Evaluating the performance of the particle finite element method in parallel architectures

**Juan M. Gimenez · Norberto M. Nigro ·
Sergio R. Idelsohn**

Abstract This paper presents a high performance implementation for the particle-mesh based method called particle finite element method two (PFEM-2). It consists of a material derivative based formulation of the equations with a hybrid spatial discretization which uses an Eulerian mesh and Lagrangian particles. The main aim of PFEM-2 is to solve transport equations as fast as possible keeping some level of accuracy. The method was found to be competitive with classical Eulerian alternatives for these targets, even in their range of optimal application. To evaluate the goodness of the method with large simulations, it is imperative to use of parallel environments. Parallel strategies for Finite Element Method have been widely studied and many libraries can be used to solve Eulerian stages of PFEM-2. However, Lagrangian stages, such as streamline integration, must be developed considering the parallel strategy selected. The main drawback of PFEM-2 is the large amount of memory needed, which limits its application to large problems with only one computer. Therefore, a distributed-memory implementation is urgently needed. Unlike a shared-memory approach, using domain decomposition the memory is automatically isolated, thus avoiding race conditions; however new issues appear due to data distribution over the processes. Thus, a domain decomposition strategy for both particle and mesh is adopted, which minimizes the communication between processes. Finally, performance analysis running over multicore and multinode architectures are presented. The Courant–Friedrichs–Lewy number used influences the efficiency of the parallelization and, in some cases, a weighted partitioning can be used to improve the speed-up. However the total cputime for cases presented is lower than that obtained when using classical Eulerian strategies.

J. M. Gimenez · N. M. Nigro
Centro de Investigación de Métodos Computacionales (CIMEC) -
UNL/CONICET, Predio Conicet-Santa Fe Colectora
Ruta Nac 168 Paraje El Pozo, Santa Fe, Argentina

N. M. Nigro · S. R. Idelsohn (✉)
Facultad de Ingeniería y Ciencias Hídricas, Universidad Nacional
del Litoral, Ciudad Universitaria, Paraje "El Pozo",
Santa Fe, Argentina
e-mail: sergio@cimne.upc.edu

S. R. Idelsohn
Institució Catalana de Recerca i Estudis Avançats (ICREA),
Passeig Lluis Companys 23, 08010 Barcelona, Spain

S. R. Idelsohn
Centro Internacional de Métodos Numéricos en Ingeniería (CIMNE),
Gran Capitan s/n, 08034 Barcelona, Spain

1 Introduction

When attempting to classify transport equation solvers, it is important to take into account the level of locality of the information needed. One can define as *implicit* a solution strategy in which a change in the solution in any part of the domain can potentially influence the solution on any other of its parts (enforcing a strong coupling between time and space). *Explicit* strategies can therefore be understood as strategies in which the solution at a point, within a time-step, is only influenced by a portion of the domain around the point (the coupling between time and space is somewhat relaxed).

While the implicit strategies are more robust the explicit ones are more efficient. This feature is attributed first to the simplicity of its computation with the minimum amount of

information needed to update the solution and principally to be in a better position, taking into account the present state of hardware technology based on the use of parallel computers and general purpose graphic processor units (GPGPU). Focusing on the efficiency, in the explicit strategies, can be also included those methods that, being implicit, lead to a linear system of equations that may be factorized once and solved each time-step with the same factorized matrix as a preconditioner.

Moreover, formulations for transport equations may be split in two classes depending on the approach selected to describe the inertial terms, namely Eulerian and Lagrangian approaches. Over the last thirty years, computer simulation of incompressible fluid flow has been mainly based on the Eulerian formulation of the fluid mechanics equations on fixed domains [1]. On the other hand, Lagrangian formulations justify their popularity solving free-surface flows or complicated multi-fluid flows in which the standard Eulerian formulations are inaccurate or, sometimes, impossible to use.

Particle-based methods, in which each fluid particle is followed in a Lagrangian manner, have been continuously used since Monaghan [2] proposed the first ideas on this approach for the treatment of astrophysical hydrodynamic problems with the so-called smoothed particle hydrodynamics method (SPH). That method was later generalized to fluid mechanics problems [3]. A similar idea to SPH was developed by Koshizuka et al. [4] for incompressible flows named moving particle simulation (MPS) methods. SPH and MPS belong to the family of the so-called meshless methods. Lately, the meshless ideas were generalized by the meshless finite element method (MFEM) [5]. This method, which uses the extended Delaunay tessellation [6] to rebuild a mesh in simulation time, takes into account the finite element type approximations in order to obtain more accurate solutions.

A natural evolution of MFEM was the particle finite element method (PFEM) [7]. The PFEM combines the particle idea with the finite element method (FEM) shape functions using an auxiliary finite element mesh. PFEM has been successfully used to solve the Navier–Stokes equations with free-surfaces [8–10], fluid-structure interaction problems [11], and fluid mechanics problems including multi-fluid flows [12]. The idea of combining meshes with moving particles is also used in the so-called material point method (MPM) [13]. However, the most important difference is that in the PFEM the particles do not represent a fixed amount of mass, but rather material points that transport only intrinsic properties. This allows using a variable number of particles, which simplifies mesh refinement due to the possibility to use more flexible element sizes.

However, only few attempts in the past thought in using Lagrangian formulation for homogeneous fluid flow can be presented. Maybe the most relevant work was done by Joe Stam [14], which solve the Navier–Stokes equations

in the context of video games. One of the reasons why the above mentioned Lagrangian methods, and particularly PFEM, are not directly applied on solving homogeneous fluid flow applications may be the important computational cost involved in the mesh, grid or neighborhood management. This severe limitation together with another imposed for the non-linearities and those proper of explicit schemes made the efficiency of original PFEM a serious problem, beyond that the method has evolved thanks to the progress done in parallel mesh generation and remeshing avoiding this serious limitation in some measure.

Although the above mentioned limitations, Lagrangian schemes have some features that show some advantages against Eulerian frames. The main one is the missing of the convective term in the balance equations, converting the non-symmetric equations in symmetric and positive definite. For Navier–Stokes equations this fact is more relevant, due to the original non-linear momentum equation is converted in linear, which allows to avoid the usage of stabilization terms with the strong consequence of not adding the typical numerical diffusion. Then, for convection dominated flows the time step in Eulerian formulations needs to be limited attending non-linearities and stability reasons. On the contrary, the Lagrangian formulations do not suffer from this inconvenience if and when the equations are integrated with good accuracy. This is the key point where the emphasis was put to develop the new generation of PFEM method's.

According to the above mentioned background, a new strategy to integrate equations which is named eXplicit Integration following the Velocity and Acceleration Streamlines (X-IVAS), was recently developed [15,16]. This form of integrating based in following the streamlines of the flow in the present time step is a better way to solve the non-linearities of the equations of the flow. Adding this strategy to the original PFEM method converges to a new methodology called particle finite element method second generation (PFEM-2). This method proves that Lagrangian formulations for homogeneous fluid flows, without free-surfaces or internal interfaces, are able to yield accurate solutions while being competitively fast when compared to state-of-the-art Eulerian solvers.

This particle-mesh method must be categorized into continuum mechanics. PFEM-2 is not a molecular dynamics method, or a method based on force equilibrium, such as smoothed particle hydrodynamics [2]. It is also not related to statistical mechanics as the Lattice-Boltzmann method [17]. PFEM-2 is intended to take advantage of the wealth that balance equations of continuum media offer, but avoiding an Eulerian formulation to ward off its excessive numerical diffusion and its limited stability in the explicit case because of the Courant–Friedrichs–Lewy (CFL) condition. The main interest of the method lies in its capacity to solve problems of industrial interest, in which usually arbitrary geometries, non-structured meshes and turbulence modeling

are employed, typically reaching extreme Reynold numbers, and also fluid-forces over solid-bodies are required. Finally, PFEM-2 is intended to be a valid alternative for that design engineers who nowadays use classical CFD software.

The competitiveness of the method depends on the performance of the implementation. A good algorithmic idea can be overshadowed if a poor programming strategy is used. Nowadays, an efficient implementation must include strategies to make intensive use of parallel hardware technologies.

An initial step is to develop a code to be executed on multi-core computers. The multi-core environment is typically non-deterministic and thread-safety issues make the implementation into multi-core a non-trivial exercise. A brief summary about the strategies to avoid race conditions and to improve the load balancing is reported in this paper. Although this work is not focused on that architecture, these issues are presented to introduce it and to compare different algorithmic solutions. Interested readers can find in [18] a more detailed description of the shared-memory implementation.

