the *smoothing length* can vary from one block to another.

The total number of *effective particles* of the whole computational domain, $N_{e,tot}$, is the sum of the numbers $N_{e,B_n}$ of the *effective particles* of each block $B_n$

$$N_{e,tot} = \sum_{n=1}^{N_B} N_{e,B_n}$$

where $N_B$ is the total number of blocks.

Virtual particles, named *interface particles* ($IP$), are added at the *block interfaces* to match the solution between neighbouring blocks and to restore the continuity of the *support domains* truncated by the interfaces. Specifically, each *effective particle* belonging to the generic block $B_n$ which has distance shorter than the starting particle distance of the block, $\Delta x_n$, from one *block interface* generates a number of $m$ *interface particles* at distances $\Delta x_n, ..., m\Delta x_n$ in the direction normal to the interface. The number $m$ of *interface particles* to be generated is selected in order to reach the contour of the *support domain* of the generating *effective particle*. As a consequence, when employing the ratio $kh/\Delta x = 2$, only two $IPs$ are generated (thus $m = 2$). The $IPs$ generated by one block are contained in the domain of the neighbouring block.

The *block interfaces* are treated as inflow and/or outflow sections for the particles going through them: when an *effective particle* leaves its own subdomain crossing a *block interface*, it is simply deactivated and thus removed from further calculations (outflow); on the other hand, a specific procedure based on the search of void spaces is employed to release new particles when the *block interface* acts as an inflow (see Fig. 6 of Monteleone et al. (2018)).

*2.4.2. Solution matching*

Considering two blocks, indicated as block 1 (with *smoothing length* $h_1$) and block 2 (with $h_2$), the hydrodynamic variables of the $IP$ particles generated by block 1 are obtained through interpolation procedures starting from the *effective particles* of the neighbouring block 2 in which the $IP$ particles are contained. Specifically, the generic $f$ hydrodynamic variable for a $IP$ particle generated by block 1 is obtained using a Taylor series expansion around the closest *effective particle* of the neighbouring block 2. *Vice versa*, when considering the $IP$ generated by block 2, the Taylor series expansion is performed around the closest effective particle of the neighbouring block 1. Therefore, a system for each variable must be solved at each block interface, containing one interpolation equation for each $IP$ particle generated by the two neighbouring blocks through the interface. In the ISPH approach, the system must be solved for both the intermediate and corrected velocities. In these systems, the *effective particle* velocities are known and only the $IP$ are unknown. For vectorial variables such as velocity, the same coefficient matrix is considered for each component, while updating the equation *right-hand-side* only. The velocity system is solved after calculating the *effective particle* values.

In the case of the $\psi$ values, due to the *elliptic* nature of the Poisson's equations, the $IP$ interpolation equations need to be solved simultaneously with the PPEs. A unique equation system is thus solved obtained by associating the single PPE sub-systems of each block (made of $N_{e,Bn}$ equations, for a total value of $N_{e,tot}$ equations) and the $N_{IP,tot}$ interpolation equations of the whole set of existing interfaces.

As the coefficient matrices are sparse, only the non-null terms of the velocity and PPE systems are saved, using the Compressed Row Storage (CRS) format (Bisseling, 2004). The nonsymmetric linear systems of equations are solved through the iterative BiConjugate Gradient STABilized (BiCGSTAB) method (Van der Vorst, 1992), using a preconditioning algorithm (Saad, 2003).

The BiCGSTAB method for the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ is described in Algorithm 1. The method relies on matrix-vector multiplications in the initializing step (line 2 of Algorithm 1) and at each BiCGSTAB $j$ iteration (lines 11, 12, 15, 16 and 19). Moreover, several scalar products are performed during the iterative solution, to calculate the coefficients $\rho_j$ (line 8), $\alpha_j$ (line 13), $\omega_j$ (line 17) and the residual $RSQ$ (line 20). The cycle at line 7 is performed until the relative residual for the computed solution approximation $\mathbf{x}_j$ reduces below an upper bound tolerance ($RSQ < tol$), or the number of iterations exceeds an imposed maximum bound ($j > iter_{max}$). In lines 11 and 15, $\mathbf{K}$ is a preconditioning matrix obtained by using the *incomplete LU factorization* discussed in Saad (2003) ($\mathbf{K} = \mathbf{L}\,\mathbf{U}$, with $\mathbf{L}$ and $\mathbf{U}$ the lower and upper triangular matrices, respectively).

**Algorithm 1** Algorithmic description of the preconditioned BiCGSTAB

$\mathbf{x}_0 = \mathbf{0}$ or any other initial guess
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ initial residual
Choose $\mathbf{r}_0^*$ such that $(\mathbf{r}_0^*, \mathbf{r}_0) \neq 0$, *e.g.* $\mathbf{r}_0^* = \mathbf{r}_0$
$\rho_0 = \alpha_0 = \omega_0 = 1$
$\mathbf{v}_0 = \mathbf{p}_0 = \mathbf{0}$
$j = 1$
**while** $(RSQ > tol$ **and** $j < iter_{max})$ **do**
  $\rho_j = (\mathbf{r}_0^*, \mathbf{r}_{j-1})$
  $\beta = \left(\frac{\rho_j}{\rho_{j-1}}\right)\left(\frac{\alpha_{j-1}}{\omega_{j-1}}\right)$
  $\mathbf{p}_j = \mathbf{r}_{j-1} + \beta\,(\rho_{j-1} - \omega_{j-1}\,\mathbf{v}_{j-1})$
  $\mathbf{y} = \mathbf{K}^{-1}\mathbf{p}_j$ (through $\mathbf{y}' = \mathbf{L}^{-1}\mathbf{p}_j$ and $\mathbf{y} = \mathbf{U}^{-1}\mathbf{y}'$)
  $\mathbf{v}_j = \mathbf{A}\,\mathbf{y}$
  $\alpha_j = \frac{\rho_j}{(\mathbf{r}_0^*, \mathbf{v}_j)}$
  $\mathbf{s} = \mathbf{r}_{j-1} - \alpha_j \mathbf{v}_j$
  $\mathbf{z} = \mathbf{K}^{-1}\mathbf{s}$ (through $\mathbf{z}' = \mathbf{L}^{-1}\mathbf{s}$ and $\mathbf{z} = \mathbf{U}^{-1}\mathbf{z}'$)
  $\mathbf{t} = \mathbf{A}\,\mathbf{z}$
  $\omega_j = \frac{(\mathbf{t},\mathbf{s})}{(\mathbf{t},\mathbf{t})}$
  $\mathbf{x}_j = \mathbf{x}_{j-1} + \alpha_j\,\mathbf{y} + \omega_j\,\mathbf{z}$
  $\mathbf{res} = \mathbf{b} - \mathbf{A}\,\mathbf{x}_j$
  $RSQ = (\mathbf{res}, \mathbf{res})$
  $\mathbf{r}_j = \mathbf{s} - \omega_j\,\mathbf{t}$
  $j = j + 1$
**end while**

## 3. Parallelization scheme

### *3.1. Domain distribution*

In order to speed up the SPH simulations, in the parallel computation, the particles are distributed among the available processors. Since the particles move throughout the domain, the list of neighbor particles must be updated at each time step, using the Cartesian *virtual grid* discussed in §2.1.

The same *virtual grid* is used in the single-domain approach to suitably subdivide the domain among processors in the initialization step of parallel computation. Specifically, the grid cells are distributed among the $N_{procs}$ processors, to allot to each of them a number of *effective particles* close to the theoretical value $N_t = N_e/N_{procs}$. All the particles in one cell of the grid are assigned to the same processor. This distribution algorithm allows to limit the difference between the number of particles assigned to the *id* processor and the theoretical value of $N_t$ below the average number of particles inside one cell (in $3D$ approximations, when using the ratio $kh/\Delta x = 2$, $N_t$ is equal to 8 ).

In the multi-domain approach, the total number of particles $N_{e,tot}$ must be distributed among different processors. Due to the variable *smoothing length*, a different *virtual* grid is used for each block $B_n$, and then the grid cells of

the different blocks are distributed among the processors. As a result, each block can be entirely assigned to one processor, or shared between one or more of them. Therefore, the *block interfaces* can be internal to one processor, or divided between two or more processors. For clarity, the different cases which may occur are explicatively summarised in Fig. 1. In particular, four blocks (with their own $kh_n$) with three *block interfaces* (indicated as bold black lines) are considered for the description of the distribution among three processors (with indices between 0 and 2). In this case, processor 0 takes part of block 1, processor 1 takes the remaining part of block 1 and a portion of block 2, processor 2 takes the remaining fraction of block 2 and the whole blocks 3 and 4. Thus, the interface between blocks 1 and 2 coincides with the domain separation surfaces (*parallel interfaces*) between processors 0 and 1 (line $AB$ in the figure), processors 0 and 2 (line $BC$), processors 1 and 2 (line $CD$). Hence, this interface is shared between three different processors. The interface between blocks 2 and 3 partly coincides with the *parallel interface* between processors 1 and 2 (line $EF$) and partly is internal to processor 2 (line $FG$). The interface between blocks 3 and 4 is entirely contained in the domain of competence of processor 2.

Correspondingly, the *parallel interface* between processors 0 and 1 is partially internal to block 1 (line $CH$), while the remaining part ($AB$) coincides with the interface between blocks 1 and 2, as described above. A similar condition occurs for the *parallel interface* between processors 1 and 2, which is partially internal to block 2 (line $BF$) and partially separating blocks 1 and 2 (line $CD$) and blocks 2 and 3 (line $EF$).

Notice that, the domain of the generic processor $id$ can neighbor the domain of any other processor whose *parallel interface* coincides with the *block interface* of one block of the processor $id$ itself (*e.g.*, processor 0 neighbors to processor 2, through the *parallel interface* $BC$ separating blocks 1 and 2). On the other hand, when the *parallel interface* is internal to one block (*e.g.*, line $CH$), the domain of processor $id$ only neighbors that of $id-1$ or $id+1$ (in the considered case, the domain of processor 0 neighbors that of processor 1), which in the reminder of this paper will be indicated as *left* and *right* processors, respectively.

Since the *virtual* grid does not change during the simulations, the spatial domain assigned to each processor is fixed in time, although the particles inside them continuously change, switching from one processor to another. As it will be discussed in §3.3, specific procedures have been implemented to address the moving particles to the appropriate processor.

The procedure for balancing the load between processors is automatic and already implemented in the *PANORMUS* code.

