HPCLatAm 2013, pp. 145-157 (full paper)
Session: CPU and Multicore Architectures and Applications
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Towards a Massively Parallel simulations with PFEM-2

Juan M. Gimenez[1,2] and Norberto M. Nigro[1,2]

[1] Centro de Investigaciones en Mecánica Computacional (CIMEC), UNL/CONICET, Predio Conicet-Santa Fe Colectora Ruta Nac 168 / Paraje El Pozo, Santa Fe, Argentina, www.cimec.org.ar
[2] Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral. Ciudad Universitaria. Paraje "El Pozo". Santa Fe. Argentina. www.fich.unl.edu.ar

Abstract. In this work an implementation of the Particle Finite Element Method Two (PFEM-2) based on the distributed-memory architecture is presented. PFEM-2 consists on a material derivative based formulation of the transport equations with an hybrid spatial discretization which uses an eulerian mesh and lagrangian particles.
Strategies for the parallelization of eulerian methods based on mesh or lagrangian solutions based on particles which solve fluid-dynamics problems are widely studied separately, however not enough works treat the use of both approaches together. Typical solutions for domain-distribution on eulerian frames are not proper to balance the work-load on some lagrangian stages, then to achieve good performance must be analyzed the use of weighted decomposition to the partitioning.
Performance analysis of the implementation running over a beowulf cluster are presented. The weighted partitioning can be used to improve the speed-up when the diffusion of the problem is low, on the other hand, with large diffusion a classical eulerian decomposition is the best choice. However the overall cpu-time required to solve the presented incompressible flow cases with the PFEM-2 method is lower than using classical eulerian solvers, which give auspicious future thinking in solving massively parallel simulations.

1 Introduction and Motivation

Formulations for transport equations may be split in two classes depending on the approach chosen for the description of the inertial terms, namely Eulerian and Lagrangian approaches. The simulation of incompressible flows has been mainly based on the Eulerian formulation (temporal derivative plus a convective term), while Lagrangian formulations (material derivative) justify their popularity solving free-surface flows or complicated multi-fluid flows in which the standard Eulerian formulations are inaccurate or, sometimes, impossible to be used.

Maybe the most used eulerian method is the Finite Element Method (FEM) where the calculations are done over a mesh which represents the geometry

allowing to discretize the problem using a set of shape functions obtaining an algebraical system of nodal equations. FEM solvers are very accurate and robust with elliptic problems (diffusion-dominant) but has problems to deal with hyperbolic problems (convection-dominant) because requires including stabilization terms and, if the problems is incompressible flow, must deal with the non-linear term of convection which includes several restrictions to the time-step[1].

On the other hand, particle methods are considered more efficient. This feature is attributed first to the simplicity of its computation with minimum amount of information needed to update the solution and principally to be in a better position attending to the present of hardware technology based on the use of parallel computers and general purpose graphic processor units (GPGPU). However they only can solve in an accurate way problems with low diffusion then they are not proper to solve engineering problems where force calculations are involved.

The Particle Finite Element Method (PFEM)[2][3] added with an eXplicit Integration following the Velocity and Acceleration Streamlines (X-IVAS)[4][5] is called Particle Finite Element Method Second Generation (PFEM-2). This method, which uses the lagrangian formulation of the equations, solves the weakness of the two above mentioned approaches applying them only in its best case, i.e. eulerian frame to solve diffusion using a mesh and lagrangian frame to solve convection with particles. Finally, PFEM-2 is a particle method which proves that even for homogeneous fluid flows, without free-surfaces or internal interfaces, it is able to yield accurate solutions while being competitively fast when compared to state-of-the-art Eulerian solvers.

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. Although the hybrid spatial discretization used by PFEM-2 allows to use the optimum strategy to calculate each equation term, keeping in memory the data of the mesh and of the cloud of particles represents an important storage cost which limits implementations that works in only one computer. Therefore, is imperative to develop a distributed-memory implementation. In order to develop a multi-machine implementation to PFEM-2 is essential not to reinvent the wheel. Reusing some 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[6][7] which can be extended by developers for their specific applications. In the current work the `libMesh` library[8] is chosen as the basis of the development. It 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 verse). To manage those features, a smart distribution of the particles over the processes is needed and in this work a dual particle-mesh distribution is adopted.