The hybrid spatial discretization used by PFEM-2 allows us to use the optimum strategy to calculate each equation term. Convection terms are solved by particles in a Lagrangian way using the X-IVAS method. On the other hand, diffusive terms are solved on a mesh in an Eulerian way using a classic FEM approach. However, keeping in memory the data of the mesh and of the cloud of particles represents an important storage cost that limits implementations that run only on a single computer. Then, a given memory capacity, FEM simulations which fit in memory might not do it with PFEM-2 simulations. Therefore, it is imperative to develop a distributed-memory implementation.

In order to develop a multi-machine implementation for PFEM-2, it is essential not to reinvent the wheel. Reusing code amortizes the formidable software development effort required to support parallel unstructured mesh-based simulations. In the open-source community there are several object-oriented toolkits and libraries [19–21], which include domain decomposition, shape functions of many orders, numerical integration, assembling and solving equation systems, etc. These packages can be extended by developers for their specific application libraries. In the current work, the `libMesh` library [22] is chosen as the basis of the development, which is an open-source OOP-C++ library to the numerical simulation using non-structured discretizations over sequential or parallel platforms.

Although libMesh solves the problem of the implementation of the FEM issues of PFEM-2, the parallel particle management remains to be developed. Particle stages in PFEM-2 include the movement of the particles along the entire domain and the updating of the nodal data from the particle data (or vice versa). To manage those features, a smart distribution of the particles over the processes is needed; in this work, a dual particle-mesh distribution is adopted.

Section 2 presents a review of the PFEM-2 method, with an explanation of the features that allow the method to run with large time-steps. Section 3 provides a summary of the shared memory implementation, focusing on load balancing problems in X-IVAS integration. Section 4 presents a detailed analysis of the extension of `libMesh` to manage particles, with an emphasis on the strategy to distribute the cloud along the sub-domains. Several Navier–Stokes simulations over Beowulf clusters are analyzed in Sect. 5, focusing on the efficiency obtained with the implementation. Moreover, the performance is compared with the widely used CFD software OpenFOAM®. Finally, some concluding remarks are provided in Sect. 6.

2 PFEM-2 method review

The goal of PFEM-2 is to solve transport-equations. The method is principally motivated by solving viscous incompressible flow equations as fast as possible. Its formulation allows to find numerical results of others scalar or vectorial transport equations, such as heat equation or turbulence modeling, and, also, the coupling between two or more of them [23].

The Lagrangian expression for a scalar transport-equation is presented in Eq. 1,

$$\frac{D\phi}{Dt} = \nabla \cdot (\alpha \, \nabla\phi) + Q \tag{1}$$

where the unknown is the scalar ϕ and Q is a external source. For example, if ϕ is the temperature, this equation is called *Heat Equation*.

On the other hand, *Navier–Stokes* equation system describes the behavior of Newtonian viscous incompressible flow. Its formulation is based on momentum-transport equation, which is normally coupled with the equation for the local mass balance. Lagrangian expressions are shown in Eqs. 2 and 3,

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \nabla \cdot (\mu(\nabla\mathbf{v}^T + \nabla\mathbf{v})) + \mathbf{f} \tag{2}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{3}$$

where the unknowns are the velocity vector $\mathbf{v}$ and the pressure p, μ is the fluid dynamic viscosity, ρ is the fluid density and $\mathbf{f}$ is an external force. Lagrangian formulation avoids solving the non-linearities of the convection in a equation system, because their issues are approached by solving particle trajectories.

It is well known [24] that the time-step selected in the solution of the transport equations is stable only for time-steps which considers the limitation imposed by two critical

dimensionless numbers: the Courant–Friedrichs–Lewy number ($CFL = \frac{|\mathbf{v}|\Delta t}{\Delta x}$) and the Fourier number ($Fo = \frac{\mu \Delta t}{\Delta x^2}$). The former concerns with the convective terms and the latter with the diffusive ones. In Eulerian formulation both numbers must be less than a constant order one to have stable algorithms. For convection dominant problems like high Reynolds number flows, the condition $CFL < 1$ becomes crucial and limits the use of explicit methods or makes the solution scheme far from being efficient. On the other hand, in diffusion dominant problems, the Fourier number becomes critical due to the time-step must decrease with the square of the grid size, doing unapproachable in problems using very refined meshes.

However, the key of the PFEM-2 algorithm is the ability to reach $CFL >> 1$ because the information is transported on the particles. One of the novelties in PFEM-2 with respect to its predecessor PFEM concerns with the integration of particle trajectory and the state variables defining the problem. This integration is performed following the streamlines computed explicitly with the information of the previous time-step. This new method was named eXplicit Integration following the Velocity and Acceleration Streamlines (X-IVAS) and it was presented in [15]. X-IVAS represents a more stable explicit time integration not limited by CFL. Briefly, the algorithm takes the streamlines as stationary on each time-step ($\mathbf{v}^n$, where n is the previous time-step), then the particle position follows that velocity field and the particle state variables are updated by the change rate determined by the physics equations (also fixed at time n).

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \int_0^1 \mathbf{v}^n(\mathbf{x}_p^{n+\tau}) \, d\tau \tag{4}$$

$$\phi_p^{n+1} = \phi_p^n + \int_0^1 \mathbf{g}^n(\mathbf{x}_p^{n+\tau}) + Q^{n+\tau} \, d\tau \tag{5}$$

$$\mathbf{v}_p^{n+1} = \mathbf{v}_p^n + \int_0^1 \mathbf{a}^n(\mathbf{x}_p^{n+\tau}) + \mathbf{f}^{n+\tau} \, d\tau \tag{6}$$

where $\mathbf{a}^n = -\nabla p^n + \nabla \cdot (\mu(\nabla \mathbf{v}^{nT} + \nabla \mathbf{v}^n))$ and $\mathbf{g}^n = \nabla \cdot (\alpha \, \nabla \phi^n)$, which are nodal variables.

Figure 1 presents a graphical description of the X-IVAS stage where each particle is transported following the streamlines fixed at time n. Temporal integration for the position and velocity can be solved using analytical expressions [25] or high-order integrators [26]. However, in this work a sub-stepping integrator inherited from STS [27] is used, which can adapt its δt depending on the local CFL number.

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \sum_{i=1}^{N} \mathbf{v}^n(\mathbf{x}_p^{n+\frac{i}{N}}) \, \delta t_p \tag{7}$$
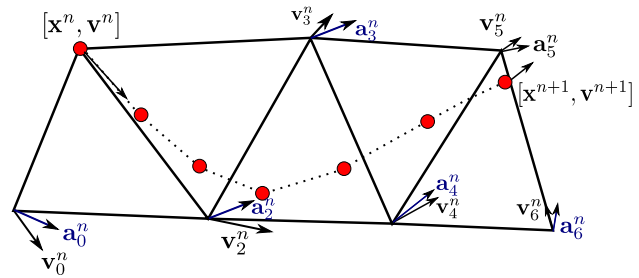
Fig. 1 X-IVAS integration. Particle position and state are updated following the frozen fields $\mathbf{v}^n$ and $\mathbf{a}^n$

The expression for δt is

$$\delta t_p = \frac{\Delta t}{K \times CFL_h} \tag{8}$$

where $CFL_h = \frac{|\mathbf{v}|\Delta t}{h}$ is the CFL number of the element that contains the particle, and K is a parameter to adjust the minimal number of sub-steps required to cross an element.

On the other hand, Eqs. (4) and (5) must be used to solve the passive scalar transport equation with a known velocity field. Also, to solve the Navier–Stokes equation, systems (4) and (6) are solved coupled with the incompressibility restriction. A typical Fractional Step Method is used to solve the coupling between the pressure and the velocity [15].

After streamline integration, nodal values must be updated with the states transported (and recently updated) by particles. There are two approaches to carry out that task, each one generating two versions of the method. The first one is called *Moving Mesh*, which creates a new mesh using the new position of the particles as nodes. The second version, named *Fixed Mesh*, projects states from particles to nodes preserving the initial mesh. These strategies are represented in Fig. 2.

This work is devoted to the *Fixed Mesh* version because avoids the remeshing at each time-step and it has the possibility of factorizing the matrices of the pressure equation and the implicit diffusion step only once. As was mentioned, avoiding the remeshing requires a projection in which a lot of particles must be employed so as not to introduce excessive numerical diffusion. However, this computational cost is lower than that of remeshing. Therefore, as it was presented in previous works, the features mentioned make PFEM-2 Fixed Mesh more efficient than Moving Mesh.

