Scalability of an Eulerian-Lagrangian large-eddy simulation solver with hybrid MPI/OpenMP parallelisation

Ouro, Pablo; Fraga, Bruno; Lopez, Unai; Stoesser, Thorsten

DOI:
10.1016/j.compfluid.2018.10.013

Computers and Fluids 179 (2019) 123–136

Contents lists available at ScienceDirect

Computers and Fluids

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

Scalability of an Eulerian-Lagrangian large-eddy simulation solver with hybrid MPI/OpenMP parallelisation

Pablo Ouro[a,*], Bruño Fraga[b], Unai Lopez-Novoa[c], Thorsten Stoesser[a]

[a] Hydro-environmental Research Centre, Cardiff School of Engineering, Cardiff University, The Parade, Cardiff CF24 3AA, UK
[b] School of Engineering, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK
[c] Data Innovation Research Institute, Cardiff University The Parade, Cardiff CF24 3AA UK

ARTICLE INFO

Article history:
Received 22 March 2018
Revised 19 September 2018
Accepted 9 October 2018
Available online 10 October 2018

Keywords:
Hybrid MPI/OpenMP
Eulerian-Lagrangian
Large-eddy simulation
Immersed boundary method
Multiphase flows
High performance computing

ABSTRACT

Eulerian-Lagrangian approaches capable of accurately reproducing complex fluid flows are becoming more and more popular due to the increasing availability and capacity of High Performance Computing facilities. However, the parallelisation of the Lagrangian part of such methods is challenging when a large number of Lagrangian markers are employed. In this study, a hybrid MPI/OpenMP parallelisation strategy is presented and implemented in a finite difference based large-eddy simulation code featuring the immersed boundary method which generally employs a large number of Lagrangian markers. A master-scattering-gathering strategy is used to deal with the handling of the Lagrangian markers and OpenMP is employed to distribute their computational load across several CPU threads. A classical domain-decomposition-based MPI approach is used to carry out the Eulerian, fixed-mesh fluid calculations. The results demonstrate that by using an effective combination of MPI and OpenMP the code can outperform a pure MPI parallelisation approach by up to 20%. Outcomes from this paper are of interest to various Eulerian-Lagrangian applications including the immersed boundary method, discrete element method or Lagrangian particle tracking.

1. Introduction

The constant evolution of High Performance Computing (HPC) systems has accelerated the development of sophisticated Computational Fluid Dynamics (CFD) codes. This has facilitated expensive Direct Numerical Simulations (DNS) and Large-Eddy Simulations (LES) of turbulent flows at low to moderately high Reynolds numbers in complex geometries in the fields of aeronautics, bio-flows or hydro-environmental engineering [1–3]. Recent applications of DNS and LES include complex multiphase flows which often require adopting Eulerian-Eulerian frameworks, e.g. scalar transport [4] or free-surface flows [5]; or Eulerian-Lagrangian frameworks, e.g. fluid-structure interaction [6] or air-gas interaction [7]. Most of the cited computations were performed at a reduced scale and for validation purposes demonstrating the methods' adequacy to reproduce the relevant physics. There is a fast-growing interest in applying these advanced eddy-resolving techniques to real-life problems, which means that the range of spatial and temporal scales to be resolved increases by one or several orders of magnitude and hence these computations become extremely expensive.

What follows is that CFD codes must not only be portable to modern HPC hardware, but also scalable to hundreds of thousands of processors. The main strategies to parallelise and scale modern CFD codes most commonly used are: distributed memory through Message Passing Interface (MPI), shared memory through Open Multi-Processing (OpenMP) or a combination of both. Alternatively, Graphics Processing Units (GPUs) can be employed to speedup particular portions of code and its application to CFD is growing [8]. To date, MPI is the most used protocol to compute DNS and LES in parallel using mesh-based methods. It allows to divide the computational domain into smaller sub-domains and to solve the same code on basis of the Single Program Multiple Data (SPMD) paradigm. The key to an efficient MPI parallelisation strategy is the communication between sub-domains as the information exchange across their interfaces must ensure the coherence and continuity of the simulation [9]. On the other hand, the efficient computation of particle-based methods is frequently performed using OpenMP [10]. Complementary to pure MPI or OpenMP codes, their combination can provide an outstanding increase in code performance as it allows to specifically tackle the code's bottleneck.

Hybrid parallelisation techniques are becoming key to efficiently perform simulations in many CFD fields such as Smoothed

* Corresponding author.
E-mail address: ourobarbap@cardiff.ac.uk (P. Ouro).

https://doi.org/10.1016/j.compfluid.2018.10.013

Particle Hydrodynamics (SPH) whose applicability is somewhat limited by the number of Lagrangian particles due to their inherent expensive computations, e.g. neighbour searching. Novel hybrid techniques are under development in order to enhance its computational efficiency, such as an MPI/CUDA scheme presented by Dominguez et al. [11], which improved load-balancing. In a similar fashion, the most efficient way to parallelise Discrete Element Methods (DEM) depends on the homogeneity of the particles' distribution. Gopalakrishnan et al. [12] presented good performance results for a pure MPI implementation of the open-source DEM code MFIX. A similar approach was followed by Yang et al. [13] achieving good scalability results in simulations with more than 10^6 particles. Liu et al. [14] implemented multi-threading in MFIX showing that a hybrid MPI/OpenMP code could overcome load-balancing problems outperforming the MPI scheme when $5.2 \cdot 10^6$ particles were simulated. Additionally, Amritkar et al. [15] showed that for several DEM applications pure OpenMP can be notably faster than MPI especially when a reduced number of processes were used in the simulations.

Yakubov et al. [16] adopted a hybrid MPI/OpenMP strategy in an Euler-Lagrange framework to simulate the flow around an airfoil where bubbles were injected into the cavitation areas, and results proved this scheme features good performance. Shi et al. [17] studied in detail the performance differences between pure MPI and hybrid MPI/OpenMP implementations of a DNS code with application to Taylor-Couette flows. They found that a mixed scheme has many benefits in reducing inter-node communications which is key to scale to hundreds or thousands of cores. Similar results were obtained by Guo et al. [18] for the finite element model Fluidity achieving better performance using a hybrid scheme compared to pure MPI due to lower communication overheads.

Overall the development of hybrid MPI/OpenMP parallelisation strategies is of interest to many CFD research areas: in DNS and LES of single-phase flows the main aim is to reduce inter-node communications between hundreds-to-thousands of cores, or in multiphase flow applications in which load-balancing problems due to Lagrangian computations (e.g. interpolation functions reconstruction or neighbour searching algorithm) need to be overcome.

In the research reported here a refined hybrid MPI/OpenMP parallelisation strategy is implemented in the open-source LES-based code Hydro3D. The code features coarse-grained MPI parallelisation the efficiency of which is studied first for the lid-driven cavity test-case. The performance of the hybrid strategy is then evaluated by comparing MPI/OpenMP to pure MPI computations for two Eulerian-Lagrangian fluid-structure interaction test-problems for which the Lagrangian part of the solution is a known bottleneck to the overall speed of the code.

The paper is organised as follows: the governing equations for fluid flow and the immersed boundary method are described in Section 2. Section 3 presents the hybrid parallelisation strategy. Section 4 presents the performance results comparing pure MPI and hybrid MPI/OpenMP schemes for the multiphase applications. A discussion based on the overall results towards the application of the hybrid parallelisation approach within Eulerian-Lagrangian applications is presented in Section 5 together with the main conclusions from this study.

2. Numerical framework

Hydro3D is an in-house open-source [19] large-eddy simulation code that has been well-validated in a number of hydro-environmental engineering flows such as compound channels [20], contact tanks [21–23], free-surface flows [24–27], aeronautical engineering applications such as flow around pitching airfoils [28] or geophysical flows [29,30].

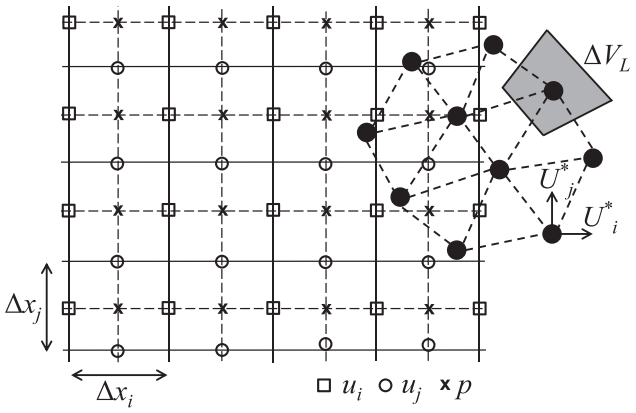

Fig. 1. 2D sketch of a distribution of Lagrangian points in an Eulerian staggered grid for a body represented by the immersed boundary method.

Hydro3D efficiently solves the filtered Navier–Stokes equations for unsteady, incompressible, viscous flows solved in an Eulerian frame reading,

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -\nabla p - \mathbf{u} \cdot \nabla \mathbf{u} + \nu \nabla^2 \mathbf{u} - \nabla \tau + \mathbf{f} \tag{2}$$

where $\mathbf{u}(\mathbf{x},t)$ represents the velocity field, $p(\mathbf{x},t)$ is the pressure, ν is the kinematic viscosity of the fluid, and $\mathbf{f}(\mathbf{x},t)$ is a volume force from a source external to the fluid, e.g. Lagrangian particles. The sub-grid scale stress (SGS) tensor, τ, can be calculated in Hydro3D with the Smagorinsky [31] or the Wall-Adapted Local Eddy viscosity (WALE) [32] SGS models. Here the latter is adopted as it is preferred when dealing with moving boundaries as it implicitly calculates the SGS viscosity near solid boundaries [6,9].

Hydro3D is based on finite differences with staggered storage of the three velocity components on three rectangular Cartesian grids and the storage of the pressure in their respective cell centre. Second- and fourth-order Central Differences Schemes (CDS) are available to approximate convective and diffusive velocity fluxes. In this research, second-order CDS is adopted as the direct forcing method used to resolve moving boundaries is also second-order accurate [33].

Hydro3D employs a refined direct forcing Immersed Boundary (IB) and Lagrangian Particle Tracking (LPT) methods to perform simulations of Eulerian-Lagrangian flow problems, allowing the simulation of moving bodies or particles inside a fixed Eulerian fluid domain. Fig. 1 depicts a fixed Cartesian rectangular grid as the Eulerian together with an unstructured Lagrangian domain moving at a given Lagrangian velocity. As depicted in this figure, lower-case variables (here coordinates, velocities and pressure) belong to the Eulerian framework whereas upper-case variables are used in the Lagrangian framework.

2.1. Time integration

The advancement in time of the governing equations is performed using the fractional-step method [34]. This adopts the Helmholtz decomposition to calculate the velocity field from a solenoidal and an irrotational vector fields obtained throughout several steps. The first is to predict the non-divergence free velocity, $\tilde{\mathbf{u}}(\mathbf{x},t)$, from the explicit computation of convection and diffusion terms using a low-storage third-order Runge-Kutta scheme together with pressure values from the previous time step as,

$$\frac{\tilde{\mathbf{u}} - \mathbf{u}^{l-1}}{\Delta t} = \alpha_l \nu \nabla^2 \mathbf{u}^{l-1} - \alpha_l \nabla p^{l-1} - \alpha_l [\mathbf{u}(\nabla \cdot \mathbf{u})]^{l-1} - \beta_l [\mathbf{u}(\nabla \cdot \mathbf{u})]^{l-2} \tag{3}$$

where l=1,2,3 is the Runge-Kutta sub-step for which l=1 denotes values from the previous time step $t-1$, and α_l and β_l stand as the Runge-Kutta coefficients with values: $\alpha_l = \beta_l$=1/3, 1/6, 1/2.

