Journal of Computational Physics 228 (2009) 6316–6332

Contents lists available at ScienceDirect

Journal of Computational Physics

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

A massively parallel fractional step solver for incompressible flows

G. Houzeaux *, M. Vázquez, R. Aubry, J.M. Cela

Barcelona Supercomputing Center (BSC-CNS), Edificio NEXUS I, Campus Nord UPC, Gran Capitán 2-4, 08034 Barcelona, Spain

ARTICLE INFO

Article history:
Received 21 November 2008
Received in revised form 17 April 2009
Accepted 14 May 2009
Available online 21 May 2009

Keywords:
Navier–Stokes equations
Fractional methods
Stabilized finite element
Predictor–corrector scheme
Incremental projection scheme
Parallel implementation
High performance computing

ABSTRACT

This paper presents a parallel implementation of fractional solvers for the incompressible Navier–Stokes equations using an algebraic approach. Under this framework, predictor–corrector and incremental projection schemes are seen as sub-classes of the same class, making apparent its differences and similarities. An additional advantage of this approach is to set a common basis for a parallelization strategy, which can be extended to other split techniques or to compressible flows.

The predictor–corrector scheme consists in solving the momentum equation and a modified "continuity" equation (namely a simple iteration for the pressure Schur complement) consecutively in order to converge to the monolithic solution, thus avoiding fractional errors. On the other hand, the incremental projection scheme solves only one iteration of the predictor–corrector per time step and adds a correction equation to fulfill the mass conservation. As shown in the paper, these two schemes are very well suited for massively parallel implementation. In fact, when compared with monolithic schemes, simpler solvers and preconditioners can be used to solve the non-symmetric momentum equations (GMRES, Bi-CGSTAB) and to solve the symmetric continuity equation (CG, Deflated CG). This gives good speedup properties of the algorithm. The implementation of the mesh partitioning technique is presented, as well as the parallel performances and speedups for thousands of processors.

1. Introduction

This paper studies efficient strategies to solve the incompressible Navier–Stokes equations, specially well-suited for large-scale supercomputing. It is well known that the straightforward way to solve the discretized Navier–Stokes equations is to consider a monolithic scheme, where momentum and continuity equations are solved simultaneously. When problems are large, iterative schemes are preferred due to their lower memory requirements and better parallelization properties. However, some hard cases degrade their convergence speed, unless a good preconditioner is used, such as ILU. But these preconditioners have an important drawback: as direct schemes and due to communication issues, they are very bad to scale when parallelized. Therefore, these preconditioners are not attractive when thousands of processors are available in a supercomputing facility (it is the same problem for direct solvers).

One well-known way of circumventing this problem is to "split" the discretized Navier–Stokes operator dividing the solution process in stages. For instance, Strang's *viscous splitting* [1] separates viscous from convection terms, solving them at different stages. From an algebraic point of view, this consists of the solution of a preconditioned Richardson iteration for the pressure Schur complement system, where the solution of the momentum equation can be viewed as an intermediate

* Corresponding author.
E-mail addresses: guillaume.houzeaux@bsc.es (G. Houzeaux), mariano.vazquez@bsc.es (M. Vázquez), romain.aubry@bsc.es (R. Aubry).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.05.019

step to solve the continuity equation. By inspecting the continuous equation behind the discrete system, the splitting can naturally take profit of the terms appearing in the original equation, that in the case of incompressible Navier–Stokes are time variation, convection, diffusion and pressure gradient. It is worth mentioning a second aspect of this strategy: the action of splitting the operator can have beneficial side effects as it is revealed in [2,3] and extended and applied in papers like [4,5] because the pressure split is proven to stabilize the incompressibility limit with equal-interpolation schemes. In this paper, we will not focus on this aspect, but in the first one.