Considering that the incompressibility restriction is non-local, an implicit scheme is needed. This feature normally diminishes the efficiency of explicit incompressible flow solvers causing that the final decision about the selection of the integration scheme be pushed on fully implicit solver. In order to keep PFEM-2 explicit and because Fixed Mesh version may exploit the benefits of having a constant Poisson matrix for the pressure equation, this matrix is initially factor-
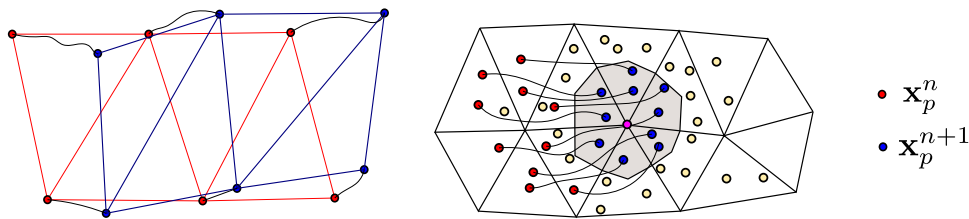
Fig. 2 Updating nodal values. *Left* the moving mesh strategy, which remeshes with the new particle positions. *Right* the approach fixed mesh is represented, where the particles project their states to the nodes of a fixed background mesh. In this paper the latter approach has been selected

ized (completely or incompletely, depending on the problem size) and then used as a preconditioner. With this fact in mind and in order to enlarge the time-step limited by the dimensionless Fourier number, the possibility to solve part of the diffusion terms in an implicit way may be included. Using the same idea for the pressure equation, we choose an initial factorization as a preconditioner for the diffusion part. This is explained in depth in [16] and [28].

Finally, a brief summary of the PFEM-2 algorithm for scalar transport problems is presented in the Algorithm 1 shown below. Next, the Algorithm 2 for PFEM-2 for incompressible flows is reported. The latter includes the implicit correction for the viscous diffusion, which allows to enlarge the time-step to overcome the critical Fourier number condition $Fo \leq 1$.

Algorithm 1 - Time Step PFEM-2 Scalar Transport

1. *Change Rate* Stage: Calculate scalar change rate on the nodes like a FEM:
   $\int_\Omega N \mathbf{g}^n d\Omega = -\int_\Omega \nabla N \alpha \nabla \phi^n d\Omega + \int_\Gamma N \nabla \phi^n \cdot \eta d\Gamma$
2. *X-IVAS* Stage: Evaluate new particles position and state following the streamlines:
   $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \int_0^1 \mathbf{v}^n(\mathbf{x}_p^{n+\tau}) d\tau$
   $\hat{\phi}_p^{n+1} = \phi_p^n + \int_0^1 (1-\theta) \mathbf{g}^n(\mathbf{x}_p^{n+\tau}) + Q^{n+\tau} d\tau$
3. *Projection* Stage: Project state to the mesh:
   $\hat{\phi}_j^{n+1} = \pi(\hat{\phi}_p^{n+1})$
4. *Implicit Correction* Stage: Implicit correction of the diffusion with FEM:
   $\phi_j^{n+1} = \hat{\phi}_j^{n+1} + \Delta t \theta \mathbf{g}^{n+1}$
   $\phi_p^{n+1} = \hat{\phi}_p^{n+1} + \Delta t \theta \pi^{-1}(\mathbf{g}^{n+1})$

---

3 Shared memory implementation

The main target of the PFEM-2 method is to look for algorithms to simulate accurately CFD problems as fast as possible, improving the current performance of general-purpose commercial and open-source codes. This goal does not mean obtaining Real Time CFD solution yet, but the aim is to change days of simulations for hours, making feasible the present challenging demands of engineering design.

Considering the goal mentioned above, a shared-memory implementation is a good option to start with the development

of efficient codes [18]. Some details relevant to the current work are explained in this section.

Algorithm 2 - Time Step PFEM-2 Incompressible Flow.

1. *Acceleration* Stage: Calculate acceleration on the nodes like a FEM:
   $\int_\Omega N \mathbf{a}_\tau^n d\Omega \approx \frac{1}{\rho} \int_\Omega N \nabla \cdot \boldsymbol{\tau}^n d\Omega = \frac{1}{\rho}(-\int_\Omega \nabla N \cdot (\mu(\nabla \mathbf{v}^n + \nabla \mathbf{v}^{nT})) d\Omega + \int_\Gamma N (\nabla \mathbf{v}^n + \nabla \mathbf{v}^{nT}) \cdot \boldsymbol{\eta} d\Gamma)$
   $\int_\Omega N \mathbf{a}_p^n d\Omega \approx \frac{1}{\rho} \int_\Omega N \nabla p^n d\Omega = \frac{1}{\rho}(-\int_\Omega \nabla N p^n d\Omega + \int_\Gamma N p^n \boldsymbol{\eta} d\Gamma)$
   $\mathbf{a}^n = -\mathbf{a}_p^n + (1-\theta) \mathbf{a}_\tau^n$
2. *X-IVAS* Stage: Evaluate new particles position and state following the streamlines:
   $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \int_0^1 \mathbf{v}^n(\mathbf{x}_p^{n+\tau}) d\tau$
   $\hat{\mathbf{v}}_p^{n+1} = \mathbf{v}_p^n + \int_0^1 \mathbf{a}^n(\mathbf{x}_p^{n+\tau}) + \mathbf{f}^{n+\tau} d\tau$
3. *Projection* Stage: Project state to the mesh:
   $\hat{\mathbf{v}}_j^{n+1} = \pi(\hat{\mathbf{v}}_p^{n+1})$
4. *Implicit Viscosity* Stage: Implicit correction of the viscous diffusion with FEM:
   $\rho \hat{\mathbf{v}}_j^{n+1} = \rho \hat{\mathbf{v}}_j^{n+1} + \Delta t \theta \mathbf{a}_\tau^{n+1}$
5. *Poisson Stage*: Search the pressure value solving the Poisson equation system with FEM:
   $\rho \nabla \cdot \hat{\mathbf{v}}_j^{n+1} = \Delta t \nabla \cdot [\nabla(\delta p^{n+1})]$
6. *Correction* Stage: Update the mesh and particle velocity with pressure and diffusion corrections:
   $\rho \mathbf{v}_j^{n+1} = \rho \hat{\mathbf{v}}_j^{n+1} - \Delta t (\mathbf{a}_p^{n+1} - \mathbf{a}_p^n)$
   $\rho \mathbf{v}_p^{n+1} = \rho \hat{\mathbf{v}}_p^{n+1} - \Delta t \pi^{-1}(\mathbf{a}_p^{n+1} - \mathbf{a}_p^n) + \Delta t \theta \pi^{-1}(\mathbf{a}_\tau^{n+1})$

The shared-memory approach introduces an issue of thread safety that can have catastrophic consequences if not addressed correctly. The load balancing problem is less severe but it can easily deteriorate the efficiency by generating non-scalable codes.

In X-IVAS stage there are no concurrence problems because a loop over all particles is done and each one updates only its own information. The adaptive sub-stepping integrator used in this work is faster than the traditional ones because it allows the algorithm not to lose computing time on particles that move more slowly (small CFL_p) and to be more accurate on critical domain zones (large CFL_p). However, using that integrator, a load balancing problem appears, especially in those cases of high variability of CFL_p. A high value of CFL_p implies to calculate a lot of sub-steps for the particle p, which typically also includes several changes of the element where the particle belongs. On the contrary, a small CFL_p requires few sub-steps normally without ele-

Fig. 3 Speed-up comparison of the X-IVAS stage on shared-memory implementation
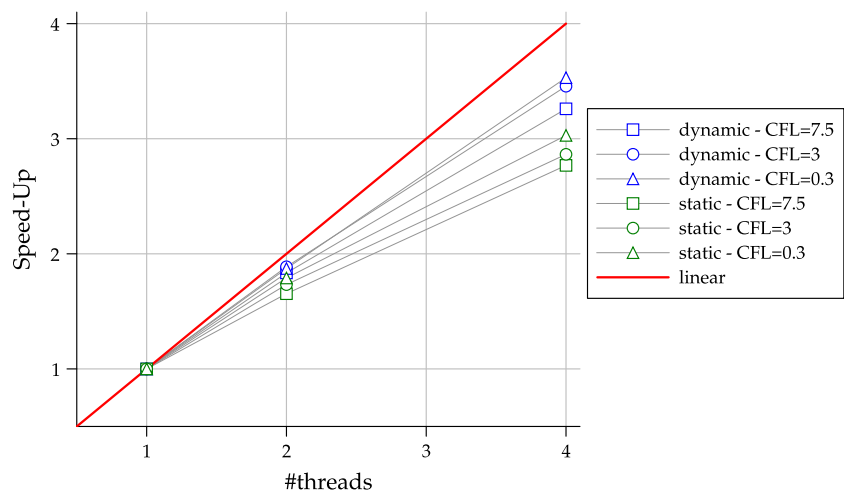
ment changes. Because PFEM-2 tries to use the largest time-step possible, high values of Courant are common.

Then, a smart particle distribution is needed. If a static scheduling is used, the load balancing problems is more likely to appear. An improved strategy is to use a dynamic scheduling.