In Eulerian-Lagrangian simulations using the IB method, the external forces are represented by the forcing term $\mathbf{f}$ in the r.h.s of Eq. (2), which is used to correct the predicted velocity $\tilde{\mathbf{u}}$ obtaining the updated intermediate velocity $\tilde{\mathbf{u}}^*(\mathbf{x},t)$ as indicated in Eq. (4). This source term $\mathbf{f}$ enforces the fluid to have the solid velocity at its location fulfilling the no-slip equation as explained in Section 2.2.

$$\tilde{\mathbf{u}}^* = \tilde{\mathbf{u}} + \mathbf{f}\Delta t \tag{4}$$

A projection scalar function $\tilde{p}$ is obtained in Eq. (5) resolving the Poisson pressure equation, which in Hydro3D is accomplished using an iterative multi-grid technique. This equation is iterated until the predicted intermediate velocity field $\tilde{\mathbf{u}}^*$ satisfies the incompressiblity condition.

$$\nabla^2 \tilde{p} = \frac{\nabla \cdot \tilde{\mathbf{u}}^*}{\Delta t} \tag{5}$$

Here, the corrected velocity field satisfies the divergence-free condition once Eq. (6) achieves a residual lower than a set tolerance ε often set to a value $\leq 10^{-7}$.

$$\nabla \cdot \tilde{\mathbf{u}}^* \leq \varepsilon \tag{6}$$

The predicted velocity field $\tilde{\mathbf{u}}^*$ is projected onto the divergence-free field following Eq. (7) to obtain the solution velocity field at the current step $\mathbf{u}^t(\mathbf{x},t)$.

$$\mathbf{u}^t = \tilde{\mathbf{u}}^* - \Delta t \nabla \tilde{p} \tag{7}$$

Note the latter velocity field differs from that obtained right after the Lagrangian correction $\tilde{\mathbf{u}}^*$, i.e. the final flow velocity is not exactly that enforced by the solid in Eq. (4). Cristallo et al. [35] showed that the error associated to this step can be deemed negligible. Finally, the pressure field at the current time step, p^t, is calculated in Eq. (8) resulting from the value at the previous time step p^{t-1} and $\tilde{p}$ field.

$$p^t = p^{t-1} + \tilde{p} - \frac{\nu \Delta t}{2} \nabla^2 \tilde{p} \tag{8}$$

2.2. Immersed boundary method

The IB method in Hydro3D was successfully validated in a series of applications ranging from fluid-structure interaction [36], flow around pitching airfoils [28], multi-chamber tanks [23], tidal turbines [6,37] and geophysical flows [38]. The numerical method used to accomplish fluid-structure interaction is a refined version presented in Kara et al. [36] based on the direct forcing IB method introduced by Fadlun et al. [39] and refined by Uhlmann [33]. In this approach, the immersed solid is comprised by a collection of N_L Lagrangian forcing points conforming the targeted geometry (Fig. 1), which directly enforce a no-slip boundary condition at their location through the forcing term $\mathbf{f}$.

The direct forcing method follows a multi-step predictor-corrector procedure which is detailed in the following. First, the predicted Eulerian velocities $\tilde{\mathbf{u}}$, calculated according to Eq. (3), are transferred to the closest Lagrangian markers. This procedure is accomplished using interpolation functions which can feature a different number of neighbours depending on their stencil, i.e. support width. Hereinafter n_e stands for the number of Eulerian cells used to transfer the information to each neighbouring Lagrangian marker, the total number of Eulerian cells and Lagrangian markers are N_e and N_L respectively, and n_L is the number of Lagrangian markers used to interpolate solid quantities to the closest Eulerian cells. Therefore, the interpolated Lagrangian velocity U_L at the marker L is obtained interpolating $\tilde{\mathbf{u}}$ from its n_e closest Eulerian neighbours as,

$$\mathbf{U}_L = \sum_{i=1}^{n_e} \tilde{\mathbf{u}}_i \cdot \delta(\mathbf{x}_i - \mathbf{X}_L) \cdot \Delta\mathbf{x}_i, \qquad L = 1, \ldots, N_L \tag{9}$$

Here $\mathbf{x}_i$ and $\mathbf{X}_L$ are the coordinates of the Eulerian cell i and Lagrangian marker L, respectively, and $\Delta\mathbf{x}_i = \Delta x_i \Delta x_j \Delta x_k$ represents the volume of an Eulerian cell. The second step of the direct forcing method is to compute the force $\mathbf{F}_L$ each Lagrangian marker exerts on the fluid to satisfy the no-slip condition at the marker's position [33]. This force term results as the difference between the desired (or forced) marker velocity, $\mathbf{U}_L^*$, and the interpolated velocity from the fluid $\mathbf{U}_L$, calculated as,

$$\mathbf{F}_L = \frac{\mathbf{U}_L^* - \mathbf{U}_L}{\Delta t}, \qquad L = 1, \ldots, N_L \tag{10}$$

The forced velocity $\mathbf{U}_L^*$ is computed depending on the solid body movement pattern, and there are three different possible scenarios. Firstly, when the body is static the forced velocity is zero. In case the body moves as a reaction to the fluid action, a fluid-structure interaction algorithm is needed to compute the forced velocity, which results from the time rate-of-change of the marker position as indicated in Eq. (11) [36,40]. The last case is when the solid body is moving with a prescribed velocity and pattern which permits a straightforward calculation of its coordinates at any time, such as the cases of vertical axis turbines rotating at fixed velocities or pitching airfoils oscillating at a given reduced frequency [6,28].

$$\mathbf{U}_L^* = \frac{\partial \mathbf{X}_L}{\partial t} \tag{11}$$

The third step constitutes the backwards procedure where the Lagrangian force $\mathbf{F}_L$ is transferred to all the Eulerian cells in the fluid domain affected by Lagrangian particles obtaining the Eulerian force $\mathbf{f}$ as,

$$\mathbf{f}_i = \sum_{L=1}^{n_L} \mathbf{F}_L \cdot \delta(\mathbf{X}_L - \mathbf{x}_i) \cdot \Delta V_L, \qquad i = 1, \ldots, N_e \tag{12}$$

Here the interpolation of $\mathbf{F}_L$ from the closest n_L Lagrangian markers to each fluid cell adopts the delta functions values from the forwards step, Eq. (9). This implies performing the neighbour searching just in the forwards step which is the most computationally expensive operation in the direct forcing method when dealing with moving boundaries.

Note that the forwards interpolation (from Eulerian to Lagrangian) uses the fluid cell volume $\Delta\mathbf{x}_i$ while the backwards process adopts the volume assigned to each of the Lagrangian markers ΔV_L, as represented in Fig. 1. The direct forcing method needs to satisfy mass and torque conservation requiring the force transferred to all fluid cells (N_e) in the backwards procedure equals the total force exerted by the solid markers (N_L). This condition is indicated in Eq. (13) which implies the same total force is interpolated between frameworks.

$$\sum_{i=1}^{N_e} \boldsymbol{f}_i \cdot \delta(\mathbf{x}_i - \mathbf{X}_L) \cdot \Delta\mathbf{x}_i = \sum_{L=1}^{N_L} \mathbf{F}_L \cdot \delta(\mathbf{X}_L - \mathbf{x}_i) \cdot \Delta V_L \tag{13}$$

Delta functions based on discrete kernels do not directly satisfy the partition of unity condition [41] and this condition is then fulfilled whenever the volume of each solid marker approximated that of a fluid cell, i.e. $\Delta V_L \approx \Delta\mathbf{x}_i$.

2.3. Reconstruction of interpolation functions

Delta functions [42–44] or Moving Least Squares (MLS) [41,45] are commonly used to reconstruct the interpolation functions used in the direct forcing method framework. To date, the

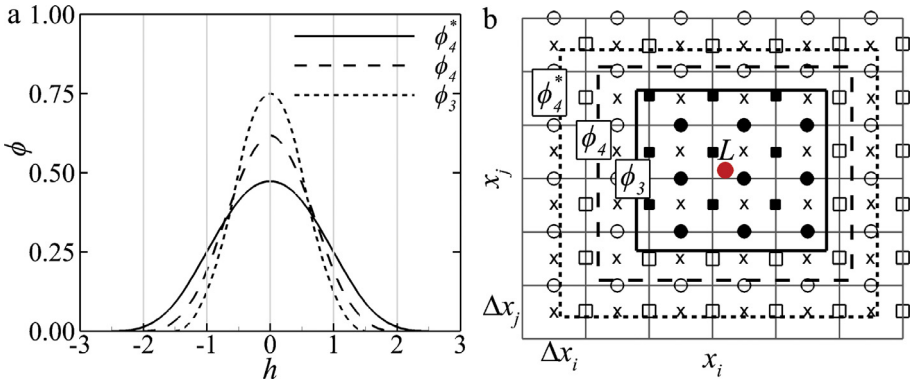

Fig. 2. (a) Kernel functions adopted to reconstruct the interpolating delta and (b) regions over which forces are transferred between Eulerian and Lagrangian frameworks.

use of delta functions in IB applications is more frequent than MLS and thus adopted herein. Note that these two techniques have a similar computational expense as the most time-consuming operation in the interpolation procedure is the neighbour searching, which is accomplished in both methods. Therefore performance results obtained here for the delta functions can be extrapolated to MLS. Delta functions δ are calculated as a result of three one-dimensional kernels ϕ as,

$$\delta\left(\frac{\mathbf{x}_i-\mathbf{X}_L}{\Delta\mathbf{x}_i}\right)=\frac{1}{\Delta\mathbf{x}_i}\phi\left(\frac{x_i-X_{i_L}}{\Delta x_i}\right)\phi\left(\frac{x_j-X_{j_L}}{\Delta x_j}\right)\phi\left(\frac{x_k-X_{k_L}}{\Delta x_k}\right) \quad (14)$$

The kernel functions of ϕ_3 by Roma et al. [44], ϕ_4 by Peskin [43], and ϕ_4^* by Yang et al. [42] as a function of the normalised grid spacing $h=(\mathbf{x}_i-\mathbf{X}_L)/\Delta\mathbf{x}_i$ read,

$$\phi_3(h)=\begin{cases}\frac{1}{3}(1+\sqrt{-3h^2+1})), & \text{if } |h|<0.5.\\ \frac{1}{6}\left(5-3|h|-\sqrt{-3(1-|h|)^2+1}\right), & \text{if } 0.5\le|h|<1.5.\\ 0, & \text{if } |h|\ge 1.5.\end{cases} \quad (15)$$

$$\phi_4(h)=\begin{cases}\frac{1}{8}(3-2|h|+\sqrt{1+4|h|-4h^2})), & \text{if } |h|<1.0.\\ \frac{1}{8}\left(5-2|h|-\sqrt{-7+12|h|-4h^2}\right), & \text{if } 1.0\le|h|<2.0.\\ 0, & \text{if } |h|\ge 2.0.\end{cases} \quad (16)$$

$$\phi_4^*(h)=\begin{cases}\frac{3}{8}+\frac{\pi}{32}-\frac{h^2}{4}, & \text{if } |h|<0.5.\\ \frac{1}{4}+\frac{1-|h|}{8}\sqrt{-2+8|h|-4h^2} & \\ \quad -\frac{1}{8}\arcsin\left(\sqrt{2}(|h|-1)\right), & \text{if } 0.5\le|h|<1.5.\\ \frac{17}{16}-\frac{\pi}{64}-\frac{3|h|}{4}+\frac{r^2}{8}+\frac{|h|-2}{16} & \\ \quad \sqrt{-14+16|h|-4h^2} & \\ \quad +\frac{1}{16}\arcsin\left(\sqrt{2}(|h|-2)\right), & \text{if } 1.5\le|h|<2.5.\\ 0, & \text{if } |h|\ge 2.5.\end{cases} \quad (17)$$

Fig. 2a shows the support width of these kernels utilising between 3 and 5 neighbours in each direction yielding a total number of neighbours, n_e, of 27, 64 and 125, respectively. Fig. 2b depicts how the interpolation area is distributed in a two-dimensional plane (bounded by solid, dashed or dotted lines, respectively) increasing with the kernel width. Yang et al. [42] showed that a larger number of neighbours reduces the non-physical force oscillations during the variable exchange procedures. Nevertheless, the improvement of the interpolations results in additional computational load that is directly proportional to the number of neighbours used.