Traditionally, in the context of Navier–Stokes equations fractional step techniques are those that segregate the two ruling equations, momentum and continuity. This means firstly to freeze or extract the pressure gradient term from the momentum equation and secondly to solve the continuity equation computing the new pressure. A last correcting step is taken to upgrade the momentum [4,6,7], the so-called "projection" or "correction" step. In this way, a Navier–Stokes fractional step scheme can be seen as a "time-splitting" strategy. On the other hand, in this paper we propose to follow the algebraic system view, as also proposed for instance in [8]. The present approach allows to design predictor–corrector methods that converge to the monolithic solution while inheriting the good algebraic properties given by the splitting of the momentum and continuity equations. A nice companion paper to the present work is that of Elman et al. [9] where similar algorithms are presented in the form of block LU decompositions.

The objective is to develop efficient and scalable solution methods by applying mesh partitioning techniques for solving both a predictor–corrector and an incremental projection schemes to the discretized Navier–Stokes equations, capable of running in thousands of CPU's with the best possible scalability properties. On the other hand, the convergence properties of the schemes are not analyzed here, because it has been done in previous papers [10,11]. It is worth mentioning that the proposed parallel implementation is general and can be applied to other techniques of the projection scheme family, for which similar speedup properties are expected.

The numerical scheme is briefly presented in the following section. It consists of a finite element method based on the variational subgrid scale concept for subgrid scale modeling and stabilization. Then, a Predictor–Corrector Scheme (PCS) and an Incremental Projection Scheme (IPS) are introduced at the algebraic level. The former one converges to the monolithic solution at each time step and the latter once steady state is reached. Next, we present the mesh partitioning technique implemented to solve the algorithm in parallel. In particular, we will point out the important aspects to take into account in order to obtain good parallel behavior on parallel supercomputers, including such topics as automatic mesh partitioning, communication scheduling or data organization. Finally, the results section will have two objectives. Firstly, they will be used to study the parallel performance of the algebraic solvers used to solve the momentum and continuity equations. Secondly, we will show the overall parallel performance of the predictor–corrector and incremental projection schemes on thousands of processors.

2. Numerical formulation

In this section, we introduce the numerical method used to solve the incompressible flow equations. We will not enter into too much details and only give the important points, referring to papers for complementary information and to [12] for the exact implementation. In the following, index i will refer to a linearization iteration and n to time step.

2.1. Space discretization and linearization

The governing equations are the Navier–Stokes' for transient and incompressible flows [13]. Let μ be the viscosity of the fluid, and ρ its constant density. The problem is stated as follows: find the velocity $\boldsymbol{u}$ and mechanical pressure p in a domain Ω such that they satisfy in a time interval

$$\rho\frac{\partial \boldsymbol{u}}{\partial t}+\rho(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}-\nabla\cdot[2\mu\boldsymbol{\varepsilon}(\boldsymbol{u})]+\nabla p=\rho\boldsymbol{f},\quad \nabla\cdot\boldsymbol{u}=0,$$

together with initial and boundary conditions. The term $\boldsymbol{f}$ is the force term, including for example gravity contributions. The velocity strain rate is $\boldsymbol{\varepsilon}(\boldsymbol{u})=\frac{1}{2}(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^t)$.

The spatial discretization is a variational multiscale (two-level) finite element method [14] using an algebraic approximation of the subgrid scale [15]. The velocity subgrid scale is taken proportional to the momentum residual and the proportionality parameter τ_1 is the so-called stabilization parameter (originally referred to as intrinsic time). It is an algebraic and local (element-wise) approximation of the momentum operator. In this work we take:

$$\tau_1=\left[\frac{4\mu}{h_e^2}+\frac{2\rho|\boldsymbol{u}_e|}{h}\right]^{-1}, \tag{1}$$

where subscript e means that the value is computed element-wise. h_e is the characteristic element size and half of it for quadratic elements [15].

The linearization is carried out using the Picard method, that is substituting the convective term $\rho(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}$ by $\rho(\boldsymbol{u}^i\cdot\nabla)\boldsymbol{u}$, $\boldsymbol{u}^i$ being taken from a previous iteration.