### *3.2. Particle data sharing within one block*

When one block is shared between two or more processors (*e.g.* block 1 shared between processors 0 and 1 in Fig. 1), the communication of data relative to the particles neighbouring the processor *parallel interface* (line $CH$ in Fig. 1) must be handled. The same situation, obviously, occurs in parallel single-domain computation, where only one block exists, distributed among the

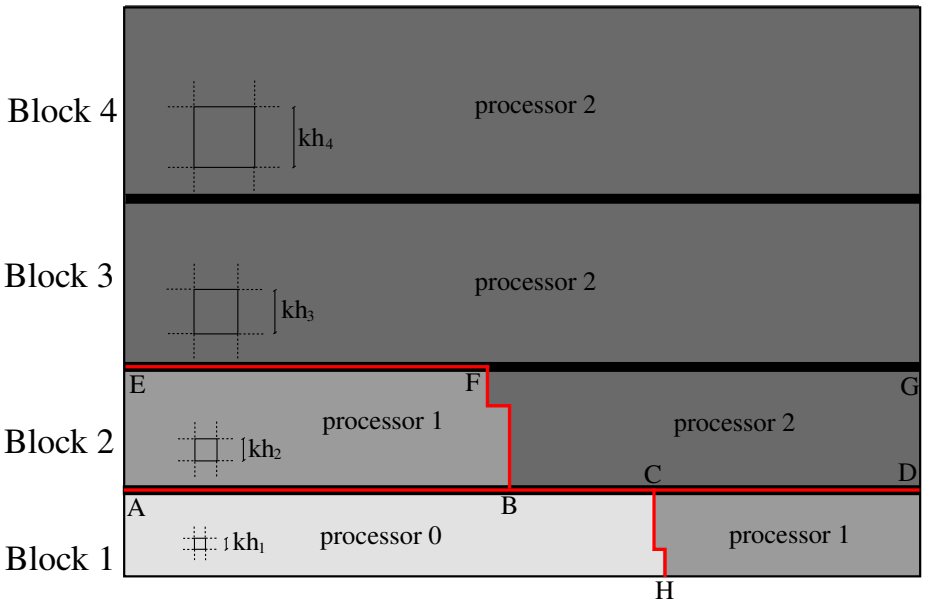



Figure 1: $N_B = 4$, $N_{procs} = 3$. Red lines: *parallel interfaces*; bold black lines: *block interfaces*. Light gray: domain of the processor 0; gray: domain of the processor 1; dark gray: domain of the processor 2.

available processors. Specifically, each MPI process needs to receive from the *left* and/or *right* processors the data of the particles falling within the interaction distance $kh_n$ from the *parallel interface*. In the example considered in Fig. 1, the block 1 with *smoothing length* $h_1$ is partitioned between processors 0 and 1, through the *parallel interface* $CH$ (this is entirely contained inside block 1). Therefore, processor 0 needs to receive the data of the particles belonging to the domain of processor 1 and having a distance less than $kh_1$ from the interface $CH$. Similarly, processor 1 needs to receive the data about the neighbouring particles in processor 0.

Fig. 2 shows an enlargement of the block 1 *virtual grid* (having cells of length $kh_1$) and the particle distribution of processor 0 near the $CH$ *parallel interface*. Referring to the case represented in the figure, processor 0 receives from the *right* processor (processor 1) the data relative to the particles inside a layer of cells neighbouring its own competence domain (red particles in Fig. 2), which are required to complete the *support domain* of the *effective particles* of processor 0, close to the *parallel interface* (*e.g.*, the particle $A$, whose *support domain* is highlighted in the figure). At the same time, processor 0 feeds processor 1 the data relative to the particles falling into the cells of its own domain, neighbouring to the *parallel interface* (full black circles in the figure). The data are thus shared between the processors 0 and 1 using the MPI function *MPI_SENDRECEIVE*.

At each time step, the required information relative to a *parallel interface* contained within one block to be shared between the processors, include:

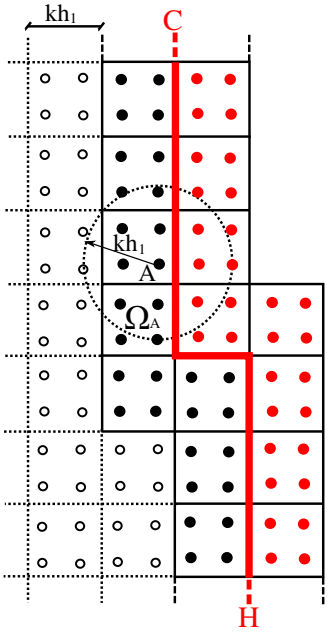


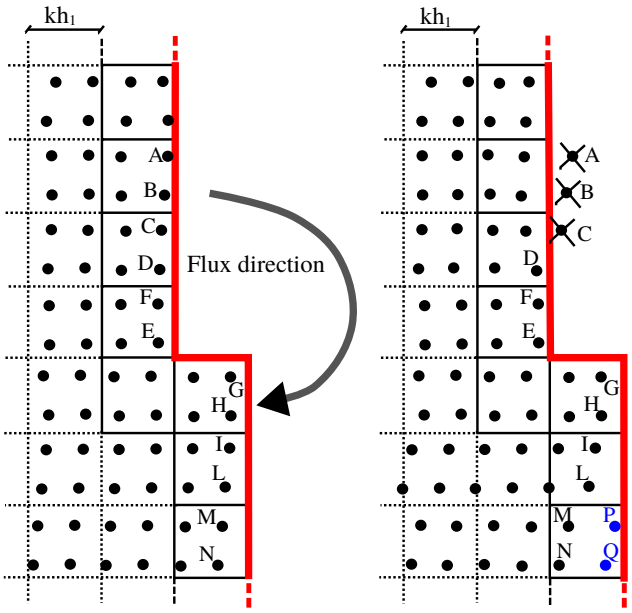

Figure 2: $2D$ Sketch of the domain competence of processor 0 in the vicinity of *parallel interface* $CH$. Bold red line: *parallel interface*; empty black circles: particles of processor 0 far away from the *parallel interface*; full black circles: particles of processor 0 to be sent to the *right* processor; full red circles: particles of processor 1, required by processor 0 to complete the *support domain* of its own particles; dotted circle: *support domain* of particle $A$.

- particle positions and velocities before the predictor-step;
- particle intermediate velocities after the predictor-step;
- *pseudo-pressure* $\psi$ values during the PPE system solution;
- particle corrected velocities after the corrector-step.

### *3.3. Management of the particle leaving/entering one processor domain*

As discussed above, due to the Lagrangian nature of the SPH method, particles transit through the processor domains. Thus, the leaving and entering of particles from and to the processor domain of competence have to be dealt with.

To this aim, at the end of each time step, the processors check if some particles have left their own domain by crossing the *parallel interfaces*. If the exit *parallel interface* is also a *block interface* (*e.g.*, line $AB$ in Fig. 1), the leaving particle is simply deactivated as in the serial case described in §2.4. Otherwise, if the particle leaves the processor domain remaining in the current block (when this is shared between two or more processors), the information relative to the particle (position, velocity, acceleration, *pseudo-pressure*, etc.) are fed to




(a) time $r$ (b) time $r+1$

Figure 3: $2D$ Sketch of the particle leaving/entering the domain competence of one processor. Bold red line: *parallel interface*; full black circles: particles belonging to the domain of competence of the processor; full blue circles: new particles received from the neighbor processor. The particles deactivated at the $(r+1)$-th time step are indicated with a cross.

the processor owning the cell occupied by the particle after its displacement. Then, the particle is deactivated from the list of those of the processor releasing the particle and added to the list of the receiving processor. This process is represented in Fig. 3, where a portion of the domain of competence of the generic processor *id*, in the proximity of the *parallel interface* (bold red line) internal to one block, is shown at time instants $r$ (Fig. 3a) and $r+1$ (Fig. 3b). The particles $A$, $B$ and $C$, which at time $r$ belong to the processor *id*, at $r+1$ cross the interface and enter the domain of the neighbouring processor, designated as $id+1$. Processor *id*, thus, deactivates these particles, after having sent their values to the processor $id+1$. Correspondingly, particles $P$ and $Q$ (blue circles in the figure), which enter the domain of processor *id* from processor $id+1$, are deactivated by processor $id+1$, after having been sent to *id*.

### *3.4. Parallel treatment of block interfaces*

The matching of the hydrodynamic variables, velocities and pressure fields, at the *block interfaces* (discussed in §2.4.2) requires specific attention in the parallelization procedure. The treatment of the velocities at the *block interfaces* is different from that of the pressure.

When considering velocities, in fact, the system of the interpolation equations is solved independently at each interface, while for the *pseudo-pressure* $\psi$

the interpolation equations for all *block interfaces* must be introduced in one single system, to be solved together with the PPEs.

For the velocities, thus, when one *block interface* is entirely within the competence domain of one processor (indicated as *case* 1 in the following), that processor can directly apply the procedure described in §2.4.2. An example of *case* 1 is the interface between blocks 3 and 4 in Fig. 1. On the other hand, when the *block interface* is (entirely or partially) shared between two or more processors, and therefore the *parallel interface* coincides with the *block interface* (*case* 2), the information required to solve the velocity field matching system must be exchanged among the involved processors.

For the *pseudo-pressure*, since all the *block interfaces* are treated simultaneously, *case* 1 never occurs and data exchanges between the processors are always necessary during the solution (*case* 2).

It is useful to highlight that when one *block interface* is also a *parallel interface*, in order to identify the data to be shared in *case* 2, the *interface particles* generated by one processor are handled by this even if they are included in the competence domain of the neighbouring processor. Nevertheless, the interpolation equations to obtain the variables relative to these particles can be written and solved only by the neighbouring processor, which contains the *effective particles* of the neighbouring block. The scheme shown in Fig. 4 is used to clarify this point, where blocks 1 and 2 are represented and the *interface particle* $P$ is highlighted. For the sake of representation, only four layers of effective particles are represented for block 1 (small black circles) and three layers of effective particle are represented for block 2 (large black circles). As discussed in §2.4.1, the number of layers of virtual particles (indicated as large and small empty circles for block 1 and 2, respectively) account for the truncation of the support domain at the interface and, for the selected values of $k$, $h$ and $\Delta x$, is equal to 2.

Specifically, the *interface particle* $P$ lies in the *right* processor, although it was generated by the *effective particle* $A$ of block 1 of the *left* processor. Obviously, the $P$ particle is surrounded by particles of block 2. These surrounding particles, due to the coincidence between the *block interface* and the *parallel interface*, are *effective particles* of the *right* processor domain (large full circles in Fig. 4) and *interface particles* lying in the *left* processor domain (large empty circles). The interpolation equation for $P$ is written by the *right* processor, which is able to identify the closest *effective particle* (indicated as $R$ in Fig. 4) of block 2 and to perform the summation relative to the surrounding *effective particles* (particles inside the dashed circles with radius $kh_2$) in the Taylor series expansion. The *right* processor, nevertheless, must receive from the *left* processor the data relative to the $P$ particle position (which depends on the position of the generating particle $A$) and to the velocities of the *interface particles* generated by block 2 (belonging to the *left* processor domain and indicated with large empty gray circles in Fig. 4). Simultaneously, the *left* processor receives from the *right* one the corresponding data.