In what follows, a review of the PFEM-2 method is presented in the section 2, where the features that allows to the method to run with large time-steps are explained. Section 3 presents a detailed analysis of the extension of `libMesh` to manage particles remarking the strategy to distribute the cloud along the sub-domains. Several Navier-Stokes simulations over Beowulf clusters are analyzed in section 4, focusing on the efficiency obtained by the implementation. Also, the performance is compared with the widely used CFD software OpenFOAM®. Finally some concluding remarks are drawn in Section 5.

2 Review of PFEM-2

The PFEM-2 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[9].

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 Equations 1 and 2,

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \nabla \cdot (\mu(\nabla \mathbf{v}^T + \nabla \mathbf{v})) + \mathbf{f} \tag{1}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{2}$$

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.

It is well known[10] that the time step selected in the solution of the transport equations is stable only for time steps which regards 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. Moreover in diffusion dominant problems where the time step decreases with the square of the grid size doing unapproachable in problems using very refined meshes.

The key of the PFEM-2 algorithm is the ability to reach $CFL >> 1$ due to the information is transported on the particles. This transport 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)[4]. 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).

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \int_0^1 \mathbf{v}^n(\mathbf{x}_p^{n+\tau})\ d\tau \tag{3}$$

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

where $\mathbf{a}^n = -\nabla p^n + \nabla \cdot (\mu(\nabla \mathbf{v}^{n^T} + \nabla \mathbf{v}^n))$ which are nodal variables. Temporal integration for the position and velocity can be solved using analytical expressions or high-order integrators. However, in this work a novel sub-stepping integrator is used where it can adapt its δt depending on its local CFL number.

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \sum_{i=1}^{N} \mathbf{v}^n(\mathbf{x}_p^{n+\frac{i}{N}})\ \delta t_p \tag{5}$$

The expression for δt is

$$\delta t_p = \frac{\Delta t}{K \times CFL_h} \tag{6}$$

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

To solve the Navier-Stokes equation, systems (3) and (4) are solved coupled with the incompressibility restriction. A typical Fractional Step Method is used to solve the coupling between the pressure and the velocity[4].

Having in mind 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 being that Fixed Mesh version may exploit the benefits of having a constant Poisson matrix for the pressure equation, this matrix is initially factorized (completely or incompletely depending on the problem size) and latterly 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, an initial factorization is chosen as a preconditioner for the diffusion part.

Finally, the algorithm 1 for PFEM-2 for incompressible flows is reported. This 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 Incompressible Flow.

1. *Acceleration* Stage: Calculate acceleration on the nodes like a FEM:
   $\int_\Omega \boldsymbol{N}\ \mathbf{a}_\tau^n\ d\Omega \approx \frac{1}{\rho}\int_\Omega \boldsymbol{N}\ \nabla\cdot\boldsymbol{\tau}^n\ d\Omega = \frac{1}{\rho}(-\int_\Omega \nabla\boldsymbol{N}\cdot(\mu(\nabla\mathbf{v}^n + \nabla\mathbf{v}^{nT}))\ d\Omega + \int_\Gamma \mathbf{N}\ (\nabla\mathbf{v}^n + \nabla\mathbf{v}^{nT})\cdot\boldsymbol{\eta}\ d\Gamma)$
   $\int_\Omega \boldsymbol{N}\ \mathbf{a}_p^n\ d\Omega \approx \frac{1}{\rho}\int_\Omega \boldsymbol{N}\ \nabla p^n\ d\Omega = \frac{1}{\rho}(-\int_\Omega \nabla\boldsymbol{N}\ p^n\ d\Omega + \int_\Gamma \boldsymbol{N}\ p^n\boldsymbol{\eta}\ d\Gamma)$
   $\mathbf{a}^n = -\mathbf{a}_p^n + (1-\theta)\ \mathbf{a}_\tau^n$
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:
   $\hat{\hat{\mathbf{v}}}_j^{n+1} = \boldsymbol{\pi}(\hat{\hat{\mathbf{v}}}_p^{n+1})$
4. *Implicit Viscosity* Stage: Implicit correction of the viscous diffusion with FEM:
   $\rho\hat{\mathbf{v}}_j^{n+1} = \rho\hat{\hat{\mathbf{v}}}_j^{n+1} + \Delta t\ \theta\ \mathbf{a}_\tau^{n+1}$
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:
   $\rho\mathbf{v}_j^{n+1} = \rho\hat{\mathbf{v}}_j^{n+1} - \Delta t\ (\mathbf{a}_p^{n+1} - \mathbf{a}_p^n)$
   $\rho\mathbf{v}_p^{n+1} = \rho\hat{\mathbf{v}}_p^{n+1} - \Delta t\ \boldsymbol{\pi}^{-1}(\mathbf{a}_p^{n+1} - \mathbf{a}_p^n) + \Delta t\ \theta\ \boldsymbol{\pi}^{-1}(\mathbf{a}_\tau^{n+1})$