In the current implementation, the widely used API specification for multi-platform shared-memory parallel programming called `openMP` is used. For dynamic scheduling, the option of the `for` directive `schedule(guided, chunk)` (where chunk depends on the number of particles of the problem) is selected. Section 3.1 presents a case in which the dependency of the speed-up with the scheduling strategy and with the maximum Courant ($CFL_{max} = max\{CFL_p\}$) is demonstrated.

Regarding memory management, sequential containers for elemental data, also for nodal and particle data, are selected. The particle loop chose in X-IVAS allows the implementation to store a cached copy of the particle data until its movement ends. However, when a particle changes its element, the data of new element should be read; this action is typically not cache-friendly. However, using sequential storage, along with a previous load of neighbor elements, reduces the occurrence of cache misses. Another possible solution to that issue consists of doing an elemental loop (moving all particles inside each element), but this alternative is not recommended when high Courant numbers are used because several elemental loops must be done.

On the other hand, the stage of the projection of the states from the particles ϕ_p to the nodes ϕ_j is also critical. Here, each node receives information from the particles P, which belong to its neighborhood, averages the data and updates its state. Then, the projection function π can be viewed as a multiple averaging, in which several weighting factors can be used. In the current version of the method, the linear shape function N_j of each node evaluated in the particle position $\mathbf{x}_p$ is used as weight. Then, the algorithm employed is presented in 9

$$\phi_j = \pi(\phi_p) = \frac{\sum_{p=1}^{P} N_j(\mathbf{x}_p)\phi_p}{\sum_{p=1}^{P} N_j(\mathbf{x}_p)} \qquad (9)$$

Implementing this projection stage with a particle loop could introduce thread locks when the nodal accumulator is updated. A solution can be using a coloring strategy [21] which allows data independence per interpolation kernel. However, in the current work, taking advantage of the fixed mesh and the data redundancy of the library, a loop over nodes is done. The last option avoids naturally the threads-locks and does not introduce extra cost because a particle only contributes to one node due to neighborhood definition [28].

3.1 Shared-memory tests

The problem to solve is the widely known test called flow around a cylinder in 2d. A node with a single Intel i7-2600K quadcore CPU (3.4 GHz) and 16 Gb RAM is used. Physical and geometric parameters are set to obtain a Reynold number (Re) of 1,000. The finite element mesh employed consists of 85,000 linear triangles, with 42,520 nodal points, and the grid is refined near the cylinder. Also, Δt is selected to obtain several CFL_{max} to analyze. Finally, the simulation was performed with 1, 2 and 4 `OpenMP` processes and its parallel performance is discussed in the following paragraph.

Figure 3 shows speed-ups of X-IVAS stage with different scheduling strategies for particles partition. The speed-up S_n is calculated by $S_n = \frac{T_1}{T_n}$, where T_1 and T_n are computing times with one and n processors, respectively. The reader can observe how the performance of the implementation varies, in the first place, according to the scheduling strategy selected and, next, by the CFL_{max} used. This assumption is reinforced by the results obtained in the following distributed memory benchmarks, which exhibit similar performance to static scheduling.

4 Distributed memory implementation

4.1 Motivation

FEM libraries must store several data about the used mesh. Each element requires connectivity data, neighboring information, and some pre-calculated properties. Moreover, nodal data must contain the position, state, element pointers and physical properties. A quick estimation shows that at least 100 bytes per element and 200 bytes per node are needed. As an example for a medium-size 3d incompressible flow problem with a mesh of 10^7 tetrahedra and about 2.5×10^6 nodes, FEM simulations require at least 1.5 Gb of RAM memory.

On the other hand, PFEM-2 simulations must store particle data (position, state and other info) which is approximately 100 bytes per particle. In a typical 3d PFEM-2 fixed-mesh simulation about of 10 particles per element are needed (for this reason approximately extras 10 Gb are needed only to store particle data). Therefore, when only one computer is used, restriction is much stronger in terms of problem-size for PFEM-2 than for FEM solvers. Consequently, a distributed-memory approach to run on multi-node architectures is urgently needed.

The parallel numerical framework developed uses as basis the library `libMesh`. For a detailed description of libMesh, see Kirk [22]; here some of the fundamental concepts are addressed. `libMesh` is an object-oriented library written in C++ to solve FEM problems with adaptive refinement and coarserning (AMC), which performs the communication between nodes through the standard message passing interface (MPI). Several libraries are also included in the suite, but the main ones are `PETSc` [29], `METIS` [30] and `ParMETIS` [31]. The first one is used for the solution of linear systems on both serial and parallel platforms, whereas the second and the third ones implement a domain-decomposition based on graph partitioning schemes for serial or distributed meshes, respectively.

Two main classes are added to the library: `Particle` and `PFEM2`. The former, which derives from the class `Point`, represents individually each particle of the system, whereas the latter encapsulates the entire library and has two main attributes: the cloud of particles (a `Particle`'s sequential array) and an instance of the libMesh class `EquationSystem`, which contains the mesh data and FEM systems to solve.

4.2 Domain decomposition

Critical to any implementation of distributed computing is the methodology used to distribute the global computational task to the local processor resources. In a numerical simulation, the tasks are generally aligned with integration points on a body in space; hence dividing the physical space may be used to parallelize a problem. That strategy is adopted by most FEM parallel implementations through domain-decomposition methodology, where a problem domain is geometrically divided into sub-domains that can then be distributed across the available computational resources. The sub-domains exchange data with one another through their boundaries.

On the other hand, most of the particle methods, including the one presented in this work, have a natural parallelism because force calculations and position updates can be done simultaneously for all particles. Two main ideas have been exploited to achieve this parallelism [32]. The first method is called atom decomposition of the workload, since the processor computes forces on its particles no matter where they move in the simulation domain because the assignment remains fixed for the duration of the simulation. The second method consists simply of the above mentioned spatial decomposition. Oriented to PFEM-2, the former has shown good performance for shared memory computers as was seen above, but the global character of the employed algorithms produces inter-processor communication overhead on distributed memory machines, because an updated copy of the entire mesh in each processor is needed. Therefore, the domain-decomposition is also selected for particles. Finally, the selection of domain-decomposition techniques for both mesh and particles in the implementation of PFEM-2 is here called dual particle-mesh distribution.

Regarding the update calculations, domain decomposition requires significant communication between the sub-domains and/or some degree of zone duplication to ensure accuracy. These zones along the segmented planes are called *ghost* or *virtual*. For the typical first order strategy of FEM used by PFEM-2, the zone simply refers to any immediately adjacent nodes. However, to calculate a particles trajectory using large time-steps, this dimension may extend through several layers of elements, unless a particle leaves the sub-domain and is immediately sent to the neighbor processor to continue the calculation. This approach adds inter-processor communication in the X-IVAS stage, but it eliminates the uncertainty of not knowing how many layers of ghost elements are needed to perform the trajectory. Layers definition can be a several problem when unstructured meshes are used.

As was mentioned above, the basic principle of parallelizing the X-IVAS algorithm in the domain-decomposition manner is that each CPU calculates the trajectories of its set of particles (those that reside in it at the given time-step). When a particle, due to its movement, changes the partition where it belongs, the particle data are transferred to the CPU in control of the partition the particle has entered and the control of that particle is given to that CPU [21].

An option to implement particle transfer could be an asynchronous transfer, in which the particle data is packed in a continuous data buffer and sent to the CPU that is in control of

the partition on the other side of the partition boundary. When there are no more particles to be tracked on the partition, the algorithm leaves the tracking loop and enters in a dummy loop where it checks if a particle data message has been sent to it from another CPU. If so, it accepts the message, unpacks the data, proceeds to the tracking loop and starts tracking the newly acquired particle. This approach has an issue to calculate the termination condition because a synchronous point for all CPUs is required [33]. Then, attempt to minimize the communication and due to the need for a synchronous point, the buffer of particles is only packed but not transferred until the particle loop ends. This strategy transfers particles and synchronizes simultaneously, and is selected for the current implementation.

To perform parallel particle tasks, a `Particle Communication` class is implemented, which through the *MPI* method `Alltoallv`, allows each sub-domain to send and receive a set of particles. This strategy is easy to implement; however, because it is a collective operation, only useful when few processors are used. Other possibilities must be analyzed, trying to reduce the communication when a large number of processors are used. Interchanging information only with the neighbor sub-domains through point-to-point communications can be the best alternative.

Because a particle can cross more than two sub-domains in a time-step, an external loop is needed which breaks when no more particles are transferred. Algorithm 3 presents a transcription of the code where the external loop is observed alongside the loop over the particles and the stop condition (when all processors do no have any more particles to transfer). The `for` loop, whose iteration starts from the last particle analyzed to the last particle in the current array, integrates the trajectory of each particle computing each sub-time-step in a `while` loop. There are four options at each sub-time-step: in the first case, the particle has completed its sub-time-step and must calculate the rest of the trajectory; in the second case the particle has left out the domain boundary and its computation finishes; in the third case, the particle has crossed to other sub-domain (in this case, it is queued to be sent to the other processor); finally, in the last case the particle finishes the entire time-step.