2.2. Time discretization

Time discretization is carried out using the trapezoidal rule. The time step can be either prescribed, or computed automatically from the critical time step. In the present work, the critical time step δt_c is computed as the minimum stabilization parameter $\rho\tau_1$ over the mesh:

$$\delta t_c = \min_{(e)}(\rho\tau_1),$$

with τ_1 given by Eq. (1). Usually, the time step is computed as a multiple of the critical time step by introducing the safety factor α such that:

$$\delta t = \alpha\delta t_c.$$

In practice, α takes values from 1 to 100 and can be adjusted to capture transient physical behavior.

3. Split scheme

This section is organized in a constructive way in different subsections. We first present the starting point consisting of the monolithic scheme. Then, we briefly review projection methods by writing the Schur complement system for the pressure and introducing a preconditioned Richardson iteration to solve the system. In the following subsection, the preconditioner as well as the correction are explained. The resulting algorithm is very similar to the one presented in [16]. We then introduce the algebraic stopping criteria which controls the algorithm. Then, the final solution procedure puts all the ingredients together.

3.1. Monolithic scheme

After space and time discretizations and linearization, the resulting algorithm consists in solving at each linearization and time steps the following algebraic system:

$$\begin{bmatrix}\mathbf{A}_{uu} & \mathbf{A}_{up}\\ \mathbf{A}_{pu} & \mathbf{A}_{pp}\end{bmatrix}\begin{bmatrix}\mathbf{u}\\ \mathbf{p}\end{bmatrix} = \begin{bmatrix}\mathbf{b}_u\\ \mathbf{b}_p\end{bmatrix}. \tag{2}$$

The four matrices and two RHS's come from the assembly of the weak form terms. The matrices consist of the following terms:

- $\mathbf{A}_{uu}$: Galerkin momentum, pressure subgrid scale, velocity subgrid scale (SUPG like term);
- $\mathbf{A}_{up}$: Galerkin pressure gradient, velocity subgrid scale;
- $\mathbf{A}_{pu}$: Galerkin velocity divergence, velocity subgrid scale;
- $\mathbf{A}_{pp}$: pressure stabilization: for that reason, no additional stabilization properties are sought by splitting.

It is well known that the convergence properties of classical iterative solvers like GMRES, Bi-CGSTAB are very poor, due to the negative effects (indefiniteness) of coupling of the velocity and pressure of system 2. Usually, complex preconditioners like Incomplete LU factorization are considered [17,18]. However, the construction of these preconditioners in parallel requires lots of communications and thus exhibit a low scalability. For that reason, we will focus on these methods that are at the same time *serially efficient*, capable of reaching the solution at a reduced rate in terms of flops, and *parallely efficient*, with good scalability properties. The quest for this double objective has guided our choice.

3.2. Projection methods

One alternative kind of solutions to the monolithic scheme that are widely used are *Projection Methods*. These methods [10]

compute the velocity and the pressure fields separately, through the computation of an intermediate auxiliary velocity, which is then projected on the subspace of the solenoidal vector functions.

Forty years ago, Chorin and Temam [19,20] proposed a first order projection method. Since then, many corrections have been proposed in the literature to improve the time accuracy of the algorithm, to get rid of the artificial Neumann condition on the pressure, etc.

One example is the incremental projection scheme [21] (see also [22] and references therein) which consists in splitting the original operator and solving an alternative system (not strictly equivalent) in three steps, usually referred to as Momentum or fractional momentum step (M), Continuity or projection step (C), and correction step (R). These steps are:

- (M) Compute intermediate velocity $\tilde{\boldsymbol{u}}^{n+1}$:

$$\rho\frac{\tilde{\boldsymbol{u}}^{n+1}-\boldsymbol{u}^{n}}{\partial t}+\rho(\boldsymbol{u}^{n}\cdot\nabla)\boldsymbol{u}^{n+1}-\nabla\cdot[2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}^{n+1})]=\rho\boldsymbol{f}-\nabla p^{n}. \tag{3}$$

- (C) Compute pressure p^{n+1}:

$$\delta t\Delta p^{n+1}=\rho\nabla\cdot\tilde{\boldsymbol{u}}^{n+1}+\delta t\Delta p^{n}. \tag{4}$$

- (R) Correct velocity $\boldsymbol{u}^{n+1}$:

$$\boldsymbol{u}^{n+1}=\tilde{\boldsymbol{u}}^{n+1}-\frac{\delta t}{\rho}\nabla(p^{n+1}-p^{n}). \tag{5}$$

Projection schemes have great computational advantages:

1. Under some conditions, the scheme retains the time accuracy one would obtain with the monolithic scheme without having to iterate for the non-linearity;
2. Algebraically speaking, the decoupling of the momentum and continuity equation enables the use of simple iterative solvers without the need for complex preconditioners.