The computational overhead of the IB method is relatively low if Lagrangian markers are static because the interpolation functions are constructed only once at the first time step. However, it increases significantly for moving bodies, especially in DNS/LES as fine grids are required and a solid body can comprise thousands of markers. Searching for the closest fluid cells in the vicinity of Lagrangian markers is the most time-consuming operation due to the fact that the staggered storage of the three components of velocity entails operations on three different grids, i.e. three interpolation functions for each marker needs to be computed. Hence, two relevant computational aspects in the present efficient implementation of the IB method are: (1) delta functions and indices of the closest neighbours to each Lagrangian marker are computed only during the forwards interpolation and stored to be used during the backwards interpolation avoiding a second neighbour searching process, i.e. $\delta(\mathbf{X}_L-\mathbf{x}_i)=\delta(\mathbf{x}_i-\mathbf{X}_L)$; and (2) a master-scattering-gathering technique (explained in Section 3.2) is developed to efficiently deal with moving Lagrangian particles that travel along different sub-domains and computed by different processes throughout the simulation.

3. Parallelisation strategy

Large eddy simulations require sufficiently fine grids to explicitly resolve the large-scale turbulence in the flow [1,46], which usually implies an enormous computational load especially when the Reynolds number is relatively high. Hydro3D features a Local Mesh Refinement (LMR) methodology that allows refining the fluid mesh in certain areas of interest or of steep gradients while using coarser grids away from them. The implementation of LMR in Hydro3D is detailed in [9] and allows performing high-resolution LES with a reasonable amount of MPI processes. The next Section 3.1 details the way that the Eulerian field is divided and mapped using pure MPI, and Section 3.2 focuses on the Eulerian-Lagrangian computation using a new hybrid MPI/OpenMP environment.

3.1. Eulerian field parallelisation using MPI

In most LES and DNS the solution of the Poisson pressure equation often constitutes the most time-consuming operation within the fractional-step method. The concept of the coarse-grained MPI parallelisation is used to divide the computational domain of the Eulerian field into sub-domains or blocks and execute them in parallel using multiple processing units by means of the Single Program Multiple Data (SPMD). These blocks are assigned to different MPI ranks and communication is accomplished via MPI operations. The blocks overlap at their boundaries by one or several layers of ghost cells (or halos) through which information is exchanged. Fig. 3 depicts an example of four sub-domains with a solid body in two of them. Each sub-domain overlaps with its immediate neighbours through two layers of ghost cells and the in-

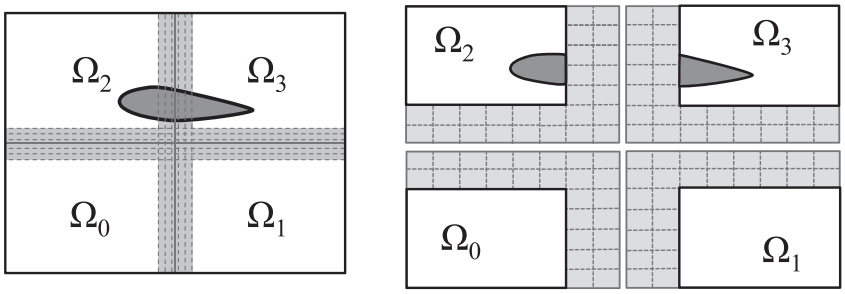

Fig. 3. A fluid domain composed of 4 sub-domains with communication via 2-layer overlapping ghost cells.

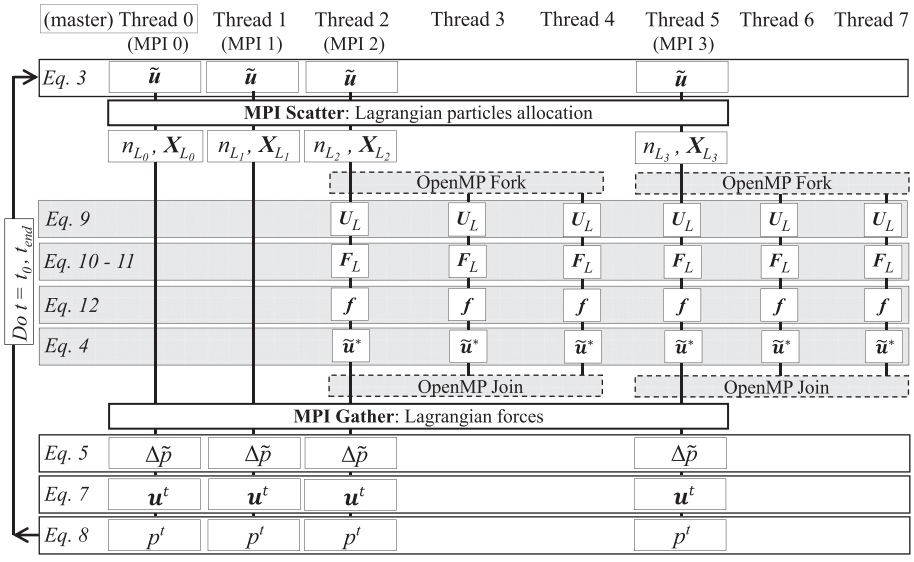

Fig. 4. Schematic representation of the time advancement procedure using 8 physical threads with MPI processes 2 and 5 spawning 3 OpenMP threads.

formation stored in these cells is exchanged, guaranteeing a continuous Eulerian field across domain interfaces. This is a standard parallelisation approach for block-structured grids and increases the calculation speed of the solution of the Poisson equation when using a multigrid solver because a big matrix solution problem is broken down into many small sub-problems [47].

3.2. Hybrid MPI/OpenMP strategy

The use of Eulerian-Lagrangian methods featuring a large number of Lagrangian markers in a relatively confined Eulerian domain [6,7,28,37,48] calls for a parallelisation strategy beyond pure MPI, as most likely Lagrangian markers are not equally distributed amongst the Eulerian sub-domains. For instance, in Fig. 3 the Lagrangian markers comprising the airfoil are all located in domains 2 and 3 and consequently there is a greater computational effort solving the flow in these domains than in domains 0 and 1. Adopting a parallelisation strategy that combines MPI and OpenMP aims at taking advantage of the benefits of each technique [49–51]. The proposed hybrid scheme in the Eulerian-Lagrangian solver combines a coarse-grained message-passing parallelism in the Eulerian calculations with a fine-grained multi-thread parallelism for the Lagrangian calculations [52].

A global layout of the proposed hybrid approach using as example 4 MPI processes with 2 of them spawning 3 OpenMP threads to fork the Lagrangian calculations is depicted in Fig. 4. Firstly, the predicted Eulerian velocities are calculated by each MPI rank according to Eq. (9). The next step, denoted as "particle allocation", concerns the distribution of the Lagrangian points among the different MPI processes. This is accomplished with a strategy combining the "master-slave" concept from Uhlmann [53] with the "scattering-and-gathering" strategy from Wang et al. [54]. The master processor (hereinafter referred to as master) gathers the general information from all the Lagrangian markers, e.g. their coordinates. Based on these data, the master calculates for each marker in which sub-domain i is located and the MPI process that is assigned to deal with it, and also generates a vector X_{L_i} with the indices of the markers contained within the sub-domain i. The latter is distributed via SCATTER together with the integer n_{L_i} that indicates the number of markers within the sub-domain i. Depending on whether the marker moves (dynamic) or is static (fixed) the particle allocation and scattering-and-gathering are performed at every time step or just once at the first one, respectively.

The computation of the IB method equations (Eqs. (9)–(12)) is performed by all MPI processes whose assigned sub-domains contain Lagrangian markers. This differs from Wang et al.'s [54] strategy in which only the master deals with these equations exchanging the Lagrangian velocities and forces via MPI communications with the other MPI processes once resolved. This is an efficient strategy whenever the number of Lagrangian points is relatively small as the communication of large arrays can lead to significant overheads. In the present cases there is a great number of markers causing an important load-unbalancing and hence the proposed alternative strategy adding multi-threading to the "master-scattering-gathering" strategy aims at improving the code's speedup by (i) reducing inter-node communication between MPI processes, and (ii) improving the locality of the IB method computations. No MPI calls are made within the OpenMP parallelised loops, which simplifies the present implementation and

Node 0 *Pure MPI*				Node 1 *Pure MPI & hybrid MPI/OpenMP*				Node 2 *Hybrid MPI/OpenMP*				Node 3 *Hybrid MPI/OpenMP*			
Socket 0		Socket 1		Socket 0		Socket 1		Socket 0		Socket 1		Socket 0		Socket 1	
MPI 0	MPI 1	MPI 8	MPI 9	MPI 16	MPI 17	MPI 24	MPI 25	MPI 28 Thrd 0	MPI 29 Thrd 0	MPI 32 Thrd 0	MPI 33 Thrd 0	MPI 36 Thrd 0	MPI 37 Thrd 0		
MPI 2	MPI 3	MPI 10	MPI 11	MPI 18	**MPI 19**	**MPI 26 Thrd 0**	**MPI 27 Thrd 0**	*MPI 28 Thrd 1*	*MPI 29 Thrd 1*	*MPI 32 Thrd 1*	*MPI 33 Thrd 1*	*MPI 36 Thrd 1*	*MPI 37 Thrd 1*		
MPI 4	MPI 5	MPI 12	MPI 13	MPI 20	**MPI 21**	*MPI 26 Thrd 1*	*MPI 27 Thrd 1*	MPI 30 Thrd 0	MPI 31 Thrd 0	MPI 34 Thrd 0	MPI 35 Thrd 0				
MPI 6	MPI 7	MPI 14	MPI 15	MPI 22	**MPI 23**			*MPI 30 Thrd 1*	*MPI 31 Thrd 1*	*MPI 34 Thrd 1*	*MPI 35 Thrd 1*				

Fig. 5. Schematic distribution of 38 MPI processes on 4 SMP nodes for which 12 MPI processes spawn 2 OpenMP threads.

avoids some of the drawbacks associated with such hybrid parallelisation schemes [49,55].