Since each processor could, in principle, be sharing *block interfaces* with all the other processors, the data indicated before are scattered to all processors

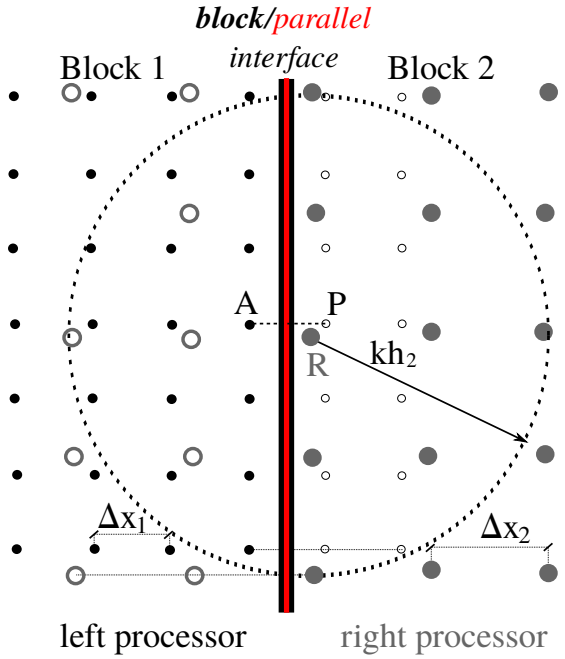



Figure 4: $2D$ Sketch of neighbouring blocks distributed between two processors. Bold black line: *block interface*; red lines: *parallel interface*; smaller full and empty circles: *effective* and *interface particles* of block 1; larger full and empty circles: *effective* and *interface particles* of block 2; black circles: particles belonging to the domain of competence of the *left* processor; gray circles: particles belonging to the domain of competence of the *right* processor.

using the function *MPI_ALLTOALLV*, which allows a varying count of data from each process. As a consequence, the data part of the message referring to processor pairs not sharing any *block interface* would remain empty.

The velocity system for each *block interface*, as in the serial case, is made of a number of equations equal to the total number of *interface particles* generated by the blocks that are separated by the interface. These equations are distributed among the processors sharing the interface. Each of these processors, in fact, after having received the above information thorough the *MPI_ALLTOALLV* function, is able to write $n_1$ equations relative to the velocities of the *interface particles* generated inside its competence domain (*e.g.* particle $P$ Fig. 4). In these equations, the unknowns are $n_1$ velocity values of the *interface particles* for which the equations are written, plus the $N_{IP}$ velocities of the *interface particles* generated by *effective particles* lying in the processor competence domain. The system coefficient matrix is thus distributed among the processors, each of which containing $n_1$ rows relative to the *interface particles* generated inside its competence domain. The length of the vector of unknowns $\mathbf{x}$ in each processor is equal to $n_2$, which is the sum of $n_1$ and $N_{IP}$.

The system is solved through the parallelized BiCGSTAB Algorithm 2, requiring, during the iterative solution, several data exchanges among the proces-

**Algorithm 2** Algorithmic description of the Parallel BiCGSTAB

1: call $MPI\_ALLTOALLV$ : Send $\mathbf{x}_{0\,(1:n_1)}$ / Receive $\mathbf{x}_{0\,(n_1+1:n_2)}$
2: $\mathbf{r}_{0\,(1:n_1)} = \mathbf{b}_{(1:n_1)} - \mathbf{A}\,\mathbf{x}_{0(1:n_2)}$
3: Choose $\mathbf{r}_0^*$ such that $(\mathbf{r}_0^*, \mathbf{r}_0) \neq \mathbf{0}$, *e.g.* $\mathbf{r}_0^* = \mathbf{r}_0$
4: $\rho_0 = \alpha_0 = \omega_0 = 1$
5: $\mathbf{v}_0 = \mathbf{p}_0 = \mathbf{0}$
6: $j = 1$
7: **while** $(RSQ > tol$ **and** $j < iter_{max})$ **do**
8: $\quad \rho_{j,proc} = (\mathbf{r}^*_{0\,(1:n_1)}, \mathbf{r}_{j-1\,(1:n_1)})$
9: $\quad$ call $MPI\_ALLREDUCE$ : $\rho_j = \sum_{proc=1}^{N_{procs}} \rho_{j,proc}$
10: $\quad \beta = \left(\frac{\rho_j}{\rho_{j-1}}\right)\left(\frac{\alpha_{j-1}}{\omega_{j-1}}\right)$
11: $\quad \mathbf{p}_{j\,(1:n_1)} = \mathbf{r}_{j-1\,(1:n_1)} + \beta\left[(\rho_{j-1} - \omega_{j-1}\,\mathbf{v}_{j-1\,(1:n_1)}\right]$
12: $\quad \mathbf{y}_{(1:n_1)} = \mathbf{K}^{-1}\mathbf{p}_{j\,(1:n_1)}$ (through $\mathbf{y}' = \mathbf{L}^{-1}\mathbf{p}_j$ and $\mathbf{y} = \mathbf{U}^{-1}\mathbf{y}'$)
13: $\quad$ call $MPI\_ALLTOALLV$ : Send $\mathbf{y}_{(1:n_1)}$ / Receive $\mathbf{y}_{(n_1+1:n_2)}$
14: $\quad \mathbf{v}_{j\,(1:n_1)} = \mathbf{A}\,\mathbf{y}_{(1:n_2)}$
15: $\quad \alpha^*_{j,proc} = (\mathbf{r}^*_{0\,(1:n_1)}, \mathbf{v}_{j\,(1:n_1)})$
16: $\quad$ call $MPI\_ALLREDUCE$ : $\alpha^*_j = \sum_{proc=1}^{N_{procs}} \alpha^*_{j,proc}$
17: $\quad \alpha_j = \frac{\rho_j}{(\alpha^*_j)}$
18: $\quad \mathbf{s}_{(1:n_1)} = \mathbf{r}_{j-1\,(1:n_1)} - \alpha_j \mathbf{v}_{j\,(1:n_1)}$
19: $\quad \mathbf{z}_{(1:n_1)} = \mathbf{K}^{-1}\mathbf{s}_{(1:n_1)}$ (solving $\mathbf{z}' = \mathbf{L}^{-1}\mathbf{s}$ and $\mathbf{z} = \mathbf{U}^{-1}\mathbf{z}'$)
20: $\quad$ call $MPI\_ALLTOALLV$ : Send $\mathbf{z}_{(1:n_1)}$ / Receive $\mathbf{z}_{(n_1+1:n_2)}$
21: $\quad \mathbf{t}_{(1:n_1)} = \mathbf{A}\,\mathbf{z}_{(1:n_2)}$
22: $\quad \omega^*_{j,proc} = (\mathbf{t}_{(1:n_1)}, \mathbf{s}_{(1:n_1)})$
23: $\quad$ call $MPI\_ALLREDUCE$ : $\omega^*_j = \sum_{proc=1}^{N_{procs}} \omega^*_{j,proc}$
24: $\quad \omega^{**}_{j,proc} = (\mathbf{t}_{(1:n_1)}, \mathbf{t}_{(1:n_1)})$
25: $\quad$ call $MPI\_ALLREDUCE$ : $\omega^{**}_j = \sum_{proc=1}^{N_{procs}} \omega^{**}_{j,proc}$
26: $\quad \omega_j = \frac{\omega^*_j}{\omega^{**}_j}$
27: $\quad \mathbf{x}_{j\,(1:n_2)} = \mathbf{x}_{j-1\,(1:n_2)} + \alpha_j\,\mathbf{y}_{(1:n_2)} + \omega_j\,\mathbf{z}_{(1:n_2)}$
28: $\quad \mathbf{res}_{(1:n_1)} = \mathbf{b}_{(1:n_1)} - \mathbf{A}\,\mathbf{x}_{j\,(1:n_2)}$
29: $\quad RSQ_{proc} = (\mathbf{res}_{(1:n_1)}, \mathbf{res}_{(1:n_1)})$
30: $\quad$ call $MPI\_ALLREDUCE$ : $RSQ = \sum_{proc=1}^{N_{procs}} RSQ_{proc}$
31: $\quad$ call $MPI\_ALLTOALLV$ : Send $\mathbf{x}_{(1:n_1)}$ / Receive $\mathbf{x}_{(n_1+1:n_2)}$
32: $\quad \mathbf{r}_{j\,(1:n_1)} = \mathbf{s}_{(1:n_1)} - \omega_j\,\mathbf{t}_{(1:n_1)}$
33: $\quad j = j + 1$
34: **end while**

sors sharing the *block interface*. Specifically, the matrix-vector multiplications at lines 2, 14 and 21 of Algorithm 2 are performed after having received the values relative to the *interface particles* generated by the *effective particles* belonging to the domain of the processor. To this aim, the function $MPI\_ALLTOALLV$ is employed (lines 1, 13 and 20 of Algorithm 2), allowing each processor to store

these information in the corresponding vectors ($\mathbf{x}_0$, $\mathbf{y}$ and $\mathbf{z}$ for lines 1, 13 and 20, respectively) from the position $n_1 + 1$ up to $n_2$. On the other hand, the values of the *interface particles* for which the equations are written (data stored in the vectors from the position 1 up to $n_1$) must be sent to the processors whose *effective particles* have generated them. In order to calculate the coefficients $\rho$, $\alpha$, $\omega$ and the residual $RSQ$, each processor firstly performs the scalar products relative to its portion of the system (lines 8 for $\rho$, 15 for $\alpha$, 22 and 24 for $\omega$ and 29 for $RSQ$). The overall values are, consequently, obtained after having summed the results of the single scalar products performed by all the involved processors, using the function $MPI\_ALLREDUCE$ (lines 9, 16, 23, 25 and 30). After solving the system, each processor sends the obtained velocities of the *interface particles* to the processors from which these particles have been generated.

As discussed before, the solution of the PPEs is partially different. In order to obtain the *pseudo-pressure* $\psi$ values, in fact, each processor writes a portion of the unique global system made of the PPEs relative to its *effective particles* plus the interpolation equations for the *interface particles* lying into its competence domain. In this case, differently from the velocity system, the *interface particles* generated through any *block interface* in the processor domain are considered. In these equations the unknowns are the $\psi$ values of: *a*) the *effective* and *inter-*

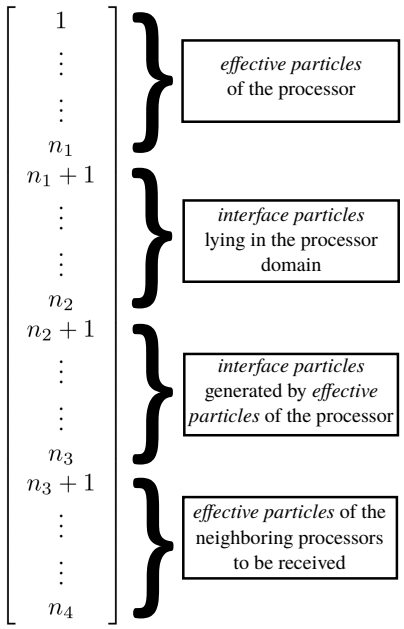



Figure 5: Sketch of the storing management of the variables in the vectors employed in the BiCGSTAB algorithm.

*face particles* in the processor domain for which the equations are written; *b*) the *interface particles* generated by the *effective particles* of the processor and belonging to other processor domains; *c*) the *effective particles* received from the *right* and *left* processor through *parallel interfaces* not coinciding with *block interfaces*. The (*b*) terms are required to complete the *support domain* for the PPEs and interpolation equations of the particles of term (*a*), while the (*c*) terms enter in the PPEs of the *effective particles* of the processor close to the *parallel interfaces*, as explained in §3.2. A sketch of the storing management of the variables in the vectors employed in the BiCGSTAB algorithm is shown in Fig. 5.