However, this scheme presents some drawbacks:

1. The time step should be relatively small;
2. From Eq. (2), it introduces an artificial Neumann condition on the pressure: $\nabla p^{n+1}\cdot\boldsymbol{n}=\nabla p^{n}\cdot\boldsymbol{n}$.

Therefore, the solution is not strictly the same as that of the original system of equations. The idea of predictor–corrector schemes [23] is to remedy this by iterating within a time step the latter scheme to recover the monolithic solution. This is accomplished by substituting p^{n+1} and p^{n} by $p^{n+1,i+1}$ and $p^{n+1,i}$ in Eqs. (3)–(5), as well as by substituting the convection velocity in (3) by $\boldsymbol{u}^{n+1,i}$. In fact, we then observe that once convergence is achieved, the following happens: Eq. (5) yields $\boldsymbol{u}^{n+1}=\tilde{\boldsymbol{u}}^{n+1}$, Eq. (4) is the original continuity equation and Eq. (3) is the original momentum equation.

Alternatively, projection methods can be re-interpreted in the algebraic context. See for example "the generalized block LU decomposition" in[8,10] or the review [11]. This approach allows to take advantage of the battery of algebraic tools to devise better algorithms, leaving aside the pressure and fractional velocity boundary conditions issues, because they are naturally included in the system matrix.

Other methods are also possible like multilevel methods, whether used as stand alone methods to substitute projection methods, as proposed in [24], coupled with projection methods as in [25], or coupled with Krylov methods as preconditioners as proposed in [26]. These methods have been successfully used in a parallel context in compressible flows in [27,28]. Apart from the fact that multilevel methods may be cumbersome in dealing with adaptive or moving meshes, they involve a much higher implementation effort. They will nevertheless be the subject of further research by the authors in a parallel context.

3.3. Richardson iteration for the pressure Schur complement

Let us manipulate the matrix system (2) to compute the Schur complement system [29] for the pressure. The Schur complement system is simply the pressure equation one obtains after eliminating the velocity from the momentum equation. We have:

$$\mathbf{S}\mathbf{p}=\mathbf{b}_{s}, \tag{6}$$

with

$$\mathbf{S}=\mathbf{A}_{pp}-\mathbf{A}_{pu}\mathbf{A}_{uu}^{-1}\mathbf{A}_{up},$$
$$\mathbf{b}_{s}=\mathbf{b}_{p}-\mathbf{A}_{pu}\mathbf{A}_{uu}^{-1}\mathbf{b}_{u}.$$

The idea is now to apply a simple preconditioned Richardson iteration method [29] to solve the Schur complement system (6). The solution at iteration $k+1$ is computed from the solution at iteration k as:

$$\mathbf{p}^{k+1}=\mathbf{p}^{k}+(\mathbf{A}_{pp}+\mathbf{P})^{-1}(\mathbf{b}_{s}-\mathbf{S}\mathbf{p}^{k}).$$

Matrix $(\mathbf{A}_{pp}+\mathbf{P})$ is the preconditioner of $\mathbf{S}$, which has been split into two matrices, such that $\mathbf{P}$ should approximate $-\mathbf{A}_{pu}\mathbf{A}_{uu}^{-1}\mathbf{A}_{up}$. A simple preconditioner will be devised in the next section.