After performing the Lagrangian forcing correction on the Eulerian field via Eq. (12), an inverse operation sending the Lagrangian forces from the processors to the master is performed via GATHER. The latter is needed when the Lagrangian forces are physically meaningful, such as in the analysis of tidal turbines [6,37]. The final step is to compute the remaining Eulerian variables of the scalar $\tilde{p}$ correcting the final velocity $\mathbf{u}^t$ and pressure p^t fields using Eqs. (5)–(8).

In many hybrid MPI/OpenMP applications the distribution of MPI/OpenMP threads is homogeneous, while in our implementation not every MPI process spawns OpenMP threads. This adds certain flexibility in the hybrid scheme especially when allocating cores dedicated to either MPI or OpenMP within the same node to reduce inter-node communications [49]. Fig. 5 shows a schematic core distribution for 38 MPI tasks among 4 nodes from which 12 ranks uses 2 threads. Different distributions among the processes are combined with nodes dedicated to pure MPI processes, mixed use of pure MPI and hybrid MPI/OpenMP, and exclusively dedicated to hybrid MPI/OpenMP.

In order to ensure a correct sequential mapping of the processes to the compute cores in the mixed MPI/OpenMP scheme, it is necessary to tune the launch configuration of the code in the job scheduler. Considering Slurm as scheduler, the environment variable SLURM_TASKS_PER_NODE is set in the batch script to indicate ensure those MPI processes spawning additional OpenMP threads are placed on the same nodes, which can be also shared with those running pure MPI processes. For instance, in the SMP-style (a sequential) assignation of the resources for the scenario depicted in Fig. 5, the environment variable would be SLURM_TASKS_PER_NODE $=' 16(\times 1), 12(\times 1), 8(\times 1), 2(\times 1)'$, in the value which is a list of comma-separated items of set as $A(xB)$ with A denoting the number of MPI processes on B consecutive nodes. This value would Hence, 16 MPI processes are assigned to the first compute node, 12 MPI processes to the second one leaving room for two of them (MPI processes 26 and 27) to spawn an additional OpenMP thread each, 8 MPI processes to the third node, leaving room for all allowing them to spawn an additional OpenMP thread each, and the remaining MPI processes (36 and 37) to the last node. Note this feature is not exclusive of Slurm as the proposed hybrid parallelisation scheme of Hydro3D could be adopted with other schedulers, e.g. PBS, and without the need for administrative privileges.

4. Parallel performance assessment

This section presents the scalability study of Hydro3D for three different flow problems. The first one is a lid-driven cavity flow case and the other two are high-resolution large-eddy simulations of complex flows of engineering interest [6,28]. The parallelisation speedup S_n is evaluated as,

$$S_n = \frac{T_0}{T_n} \tag{18}$$

where T_0 corresponds to the wall-time or runtime obtained with the configuration using the lowest number of cores and T_n corresponds to the runtime when using n cores. The runtime is obtained with the `MPI_WTIME()` directive from time-averaging 200 time steps executed under fully developed flow conditions, which are enough to obtain representative runtime values as there is an almost negligible difference in computing time among these time steps. Generation of input/output files (e.g. IB method forces) is omitted during the simulations in order to focus the analysis on the flow solver performance.

All simulations are carried out on a cluster of the Supercomputing Wales[1] project hosting 128 nodes, each with 2 Intel Xeon E5-2670 (Sandy Bridge) processors and 64GB (DDR3-1600Mhz ECC SDR) RAM. Nodes are interconnected with an Infiniband (Connect2-X) 4x QDR/PCIe gen2 16x network infrastructure (40Gbps HS/LL QDR, 1.2μs latency). The code was compiled with Intel Fortran compiler version 16.0 using $-$`O2` $-$ `AVX` $-$ `qopenmp` flags and linked with Intel MPI library version 5.1.

4.1. Lid-driven cavity flow

Firstly, the scalability of the MPI parallelisation is analysed using the lid-driven cavity flow [56], a common benchmark case used for testing incompressible flow solvers [57]. Here a similar setup to that by Wang et al. [54] is adopted. The domain is a three-dimensional cube the sides of which are equal to one and the Reynolds number is set to 400. A Dirichlet boundary condition is set at the top lid with an imposed velocity of $(u, v, w) = (1, 0, 0)$, no-slip conditions are used at the bottom, west and east walls and slip conditions are adopted on the north and south walls. The grid resolution is uniform in the whole domain with an even distribution of Eulerian cells per processor. The time step is set variable with a CFL value of 0.5 in all cases.

The resulting flow field obtained with the mesh resolution comprising 160 grid cells along each spatial direction is shown in Fig. 6 with the contour plot of u-velocity at a transversal plane along the mid-width of the domain. Profiles of u- and w-velocities confirm the accuracy of Hydro3D to predict the flow developed in the cavity achieving a good match with those of Ghia et al. [56]. Scalability of the code is assessed for three grid-resolutions using five different number of cores, namely 1, 8, 64, 125 and 512 (512 is the largest number of cores available on the cluster). Details of mesh resolution (Δx_i), number of divisions along each direction ($n_i = 1/\Delta x_i$) and total number of cells (N_e) are provided in Table 1. Results of the code's speedup are presented in Fig. 7a demonstrating that Hydro3D features a good strong scalability for all grid resolutions (and especially for the finest mesh n_i=320) except when 512 CPUs are used in cases 1 and 2, i.e. those with the least number of grid cells.

As mentioned before the most expensive computation at every time step in LES solvers is the solution of the Poisson pressure equation (Eq. (10)). Fig. 7b shows that in the present cases com-

[1] Supercomputing Wales homepage: http://www.supercomputing.wales.

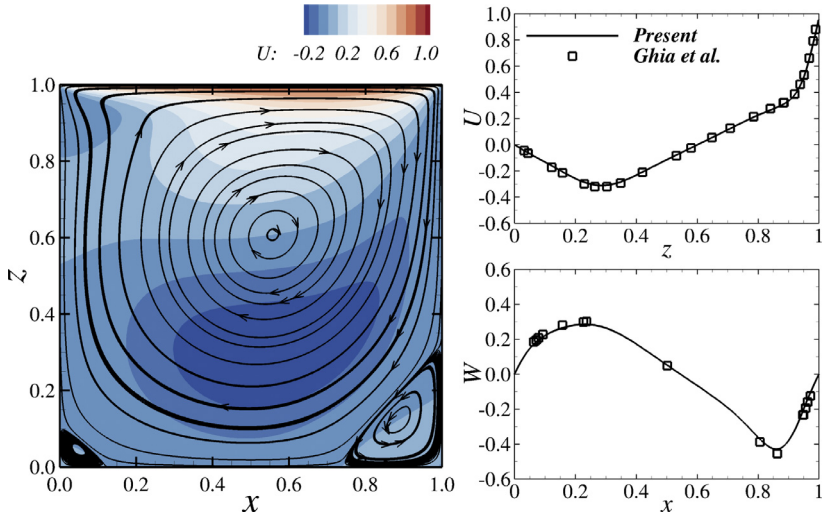

Fig. 6. Velocity field and validation of the coarse-grid lid-driven cavity flow using data of Ghia et al. [56].

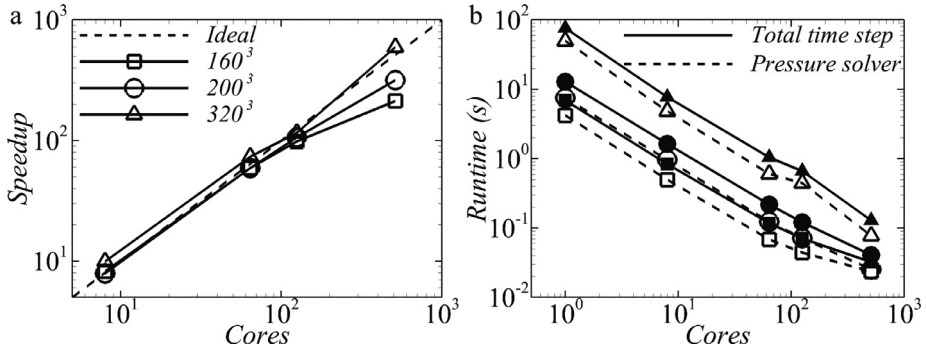

Fig. 7. Lid-driven cavity flow case results. (a) Speedup obtained for the different resolutions using up to 512 cores and (b) runtime associated with the pressure solver (blank symbols) and total time step (filled symbols).

Table 1
Details of mesh resolution, number of divisions per spatial direction and total number of fluid cells for the three scenarios used in the lid-driven cavity flow.

Case	Δx_i	n_i	N_e
1	0.00625	160	$4.096 \cdot 10^6$
2	0.00500	200	$8.000 \cdot 10^6$
3	0.003125	320	$32.768 \cdot 10^6$

Table 2
Details of the mesh resolutions tested, number of Eulerian cells in the fluid domain, Lagrangian markers conforming the airfoil shape, sub-domains and sub-domains hosting Lagrangian markers.

Case	Mesh	Δx/c	N_e	N_L	N_d	N_{dLm}
1.a	M1	0.0125	$13.32 \cdot 10^6$	186,240	57	24
1.b	M1	0.0125	$13.32 \cdot 10^6$	186,240	114	48
1.c	M1	0.0125	$13.32 \cdot 10^6$	186,240	171	72
2.a	M2	0.0100	$52.59 \cdot 10^6$	357,000	114	48
2.b	M2	0.0100	$52.59 \cdot 10^6$	357,000	171	72

puting the pressure solver always takes more than 50% of the total time and for those cases using less than 8 tasks this increases up to 70%. Overall, good strong scalability is obtained as the mean runtime needed to compute the pressure equation decreases almost linearly when increasing the number of cores.

4.2. Simulation of a vertical axis tidal turbine

This section assesses the parallelisation performance of Hydro3D for the simulation of the flow past a Vertical Axis Tidal Turbine (VATT). In contrast to the lid-driven cavity flow, this simulation requires the computation of moving Lagrangian markers and the hybrid MPI/OpenMP parallelisation approach of the code is employed. The accuracy of these simulations using the immersed boundary method was successfully validated in previous works [6,58].

The turbine operates in a hydraulic flume with an incoming velocity U_0 of 2.3m/s and comprises three cambered NACA 0018 blades with a chord length of $c = 0.032$ m. The radius of the turbine is $R = 2.73c$ and it rotates at an imposed rotational speed of $52.57 rad/s$ attaining a tip speed ratio of 2, i.e. the tangential speed of the blades is twice the incoming flow velocity. The resulting Reynolds number based on the blade's chord is $Re_c = cU_0/\nu = 73,600$. The flume's cross-section measures $32c$ by $18c$ in streamwise and transversal directions, and these dimensions are kept in the computational domain while extending vertically $2c$. The turbine centre is $x = 10c$ away from the inlet and centered in the spanwise direction. The mesh resolution is uniform in x- and y-directions, i.e. $\Delta x = \Delta y$, while it is doubled in the normal direction. The kernel function ϕ_4 is used in the immersed boundary method [6,58].