4.3 Partitioning the physical space: load balancing

In a parallel FEM calculation, the domain distribution must be done so that the number of elements assigned to each processor is the same, and the number of adjacent elements assigned to different processors is minimized. The goal of the first condition is to balance the computations among the processors. The goal of the second condition is to minimize the communication resulting from the placement of adjacent elements to different processors.

Algorithm 3 - X-IVAS over distributed-memory

```
int ini_ip = 0;
while(1){
  std::map<int,std::vector<int>
    >particles2send;
  np = vP.size();
  for(unsigned int ip=ini_ip;ip<np;ip++){
    int c=0, pid_send;
    bool next_ddt = true;
    while(next_ddt){
      c = integrateSubStep(vP[ip],pid_send);
      switch(c){
        case 0://substep complete
          break;
        case 1://out of domain
          next_ddt = false;
          break;
        case 2://out of sub-domain
          next_ddt = false;
       particles2send[pid_send].push_back(ip);
          break;
        case 3://time-step complete
          next_ddt = false;
          break;
      }
    }
  }
  ini_ip=np;
  next = particles2send.size();
  Parallel::max(next); //max All_reduce
  if(!next) break; //no particles to send
  ParticleCommunication().interchangeParticles
    (particles2send,vP); //Alltoallv
}
```

The graph partitioning strategy implemented by the library `Metis` can be used to successfully satisfy these conditions by first modeling the finite element mesh with a graph, and then partitioning it into equal parts. The user can control the distribution associating a positive weight $\eta(v)$ with each edge v of the graph.

For the case of FEM simulations, a typical selection of the weights is $\eta(v) = \eta_n(v) = \#dofs\_of\_element$, because the amount of computational task of each element is directly proportional to the number of degrees of freedom. Then, because the current implementation uses only linear simplices, after a domain decomposition each processor has approximately the same number of elements. At the beginning of the simulation, each processor creates a fixed number of particles on each own element; this guarantees that the initial load of particles over processors is also balanced. However, when the simulation starts, this distribution can be modified according to the particle movement itself.

The number of particles on each element rarely keeps constant, being in general very common to find some elements empty of particles or some elements with many more particles than that desired. Beyond the balancing problem, the main drawback is the loss of accuracy of the solution; then PFEM-2 solves this issue seeding and removing particles, as clearly explained in [28]. In addition, using those strategies, the number of particles is kept approximately constant.

An approximately constant number of particles in each sub-domain only guarantees a proper load of particles on each processor. However, due to the adaptive integrator used, that balanced distribution does not ensure a balanced work-load over each CPU, similarly to the case of static scheduling over shared-memory.

The parallel architecture in multi-core environments makes possible a dynamic scheduling to distribute the work over processors because each particle has access to the entire domain data. This feature is not possible in multi-node environments with the domain geometrically divided but, as it was explained above, the atom decomposition is not a proper choice because adds more serious problems.

A possible solution to the work-load balancing of the X-IVAS stage on distributed-memory is including information about the CFL_h on the partition algorithm through a weights array $\eta_w(V) \neq \eta_n(V)$. A dynamic solution consists of, starting from an initial partition, interchanging few elements between processors according to how the solution temporally varies, optimizing both the number of elements that are moved and the edge-cut of the resulting partitioning. However, many other parameters must be optimized, as the time interval between those repartitions and partition quality; therefore, exhaustive research is needed to achieve successful results with this solution.

An easier option to implement is using a static approach, in which the weighted partition is done only at the beginning of the simulation. It requires knowing or estimating the solution previously, which is impossible in most of the cases. Therefore, an initial simulation must be done in which the weights are calculated, saved, and used in the partitioning algorithm of a new run.

4.4 Other stages

X-IVAS implementation on distributed-memory is the main novelty of this paper. However, there are other relevant stages of the algorithm, whose implementation should be detailed.

There are many Finite Element calculations in a PFEM-2 time-step: acceleration, viscous-diffusion correction and pressure are calculated by FEM. In the PFEM-2 fixed mesh version, the benefits of having a constant Poisson matrix for the pressure equation may be exploited. This matrix is initially factorized and then used as a preconditioner. This feature also appears for the implicit diffusion step if the viscosity is not time-dependent. Then, `libMesh` library must be used smartly, assembling only once those matrices and specifying parameters to reuse the preconditioner. While in shared-memory implementation the complete Cholesky factorization is used, this option is not affordable with large problems or in parallel executions with `PETSc`. Therefore, they are being used as iterative preconditioners like diagonal, incomplete Cholesky or algebraical multigrid.

On the projection stage, the main parallel issue is the calculation of the state of the nodes over the boundary between sub-domains. To perform this task, ghost nodes are also calculated by processors. In the projection algorithm, at the end of the loop over particles, the processor owner of the boundary node receives the partial state of ghost node from the neighbor processor and calculates the complete nodal state. Because projection methods are based on weighted averages (which is a non associative operation), the parallel calculation must be done carefully: they must not be calculated as partial averages on each processor and later averaging the result again on the owner processor. Proper implementation requires, for each ghost node, calculating and sending two partial summations (numerator and denominator) and performing the division only in the owner CPU.

5 Distributed-memory tests

The distributed-memory implementation has been evaluated on our local Beowulf cluster at the Research Center on Computational Mechanics (CIMEC) [34]. The cluster has a server Intel i7-2600K 8 Gb RAM and six single socket nodes with i7-3930K hexacores CPUs (16 Gb RAM) connected by Gigabit Ethernet. In order not to introduce disturbance into the results, technologies Intel Turbo Boost and Intel Hyper-threading are disabled on the processors, giving a total of 36 computational cores of 3.4 GHz. The code was compiled with g++ 4.7.2, and it uses the libraries mpich2-1.4, petsc-3.3-p7 and libmesh-0.8.0.

5.1 Advective transport of a Gaussian hill

The transport of a Gaussian Hill problem was used to demonstrate the goodness of PFEM-2 method to solve a scalar transport problem [25]. This case also made evident the pathology that explicit Eulerian approaches suffer in solving a pure advective transport problem with $CFL > 1$. The problem consists of a Gaussian hill signal used as initial condition with no diffusion. The velocity field is a flow rotating around the center of a square domain. The Gaussian signal is displaced from the center of the domain at a certain radius and its shape makes the transported signal have a non-zero value in a limited region of the domain initially. The signal should be transported following circular path lines and preserving its original shape and its original amplitude. Figure 4 shows the problem definition.

In this paper simulations with a finer mesh are presented, in which it is possible to analyze the performance of the X-IVAS implementation, leaving the physical results of the problem at a second level. A structured 2d finite element mesh is used with a million of nodes and two million of triangular ele-
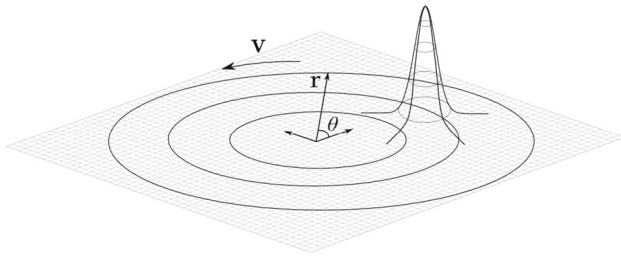
Fig. 4 Initial temperature distribution for the advective transport problem

ments to represent the fluid velocity and the temperature distributions. Because the velocity is proportional to the radius $\mathbf{v} = \mathbf{v}(\mathbf{r})$ a non-constant field of CFL_h is obtained using a structured mesh non-aligned with $\mathbf{v}$, which also depends on $\mathbf{r}$, then $CFL_h = CFL_h(r)$. Fixing $\mathbf{v}$ and δx, the only one free variable on CFL_h is Δt; then several time-steps are chosen in order to have different magnitudes of local CFL numbers.