As for solution of the velocity system, the parallel BiCGSTAB algorithm employed to solve the PPE system is made of matrix-vector multiplications which are performed after having exchanged some information. Specifically, each processor employs the function *MPI_ALLTOALLV* to send the calculated values of the *interface particles* lying into its domain of competence to the processors owning the generating *effective particles*. Simultaneously, the processor receives from other processors the values of the *interface particles* generated by its *effective particles*. Moreover, employing the function *MPI_SENDRECEIVE*, each processor sends and receives to/from the *left* and *right* processors the values of the *effective particles* close to the *parallel interfaces*. As discussed for the velocity system, the results of the scalar products performed by each processor are then summed to the corresponding values calculated by the other processors to obtain the parameters $\rho$, $\alpha$ and $\omega$ of the global system.

### *3.5. Flow chart of the parallelized code*

An overall summary of the proposed parallelization scheme is shown in the flow chart of Fig. 6. The actions, which are performed by each processor with reference to the particles in the relative competence domain, are briefly explained as follows:

- **Action** 1: The domain is decomposed into non-overlapping blocks (§2.4.1);
- **Action** 2: The domain is subdivided among the available processors and the corresponding particles are distributed accordingly (§3.1);
- **Action** 3: The *mirror* and *interface particles* are generated (see §2.2 and §2.4.1, respectively);
- **Action** 4: The positions of the *interface particles* are shared among the involved processors (§3.4);
- **Action** 5: The positions and velocities of the *effective particles* close to the *parallel interfaces* are sent to the neighbor (*right* and *left*) processors (see §3.2);
- **Action** 6: The list of neighbor particles in the *support domain* of each *effective particle* is built;

- **Action 7**: The predictor-step is performed to calculate the *intermediate* velocities;
- **Action 8**: The system of interpolation equations is solved at each *block interface* by the involved processors employing the parallelized BiCGSTAB method (Algorithm 2) to obtain the *intermediate* velocities of the *interface particles*;
- **Action 9**: The *intermediate* velocities of the *effective particles* close to the *parallel interfaces* are sent to the neighbor processors (similar to Action 5);
- **Action 10**: A global system is solved to obtain the *pseudo-pressure* val-

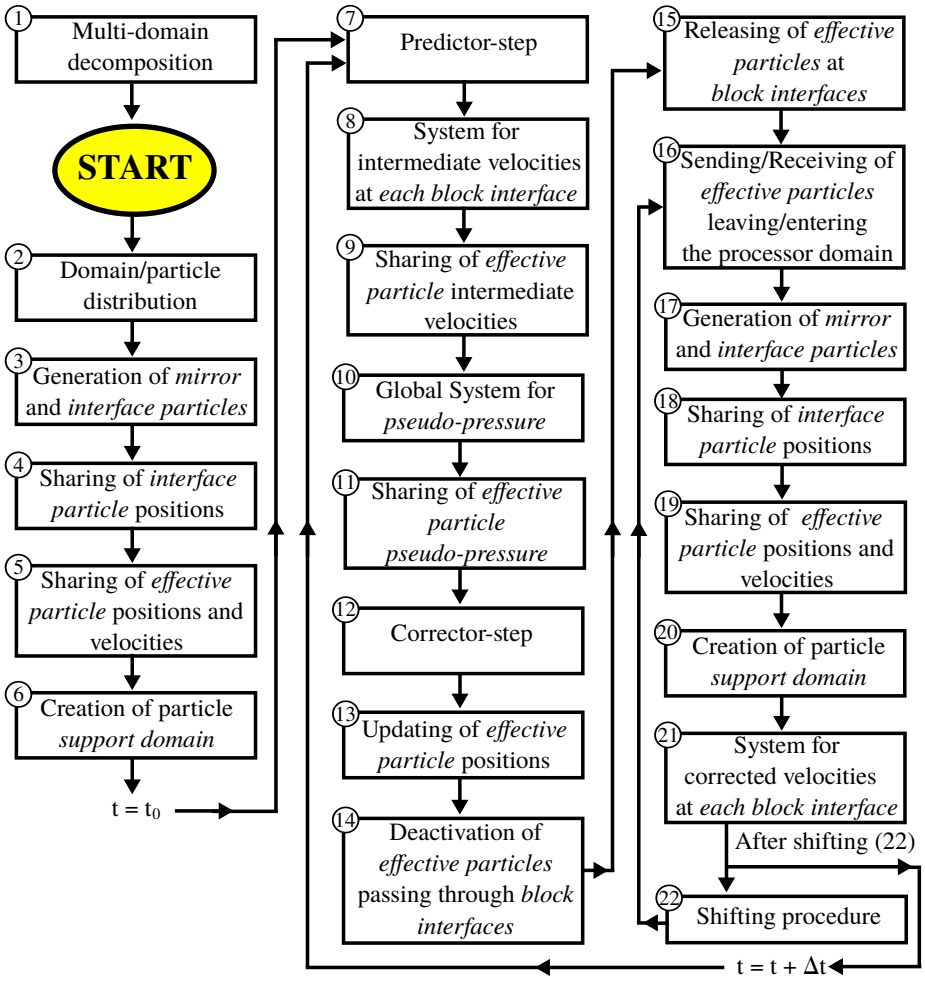



Figure 6: Flow chart of the parallelized code.

ues. For each processor the number of equations is equal to the sum of the number of *effective* and *interface particles* lying into its competence domain;

- **Action** 11: The *pseudo-pressure* values of the *effective particles* are shared with the *right* and *left* processors;
- **Action** 12: The corrector-step is performed to obtain the updated velocities;
- **Action** 13: The positions of the *effective particles* are updated;
- **Action** 14: The *effective particles* leaving the processor competence domain through *block interfaces* are deactivated;
- **Action** 15: At the *block interfaces* new *effective particles* are generated whenever required to maintain mass conservation;
- **Action** 16: The variables (position, velocity, *pseudo-pressure*, etc.) relative to the *effective particles* entering or leaving the domain of competence of each processor through *parallel interfaces* are shared;
- **Action** 17: As in action 3, the *mirror* and *interface particles* are generated;
- **Action** 18: As in action 4, the updated positions of the *interface particles* are shared among the involved processors;
- **Action** 19: As in action 5, the updated positions and velocities of the *effective particles* are shared;
- **Action** 20: As in action 6, the list of neighbor particles in the *support domain* of each *effective particle* is built;
- **Action** 21: Action 8 is repeated with reference to the corrected velocities of the *interface particles*;
- **Action** 22: The shifting procedure of Xu et al. (2009) is performed.

After shifting the *effective particle* positions, the actions from 16 to 21 are repeated. The time-marching procedure is then restarted from the predictor-step (action 7).

## 4. Performance analysis and results

All the presented simulations were carried out on an AMD EPYC 7402 – 2.8 GHz processor with 2 sockets and 24 cores per sockets. Therefore, timing analysis was performed up to 48 cores.

Moreover, Open MPI Fortran 90 wrapper compiler (*mpif90*) was used to compile the code.

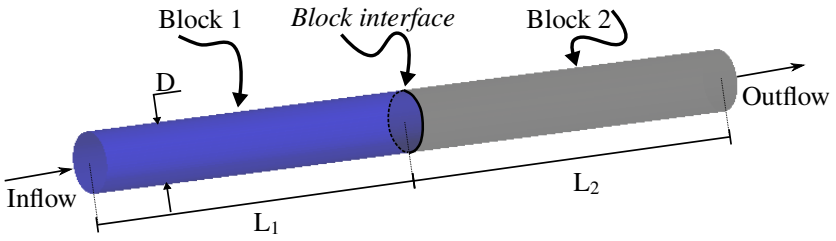



Figure 7: Test case 1. Computational domain. $L_1 = L_2 = 6 \cdot 10^{-3}$ $m$, $D = 1 \cdot 10^{-3}$ $m$; $h_1 = h_2 = 2.5 \cdot 10^{-5}$ $m$; $N_1 = N_2 = 303\,360$.

### *4.1. Test case* 1

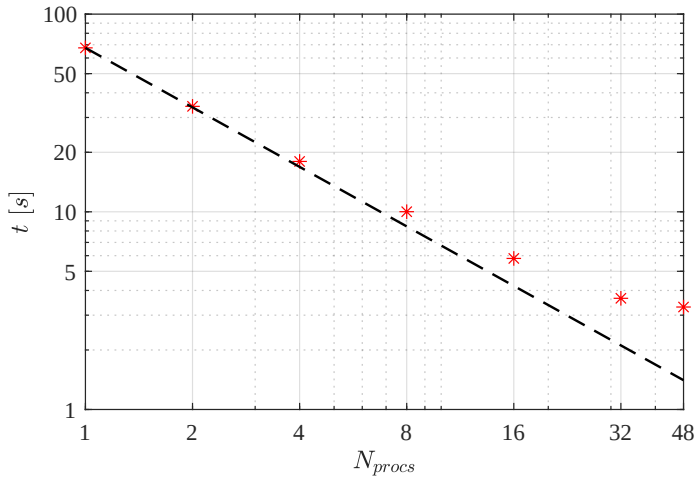
The implemented parallel computing scheme was validated considering the simple test case of steady-state flow in a circular pipe. The pipe, having diameter $D = 1 \cdot 10^{-3}$ m and total length $L = 12 \cdot 10^{-3}$ m, was partitioned into 2 identical blocks, as shown in Fig. 7. While the multi-domain procedure is based on the employment of different values of the *smoothing length*, in this test case, the same value was used ($h_1 = h_2 = 2.5 \cdot 10^{-5}$ m, corresponding to 20 particles along the pipe radius) in order to make the scalability test independent on the refinement level. The total number of *effective* and *interface particles* was equal to $N_{e,tot} = 606\,720$ and $5\,625$, respectively.

A stationary flow rate of $3.93 \cdot 10^{-7}$ m$^3$/s, corresponding to a mean velocity of 0.5 m/s, was imposed at the inflow section of the first block, with Poiseuille velocity profile. The resulting Reynolds number was $Re = 500$ ($\nu = 1.0 \cdot 10^{-6}$ m$^2$/s) which is well within the laminar regime. Dirichlet and Neumann boundary conditions for the pressure and velocity, respectively, were imposed at the outflow section of the second block. Specifically, the pressure was set to zero and null value was assigned to the velocity derivative as well. The multi-domain interpolation procedure was applied at the *block interface*, which corresponds to the outflow and inflow sections for the first and second blocks, respectively.