The efficiency of the hybrid parallelisation strategy is tested with two different mesh resolutions, namely M1 and M2, which are chosen based on the mesh convergence study of Ouro et al. [6]. Table 2 details the mesh configurations regarding the nor-

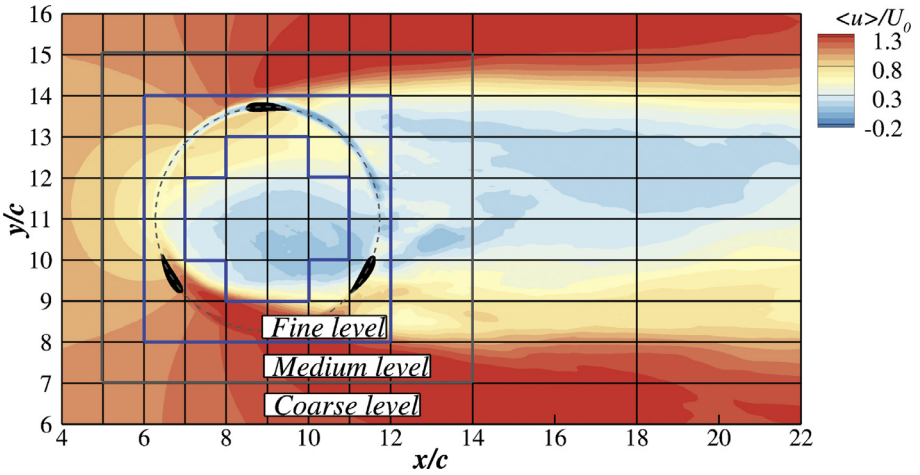

Fig. 8. Zoom-in of the Eulerian sub-domains distribution and the VATT blades with contour of the normalised time-averaged streamwise velocity. The LMR levels are also indicated.

malised grid resolution ($\Delta x/c$), number of Eulerian cells in the entire fluid domain (N_e), Lagrangian markers comprising the turbine (N_L), number of domains (N_d) and number of domains hosting Lagrangian markers (N_{dLm}).

Fig. 8 shows a zoom-in of the fluid domain of the horizontal plane at $z/c = 1$ with contours of normalised mean streamwise velocity. The domain is divided into 57 sub-domains in the x-y plane, meaning that the code runs 57 MPI processes when using one vertical layer of sub-domains, whilst 114 and 171 MPI processes are used when adopting two or three layers in the vertical, respectively. The turbine blades move within the finest LMR level but only 24 of the 36 sub-domains deal with Lagrangian markers during the simulation, which means that 24 MPI processes can require to spawn OpenMP threads.

The scalability of the scheme is assessed for 57, 114 and 171number of sub-domains for the coarse resolution (M1) while for the fine resolution (M2) only 114 and 171number of sub-domains are used. Each case is tested with a pure MPI configuration (meaning that no additional OpenMP thread is spawned), and using the hybrid MPI/OpenMP scheme with 2 and 3 OpenMP threads for those MPI processes computing IB markers, i.e. each MPI process spawns 1 or 2 additional OpenMP threads. The procedure to determine the processes computing the IB bodies is straightforward when the body moves with a prescribed motion, e.g. a circular movement described by a VATT, and for this case the sub-domains requiring multi-threading are the ones enclosed within the blue boundary in Fig. 8.

Results of runtime and speedup obtained with the pure MPI and hybrid MPI/OpenMP schemes on the VATT simulations are presented in Fig. 9. Note that the OpenMP Dynamic schedule directive with chunk size of 50 is used as this gave the best performance according to the test described in Appendix A. For case 1.a, using 1 and 2 additional OpenMP threads reduces the time to compute the IB method by 43% and 55% respectively, which diminish the overall time to 25% and 32%. Nonetheless, the configuration using 105 threads (57 MPI processes with 48 additional OpenMP threads) in case 1.a takes longer to run than using 114 MPI processes with no OpenMP in case 1.b. Note that having two vertical domains reduces the number of Eulerian cells used in the neighbour searching, thus the computation of the IB method also benefits from a larger domain partitioning.

For case 1.b, using 114 MPI tasks with 2 additional OpenMP threads (162 cores in total) the hybrid scheme outperforms by 12% the results obtained with the pure MPI with 171 cores. In view of these runtimes, the effectiveness of the hybrid scheme is deemed conditioned by the time spent solving the Poisson equation, as shown in Table 3. For the finer mesh resolution, M2, similar results are obtained. In case 2.a, executing the code with 2 additional OpenMP threads drops by 42% the time spent computing the IB method leading to a smaller runtime than that obtained for case 2.b with 171 MPI processes.

The speedup obtained with every hybrid configuration relative to the pure MPI scenario is presented in Fig. 9c and d for meshes M1 and M2, respectively. The average speedup gained to compute the IB method with 1 additional thread is about 1.7, which in terms of speedup based on the total runtime is 1.3. However, when using 3 threads the speedup increases up to about 2.2 and 1.4 related to the IB method and total runtime per time step. Note that the slope of the speedup curves related to the total runtime flattens when adopting 3 threads as a result of the time spent solving the Poisson equation again becoming the most expensive computation over the IB method.

The percentage of time spent on the different stages of the LES solver to advance the simulation in time for the pure MPI configurations is summarised in Table 3. It is appreciated that the IB method consumes from 48% to almost 60% of the computing time whilst resolving the Poisson equation takes about 30–38%, which indicates that the IB method arises as the code bottleneck. This is noticeable when increasing from 114 to 171 cores, i.e. 1.5 times more resources, runtime time drops by 14% and 21% for cases 1 and 2 respectively.

Hybrid MPI/OpenMP performance for different kernel functions

In the direct forcing IB method the accuracy and smoothness of the interpolation can be improved increasing the number of neighbours [42]. However, this is not free-of-charge and brings additional computational overhead due to a larger number of operations in the neighbour searching. Here, three delta functions ϕ_3, ϕ_4 and ϕ_4^* are examined which use 27, 64 and 125 neighbours respectively. Fig. 10 presents the mean runtime per time step and that corresponding to the IB method with mesh M1. For all cases with pure MPI configuration the computing time spent on the IB method increases about 1.8 times using ϕ_4 and 3.4 times using ϕ_4^* when compared to the runtime obtained with ϕ_3, which uses the least number neighbours in the interpolation.

The runtime obtained with 114 MPI processes and 1 additional OpenMP thread (162 cores in total) using ϕ_4^* and ϕ_4 is 22% and 12% lower, respectively, than that with 171 MPI processes without OpenMP threads. In the case of ϕ_3, the hybrid scheme does not outperform the pure MPI performance as the resolution of the

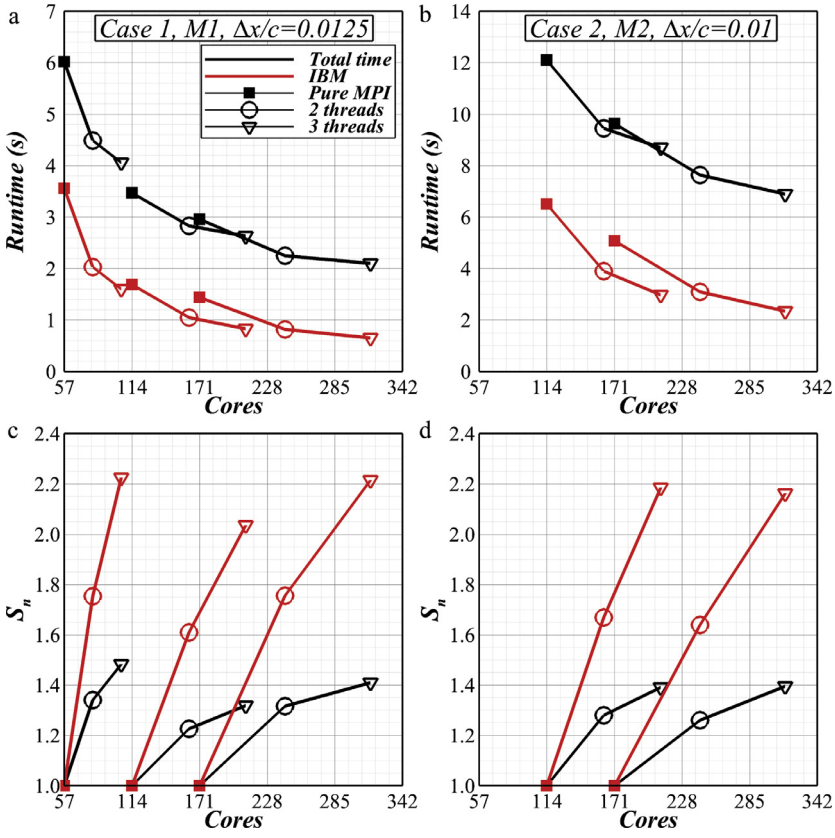

Fig. 9. Physical average runtime spent computing the IB method and total time per time step for meshes M1 (a) and M2 (b). Speedup in the code's performance regarding the IB method computations and the total time per time step for meshes M1 (c) and M2 (d).

Table 3
Percentage of the mean runtime spent on computing convection-diffusion, pressure equation, IB method, and average time (in seconds) per time step for the pure MPI configurations.

Case	N_d	Convection-diffusion	Pressure	IB method	Mean time step (s)
1.a	57	10.80 %	28.41%	59.14%	6.02
1.b	114	11.05%	38.66%	48.26%	3.44
1.c	171	10.98%	38.18%	48.65%	2.96
2.a	114	9.13 %	34.71%	53.80%	12.10
2.b	171	9.55 %	35.62%	52.75%	9.63

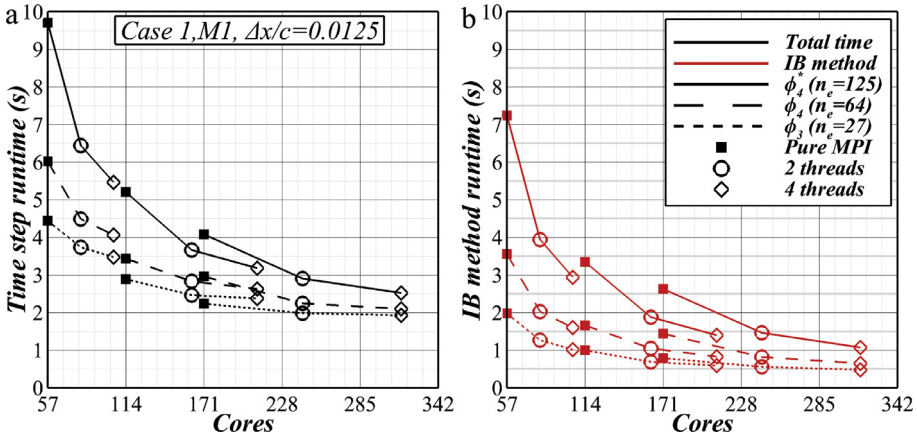

Fig. 10. Comparison of the mean (a) total time step runtime and (b) runtime computing the IB method using different kernels.

Table 4
Values of speedup in the IB method calculation, speedup based on the average time per time step and parallel efficiency.