3 Brief Summary of Distributed Memory Implementation

PFEM-2 simulations must store several data about the used mesh and the cloud of particles (approximately ten particle by element in 3d). Comparing with classical FEM solvers which only stores mesh data, when only one computer is used, there is a much stronger restriction in terms of problem-size for PFEM-2. Consequently, a distributed-memory approach to run on multi-node architectures is quickly needed.

The parallel numerical framework developed uses as basis the library `libMesh`. For a detailed description of libMesh see Kirk [8]; 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 MPI (Message Passing Interface). Several libraries are also included in the suite, but the main ones are `PETSc`[11] for the solution of linear systems on parallel platforms and `METIS`[12] and `ParMETIS`[13], which implement a domain-decomposition based on graph partitioning schemes for serial or distributed meshes respectively.

Critical to any implementation of distributed computing is the methodology used to distribute the global computational task to the local processor resources. In numerical simulation, tasks are generally aligned with integration points on a body in space and so dividing physical space may be used to parallelize a problem. That strategy is adopted by most FEM parallel implementations

through domain-decomposition methodology, where some problem domain is geometrically divided into sub-domains which can then be distributed across the available computational resources. The sub-domains exchange data with each other through their boundaries.

On the other hand, particle methods has a natural parallelism due to force calculations and position updates can be simultaneously done for all particle. Two main ideas can be exploited to achieve parallelism[14]. The former method, which is called atom decomposition of the workload, distributes a fixed set of particles over each processor no matter where they move in the simulation domain. The latter method consists simply in the above mentioned spatial decomposition where each processor computes only those particles that belong to its geometrical domain. Applied to PFEM-2 the first produces inter-processor communication overhead due to it is needed an updated copy of the entire mesh in each processor. Therefore the domain-decomposition is also selected for particles.

Regarding to 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, for a particles trajectory calculation using large time-steps, this dimension may be extended through several layers of elements, unless that when a particle leaves its sub-domain 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, without even mentioning the problems with defining layers when unstructured meshes are used. Figure 1 presents a graphical interpretation of the particle trajectory calculation. When a particle leaves its initial sub-domain must be communicated to the neighbor processor instead of continue in its initial processor which requires a number of ghost layers which depends approximately on CFL_p.

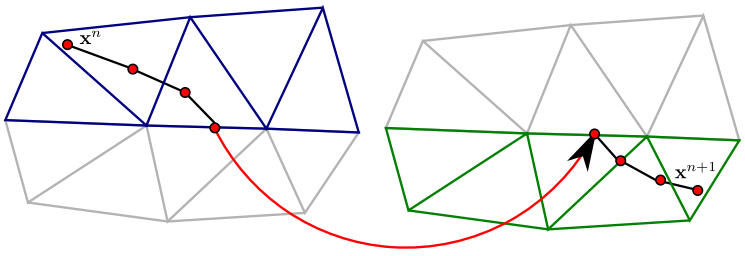

Fig. 1: Particle trajectory integration with sub-domain interchange.

To implement particle interchange the strategy that has been selected is the synchronous transference. This approach attends to minimize the communication: each particle is packed in a buffer and is not transferred until the particle loop ends. This strategy transfer particles and synchronize simultaneously, contrary to asynchronous approach where each particle is immediately sent to the CPU which is in control of the partition on the other side of the partition boundary and a synchronization point is required to finish the processors work. 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 2 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 have not 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: the first case is the particle has completed its sub-time step and it must calculate the rest of the trajectory, in the second case the particle has left out the domain boundary and its computation finishes, the third case is the particle has crossed to other sub-domain (in whose case is queued to send it to the other processor) and finally the last case is that the particle finishes the entire time step.

In any parallel calculation, the domain distribution must be done so that the computations are balanced among the processors and the communication resulting from the placement of adjacent elements to different processors is minimized. The graph partitioning strategy implemented by the library `Metis` can be used to successfully satisfy these conditions by first modeling the finite element mesh by a graph, where each vertex v of the graph can have a associate weight $\eta(v)$ to declare the work-load over each element, and then partitioning it into equal parts.