For implementation purposes, the Schur complement system is not implemented as shown above as it would require to compute explicitly the inverse of matrix $\mathbf{A}_{uu}$. In fact, we can easily see that

$$\begin{cases} \mathbf{A}_{uu}\mathbf{u}^{k+1} = \mathbf{b}_u - \mathbf{A}_{up}\mathbf{p}^k, \\ (\mathbf{A}_{pp}+\mathbf{P})\mathbf{p}^{k+1} = (\mathbf{A}_{pp}+\mathbf{P})\mathbf{p}^k + \mathbf{b}_s - \mathbf{S}\mathbf{p}^k, \end{cases}$$

is equivalent to the following system:

$$\begin{cases} \mathbf{A}_{uu}\mathbf{u}^{k+1} = \mathbf{b}_u - \mathbf{A}_{up}\mathbf{p}^k, \\ (\mathbf{A}_{pp}+\mathbf{P})\mathbf{p}^{k+1} = \mathbf{b}_p - \mathbf{A}_{pu}\mathbf{u}^{k+1} + \mathbf{P}\mathbf{p}^k, \end{cases} \tag{7}$$

that is the Schur complement is solved using an intermediate step corresponding to the solution of the momentum equation using the previous pressure iterate. We observe that in this way, the Schur complement matrix $\mathbf{S}$ is not assembled explicitly but is computed using the previous velocity solve. In the current implementation, the solver iteration k is coupled to the linearization iteration; that is k is substituted by i. So both indices will be used indifferently. Note also that matrices $\mathbf{A}_{pu}$ and $\mathbf{A}_{pp}$ respectively depend on the velocity due to the convective and stabilization terms. In the current implementation of System (7), these matrices are recomputed after the velocity is obtained. Note finally that once converged, the solution of this algorithm is exactly the same as that of the monolithic scheme.

3.4. Schur complement preconditioner and correction

A simple preconditioner $\mathbf{P}$ is considered such that

$$\mathbf{P} \approx -\mathbf{A}_{pu}\mathbf{A}_{uu}^{-1}\mathbf{A}_{up}. \tag{8}$$

On the one hand, $\mathbf{A}_{pu}$ comes from the divergence of the velocity linear form and the stabilization; on the other hand, $\mathbf{A}_{up}$ comes from the linear form representing the pressure term of the momentum equations and the stabilization. These two matrices are almost minus the transpose of each other because of the boundary conditions imposed on the velocity. Finally, we are only left with the inverse of matrix $\mathbf{A}_{uu}$ which comes from the discretization of the momentum operator. Let us define the momentum operator $\mathcal{M}$ (after time discretization) acting on the velocity as

$$\mathcal{M} = \frac{\rho}{\theta\delta t} + \boldsymbol{u}^i \cdot \nabla - \nabla \cdot [2\mu\boldsymbol{\varepsilon}].$$

We then look at the weak form of the classical Uzawa's operator $-\nabla\cdot\mathcal{M}^{-1}\nabla$, which is the continuous counterpart of Eq. (8), from which $\mathbf{P}$ is computed [30,31]. We have

$$\mathbf{P} \Leftarrow -\int_\Omega (\nabla\cdot\mathcal{M}^{-1}\nabla p)\tilde{q}\,d\Omega, \tag{9}$$

where $\tilde{q}$ is a subspace of the pressure space imposing Dirichlet boundary conditions on the Neumann part of the boundary.

For transient problems and sufficiently small time steps, $\mathcal{M}$ can be approximated element-wise by

$$\mathcal{M} \approx \frac{\rho}{\theta\delta t}.$$

In [32], the same authors will present a different approximation based on the common approximation used when subgrid scale are tracked in time [15]. Integrating by parts Eq. (9) and neglecting the boundary terms, we find

$$\begin{aligned} \mathbf{P} &\Leftarrow \int_{\Omega'} \mathcal{M}^{-1}\nabla p\cdot\nabla\tilde{q}\,d\Omega, \\ &\Leftarrow \int_{\Omega'} \theta\delta t/\rho\nabla p\cdot\nabla\tilde{q}\,d\Omega, \end{aligned} \tag{10}$$

where $\int_{\Omega'}$ is the sum of element integrals.

Finally, the correction step is the same as given by Eq. (5).