Case	N_d	N_{omp}	$N_{threads}$	T_{IB}/T_{Press}	S_n^{IB}	$S_n^{time\_step}$	E_n
ϕ_3							
1.a	57	2	81	0.743	1.559	1.187	0.835
1.b	114	2	162	0.520	1.441	1.170	0.823
1.c	171	2	243	0.523	1.402	1.126	0.792
2.a	57	3	105	0.596	1.941	1.227	0.693
2.b	114	3	210	0.440	1.645	1.214	0.659
2.c	171	3	315	0.444	1.635	1.161	0.630
ϕ_4							
1.a	57	2	81	1.187	1.754	1.341	0.943
1.b	114	2	162	0.789	1.518	1.216	0.855
1.c	171	2	243	0.766	1.756	1.316	0.926
2.a	57	3	105	0.936	2.225	1.483	0.805
2.b	114	3	210	0.624	2.000	1.308	0.710
2.c	171	3	315	0.607	2.215	1.410	0.765
ϕ_4^*							
1.a	57	2	81	2.304	1.838	1.508	1.061
1.b	114	2	162	1.417	1.778	1.421	1.000
1.c	171	2	243	1.364	1.794	1.402	0.987
2.a	57	3	105	1.690	2.463	1.777	0.965
2.b	114	3	210	1.053	2.393	1.635	0.888
2.c	171	3	315	1.005	2.447	1.619	0.879

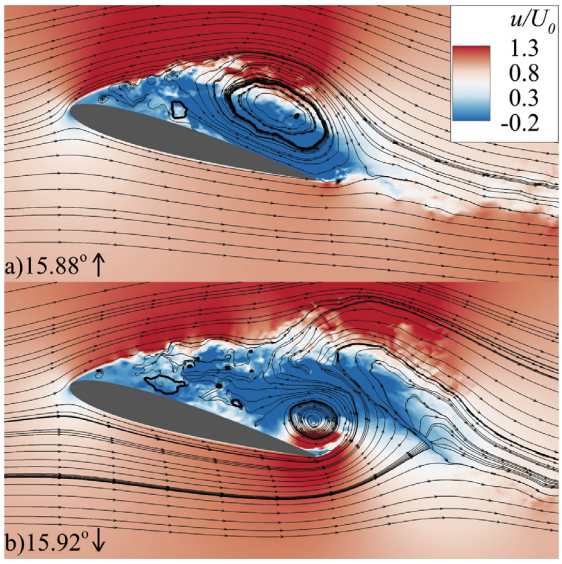

Fig. 11. Contours of normalised streamwise velocity contours and flow streamlines around the pitching NACA 0012 at a) $\alpha = 15.88^\circ \uparrow$ and b) $15.92^\circ\downarrow$.

Poisson equation is constantly more time-consuming than the IB method. An effective measure of multi-threading performance is through parallel efficiency evaluated as,

$$E_n = \frac{nT_n}{mT_m} \tag{19}$$

which compares the runtime obtained with n and m number of cores. Details of the different hybrid configurations with their speedup and parallel efficiency calculated in reference to the respective pure MPI configurations are given in Table 4, together with the ratio between time computing the IB method (T_{IB}) and pressure solver (T_{Press}) for the different kernel functions. N_{omp} represents the number of additional OpenMP threads and $N_{threads}$ the total number of threads, summing the ones running MPI processes and OpenMP threads.

The performance of adding multi-threading improves for ϕ_4^* as speedup values based on the total time step are above 1.4 for all cases using 2 threads and 1.6 when using 3 threads. This is also reflected in good parallel efficiency values being approx. 1.0 and 0.88 for 2 and 3 threads, respectively. Values of E_n for ϕ_4 decrease to a range between 0.85 and 0.94 when using 2 threads and from 0.71 to 0.80 for 3 threads. Speedup results for ϕ_3 are relatively low (< 1.2) as the computational load of the IB method compared to that of Poisson pressure equation, i.e. T_{IB}/T_{Press} is constantly lower than 0.75. The resulting parallel efficiency shows values mostly over 0.80 for 2 OpenMP threads while dropping to 0.60 for 3 threads. Results suggest that it should be borne in mind the balance between the higher-order kernels being more accurate, e.g. ϕ_4^*, and their computational expense.

Overall good parallel efficiency is obtained with the hybrid MPI/OpenMP scheme whenever the ratio T_{IB}/T_{Press} is greater than one, which seems a good indicator of whether extra computational resources should be added to pure MPI ($T_{IB}/T_{Press} < 1$) or multi-threading ($T_{IB}/T_{Press} > 1$).

4.3. Simulation of a pitching airfoil under dynamic stall

The simulation of pitching airfoils undergoing dynamic stall is of interest to many fields, as it is a key phenomenon in the aerodynamics of helicopters, micro-aerial vehicles or wind and tidal turbines. Despite its relevance in such variety of flows, the understanding of the dynamic stall is still not complete due to its remarkably complex nature depending on a large number of flow and kinematic variables, e.g. flow or pitching conditions [59].

In the simulation of pitching airfoils, the velocity gradients over the airfoil surfaces need to be well-resolved in order to accurately capture flow phenomena such as flow separation, laminar-to-turbulent boundary layer transition during upstroke motion, or boundary layer reattachment during pitch down. Therefore, eddy-resolving approaches, such as LES, provided with very fine meshes are required to obtain trustworthy results. The computational load of simulating these moving bodies is notably high mainly due to extra computations to re-allocate variables when body-fitted or chimera methods are used [60] or to re-construct the interpolation functions using the immersed boundary method [28].

Here the hybrid parallelisation scheme is tested in the simulation of a NACA 0012 [61] under a pitching motion using the IB method with kernel function ϕ_4 and whose accuracy was successfully proven in Ouro et al. [28]. In the present case an airfoil with chord length $c = 0.15$ m oscillates sinusoidally with a constant pitching frequency (ω) around its gravity centre located at $0.25c$ away from the leading edge. The angle described by the solid at any time is calculated as,

$$\alpha(t) = \alpha_0 + \Delta\alpha \cdot \sin(\omega t) \tag{20}$$

where α_0 is the pre-set angle with value 10°, $\Delta\alpha$ is the angle amplitude equal to 6°, and ω is the oscillation frequency equal to 0.32rad/s. The resulting maximum and minimum pitch angles are 16° and 4°, respectively. Fig. 11 shows the flow developed over the moving airfoil at $\alpha = 15.88^\circ \uparrow$ and $15.92^\circ\downarrow$. Fig. 11a represents the last stage of the upstroke motion with the leading edge vortex dominating the flow over the airfoil's suction side also observed during the experiments [61]. This energetic large-scale structure is shed once the airfoil achieves its maximum angle of attack and a trailing edge vortex is then generated as observed in Fig. 11b.

The performance of the hybrid MPI/OpenMP scheme is tested against pure MPI runs using two grid resolutions. Details are provided in Table 5 together with total number of Eulerian cells N_e and Lagrangian markers N_L. Mesh P1 features 320 markers uniformly distributed over both pressure and suction sides of the airfoil while 400 markers are adopted in mesh P2. These are selected based on the mesh sensitivity study performed in [28] that provided an accurate resolution of the flow. The grid is uniform in x- and y-directions while in the spanwise direction it is $\Delta z = 2\Delta x$.

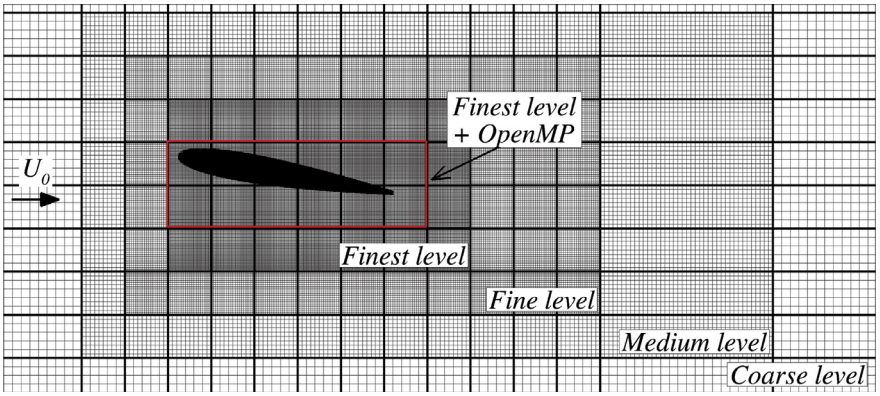

Fig. 12. Distribution of the different resolution levels on the mesh used for the pitching airfoil simulations. Red contour bounds the sub-domains requiring OpenMP threads.

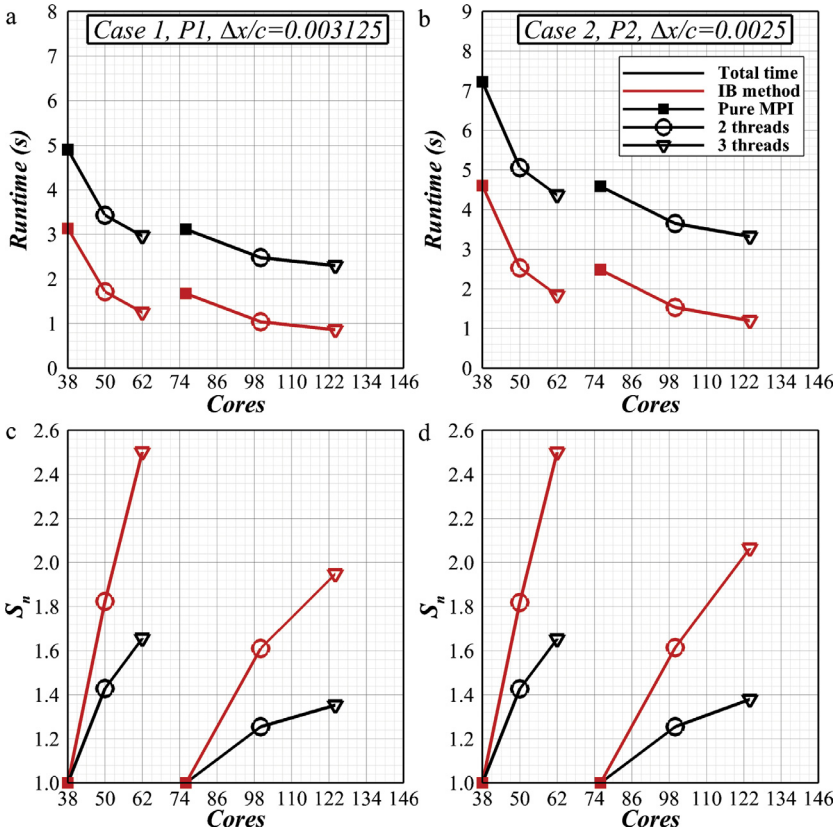

Fig. 13. Performance of the hybrid MPI/OpenMP parallelisation on the pitching airfoil simulations.

Table 5
Details of the mesh resolutions used for the pitching airfoil simulations.

Case	Mesh	$\Delta x/c$	N_e	N_L	N_d	N_{dLm}
1.a	P1	0.003125	$8.0 \cdot 10^6$	235,040	38	12
1.b	P1	0.003125	$8.0 \cdot 10^6$	235,040	76	24
2.a	P2	0.002500	$10.14 \cdot 10^6$	364,224	38	12
2.b	P2	0.002500	$10.14 \cdot 10^6$	364,224	76	24

Four levels of LMR are adopted to construct the fluid mesh as depicted in Fig. 12 with the airfoil embedded within the finest level. This allows to reduce the total number of fluid cells required to perform these fine-grid simulations with a relatively affordable number of computational resources. The sub-domains dealing with IB markers that may require multi-threading are indicated by the red boundary in Fig. 12. Analogously to the VATT case, the fluid mesh P1 comprises a single layer of sub-domains in the spanwise direction, so up to 12 MPI processes can spawn OpenMP threads. In mesh P2 this number is doubled as two domains are distributed along the spanwise direction. Note the Dynamic OpenMP scheduling directive with chunk size of 50 is used as it obtained the best performance results in Appendix A.