For the case of FEM simulations, a typical selection of the weights is $\eta(v) = \eta_n(v) = \#dof\_e^3$, because the amount of computational task of each element is directly proportional to the number of degrees of freedom. Then, due to 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 but PFEM-2 solves this issue seeding and removing particles preserving certain quantity $Q : Q_{min} < Q < Q_{max}$ of particles on each region (more detail can be found in [15]) and using these strategies, the overall number of particles is kept approximately constant.

An approximately constant number of particles in each sub-domain only guarantees a proper load of particle on each processor. However, due to the adaptive integrator used, that balanced distribution does not ensure a balanced work-load over each cpu. The parallel architecture in multi-core environments

[3] $\#dof\_e$: number of degrees of freedom of the element e

Algorithm 2 - 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); //using MPI_Allreduce
  if(!next) break; //no particles to send
  ParticleCommunication().interchangeParticles(particles2send,vP); //using ↩
      MPI_Alltoallv
}
```

makes impossible a dynamic scheduling to distribute the work over processors because each particle has not access to the entire domain data. A possible solution for 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 $\eta_w(V) \neq \eta_n(V)$. It requires to know or to estimate the solution previously which is impossible most of the cases, therefore an initial simulation must be done where the weights are calculated, saved, and used in the partitioning algorithm of a new run.

4 Distributed-Memory Tests

The evaluation of the distributed-memory implementation has been performed on our local Beowulf cluster at the Research Center on Computational Mechanics (CIMEC). The cluster has a server Intel i7-2600K 8Gb RAM and six nodes i7-3930K 16Gb RAM connected by Gigabit Ethernet. In order to not introduce disturbance on the results, technologies Intel Turbo Boost and Intel Hyperthreading are deactivated on the processors, giving a total of 36 computational cores of 3.4 Ghz.

4.1 Flow around a cylinder

The flow around a cylinder is a typical benchmark for incompressible flow. For the three dimensional (3d) case the geometry used has the reference unit $D = 1$, which 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 to boundary conditions, the test is run for the case $Re = 1000$, 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 $\rho = 1$. Should be discussed that the low number of Reynolds selected is because the possiblity of comparing with experimental and numerical results and due to higher values requires turbulence modeling.

The calculations in 2d are done using a mesh containing 88000 triangular elements with 43000 nodes refined towards the cylinder, whereas the three dimensional mesh used has 1.6 million of tetrahedral elements and 356000 nodes also refined towards the cylinder.

The mesh refinement is designed in order to capture more accurately the fluid forces over the cylinder. Due to the CFL number depends on the inverse of the element size h, its value increases next to the cylinder. As was widely mentioned above, the elements with large CFL will have more work-load in the X-IVAS stage then, to balance this work-load can be done a partitioning weighting with a factor proportional to CFL_h. 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 which calculates the number of sub-steps of each particle in the streamline integration $\eta_w(v_j) = min\{N_{max}, max\{N_{min}, K \times (CFL_h)_j\}\}$ where N_{max} and N_{min} are the maximum and minimum number of steps required to move a particle along the entire 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 the Figure 2 which shows the speed-up obtained by each stage of the method attending 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 $4\times$ comparing with $\eta(v)$ (from $8x$ to $12x$), the influence of this stage in the total time is not very relevant, so the strategy with $\eta_n(v)$ reaches the best overall results. Must be noted that the size of the problem is not large enough to reaches good performance with more than approximately 10 processors due to communication overheads.