The weight array selected for the partitioning algorithm is $\eta_n(v)$, which ensures a proper division of the total number of elements and particles over each processor. Figure 5 shows the distribution after partitioning with 1 to 16 sub-domains. However, as it was mentioned above, this strategy can insert issues of work-load balancing on streamline integration and these are more evident when large CFL are used. Due to the independence of CFL_h of the angle $CFL_h \neq CFL_h(\theta)$, the best option to balance the X-IVAS workload is partitioning by the angle but, using the *k-way* partitioner, this happens only with 2 and 4 processors. This behavior is reported in Fig. 6, which shows the speed-up S_n obtained by X-IVAS stage choosing different CFL (the maximum local CFL is reported in the Figure). For #$processors <= 4$ the speed-up is practically independent of CFL_h; however when the number of sub-domains is increased, imbalance appears and the scalability worsens with large CFL_h. The other stages of the algorithm (change rate calculation and projection) are not presented in the graph because their scalability is not dependent on the CFL_h (being approximately $S_16 = 12x$ and $S_16 = 11x$ respectively) and also their relevance in the entire wall-clock reaches only approximately 20 %.
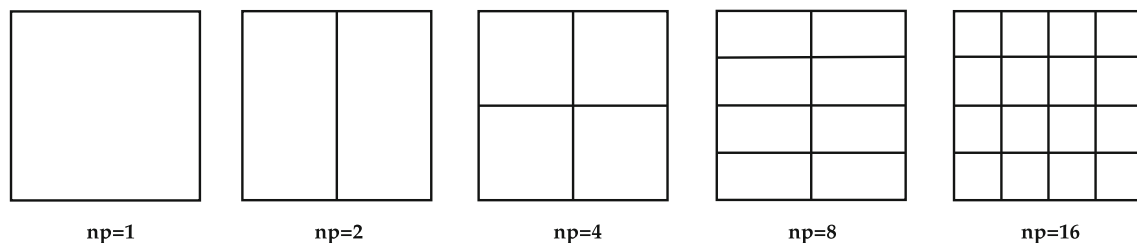
5.2 Flow around a cylinder

The flow around a cylinder is a typical benchmark for incompressible flow. Figure 7 shows the geometry used for the three dimensional (3d) case, where $D = 1$ is the diameter of the cylinder, the bounding box is $[-2.5D, -5.5D, -5.5D]$ to $[2.5D, 15.5D, 5.5D]$, with the axial direction of the cylinder over the x axis, and centered in [0 0 0]. The two dimensional (2d) case uses the same geometry without the extrusion in the axial direction.

Regarding boundary conditions, the test is run for the case $Re = 1,000$, then $U = [0; 1; 0]$ is imposed on the inflow surface, fixed pressure on the outflow, slip condition on front and back surfaces (3d cases), non-slip boundary condition on the cylinder, and $U = [0; 1; 0]$ on the upper and bottom surfaces. The initial fields are $U = [0; 1; 0]$, $p = 0$ with the fluid properties viscosity $\mu = 10^{-3}$ and density $\rho = 1$.

The calculations in 2d are done using a mesh containing 88 thousand triangular elements with 43 thousand nodes refined towards the cylinder, whereas the three dimensional mesh used has 1.6 million tetrahedral elements and 356 thousand nodes also refined towards the cylinder.

The mesh refinement is designed in order to capture the fluid forces over the cylinder more accurately. Since the CFL number depends on the inverse of the element size h, its value increases next to the cylinder. As was extensively mentioned above, the elements with large CFL will have more workload in the X-IVAS stage; then, to balance this work-load a partitioning weighting with a factor proportional to CFL_h can be done. However, this strategy unbalances the workload in all other stages, where the number of elements (FEM calculations) or the number of particles (projection and correction) must be balanced on each processor to optimize the performance.

The formula used to calculate the weight of the vertex v_j of the partitioning graph (i.e. the weight of the element e_j) is the same as that which calculates the number of sub-steps of each particle in the streamline integration:

$$\eta_w(v_j) = min\{N_{max}, max\{N_{min}, K \times (CFL_h)_j\}\} \quad (10)$$

where N_{max} and N_{min} are the maximum and minimum number of steps required to move a particle along the entire

Fig. 5 Domain partitioning with 1–16 sub-domains. Case: advective transport problem. The entire work-load is balanced with 1, 2 and 4 processors, but with 8 and 16 (and more) processors, only the number of elements and the number of particles are balanced

Fig. 6 SpeedUp X-IVAS.
Case: advective transport of a
Gaussian hill. In explicit scalar
transport PFEM-2, X-IVAS
stage represents about 80 % of
the entire wall-clock time
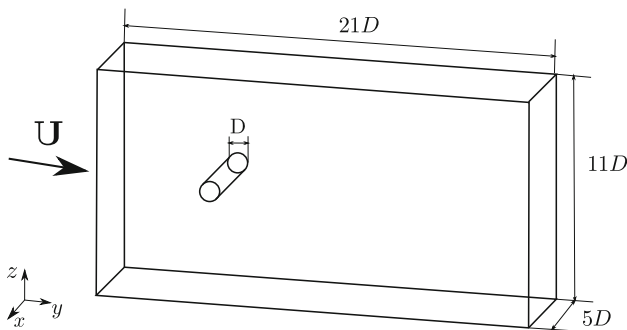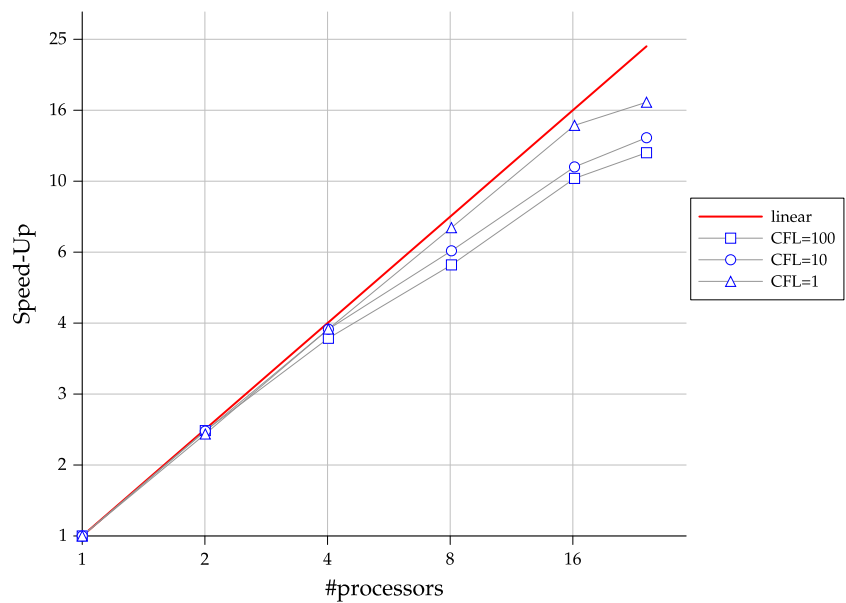
Fig. 7 Geometry flow around a cylinder in three dimensions. Two-dimensional geometry excludes the axial propagation over the x axis

time-step and K is a parameter to adjust the minimal number of sub-steps required to cross an element.

The performance of the parallelization solving the flow around a cylinder 2d with $\Delta t = 0.025$ and $CFL_{max} \approx 15$ is presented in Fig. 8 which shows the speed-up obtained by each stage of the method according to the weighted partitioning strategy selected. Table 1 presents a summary with the CPU-times for each stage. Note that although $\eta_w(v)$ improves the X-IVAS performance around $4\times$ compared with $\eta(v)$ (from $8x$ to $12x$), the influence of this stage in the total time is not very relevant. Therefore, the strategy with $\eta_n(v)$ reaches the best overall results. It should be noted that the size of the problem is not large enough to get good performance with approximately more than 10 processors.

In the three-dimensional case, the current PFEM-2 implementation and the widely used CFD software OpenFOAM® are compared. OpenFOAM® is an open source code increasingly used on engineering and industrial environments. Therefore, considering the main aim of the method here pre-

sented, it is necessary to show a screen-shot of the current status.

The main idea of the comparison is to force the time-step Δt to be the maximum such that the solver is stable and accurate. For the PFEM-2 results, the CPU-times obtained treating implicitly the viscous term (implicit diffusion as described at the beginning of the paper) are reported, and for OpenFOAM® results the solver `pimpleFoam` is selected, which is the fastest and most robust for incompressible flow because it allows the time-step to grow more than many other solvers as `icoFoam`. Absolute tolerances for `pcg` are set to 10^{-6} for each solver.

Table 2 presents the time-steps (Δt) used by each solver and shows the CFL values that each simulation reaches. In `pimpleFoam` $maxCo = 10$ is set and the time-step shown is an average of the instantaneous values used by the solver. Both solvers are able to solve with large time-steps, but results are too diffusive (drag and lift coefficients were checked) and they are considered inaccurate. Finally, in the mentioned table, the speed-up and the total CPU-time to compute 1 s of real time with 16 processors are reported. From the results it can be concluded that, keeping a similar parallel efficiency, PFEM-2 is about three times faster than the fastest incompressible solver of OpenFOAM®.

The performance of the parallelization is presented in Fig. 9, which shows the speed-up obtained by each stage of the method considering the weighting strategy selected. In this case the problem is large enough to reach good speedup with more than 10 processors. The stage of velocity correction by the pressure gradient is the most efficient because of its simplicity and the locality of the data, and it reaches approximately $S_{16} \approx 14x - 15x$.