Fig. 13 presents the performance comparison between the hybrid MPI/OpenMP and pure MPI parallelisation schemes with mean runtime and speedup values for meshes P1 and P2. Adding an additional OpenMP thread to the MPI processes computing the Lagrangian markers in the 38 sub-domain configuration results in a reduction of the runtime from 4.9s to 3.4s, about a 30% of the total mean time step. This achieves an almost perfect parallel efficiency as 31% more cores are added. The hybrid scheme takes 3.4s per time step compared to 3.11s from the 78 pure MPI setup, i.e. needs about 9% more time to compute each time step but with 36% less

Table 6
Values of speedup in the IB method calculation, speedup based on the average time per time step and parallel efficiency for the pitching airfoil simulations.

Case	N_d	N_{omp}	$N_{threads}$	T_{IB}/T_{Press}	S_n^{IB}	$S_n^{time\_step}$	E_n
1.a	38	2	50	1.351	1.824	1.429	1.086
1.b	76	2	100	1.003	1.611	1.245	0.954
2.a	38	3	62	0.981	2.502	1.656	1.015
2.b	76	3	124	0.835	1.948	1.353	0.829

number of cores. Furthermore, the hybrid scheme in case 1.a with 3 OpenMP threads (62 cores in total) outperforms case 1.b with 78 MPI processes being the latter 5% faster using 20% less computational resources.

Analogous results are obtained for mesh P2 with the hybrid scheme using 50 cores (38 MPI processes and 12 additional OpenMP threads) performing similarly to 78 pure MPI cores and becoming faster when 2 additional OpenMP threads ($N_{dLm} = 24$) are adopted. Fig. 13c and d show that the hybrid MPI/OpenMP features a good weak scalability, as similar speedup is achieved between meshes P1 and P2 comparing analogous hybrid configurations.

Results of the relative time computing the IB method and pressure solver, speedup related to the IB method and total time step, and parallel efficiency are presented in Table 6. For cases 1.a with 2 and 3 threads, the ratio T_{IB}/T_{Press} is consistently almost equal to or above 1.0 identifying the IB method as the code's bottleneck, and consequently the parallel efficiency for these cases is over 1.0 proving the feasibility of the multi-threading scheme. For case 1.b, the relative time spent on the IB method reduces attaining an E_n of 0.954 using 2 and 0.829 using 3 threads as the IB method computation becomes faster than computing the pressure equation, i.e. the Lagrangian part is not the code bottleneck after 2 additional threads are used.

It is noteworthy that Ouro et al. [28] performed almost 300,000 iterations to simulate four full cycles of a pitching airfoil albeit their flow and kinematic conditions were different to the present case. The computational load of their LES was equivalent to 38,000 CPU hours using a pure MPI scheme with 76 cores. In view of the present results, a hybrid scheme with 38 MPI processes and 2 additional OpenMP threads would lead to approx. 22% lower computational cost as the total number of cores is 62 (38 MPI processes + 24 additional OpenMP threads) and the average runtime per time step reduces from 4.58s to 4,37s, i.e 5% less. An even better performance improvement could be achieved for those cases with large stencil kernels, e.g. ϕ_4^*.

5. Conclusions

A hybrid MPI/OpenMP parallelisation methodology designed for Eulerian-Lagrangian simulations using the in-house code Hydro3D has been presented and applied to a series of benchmark cases. The hybrid scheme features multi-threading capabilities by means of OpenMP that has been added to an already existing MPI coarse-grained parallelisation approach. OpenMP has been included into a complex master-scattering-gathering strategy which targets the load imbalance of executing the Lagrangian computations and efficiently solves the movement of dynamic bodies across Eulerian sub-domains during the simulation.

In this work, the performance of the enhanced code has been assessed for three different flow problems. The first one comprises various lid-driven cavity flow simulations with up to 512 CPUs which demonstrate the excellent scalability of the pure MPI parallelisation of Hydro3D. Next, the efficiency of the hybrid strategy has been assessed for two challenging fluid-structure interaction problems in which the Lagrangian-framework-based immersed boundary method is employed to simulate a moving solid body. The mixed parallelisation strategy outperforms the performance of pure MPI schemes in those cases in which the load from Lagrangian computations is considerably larger than the Eulerian ones, and here mainly the solution of the Poisson pressure equation requires most of the resources. It has been shown that the hybrid MPI/OpenMP scheme achieves a reduction of approx. 20% of the computational cost needed for the simulation of the pitching airfoil due to a larger computational load of the Lagrangian part than in the case of the turbine. This performance increase can have a notable impact in many CFD fields considering the huge expense of most LES or DNS applications allowing to carry out a certain simulation in a shorter time using less computational resources.

The hybrid MPI/OpenMP scheme was further analysed for different kernel functions used in the immersed boundary method, which showed that the relative performance of the mixed strategy improves when the number is larger. Good parallel efficiency values close to the unity are reported when 64 and 125 neighbours were adopted in the interpolations. The OpenMP parallelisation was further refined using Dynamic scheduling with chunk size of 50 which performed best with a speedup of 1.3–1.4 times the default Static directive.

The presented test-cases used the standard direct forcing equations without additional fluid-structure interaction algorithm, such as the ones use for deformable bodies, which require to solve a larger number of equations meaning they can benefit even more from the proposed hybrid strategy. Multiphase techniques, such as Lagrangian particle tracking, can also take advantage of the mixed MPI/OpenMP strategy which will be analysed in the future.

Acknowledgements

This research was partially funded by EPSRC under the grant EP/K502819/1 and information on the data underpinning the results presented here can found at Cardiff University data catalogue at DOI: 10.17035/d.2017.0033982819. The simulations were carried out in the facilities of the Supercomputing Wales, a project partly funded by the European Regional Development Fund (ERDF) via Welsh Government. The authors would like to thank Thomas Green, from the Advance Research Computing at Cardiff University (ARCCA) for his generous advice.

Appendix A. Analysis of scheduling directives

OpenMP allows the developer to control the way threads are scheduled and assigned to physical cores. The scheduling of the threads can have a significant impact on the performance of the code, as it directly affects the way that memory is accessed [62]. The impact of the following OpenMP directives is assessed:

- *Static*: the number of chunks the loop is split into is equal to the number of threads. This is the default schedule directive.
- *Dynamic*: the iterations from the loop are divided in chunks of *n*-size. By default *n* is 1 but can be modified. This schedule works with a first-come first-served basis.
- *Guided*: similar to Dynamic but the specified *n*-size chunk corresponds to the largest piece of work. Thereafter, the new chunk size is approximately equal to the iterations left in the loop divided by the number of threads. This exponential decreasing of the chunk size makes Guided to have fewer synchronizations than Dynamic but adds an extra computational cost due to communication and distribution.

The performance of these OpenMP scheduling policies are assessed for the VATT configuration with case 1.b using 2 and 3

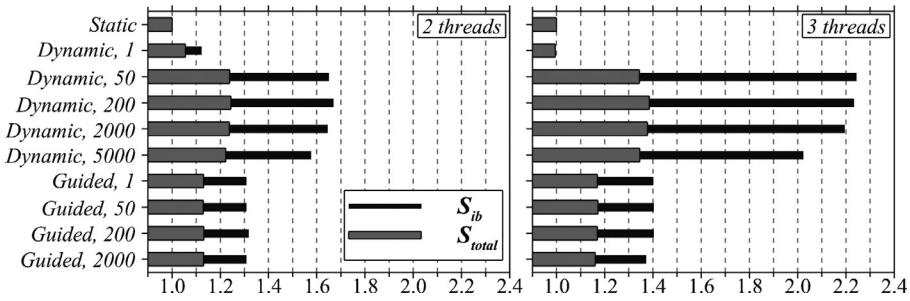

Fig. A.1. Comparison of the relative speedup computing the IB method (S_{IB}) and total time step (S_{total}) using different schedule directives in OpenMP with 2 and 3 threads.

threads. Results are presented in Fig. A.1 comparing the speedup values relative to the IB method S_{IB} and total time step S_{total} which are calculated based on the runtime obtained the Static directive. The greatest speedup is achieved with the Dynamic directive using a chunk size larger or equal to 50 with $S_{IB} = 1.6$ and $S_{total} = 1.25$ using 2 threads, and $S_{IB} = 2.25$ and $S_{total} = 2.35$ using 3 threads. Chunk sizes between 50 and 2000 achieve a significant performance increase with any thread number, while a larger chunk size of 5000 experiences a decrease in speedup, being this noticeable for 3 threads. The Guided directive provides better performance than Static but lower than that achieved with Dynamic.

The speedup obtained with the Dynamic directive and chunks equal or larger than 50 are remarkable, suggesting that the performance of multi-threading in the hybrid paralellisation scheme is somewhat compromised by the scheduling directives. This becomes more relevant when a large number of threads are used. Therefore, the improved speedup obtained in case 1.b using 114 MPI processes and 2 additional OpenMP threads over the pure MPI with 171 processes could be only achievable using the Dynamic directive with chunk sizes of 50–200.

References

[1] Stoesser T. Large-eddy simulation in hydraulics: Quo Vadis? J Hydraulic Res 2014;52(4):441–52. doi:10.1080/00221686.2014.944227.

[2] Sotiropoulos F. Hydraulics in the era of exponentially growing computing power. J Hydraulic Res 2015;53(5):547–60. doi:10.1080/00221686.2015.1119210.

[3] Rodi W. Turbulence modeling and simulation in hydraulics: a historical review. J Hydraul Eng 2017;143(5):1–20. doi:10.1061/(ASCE)HY.1943-7900.0001288.

[4] Constantinescu G. LE Of shallow mixing interfaces: a review. Environ Fluid Mech 2014;14(5):971–96. doi:10.1007/s10652-013-9303-6.

[5] Xie Z. A two-phase flow model for three-dimensional breaking waves over complex topography. Proc R Soc A 2015;471(2180). doi:10.1098/rspa.2015.0101.

[6] Ouro P, Stoesser T. An immersed boundary-based large-eddy simulation approach to predict the performance of vertical axis tidal turbines. Comput Fluids 2017;152:74–87. doi:10.1016/j.compfluid.2017.04.003.

[7] Fraga B, Stoesser T, Lai CC, Socolofsky SA. A LES-based EulerianLagrangian approach to predict the dynamics of bubble plumes. Ocean Modell 2016;97:27–36. doi:10.1016/j.ocemod.2015.11.005.

[8] Tutkun B, Edis FO. An implementation of the direct-forcing immersed boundary method using GPU power. Eng Appl Comput Fluid Mech 2017;11(1):15–29. doi:10.1080/19942060.2016.1236749.

[9] Cevheri M, McSherry R, Stoesser T. A local mesh refinement approach for large-eddy simulations of turbulent flows. Int J Numer Methods Fluids 2016;82:261–85. doi:10.1002/fld.421.

[10] Valero-Lara P, Igual FD, Prieto-Matías M, Pinelli A, Favier J. Accelerating fluid-solid simulations (Lattice-Boltzmann & Immersed-Boundary) on heterogeneous architectures. J Comput Sci 2015;10:249–61. doi:10.1016/j.jocs.2015.07.002.