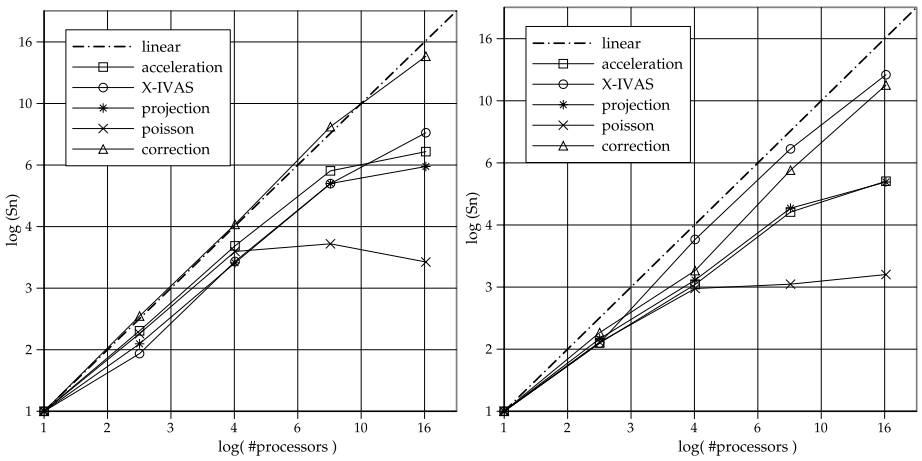

Fig. 2: Speed-Up comparison between partitioning weighted by number of degrees of freedom $\eta_n(v)$ (Figure 2a) and partitioning weighted by $\eta_w(v_j)$ (Figure 2b). Case: flow around a cylinder in 2d.

In the three-dimensional case, a comparison between the current PFEM-2 implementation and the widely used CFD software OpenFOAM®is done. The main idea is to force the time-step Δt to be the maximum such that the solver be 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. On the other hand, for OpenFOAM®results the solver `pimpleFoam` is selected, which implements the segregated PIMPLE (merged PISO-SIMPLE) algorithm, this is presented as the fastest for incompressible flow because it allows to use time-steps larger than many the other solvers. Symmetric linear equation systems are solved with `pcg` and non-symmetric with

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

Table 1: Comparison table for cputimes for the different PFEM-2 stages. Case: flow around a cylinder in 2d.

`bicg`, preconditioning both with algebraical multigrid. Absolute tolerances are set to 10^{-6}.

Table 2 presents the time-steps (Δt) used by each solver and shows the CFL values that each simulation reaches. In `pimpleFoam` is set $maxCo = 10$ and the time-step shown is an average of the instantaneous values used by the solver. Both solvers are able to solve with longer time-steps but Δt is selected checking accuracy over drag and lift coefficient values (see [5]). 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 can be concluded that, keeping a similar parallel efficiency, PFEM-2 is around three times faster than the fastest incompressible solver of OpenFOAM®.

Solver	Re	Δt	Co_{mean}	Co_{max}	S_{16}	CPU-time
PFEM-2	1000	$0.05[s]$	≈ 0.75	≈ 8	10.45x	202.56[s]
OpenFOAM®	1000	$\approx 0.025[s]$	≈ 0.5	≈ 10	9.41x	613.98[s]

Table 2: Comparison table for time-steps with different solvers. Case: flow around a cylinder in 3d. PFEM-2 partitioning with $\eta_n(v)$

The performance of the parallelization is presented in the Figure 3 which shows the speed-up obtained by each stage of the method attending to 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 $S_{16} \approx 14x - 15x$. 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 reaches values from $S_{16} \approx 7x$ to $S_{16} \approx 9x$ 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.

It must be emphasized that an important reason for the loss of efficiency with 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 then introduces several delays in a congested network as happens when many nodes are computing and sending data. Dedicated architectures as Infiniband should be employed in the future and their results should be compared[16].

5 Conclusions

The methodology for parallelizing PFEM-2 presented in this paper is an initial approach towards massively parallel computations using the method to solve both scalar and vectorial transport equations.

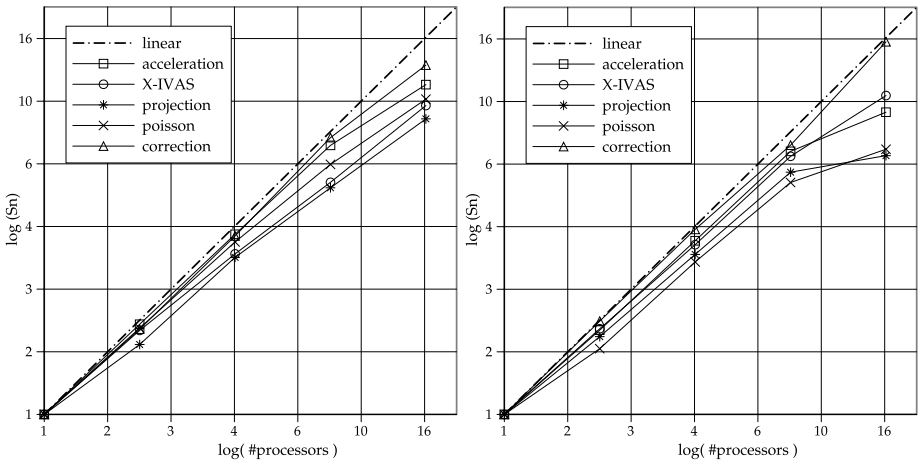

Fig. 3: Speed-Up comparison between partitioning weighted by number of degrees of freedom $\eta_n(v)$ (Figure 3a) and partitioning weighted by $\eta_w(v)$ (Figure 3b). Case: flow around a cylinder in 3d.

The domain decomposition through a graph partitioner using nodal weights allows to select, depending on the problem to solve, which stage of the algorithm will be solved more efficiently. Where X-IVAS stage dominates the calculation, this is for scalar transport problems and flow problems with low Fourier number (no implicit viscous-diffusion is needed), having a weights 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 add the implicit correction of the viscous diffusion, so the weights array must contain values proportional to the number of degrees of freedom of each element to optimize the performance.