3.5. Stopping criteria

As mentioned previously, only one Richardson iteration is carried out for one velocity guess, that is the linearization loop is coupled to the Richardson loop. One convergence criterion is therefore needed for the iteration i inside a time step (apart

from the iterative solver one). For a generic linearized problem $\mathbf{A}^i\mathbf{u}^{i+1} = \mathbf{b}$, we have among others the following option to compute the stopping criterion [33]:

$$\text{Relative difference} = \frac{\|\mathbf{u}^{i+1} - \mathbf{u}^i\|}{\|\mathbf{u}^{i+1}\|},$$
$$\text{Algebraic residual} = \frac{\|\mathbf{b} - \mathbf{A}^{i+1}\mathbf{u}^{i+1}\|}{\|\mathbf{b}\|}.$$

On the one hand, the relative difference convergence criteria is easy to compute. However, when the solving iterative process for the algebraic system $\mathbf{A}^i\mathbf{u}^{i+1} = \mathbf{b}$ stagnates (typical behavior of the GMRES solver), we may have $\mathbf{u}^{i+1} \approx \mathbf{u}^i$ so that the relative residual is zero and this would mislead us. On the other hand, in order to compute the algebraic residual, the matrix must be computed at iteration $i+1$. In order to avoid this extra calculation, we propose to compute the residual of the previous iteration (i) just after matrix $\mathbf{A}^i$ is assembled, that is

$$\text{Lagged algebraic residual} = \frac{\|\mathbf{b} - \mathbf{A}^i\mathbf{u}^i\|}{\|\mathbf{b}\|}.$$

Very few additional computation is involved in the calculation if we compare the number of matrix-vector products appearing in iterative solvers.

3.6. Solution procedures

We present in the following two algorithms, namely an incremental projection scheme (IPS) and a predictor–corrector scheme (PCS), both solving the preconditioned Schur complement system for the pressure presented earlier. The differences between both are the following:

- In the case of the predictor–corrector method, the velocity correction is not necessary to fulfill the continuity equation.
- When using the Crank–Nicolson method, the velocity update is performed just after solving the momentum equation.
- In the incremental projection method, the momentum, continuity and correction steps are performed only once per time step.

The algebraic incremental projection method is illustrated by Algorithm 1 and the predictor–corrector algorithm is illustrated by Algorithm 2. In the correction step, a closed quadrature rule is used to obtain a diagonal mass matrix $\mathbf{M}$. The same rule is used to compute the weighted gradient matrix $\mathbf{G}'$. Finally, recall that $\mathbf{P}$ is computed using weak form (10).

Algorithm 1. Incremental projection method

Set time initial values.
$t = 0$.
for time steps $n = 0, 1, 2 \ldots$
  Compute time step δt.
  $t = t + \delta t$.

$$\text{(M)}\begin{cases} \text{Assemble} : \mathbf{A}_{uu} \text{ and } \tilde{\mathbf{b}}_u \text{ with } \tilde{\mathbf{b}}_u := \mathbf{b}_u - \mathbf{A}_{up}\mathbf{p}^n. \\ \text{Compute residual} : \|\tilde{\mathbf{b}}_u - \mathbf{A}_{uu}\mathbf{u}^n\|/\|\tilde{\mathbf{b}}_u\|. \\ \text{Solve} : \mathbf{A}_{uu}\tilde{\mathbf{u}} = \tilde{\mathbf{b}}_u. \\ \text{Update } \tilde{\mathbf{u}}^{n+1} = 1/\theta\tilde{\mathbf{u}} + (1 - 1/\theta)\mathbf{u}^n \end{cases}$$

$$\text{(C)}\begin{cases} \text{Assemble } \mathbf{A}_{pp}, \mathbf{P} \text{ and } \tilde{\mathbf{b}}_p \text{with } \tilde{\mathbf{b}}_p = \mathbf{b}_p - \mathbf{A}_{pu}\tilde{\mathbf{u}}^{n+1}. \\ \text{Compute residual} : \|\tilde{\mathbf{b}}_p - \mathbf{A}_{pp}\mathbf{p}^i\|/\|\tilde{\mathbf{b}}_p\|. \\ \text{Solve} : (\mathbf{A}_{pp} + \mathbf{P})\mathbf{p}^{n+1} = \tilde{\mathbf{b}}_p + \mathbf{P}\mathbf{p}^n. \end{cases}$$