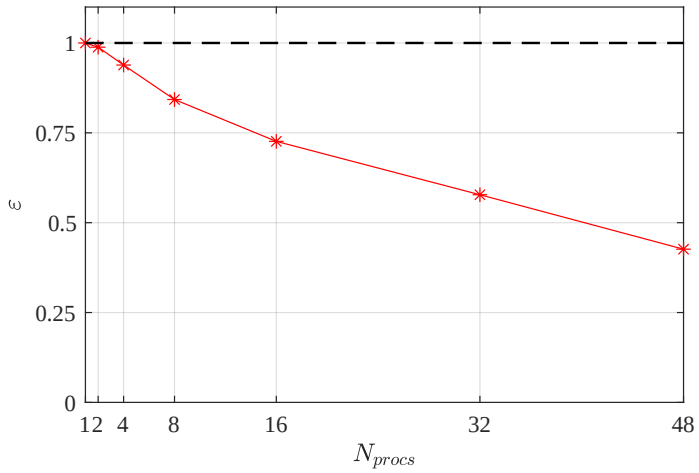
The computational time spent to advance the solution in time (actions from 7 to 22 of the flow chart represented in Fig. 6) was quantified employing different numbers of processors, from $N_{procs} = 1$ (serial mode) to the available number of processors ($N_{procs} = 48$). The results are shown in Fig. 8a, where the ideal line corresponding to a linear scalability (dashed black line with $-1$ slope) is plotted too, for comparison. As it can be observed, CPU wall-clock time (red stars) is close to the ideal time (dashed black line) up to 8 processors. Further increase in the number of processors determines a rise in the relative weight of the communication time among processors, resulting in a loss of parallel efficiency $\varepsilon$, calculated as

$$\varepsilon = \frac{1}{N_{procs}} \frac{T_s}{T_p} \tag{3}$$

where $T_s$ is the CPU wall-clock time of the code in serial mode, and $T_p$ is the CPU wall-clock time of the code in parallel mode executed by the number of




(a)




(b)

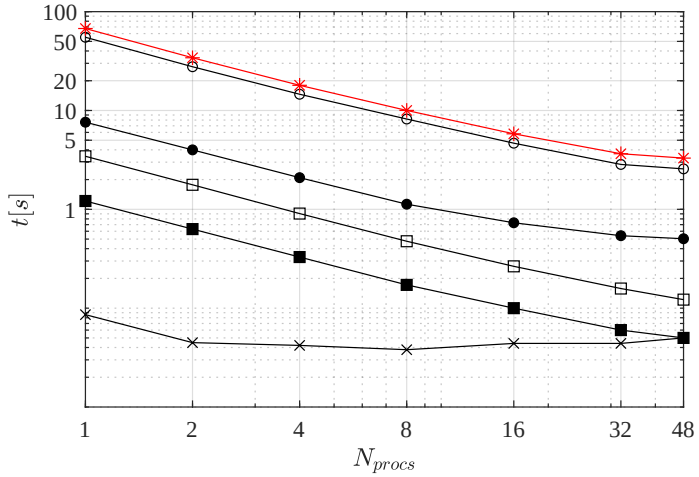
Figure 8: Test case 1. a) Scalability test: time versus $N_{procs}$. Dashed black line: ideal scalability; red stars: CPU wall-clock time for each $N_{procs}$ value. Double logarithmic scale; b) Parallel efficiency ($\epsilon$) versus $N_{procs}$. Dashed black line: ideal parallel efficiency.

processors $N_{procs}$. The diagram of in Fig. 8b indicates a progressive decay in the parallel efficiency, that for 48 processors reduces to 43%.

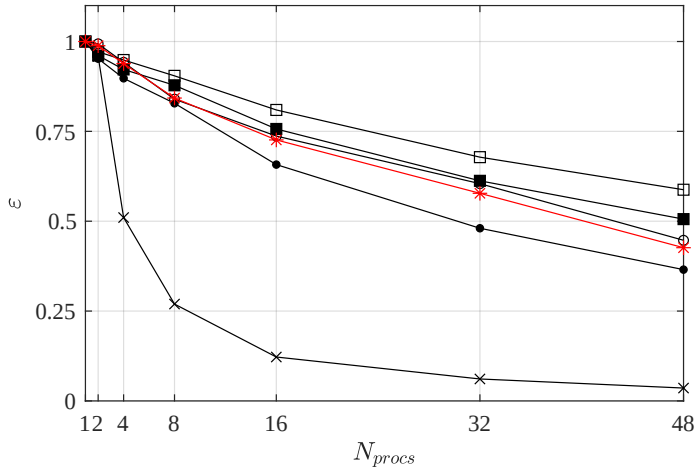
A detailed evaluation of the CPU wall-clock time is provided in Fig. 9a where sub-steps internal to one time step executed by the code are analysed individually. The greatest computational weight is associated with the solution of the Pressure Poisson Equation (empty circles), which contains a number of equations equal to the sum of the *effective* ($N_{e,tot}$) and *interface particles* ($N_{IP,tot}$). This is followed by the step including neighbouring search, release/deactivate particles, sharing *effective* and *interface particle* positions/velocities and shifting procedure (full circles), the corrector step and particle positions update (empty squares), the predictor step (full squares) and, finally, the velocity systems for the *interface particles* (crosses).

It is interesting to highlight that, in this simple test case, the *block interface* corresponds to a *parallel interface* (whatever the number of employed processors). Hence, only two processors solve the interface equations, in addition to the Navier-Stokes equations relative to the effective particles. This is evident in Fig. 9b where the time spent to solve the interpolation for the interface particle velocities (cross markers) is halved going from 1 to 2 processors and it remains constant when the number of processors growth. For example, for $N_{procs} = 4$, each processor is assigned $606\,720/4 = 151680$ particles and, in addition, the second and third processors are also assigned 2 528 additional *interface particles*. On the other hand, for $N_{procs} = 48$, the number of *effective particles* of each processor reduces to $606\,720/48 = 12\,640$, while the number of interface particles (assigned to processors 23 and 24 which are neighbors to the *block/parallel interface*) is again equal to 2 528, with a clear increase of the relative weight. Obviously, the uneven distribution of *interface particles* negative effects both the time for the solution of the systems for the velocities and *pseudo-pressure* and for the generation and the sharing of the value of the *interface particles*. This effect is partially responsible for the efficiency reduction observed in Fig. 8, which is also caused by the obvious increase in the relative weight of the communication time among processors. Fig. 9b shows the parallel efficiency of each step. Confirming the above, $\varepsilon$ tends to zero when considering the velocity systems for the *interface particles*, while the global efficiency (red stars) is heavily dependent on the global solution for *pseudo-pressure* (empty circles).

Further, an analysis of the parallel efficiency with the single-domain approach (Napoli et al., 2015) was performed and compared to that of the multi-domain approach. The results, shown in Fig. 10, confirm that in the single-domain (blue starts in Fig. 10) the parallel efficiency is higher than in the multi-domain approach (red starts in Fig. 10). In fact, when only one domain is considered no extra equations must be solved for the matching of the results at the *block-interfaces* and the computational loads are uniformly distributed among processors. On the contrary, in the multi-domain scheme, the load unbalance resulting to the uneven distribution of the *interface particles* reduces the parallel efficiency. However, in both cases, the parallel efficiency is affected by the increase in the relative weight of the communication time among processors occurring as the number of processors increases.




(a)




(b)

Figure 9: Test case 1. Full squares: predictor step (action 7 of Fig. 6); crosses: system for intermediate and corrected velocities at each *block interface* (actions 8 and 21); empty circles: global system for *pseudo-pressure* (actions 10 and 11); empty squares: corrector step and particle positions update (actions 12 and 13); full circles: creation of *support domain*, release/deactivate particles and sharing *effective/interface* particle information (actions $14-20$) and shifting procedure (action 22); red stars: total time. a) Time versus $N_{procs}$; b) Parallel efficiency versus $N_{procs}$.

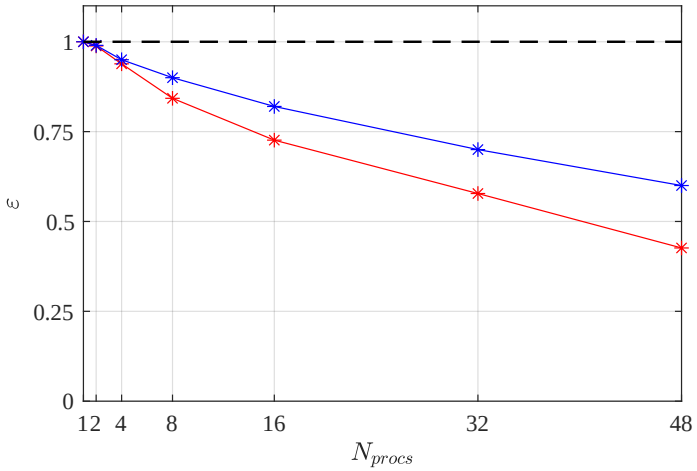



Figure 10: Test case 1. Parallel efficiency ($\epsilon$) versus $N_{procs}$. Dashed black line: ideal parallel efficiency; red stars: multi-domain approach; blue stars: single-domain approach.

Velocity profiles did match the theoretical predictions.

### *4.2. Test case* 2

In order to investigate the performance of the implemented parallel scheme with the subdivision of the domains into blocks, the pipe of *Test case 1* (having diameter $D = 1 \cdot 10^{-3}$ $m$ and total length $L = 12 \cdot 10^{-3}$ $m$) was considered again. As shown in Fig. 11, here the pipe was partitioned in two coaxial blocks: block 1 is a cylinder with diameter $D_1 = 0.6 \cdot 10^{-3}$ $m$, block 2 is a hollow cylinder with external and internal diameters equal to $D$ and $D_1$, respectively. Therefore, the *block interface* coincides with the external lateral surface of block 1 and the internal lateral surface of block 2. The *smoothing lengths* imposed at the two blocks are equal to that of §4.1 ($h_1 = h_2 = 2.5 \cdot 10^{-5}$ $m$) and thus the total number of *effective particles* remains unchanged with respect to *Test case 1* ($N_{e,tot} = 606\,720$). The total number of *interface particles* are equal to $N_{IP,tot} = 144\,960$ corresponding to 24 % of the *effective particles*, fraction much higher than that of *Test case* 2 where $N_{IP,tot}/N_{e,tot} = 0.4$ %.

Boundary conditions and fluid properties are the same of §4.1.

Fig. 12a shows the scalability test. Although the number of particles is the same of *Test case* 1, here the time spent is higher due to the larger ratio *interface*/*effective* particles. However, the global parallel efficiency shown in Fig. 8b is quite similar to that of *Test case* 1 (Fig. 8b). Considering the maximum number of processors employed ($N_{procs} = 48$), $\varepsilon = 46\%$ for the present test case and $\varepsilon = 43\%$ for *Test case* 1. In fact, due to the different domain subdivision, the *interface particles*, albeit in greater numbers, are distributed between all the

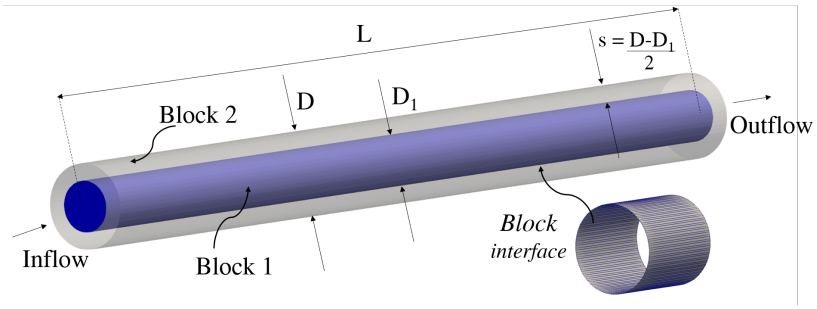


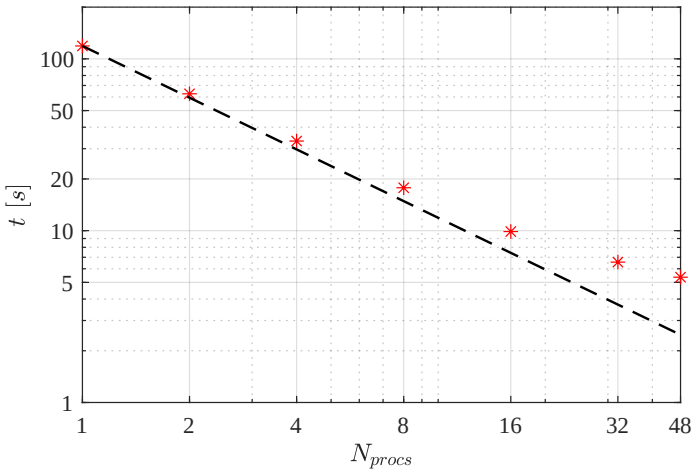

Figure 11: Test case 2. Computational domain. $L = 12 \cdot 10^{-3}\ m$; $D = 1 \cdot 10^{-3}\ m$; $D_1 = 0.6 \cdot 10^{-3}\ m$; $s = 0.2 \cdot 10^{-3}\ m$; $h_1 = h_2 = 2.5 \cdot 10^{-5}\ m$; $N_1 = 391\,680$; $N_2 = 215\,040$; $N_{e,tot} = 606\,720$.

processors. For example, employing $N_{procs} = 48$, all the processors have 12 640 *effective particles*, the least loaded has 2 400 *interface particles*, whilst the most loaded processor has 4 400 *interface particles*. Therefore, the non-uniformity in loads is similar to that found in *Test case* 1 where only 2 processors handle the *interface particles*.