Finally, the efficiency measured in terms of CPU time for reaching a given final time in the simulation had shown important advantages of the present method against OpenFOAM®. In this paper a factor approximately 3× was obtained at the same level of accuracy, which place the method among the fastest. Moreover if it is considered that currently the speed-up reached by PFEM-2 is similar to that achieved by the CFD software and we are researching to improve these marks.

6 Acknowledgments

The authors are grateful to CONICET, Universidad Nacional del Litoral (CAI+D Tipo II 65-333 (2009)) and ANPCyT-FONCyT (grants PICT 1645

BID (2008)) for their financial support. J. Gimenez gratefully acknowledges the support of the argentinian Agencia Nacional de Promoción Científica y Técnica (ANPCyT) through a doctoral grant in the FONCyT program.

References

1. Donea, J., Huerta, A.: Finite Element Method for Flow Problems. Wiley, Chichester England (1983)
2. Oñate, E., Idelsohn, S., Del Pin, F., Aubry, R.: The particle finite element method, an overview. International Journal of Computational Methods **1** (2004) 267–307
3. Idelsohn, S., Oñate, E., Del Pin, F.: The particle finite element method a powerful tool to solve incompressible flows with free-surfaces and breaking waves. International Journal of Numerical Methods **61** (2004) 964–989
4. Idelsohn, S., Nigro, N., Limache, A., Oñate., E.: Large time-step explicit integration method for solving problems with dominant convection. Comp. Meth. in Applied Mechanics and Engineering **217-220** (2012) 168–185
5. Idelsohn, S.R., Nigro, N.M., Gimenez, J.M., Rossi, R., Marti., J.: A fast and accurate method to solve the incompressible navier-stokes equations. Engineering Computations **30-Iss:2** (2013) 197–222
6. Mackerle, J.: Object-oriented programming in fem and bem: a bibliography (1990–2003). Advances in Engineering Software **35**(6) (2004) 325 – 336
7. Sbalzarini, I., Walther, J., Bergdorf, M., Hieber, S., Kotsalis, E., Koumoutsakos, P.: {PPM} – a highly efficient parallel particle–mesh library for the simulation of continuum systems. Journal of Computational Physics **215**(2) (2006) 566 – 588
8. Kirk, B.S., Peterson, J.W., Stogner, R.H., Carey, G.F.: `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. Engineering with Computers **22**(3–4) (2006) 237–254 `http://dx.doi.org/10.1007/s00366-006-0049-3`.
9. Sklar, D.M., Gimenez, J.M., Nigro, N.M., Idelsohn, S.R.: Thermal coupling in particle finite element method - second generation. Mecánica Computacional **XXXI** (2012) 4143–4152
10. Donea, J., Huerta, A.: Finite element method for flow problems. 1st edn. Springer-Verlag. (2003)
11. Balay, S., Brown, J., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc Web page (2012) http://www.mcs.anl.gov/petsc.
12. Karypis, G., Kumar, V.: A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing **20** (1999) 359–392
13. Karypis, G., Kumar, V.: A parallel algorithm for multilevel graph partitioning and sparse matrix reordering. Parallel and Distributed Computing **48** (1998) 71–95
14. Kacianauskas, R., Maknickas, A., Kaceniauskas, A., Markauskas, D., Balevicius, R.: Parallel discrete element simulation of poly-dispersed granular material. Advances in Engineering Software **41** (2010) 52–63
15. Gimenez, J.M., Nigro, N.M., Idelsohn, S.R.: Improvements to solve diffusion-dominant problems with pfem-2. Mecánica Computacional **XXXI** (2012) 137–155
16. Yeo, C., Buyya, R., Pourreza, H., Eskicioglu, R., Graham, P., Sommers, F.: Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. In Zomaya, A., ed.: Handbook of Nature-Inspired and Innovative Computing. Springer US (2006) 521–551