(a)

(b)
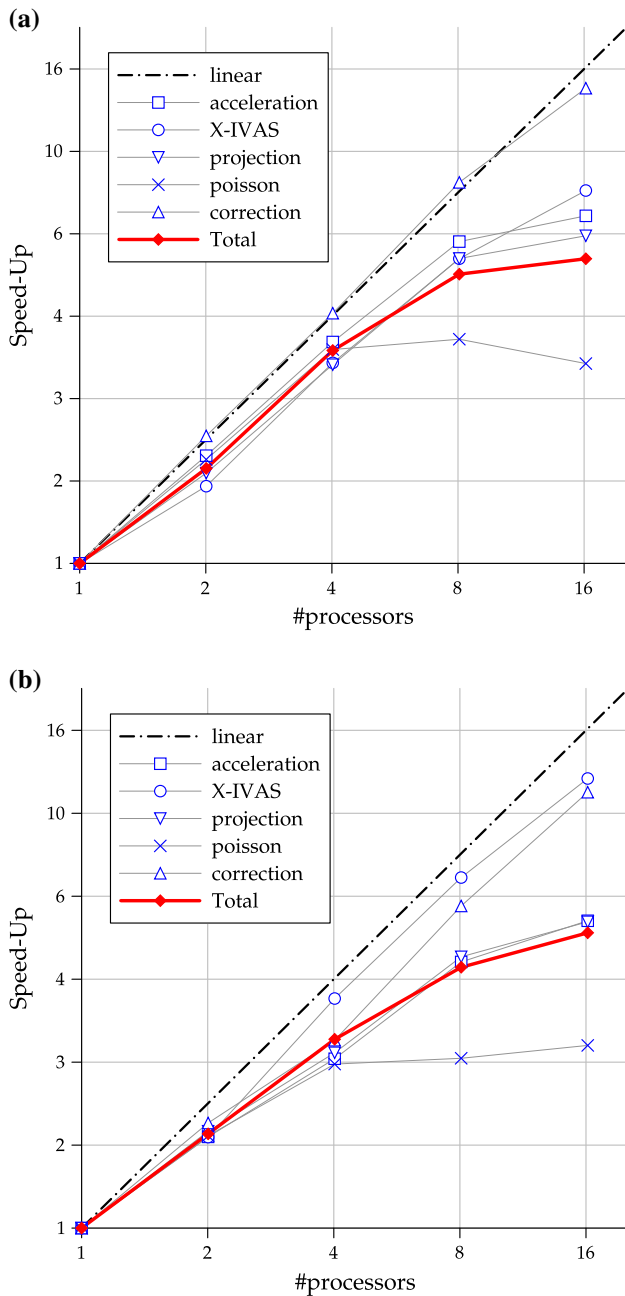
Fig. 8 Speed-Up comparison between partitioning weighted by number of degrees of freedom $\eta_n(v)$ (**a**) and partitioning weighted by $\eta_w(v_j)$ (**b**). Case: flow around a cylinder in 2d

The scalability of the FEM stages, which is inherited from the `libMesh` implementation, obtains values from $S_{16} \approx 10x$ to $S_{16} \approx 12x$ using the weighting formula $\eta_n(v)$, whereas with $\eta_w(v)$ only values from $S_{16} \approx 7x$ to $S_{16} \approx 9x$ are obtained, indicating an imbalance of the number of elements. X-IVAS stage is improved by $\eta_w(v)$, but that does not compensate the worsening of the remainder stages of the algorithm.

Table 1 Comparison table for CPU-times for the different PFEM-2 stages

Stage	$1\times$ (s)	$16 \times \eta_n(v)$ (s)	$16 \times \eta_w(v)$ (s)
Acceleration	40.5	5.81	7.36
X-IVAS	88.55	11.02	7.3
Projection	42.87	6.87	7.83
Implicit correction	33.71	11.58	12.84
Poisson	50.23	16.44	18.22
Correction	34.73	2.44	3.09
Total	290.5	52.93	56.48

Case: flow around a cylinder in 2d

Table 2 Comparison of time-steps with different solvers

Solver	Re	Δt (s)	Co_{mean}	Co_{max}	S_{16}	CPU-time (s)
PFEM-2	1,000	0.05	≈ 0.75	≈ 8	$10.45\times$	202.56
OpenFOAM®	1,000	≈ 0.025	≈ 0.5	≈ 10	$9.41\times$	613.98

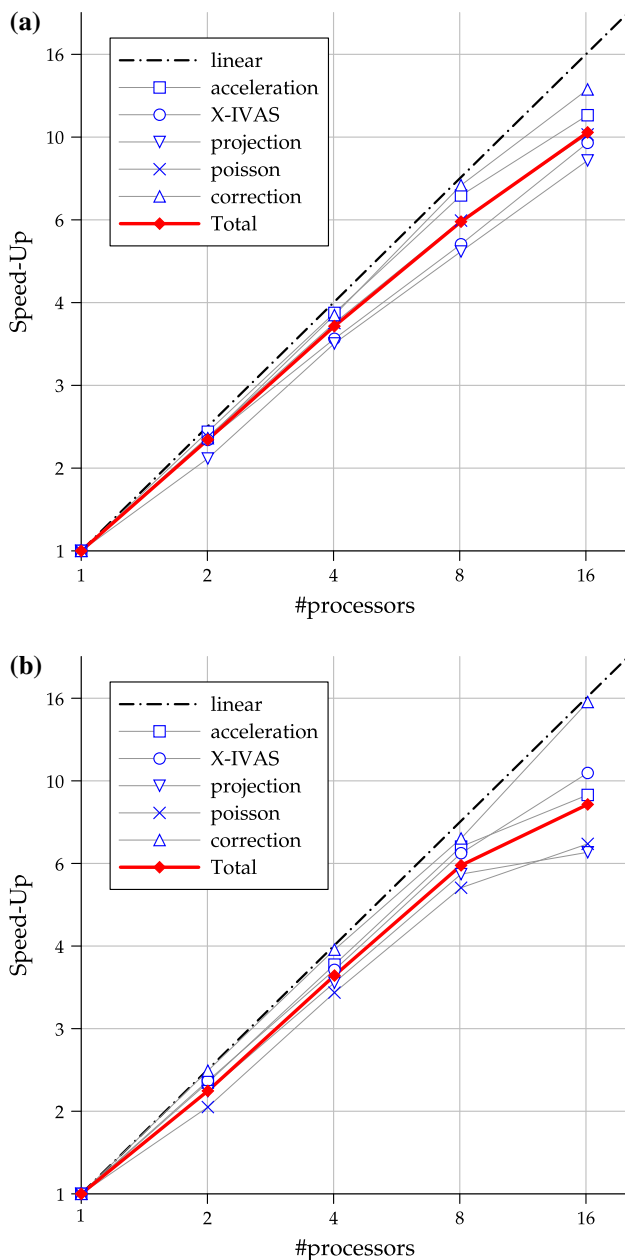Case: flow around a cylinder in 3d. PFEM-2 partitioning with $\eta_n(v)$

5.2.1 Test over infiniband cluster

It must be emphasized that an important reason for the loss of efficiency for a large number of processors in all PFEM-2 tests and OpenFOAM® is due to the interconnection network used: Gigabit Ethernet is a multi-purpose architecture, which introduces several delays in a congested network, as it happens when many nodes are computing and sending data. The employement of dedicated architectures should have a big impact over the quality of the results. Then, in this section the results for the same case of the flow around of the cylinder in 3d are presented, but executing the code over a Infiniband interconnected cluster.

The mentioned cluster has dual socket nodes, with Intel Xeon E5-2600 CPUs and 64 Gb RAM, interconnected with IB-QDR 40 Gbps (libraries used were the same as presented above). Although the cluster is more powerful than the used in the previous test, the main aim of this section is to show the dependence of the results with the interconnection network; leaving the design and solution of larger problems, that requires the entire capacity of the cluster, for next works. For the comparison, the same mesh and configuration of the test is taken. Also the partitioned domains are distributed in the same way over the nodes, it is: four processes per node. This configuration allow us to reach the main aim proposed.