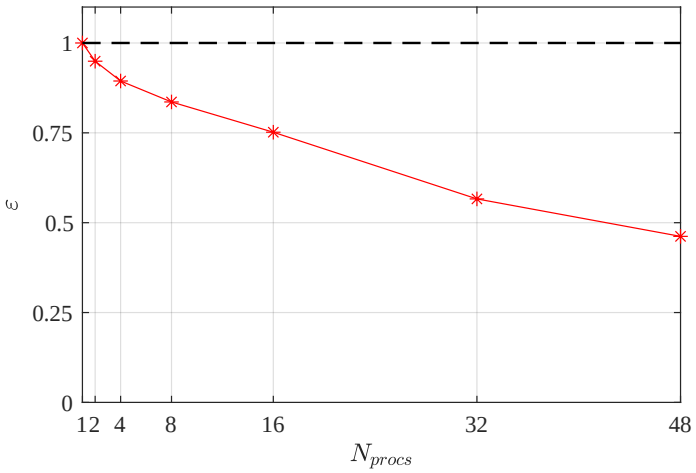
A detailed analysis of the time spent for each step of the code is shown in Fig. 13. Differently from *Test case* 1, the time spent to solve the velocity systems for the *interface particles* (cross markers in the figure) is not the lowest (in fact it is higher than the time associated to the predictor step indicated with full squares). Moreover, this time has a similar trend to that associated with the other steps (on the contrary of *Test case* 1 where this time remained unchanged with the increase of $N_{procs}$, see Fig. 9).

### *4.3. Test case* 3

A more complex test case, already presented in Monteleone et al. (2018), was considered to show the performance of the proposed multi-domain parallel computing scheme in a more relevant context. The domain, representing a system of cerebral vessels with a giant aneurysm, was subdivided into 6 blocks (see Fig. 14). The largest *smoothing length* of $h_4 = 1.25 \cdot 10^{-4}$ m was imposed for the aneurysm sac (block 4), which is 5 times larger than the minimum value used for the smallest branch (block 6, with $h_6 = 0.25 \cdot 10^{-4}$ m). At the inlet section $A$, a pulsatile flow rate with Womersley velocity profile (Womersley, 1955) was imposed. A constant, uniformly distributed, pressure was set at the three outflow sections ($B$, $C$ and $D$ in Fig. 14), and the multi-domain procedure was applied at the five *block interfaces*. The employment of the multi-domain procedure with the values of $h$ indicated above allows to solve the problem using 902 576 *effective particles*. A significant load reduction is thus obtained with respect to the use of a single-domain approach with the smaller value of $h = 0.25 \cdot 10^{-4}$ m, which would have resulted in a number of particles about 50 times larger.

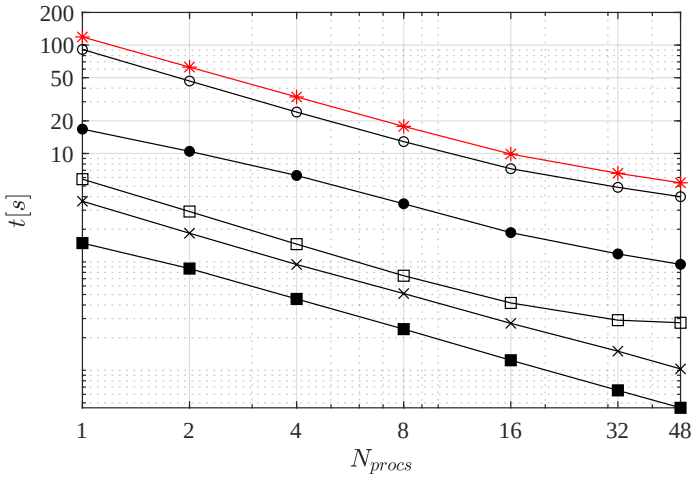



(a)




(b)

Figure 12: Test case 2. a) Scalability test: time versus $N_{procs}$. Dashed black line: ideal scalability; red stars: CPU wall-clock time for each $N_{procs}$ value. Double logarithmic scale; b) Parallel efficiency ($\epsilon$) versus $N_{procs}$. Dashed black line: ideal parallel efficiency.




(a)

Figure 13: Test case 2. Time versus $N_{procs}$. Full squares: predictor step (action 7 of Fig. 6); crosses: system for intermediate and corrected velocities at each *block interface* (actions 8 and 21); empty circles: global system for *pseudo-pressure* (actions 10 and 11); empty squares: corrector step and update particle positions (actions 12 and 13); full circles: creation of *support domain*, release/deactivate particles and sharing *effective/interface* particle information (actions $14-20$) and shifting procedure (action 22); red stars: total time.

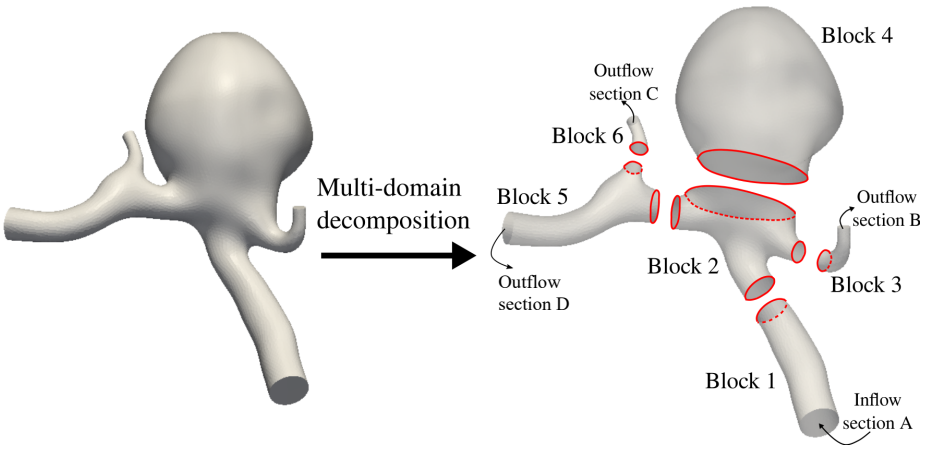
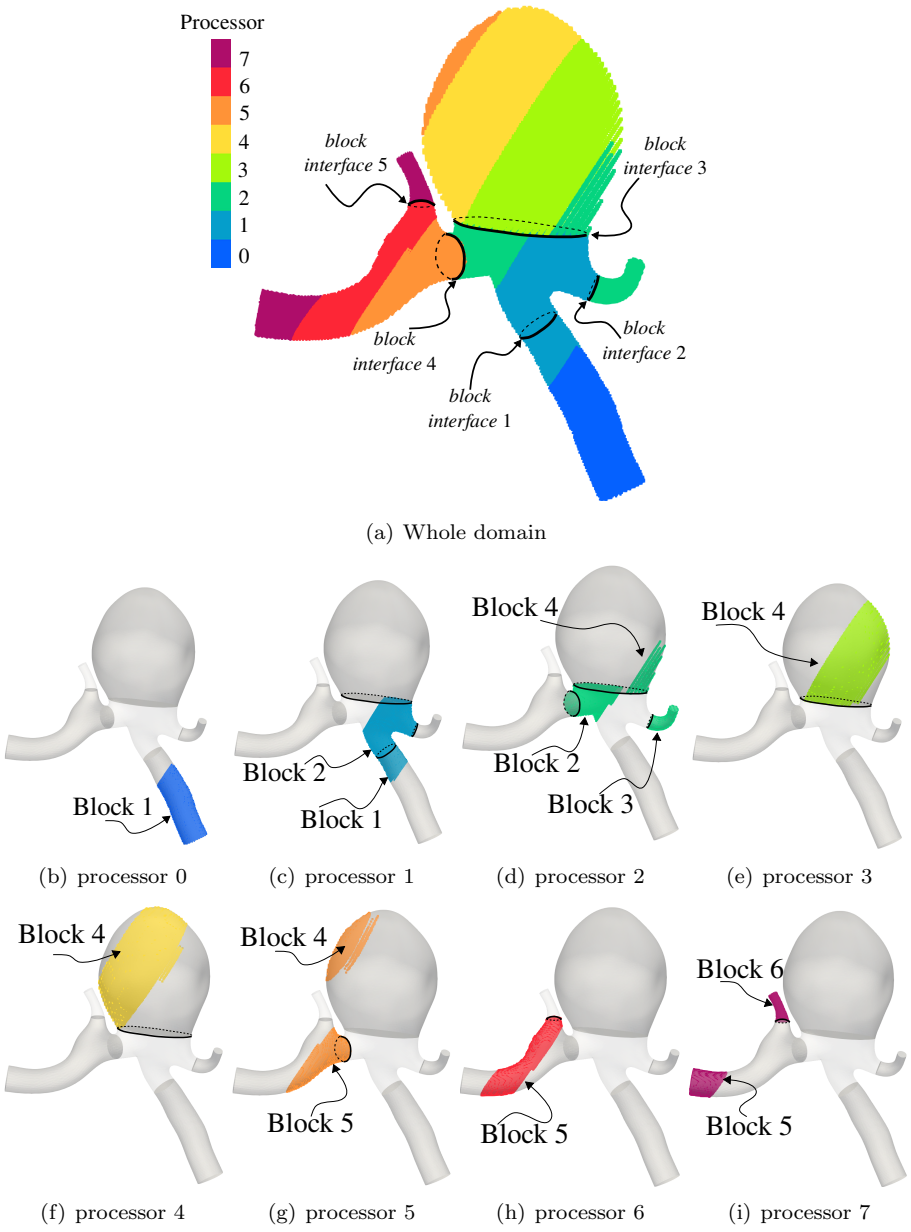



Figure 14: Test case 3. Multi-domain decomposition. $h_1 = 0.625 \cdot 10^{-4}\, m$, $h_2 = 0.75 \cdot 10^{-4}\, m$, $h_3 = 0.375 \cdot 10^{-4}\, m$, $h_4 = 1.25 \cdot 10^{-4}\, m$, $h_5 = 0.5 \cdot 10^{-4}\, m$, $h_6 = 0.25 \cdot 10^{-4}\, m$. Bold red lines: *block interface* contours.