[11] Domínguez JM, Crespo AJC, Valdez-Balderas D, Rogers BD, Gómez-Gesteira M. New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. Comput Phys Commun 2013;184(8):1848–60. doi:10.1016/j.cpc.2013.03.008.

[12] Gopalakrishnan P, Tafti D. Development of parallel DEM for the open source code MFIX. Powder Technol 2013;235:33–41. doi:10.1016/j.powtec.2012.09.006.

[13] Yang S, Luo K, Fang M, Zhang K, Fan J. Parallel CFD-DEM modeling of the hydrodynamics in a lab-scale double slot-rectangular spouted bed with a partition plate. Chem Eng J 2014;236:158–70. doi:10.1016/j.cej.2013.09.082.

[14] Liu H, Tafti DK, Li T. Hybrid parallelism in MFIX CFD-DEM using OpenMP. Powder Technol 2014;259:22–9. doi:10.1016/j.powtec.2014.03.047.

[15] Amritkar A, Deb S, Tafti D. Efficient parallel CFD-DEM simulations using openmp. J Comput Phys 2014;256:501–19. doi:10.1016/j.jcp.2013.09.007.

[16] Yakubov S, Cankurt B, Abdel-Maksoud M, Rung T. Hybrid MPI/OpenMP parallelization of an euler-lagrange approach to cavitation modelling. Computers & Fluids 2013;80(1):365–71. doi:10.1016/j.compfluid.2012.01.020.

[17] Shi L, Rampp M, Hof B, Avila M. A hybrid MPI-Openmp parallel implementation for pseudospectral simulations with application to taylor-Couette flow. Comput Fluids 2015;106:1–11. doi:10.1016/j.compfluid.2014.09.021.

[18] Guo X, Lange M, Gorman G, Mitchell L, Weiland M. Developing a scalable hybrid MPI/OpenMP unstructured finite element model. Comput Fluids 2015;110:227–34. doi:10.1016/j.compfluid.2014.09.007.

[19] Ouro P., Stoesser T., Fraga B., Lopez-Novoa U.. Hydro3D. 2018. doi:10.5281/zenodo.1200187.

[20] Kara S, Stoesser T, Sturm TW. Turbulence statistics in compound channels with deep and shallow overbank flows. J Hydraulic Res 2012;50(5):482–93. doi:10.1080/00221686.2012.724194.

[21] Kim D, Kim DI, Kim JH, Stoesser T. Large eddy simulation of flow and tracer transport in multichamber ozone contactors. J Environ Eng 2010;136:22–31. doi:10.1061/(ASCE)EE.1943-7870.0000118.

[22] Kim D, Stoesser T, Kim JH. The effect of baffle spacing on hydrodynamics and solute transport in serpentine contact tanks. J Hydraulic Res 2013;51(5):558–68. doi:10.1080/00221686.2013.777681.

[23] Ouro P, Fraga B, Viti N, Angeloudis A, Stoesser T, Gualtieri C. Instantaneous transport of a passive scalar in a turbulent separated flow. Environ Fluid Mech 2018;18(2):487–513. doi:10.1007/s10652-017-9567-3.

[24] Kara S, Kara MC, Stoesser T, Sturm TW. Free-Surface versus rigid-Lid LES computations for bridge-Abutment flow. J Hydraul Eng 2015;141(9):04015019. doi:10.1061/(ASCE)HY.1943-7900.0001028.

[25] Kara S, Stoesser T, Sturm TW, Mulahasan S. Flow dynamics through a submerged bridge opening with overtopping. J Hydraulic Res 2015;53(2):186–95. doi:10.1080/00221686.2014.967821.

[26] McSherry RJ, Chua KV, Stoesser T. Large eddy simulation of free-surface flows. J Hydrodyn Ser-B 2017;29(1):1–12. doi:10.1016/S1001-6058(16)60712-6.

[27] Chua KV, Fraga B, Stoesser T, Hong S, Sturm TW. Free surface flow through bridge openings in an asymmetrical compound channel. Submitted to, Under review in J Hydraulic Eng 2018.

[28] Ouro P, Stoesser T, Ramirez L. Effect of blade cambering on dynamic stall in view of designing vertical axis turbines. ASME J Fluids Eng 2018;140(6):061104. doi:10.1115/1.4039235.

[29] Bomminayuni S, Stoesser T. Turbulence statistics in an open-Channel flow over a rough bed. J Hydraul Eng 2011;137(11):1347–58. doi:10.1061/(ASCE)HY.1943-7900.0000454.

[30] Liu Y, Stoesser T, Fang H, Papanicolaou A, Tsakiris AG. Turbulent flow over an array of boulders placed on a rough, permeable bed. Comput Fluids 2017;158:120–32. doi:10.1016/j.compfluid.2017.05.023.

[31] Smagorinsky J. General circulation experiments with the primitive equations. Mon Weather Rev 1963;91(3):99–164. doi:10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2.

[32] Nicoud F, Ducros F. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. Flow Turbul Combust 1999;62(3):183–200. doi:10.1023/A:1009995426001.

[33] Uhlmann M. An immersed boundary method with direct forcing for the simulation of particulate flows. J Comput Phys 2005;209(2):448–76. doi:10.1016/j.jcp.2005.03.017.

[34] Chorin AJ. Numerical solution of the Navier-Stokes equations. Math Comput 1968;22(104):745–62. doi:10.1090/S0025-5718-1968-0242392-2.

[35] Cristallo A, Verzicco R. Combined immersed boundary/large-Eddy-Simulations of incompressible three dimensional complex flows. Flow Turbul Combust 2006;77:3–26. doi:10.1007/s10494-006-9034-6.

[36] Kara MC, Stoesser T, McSherry R. Calculation of fluidstructure interaction: methods, refinements, applications. Proc ICE Eng Comput Mech 2015;168(2):59–78. doi:10.1680/eacm.15.00010.

[37] Ouro P, Harrold M, Stoesser T, Bromley P. Hydrodynamic loadings on a horizontal axis tidal turbine prototype. J Fluids Struct 2017;71:78–95. doi:10.1016/j.jfluidstructs.2017.03.009.

[38] Ouro P, Wilson CAME, Evans P, Angeloudis A. Large-eddy simulation of shallow turbulent wakes behind a conical island. Phys Fluids 2017;29(12):126601. doi:10.1063/1.5004028.

[39] Fadlun E, Verzicco R, Orlandi P, Mohd-Yusof J. Combined immersed-Boundary finite-Difference methods for three-Dimensional complex flow simulations. J Comput Phys 2000;161(1):35–60. doi:10.1006/jcph.2000.6484.

[40] Yang J, Stern F. A non-iterative direct forcing immersed boundary method for strongly-coupled fluidsolid interactions. J Comput Phys 2015;295:779–804. doi:10.1016/j.jcp.2015.04.040.
[41] Ouro P, Cea L, Ramírez L, Nogueira X. An immersed boundary method for unstructured meshes in depth averaged shallow water models. Int J Numer Methods Fluids 2016;81(11):672–88. doi:10.1002/fld.4201.
[42] Yang X, Zhang X, Li Z, He G. A smoothing technique for discrete delta functions with application to immersed boundary method in moving boundary simulations. J Comput Phys 2009;228(20):7821–36. doi:10.1016/j.jcp.2009.07.023.
[43] Peskin CS. The immersed boundary method. Acta Numerica 2002;11:479–517. doi:10.1017/S0962492902000077.
[44] Roma AM, Peskin CS, Berger MJ. An adaptive version of the immersed boundary method. J Comput Phys 1999;153(2):509–34. doi:10.1006/jcph.1999.6293.
[45] Vanella M, Balaras E. A moving-least-squares reconstruction for embedded-boundary formulations. J Comput Phys 2009;228(18):6617–28. doi:10.1016/j.jcp.2009.06.003.
[46] Rodi W, Constantinescu G, Stoesser T. Large-Eddy simulation in hydraulics. CRC Press; 2013. ISBN 978-0-203-79757-0.
[47] Versteeg H, Malalasekera W. An introduction to computational fluid dynamics. 2nd ed. Pearson Prentice Hall; 2007.
[48] Fraga B, Stoesser T. Influence of bubble size, diffuser width, and flow rate on the integral behavior of bubble plumes. J Geophys Res 2016;121(6):3887–904. doi:10.1002/2015JC011381.
[49] Rabenseifner R, Hager G, Jost G, Keller R. Hybrid MPI and OpenMP parallel programming MPI + OpenMP and other models on clusters of SMP nodes. In: 17th Euromicro International Conference on Parallel, Distributed and Network based Processing. IEEE; 2009. p. 427–36. doi:10.1109/PDP.2009.43.
[50] Smith L, Bull M. Development of mixed mode MPI/OpenMP applications. Sci Program 2001;9:83–98.
[51] He Y, Ding CHQ. MPI and OpenMP paradigms on cluster of SMP architectures: the vacancy tracking algorithm for multi-dimensional array transposition. In: ACM/IEEE SC 2002 Conference; 2002. ISBN 0-7695-1524-X. https://doi.org/10.1109/SC.2002.10065.
[52] Ouro P. Large eddy simulation of tidal turbines. Phd thesis. Cardiff University, United Kingdom; 2017. 10.5281/zenodo.1340658.
[53] Uhlmann M. Simulation of particulate flows on multi-processor machines with distributed memory. Technical Report. May. Madrid. Spain: Department of Combustibles Fosiles, CIEMAT; 2003.
[54] Wang S, He G, Zhang X. Parallel computing strategy for a flow solver based on immersed boundary method and discrete stream-function formulation. Comput Fluids 2013;88(8):210–24. doi:10.1016/j.compfluid.2013.09.001.
[55] Aversa R, Di Martino B, Rak M, Venticinque S, Villano U. Performance prediction through simulation of a hybrid MPI/OpenMP application. Parallel Comput 2005;31:1013–33. doi:10.1016/j.parco.2005.03.009.
[56] Ghia U, Ghia KN, Shin CT. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. J Comput Phys 1982;48(3):387–411. doi:10.1016/0021-9991(82)90058-4.
[57] Ramírez L, Foulquié C, Nogueira X, Khelladi S, Chassaing J-C, Colominas I. New high-resolution-preserving sliding mesh techniques for higher-order finite volume schemes. Comput Fluids 2015;118:114–30. doi:10.1016/j.compfluid.2015.06.008.
[58] Ouro P, Stoesser T. Wake generated downstream of a vertical axis tidal turbine. In: 12th European Wave and Tidal Energy Conference (EWTEC). Cork, Ireland; 2017.
[59] Choudhry A, Leknys R, Arjomandi M, Kelso R. An insight into the dynamic stall lift characteristics. Exp Therm Fluid Sci 2014;58:188–208. doi:10.1016/j.expthermflusci.2014.07.006.
[60] Ramírez L, Nogueira X, Ouro P, Navarrina F, Khelladi S, Colominas I. A higher-order chimera method for finite volume schemes. Arch Comput Methods Eng 2018;25(3):691–706. doi:10.1007/s11831-017-9213-8.
[61] Lee T, Su Y. Surface pressures developed on an airfoil undergoing heaving and pitching motion. ASME J Fluids Eng 2015;137(5):1–11. doi:10.1115/1.4029443.
[62] Zhang Y, Burcea M, Cheng V. An adaptative OpenMP loop scheduler for hyperthreaded SMPs. In: PDCS-2004: International Conference on Parallel and Distributed Computing Systems; 2004.