Figure 10 presents the scalability of each PFEM stage and of the entire simulation using the weighting formula $\eta_n(v)$. The differences are notable: with sixteen cores a scalability of $S_{16} \approx 14.5x$ is obtained comparing with $S_{16} \approx 10.45x$ obtained with the Gigabit Ethernet cluster. Moreover, the efficiency in the Infiniband cluster is good enough also running with 32 cores, reaching a global $S_{32} \approx 26x$. Using more cores
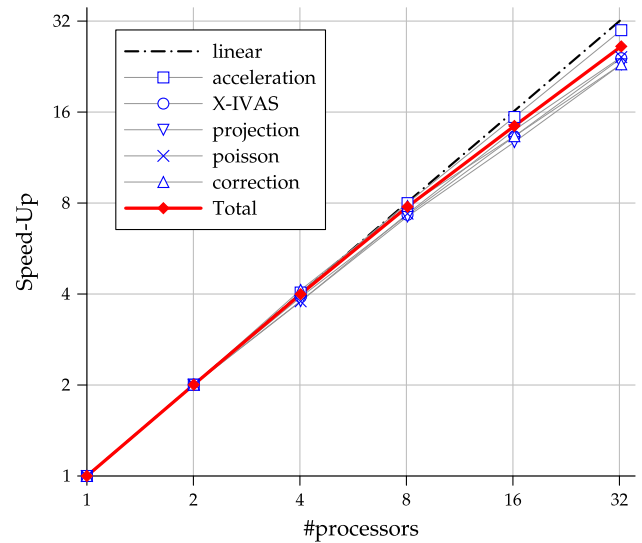
(a)

(b)

Fig. 9 Speed-Up comparison between partitioning weighted by: **a** number of degrees of freedom $\eta_n(v)$ and **b** partitioning weighted by $\eta_w(v)$. Case: flow around a cylinder in 3d

Fig. 10 Speed-up over an infiniband cluster. Case: flow around a cylinder in 3d

putations using the method to solve both scalar and vectorial transport equations.

The domain decomposition through a graph partitioner using nodal weights allows us to select, depending on the problem to solve, the stage of the algorithm that will be solved more efficiently. In simulations where X-IVAS stage dominates the calculation (this is for scalar transport problems and flow problems with low Fourier number, in which no implicit viscous-diffusion is needed), using a weight array with values proportional to the elemental Courant number improves the parallel performance of the streamline integration, therefore, of the overall simulation. In incompressible flow problems with large Fourier the overall time is controlled by FEM calculations due to the addition of the implicit correction of the viscous diffusion; therefore, the weights array must contain values proportional to the number of degrees of freedom of each element to optimize the performance.

It should be noted that the current work does not deeply analyze the performance of the solution of equation systems. The reason for this is that in this first stage, we was focused on the Lagrangian–Eulerian parallel coupling. In next works, as it was reported in this work, faster clusters with high bandwidth and low latency are required. It remains to perform a weak scalability analysis of the code, with special emphasis on analyzing a correct selection of preconditioners of linear equation systems.

Finally, the efficiency measured in terms of CPU-time for reaching a given final time in the simulation showed important advantages of the present method against OpenFOAM®. In this paper, the speed-up reached by PFEM-2 is similar to that achieved by the CFD software, but a factor approximately $3\times$ was obtained at the same level of accuracy, which places the method among the fastest. Also, the implementa-

the efficiency decays because there is not enough work for each process to overweight the communication time. But, it can be noted that this minimum limit is reached using smaller domain partitions in the Infiniband cluster.

6 Conclusions and future work

The methodology for parallelizing PFEM-2 presented in this paper is an initial approach towards massively parallel com-

tion has demonstrated good behavior over a cluster dedicated to scientific computing, which give us warranties to use it as starting point of a massively parallel code.

References

1. Donea J, Huerta A (1983) Finite element method for flow problems. Wiley, Chichester
2. Gingold RA, Monaghan JJ (1977) Smoothed particle hydrodynamics, theory and application to non-spherical stars. R Astron Soc 181:375–389
3. Monaghan J (1988) An introduction to SPH. Comput Phys Commun 48:89–96
4. Koshizuka S, Tamako H, Oka Y (1995) A particle method for incompressible viscous flow with fluid fragmentation. Comput Fluid Mech J 113:134–147
5. Idelsohn S, Oñate E, Calvo N, Pin FD (2003) The meshless finite element method. Int J Numer Methods Eng 58(6):893–912
6. Idelsohn S, Calvo N, Oñate E (2003) Polyhedrization of an arbitrary 3d point set. Comput Method Appl Mech Eng 192(22–24):2649–2667
7. Idelsohn S, Oñate E, Del Pin F (2004) The particle finite element method a powerful tool to solve incompressible flows with free-surfaces and breaking waves. Int J Numer Methods 61:964–989
8. Larese A, Rossi R, Oñate E, Idelsohn S (2008) Validation of the particle finite element method (PFEM) for simulation of the free-surface flows. Eng Comput 25(4):385–425
9. Aubry R, Idelsohn S, Oñate EO (2005) Particle finite element method in fluid mechanics including thermal convection–diffusion. Comput Struct 83(17–18):1459–1475
10. Idelsohn SR, Oñate E, Del Pin F, Calvo N (2002) Lagrangian formulation: the only way to solve some free-surface fluid mechanics problems. In: 5th world congress on computational mechanics, pp 1459–1475
11. Idelsohn SR, Oñate E, Del Pin F, Calvo N (2006) Fluidstructure interaction using the particle finite element method. Comput Methods Appl Mech Eng 195:2100–2113
12. Idelsohn SR, Mier-Torrecilla M, Oñate E (2009) Multi-fluid flows with the particle finite element method. Comput Methods Appl Mech Eng 198:2750–2767
13. Wieckowsky Z (2004) The material point method in large strain engineering problems. Comput Methods Appl Mech Eng 193(39):4417–4438
14. Stam J (2010) Stable fluids. Document Unpublished
15. Idelsohn S, Nigro N, Limache A, Oñate E (2012) Large time-step explicit integration method for solving problems with dominant convection. Comput Methods Appl Mech Eng 217–220:168–185
16. Idelsohn SR, Nigro NM, Gimenez JM, Rossi R, Marti J (2013) A fast and accurate method to solve the incompressible navier–stokes equations. Eng Comput 30(2):197–222
17. Chen S, Doolen G (1998) Lattice boltzmann method for fluids flows. Annu Rev Fluids Mech 30:329–364
18. Gimenez JM, Nigro NM (2011) Parallel implementation of the particle finite element method. Mecánica Computacional XXX:3021–3032
19. Dadvand P, Rossi R, Oñate E (2010) An object-oriented environment for developing finite element codes for multi-disciplinary applications. Arch Comput Methods Eng 17:253–297
20. Mackerle J (2004) Object-oriented programming in fem and bem: a bibliography (1990–2003). Adv Eng Softw 35(6):325–336
21. Sbalzarini I, Walther J, Bergdorf M, Hieber S, Kotsalis E, Koumoutsakos P (2006) PPM: a highly efficient parallel particlemesh library for the simulation of continuum systems. J Comput Phys 215(2):566–588. doi:10.1016/j.jcp.2005.11.017
22. Kirk BS, Peterson JW, Stogner RH, Carey GF (2006) libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations. Eng Comput 22(3–4):237–254. doi:10.1007/s00366-006-0049-3
23. Sklar DM, Gimenez JM, Nigro NM, Idelsohn SR (2012) Thermal coupling in particle finite element method-second generation. Mecánica Computacional XXXI:4143–4152
24. Donea J, Huerta A (2003) Finite element method for flow problems, 1st edn. Springer, Berlin
25. Nigro N, Gimenez J, Limache A, Idelsohn S, Oñate E, Calvo N, Novara P, Morin P (2011) A new approach to solve incompressible navier–stokes equation using a particle method. Mecánica Computacional XXX:451–483
26. Nair RD, Scroggs JS, Semazzi FH (2003) A forward-trajectory global semi-lagrangian transport scheme. J Comput Phys 190(1):275–294. doi:10.1016/S0021-9991(03)00274-2
27. Alexiades V, Amiez G, Gremaud P (1996) Super-time-stepping acceleration of explicit schemes for parabolic problems. Commun Numer Methods Eng 12:31–42
28. Gimenez JM, Nigro NM, Idelsohn SR (2012) Improvements to solve diffusion-dominant problems with PFEM-2. Mecánica Comput XXXI:137–155
29. Balay S, Brown J, Buschelman K, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Smith BF, Zhang H (2012) PETSc web page. http://www.mcs.anl.gov/petsc
30. Karypis G, Kumar V (1999) A fast and highly quality multi-level scheme for partitioning irregular graphs. SIAM J Sci Comput 20:359–392
31. Karypis G, Kumar V (1998) A parallel algorithm for multilevel graph partitioning and sparse matrix reordering. Parallel Distrib Comput 48:71–95
32. Kacianauskas R, Maknickas A, Kaceniauskas A, Markauskas D, Balevicius R (2010) Parallel discrete element simulation of polydispersed granular material. Adv Eng Softw 41:52–63
33. Kaludercic B (2004) Parallelisation of the lagrangian model in a mixed eulerian–lagrangian cfd algorithm. Parallel Distrib Comput 64:277–284
34. Centro de investigaciones en mecánica computacional (2013). http://www.cimec.org.ar