An example of domain distribution with 8 MPI partitions is shown in Fig. 15, where the particles belonging to the domain competence of each processor are highlighted with different colors. In order to show the different cases that can occur in the multi-domain approach with several processors, the obtained domain distribution is discussed in detail for the first three processors. The first processor is allocated a portion of block 1 and, due to the distance from the *block interfaces*, it does not handle any *interface particle* (Fig. 15.b). The second processor is allocated the remaining portion of block 1, a fraction of block 2, the whole first *block interface*, one side of the second *block interface* separating blocks 2 and 3, and a portion of the third *block interface* separating blocks 2 and 4 (Fig. 15.c). The third processor is allocated the remaining portion of block 2, the whole block 3, a portion of block 4, one side of the second *block interface* between blocks 2 and 3, and a portion of the third *block interface* separating blocks 2 and 4 (Fig. 15.d). Similar conditions occur to the remaining processors.

Table 1 shows the number of particles belonging to the domain competence of each processor with reference to each block. The last two rows of the table summarize the distribution of the total number of *effective* and *interface particles* ($N_{e,tot}$ and $N_{IP,tot}$, respectively). As it can be seen, a fair distribution of the *effective particles* ($N_{e,tot}/N_{procs} = 902\,576/8 \approx 112\,822$) is employed, resulting in an efficient load balancing among processors. The number of *interface particles* changes according to the variable distance of the *effective particles* from the *block interfaces* handled by each processor. The uneven distribution of the *interface particles*, obviously, results in a relative loss of efficiency in the parallelization algorithm when their number is a large fraction of that of the *effective particles*. In this test case, where the number of *interface particles* $N_{IP,tot}$ is about 3.3% of $N_{e,tot}$, the most loaded processor is the fourth (proces-




(a) Whole domain

(b) processor 0 (c) processor 1 (d) processor 2 (e) processor 3

(f) processor 4 (g) processor 5 (h) processor 6 (i) processor 7

Figure 15: Test case 3. a) Domain distribution. From b) to i): domain competence of the processors, indicated with different colors. Bold black lines: *block interfaces*.

sor 3 in Table 1), having 9 158 *interface particles*. This is substantially larger than the average number of 30 079/8 = 3 760. Summing this number to the number of *effective particles*, a total of 112 823 + 9 158 = 121 981 is obtained,

| **Block/Proc** | $Proc\ 0$ | $Proc\ 1$ | $Proc\ 2$ | $Proc\ 3$ |
|---|---|---|---|---|
| $B_1 : N_e = 145\,375$ | 112 825 | 32 550 | – | – |
| $B_2 : N_e = 140\,376$ | – | 80 273 | 60 103 | – |
| $B_3 : N_e = 46\,144$ | – | – | 46 144 | – |
| $B_4 : N_e = 241\,888$ | – | – | 6 571 | 112 823 |
| $B_5 : N_e = 256\,404$ | – | – | – | – |
| $B_6 : N_e = 72\,389$ | – | – | – | – |
| $N_{e,tot} = 902\,576$ | 112 825 | 112 823 | 112 818 | 112 823 |
| $N_{IP,Tot} = 30\,079$ | – | 5 837 | 8 186 | 9 158 |

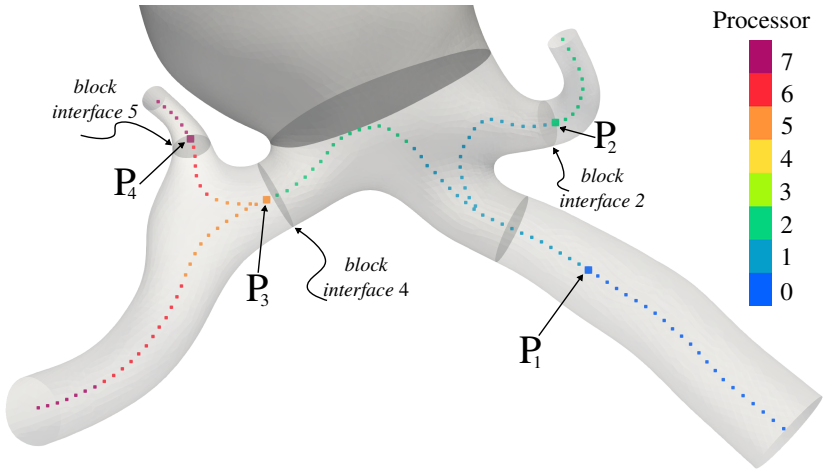
| **Block/Proc** | $Proc\ 4$ | $Proc\ 5$ | $Proc\ 6$ | $Proc\ 7$ |
|---|---|---|---|---|
| $B_1 : N_e = 145\,375$ | – | – | – | – |
| $B_2 : N_e = 140\,376$ | – | – | – | – |
| $B_3 : N_e = 46\,144$ | – | – | – | – |
| $B_4 : N_e = 241\,888$ | 112 826 | 9 668 | – | – |
| $B_5 : N_e = 256\,404$ | – | 103 153 | 112 823 | 40 428 |
| $B_6 : N_e = 72\,389$ | – | – | – | 72 389 |
| $N_{e,tot} = 902\,576$ | 112 826 | 112 821 | 112 823 | 112 817 |
| $N_{IP,Tot} = 30\,079$ | 531 | 1 496 | 3 886 | 985 |

Table 1: Test case 3. Number of particles for each processor with $N_{procs} = 8$.

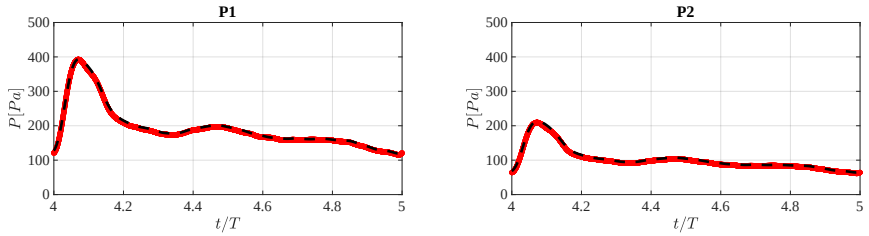
which is 4.6% larger than the number corresponding to the *ideal* distribution of $112\,822 + 3\,760$.

Fig. 16 shows a comparison of the parallel simulation results (8 processors) with the serial simulation so as confirm that the domain distribution does not affect the estimated hydrodynamic fields. Specifically, the cross-section averaged pressure determined at points $P_1$ to $P_4$ of the vessel centerline (Fig. 16.a) was calculated. Point $P_1$, belonging to the competence domain of processor 0, is close to the *parallel interface* between processors 0 and 1, point $P_2$ (inside the domain competence of processor 2) is close to the *block/parallel interface* separating the domain competence of processors 1 and 2, while point $P_2$ of processor 5 and point $P_4$ of processor 7 are close to the fourth and fifth *block interfaces*, respectively. As it can be seen from the Figs. 16.b-e, the results of the parallel simulation are identical to those of the serial simulation.

The results of the performed scalability test are shown in Fig. 17. As discussed with reference to Test case 1 (§4.1) and 2 (§4.2), the load unbalance related to the uneven distribution of the *interface particles* contributes to a progressive decay of the efficiency of the parallelization procedure with the increase of the number of processors. Moreover, in this complex test case, the efficiency is also reduced due to the presence of five *block interfaces* whose interpolation equations for the velocities are solved separately for each interface, as



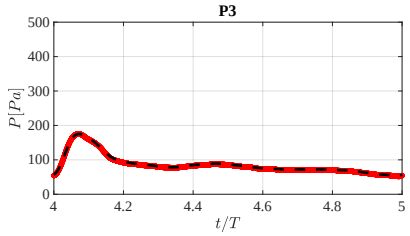

(a) scheme



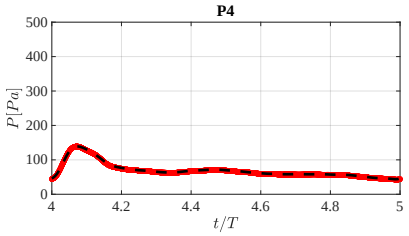

(b) $P_1$




(c) $P_2$



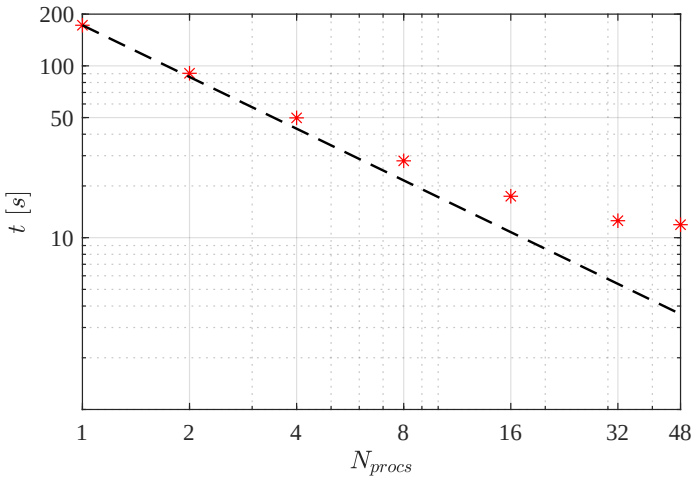

(d) $P_3$



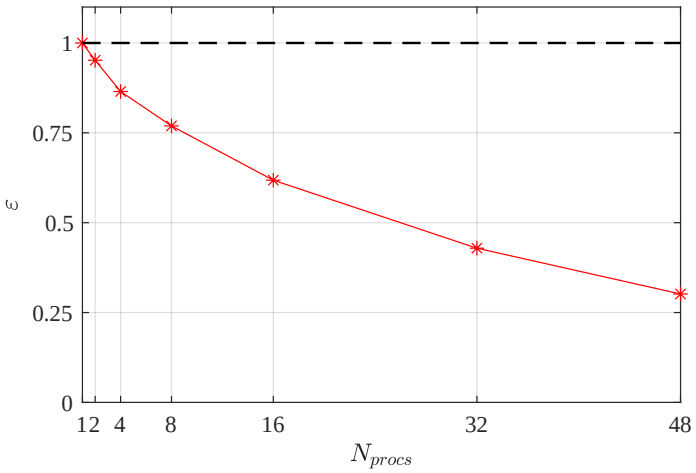

(e) $P_4$

Figure 16: Test case 3. Cross-section averaged pressure at points $P_1$, $P_2$, $P_3$, $P_4$ of the vessel centerline. Bold red line: parallel simulation with $N_{procs} = 8$; dashed black line: serial simulation.

discussed in §3.4. Therefore, in order to improve the efficiency of the parallelization scheme, the *interface particle* computational weight could be considered in the domain distribution procedure. Nonetheless, the implemented parallelization technique is quite satisfactory results in a major improvement of the overall computational efficiency.




(a)




(b)

Figure 17: Test case 3. a) Scalability test: time versus $N_{procs}$. Dashed black line: ideal scalability; red stars: CPU wall-clock time for each $N_{procs}$ value. Double logarithmic scale; b) Parallel efficiency ($\epsilon$) versus $N_{procs}$. Dashed black line: ideal parallel efficiency.

## 5. Conclusions

A parallel computing scheme for a multi-domain ISPH approach was implemented. The combination of variable resolution and parallel computing approaches allows to greatly improve the computational efficiency of the SPH method. The domain is firstly subdivided into non-overlapping blocks having their own *smoothing length* values. *Interface particles* are then introduced, to ensure seamless flow transition through *block interfaces* separating neighbouring blocks. An automatic procedure for the domain distribution, based on the assignment of the cells of a *virtual grid*, allows to efficiently distribute the computational load among the processors. The domain competence of each processor is fixed in time, whilst the corresponding leaving/entering of particles is handled at each time step. The processors can be assigned one or more whole blocks and/or a portion of a block, as well as entire and/or portions of *block interfaces*.

A system is solved at each *block interface* for the intermediate and corrected velocities, employing an efficient parallelized BiCGSTAB algorithm. On the other hand, a whole system is built for the *pseudo-pressure* values, solving simultaneously the PPEs for the *effective particles* and the interpolation equations for the *interface particles*.

The scalability tests, performed up to 48 processors, have confirmed the good efficiency of the proposed parallel scheme. Moreover, future improvement of the procedure should take into account the weight of the *interface particles* while distributing the domain among the processors.

The proposed parallel scheme is general and applicable even when a single-domain approach, consisting of a single block with a constant resolution, is considered.

The approach is well suited for confined flows where the number of particles in each cell of the *virtual grid* is, in the average, constant in time, due to the continuity constraint. When considering free-surface flows, the number of particles can change, as some cells which are empty in one time step could be full in the next one. Consequently, for unconfined flows, the assignment of the cells of the *virtual grid* should be dynamically updated to maintain a constant in time workload among the involved processors. To this purpose, the balancing algorithm could be set off whenever the most loaded processor has a number of particles that exceeds the average per processor by a fixed threshold.

## References

Antoci, C., Gallati, M., Sibilla, S., 2007. Numerical simulation of fluid–structure interaction by sph. Computers & structures 85, 879–890.

Barcarolo, D.A., Le Touzé, D., Oger, G., De Vuyst, F., 2014. Adaptive particle refinement and derefinement applied to the smoothed particle hydrodynamics method. Journal of Computational Physics 273, 640–657.

Bisseling, R.H., 2004. Parallel scientific computation: a structured approach using BSP and MPI. OUP Oxford.

Chiron, L., Marrone, S., Di Mascio, A., Le Touzé, D., 2018. Coupled sph–fv method with net vorticity and mass transfer. Journal of Computational Physics 364, 111–136.

Chorin, A.J., 1968. Numerical solution of the navier-stokes equations. Mathematics of computation 22, 745–762.

Chow, A.D., Rogers, B.D., Lind, S.J., Stansby, P.K., 2018. Incompressible sph (isph) with fast poisson solver on a gpu. Computer Physics Communications 226, 81–103.

Crespo, A.J., Domínguez, J.M., Rogers, B.D., Gómez-Gesteira, M., Longshaw, S., Canelas, R., Vacondio, R., Barreiro, A., García-Feal, O., 2015. Dualsphysics: Open-source parallel cfd solver based on smoothed particle hydrodynamics (sph). Computer Physics Communications 187, 204–216.

De Marchis, M., Milici, B., 2016. Turbulence modulation by micro-particles in smooth and rough channels. Physics of Fluids 28, 115101.

Domínguez, J.M., Crespo, A.J., Valdez-Balderas, D., Rogers, B.D., Gómez-Gesteira, M., 2013. New multi-gpu implementation for smoothed particle hydrodynamics on heterogeneous clusters. Computer Physics Communications 184, 1848–1860.

Ferrari, A., Dumbser, M., Toro, E.F., Armanini, A., 2009. A new 3d parallel sph scheme for free surface flows. Computers & Fluids 38, 1203–1217.

Gingold, R.A., Monaghan, J.J., 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. Monthly notices of the royal astronomical society 181, 375–389.

Guo, X., Rogers, B.D., Lind, S., Stansby, P.K., 2018. New massively parallel scheme for incompressible smoothed particle hydrodynamics (isph) for highly nonlinear and distorted flow. Computer Physics Communications 233, 16–28.

Hu, W., Pan, W., Rakhsha, M., Tian, Q., Hu, H., Negrut, D., 2017. A consistent multi-resolution smoothed particle hydrodynamics method. Computer Methods in Applied Mechanics and Engineering 324, 278–299.

Huang, C., Zhao, L., Niu, J., Di, J., Yuan, J., Zhao, Q., Zhang, F., Zhang, Z., Lei, J., He, G., 2022. Coupled particle and mesh method in an euler frame for unsteady flows around the pitching airfoil. Engineering Analysis with Boundary Elements 138, 159–176.

Kim, J., Moin, P., 1985. Application of a fractional-step method to incompressible navier-stokes equations. Journal of computational physics 59, 308–323.

Lind, S.J., Xu, R., Stansby, P.K., Rogers, B.D., 2012. Incompressible smoothed particle hydrodynamics for free-surface flows: A generalised diffusion-based algorithm for stability and validations for impulsive flows and propagating waves. Journal of Computational Physics 231, 1499–1523.

Liu, M., Liu, G., 2010. Smoothed particle hydrodynamics (sph): an overview and recent developments. Archives of computational methods in engineering 17, 25–76.

Liu, M., Zhang, Z., 2019. Smoothed particle hydrodynamics (sph) for modeling fluid-structure interactions. Science China Physics, Mechanics & Astronomy 62, 1–38.

Lucy, L.B., 1977. A numerical approach to the testing of the fission hypothesis. The astronomical journal 82, 1013–1024.

Marrone, S., Bouscasse, B., Colagrossi, A., Antuono, M., 2012. Study of ship wave breaking patterns using 3d parallel sph simulations. Computers & Fluids 69, 54–66.

Milici, B., De Marchis, M., 2016. Statistics of inertial particle deviation from fluid particle trajectories in horizontal rough wall turbulent channel flow. International Journal of Heat and Fluid Flow 60, 1–11.

Monaghan, J.J., 1992. Smoothed particle hydrodynamics. Annual review of astronomy and astrophysics 30, 543–574.

Monaghan, J.J., 2012. Smoothed particle hydrodynamics and its diverse applications. Annual Review of Fluid Mechanics 44, 323–346.

Monteleone, A., Borino, G., Napoli, E., Burriesci, G., 2022. Fluid–structure interaction approach with smoothed particle hydrodynamics and particle–spring systems. Computer Methods in Applied Mechanics and Engineering 392, 114728.

Monteleone, A., De Marchis, M., Milici, B., Napoli, E., 2018. A multi-domain approach for smoothed particle hydrodynamics simulations of highly complex flows. Computer Methods in Applied Mechanics and Engineering 340, 956–977.

Monteleone, A., Monteforte, M., Napoli, E., 2017. Inflow/outflow pressure boundary conditions for smoothed particle hydrodynamics simulations of incompressible flows. Computers & Fluids 159, 9–22.

Morris, J.P., Fox, P.J., Zhu, Y., 1997. Modeling low reynolds number incompressible flows using sph. Journal of computational physics 136, 214–226.

Napoli, E., De Marchis, M., Gianguzzi, C., Milici, B., Monteleone, A., 2016. A coupled finite volume–smoothed particle hydrodynamics method for incompressible flows. Computer Methods in Applied Mechanics and Engineering 310, 674–693.

Napoli, E., De Marchis, M., Vitanza, E., 2015. Panormus-sph. a new smoothed particle hydrodynamics solver for incompressible flows. Computers & Fluids 106, 185–195.

O'connor, J., Domínguez, J.M., Rogers, B.D., Lind, S.J., Stansby, P.K., 2021. Eulerian incompressible smoothed particle hydrodynamics on multiple gpus. Computer Physics Communications , 108263.

Oger, G., Doring, M., Alessandrini, B., Ferrant, P., 2007. An improved sph method: Towards higher order convergence. Journal of Computational Physics 225, 1472–1492.

Oger, G., Le Touzé, D., Guibert, D., De Leffe, M., Biddiscombe, J., Soumagne, J., Piccinali, J.G., 2016. On distributed memory mpi-based parallelization of sph codes in massive hpc context. Computer Physics Communications 200, 1–14.

Saad, Y., 2003. Iterative methods for sparse linear systems. volume 82. siam.

Spreng, F., Schnabel, D., Mueller, A., Eberhard, P., 2014. A local adaptive discretization algorithm for smoothed particle hydrodynamics. Computational Particle Mechanics 1, 131–145.

Swegle, J.W., Hicks, D.L., Attaway, S., 1995. Smoothed particle hydrodynamics stability analysis. Journal of computational physics 116, 123–134.

Tafuni, A., Domínguez, J., Vacondio, R., Crespo, A., 2018. A versatile algorithm for the treatment of open boundary conditions in smoothed particle hydrodynamics gpu models. Computer Methods in Applied Mechanics and Engineering 342, 604–624.

Vacondio, R., Rogers, B., Stansby, P., Mignosa, P., 2016. Variable resolution for sph in three dimensions: Towards optimal splitting and coalescing for dynamic adaptivity. Computer Methods in Applied Mechanics and Engineering 300, 442–460.

Vacondio, R., Rogers, B., Stansby, P., Mignosa, P., Feldman, J., 2013. Variable resolution for sph: a dynamic particle coalescing and splitting scheme. Computer Methods in Applied Mechanics and Engineering 256, 132–148.

Van der Vorst, H.A., 1992. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. SIAM Journal on scientific and Statistical Computing 13, 631–644.

Wang, S., Hu, J., Huang, C., Yu, Y., 2021. Graphics processing unit-accelerated smoothed particle hydrodynamics—finite difference method and the application for the flow around a cylinder with forced motions. Physics of Fluids 33, 127122.

Wendland, H., 1995. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. Advances in computational Mathematics 4, 389–396.

Winkler, D., Rezavand, M., Rauch, W., 2018. Neighbour lists for smoothed particle hydrodynamics on gpus. Computer Physics Communications 225, 140–148.

Womersley, J.R., 1955. Method for the calculation of velocity, rate of flow and viscous drag in arteries when the pressure gradient is known. The Journal of physiology 127, 553.

Xiong, Q., Li, B., Xu, J., 2013. Gpu-accelerated adaptive particle splitting and merging in sph. Computer Physics Communications 184, 1701–1707.

Xu, R., Stansby, P., Laurence, D., 2009. Accuracy and stability in incompressible sph (isph) based on the projection method and a new approach. Journal of computational Physics 228, 6703–6725.

Zhan, L., Peng, C., Zhang, B., Wu, W., 2019. A stabilized tl–wc sph approach with gpu acceleration for three-dimensional fluid–structure interaction. Journal of Fluids and Structures 86, 329–353.