$$\text{(R)}\{\ \text{Correct velocity} : \mathbf{u}^{n+1} = \tilde{\mathbf{u}}^{n+1} - \mathbf{M}^{-1}\mathbf{G}'(\mathbf{p}^{n+1} - \mathbf{p}^n).$$

end for

Algorithm 2. Predictor–corrector method

Set time initial values.
$t = 0$.
for time steps $n = 0, 1, 2, \ldots$
  Compute time step δt.
  $t = t + \delta t, i = 0$.
  while Convergence not achieved

$$(\mathrm{M})\begin{cases}\text{Assemble}: \mathbf{A}_{uu} \text{ and } \tilde{\mathbf{b}}_u \text{ with } \tilde{\mathbf{b}}_u := \mathbf{b}_u - \mathbf{A}_{up}\mathbf{p}^i.\\ \text{Compute momentum residual}: \|\tilde{\mathbf{b}}_u - \mathbf{A}_{uu}\mathbf{u}^i\|/\|\tilde{\mathbf{b}}_u\|\\ \text{Solve}: \mathbf{A}_{uu}\tilde{\mathbf{u}}^{i+1} = \tilde{\mathbf{b}}_u.\end{cases}$$

$$(\mathrm{C})\begin{cases}\text{Assemble}: \mathbf{A}_{pp}, \mathbf{P} \text{ and } \tilde{\mathbf{b}}_p \text{ with } \tilde{\mathbf{b}}_p = \mathbf{b}_p - \mathbf{A}_{pu}\tilde{\mathbf{u}}^{i+1}.\\ \text{Compute residual}: \|\tilde{\mathbf{b}}_p - \mathbf{A}_{pp}\mathbf{p}^i\|/\|\tilde{\mathbf{b}}_p\|.\\ \text{Solve}: (\mathbf{A}_{pp} + \mathbf{P})\mathbf{p}^{i+1} = \tilde{\mathbf{b}}_p + \mathbf{P}\mathbf{p}^i.\end{cases}$$

$$(\mathrm{R})\{\ \text{Optionally Correct velocity}: \mathbf{u}^{i+1} = \tilde{\mathbf{u}}^{i+1} - \mathbf{M}^{-1}\mathbf{G}'(\mathbf{p}^{i+1} - \mathbf{p}^i).$$

  $i = i + 1$.
  endwhile
  Update $\mathbf{u}^{n+1} = 1/\theta\mathbf{u} + (1 - 1/\theta)\mathbf{u}^n$
end for

4. Parallel implementation

The parallelization strategy is based on a mesh partitioning technique using a Master-Slave strategy. Mesh partitioning is done by METIS [34] and parallelized with an MPI-based strategy. Although not discussed in this paper, it is worth mentioning that a second and deeper degree of parallelization is achieved by opening threads in internal loops using OpenMP directives, resulting in an hybrid-parallelization strategy. All the examples presented above were run using only the MPI-based strategy.

Iterative solvers are used to solve both the momentum and continuity equations. The momentum equations are non-symmetrical so a GMRES or Bi-CGSTAB method with diagonal preconditioning is used. Both methods usually perform very well in terms of rate of convergence for solving these equations. The advantage of the Bi-CGSTAB method is that no Krylov dimension must be provided so that the convergence is not user-dependent. The modified continuity equation is symmetric so the conjugate gradient (CG) method with diagonal preconditioning is used.

What can we expect from the performance of the algorithm? One common operation involved in the GMRES, Bi-CGSTAB and CG method is the multiplication of a matrix by a vector. We expect for these two operations a nice speedup property as the ratio local computations/communications is high. However, the inner loop of the GMRES method (the modified Gram-Schmidt orthogonalization) involves a scalar product, performed through global communications, which can have a small ratio local computations/communications. We therefore expect that, apart from other parallelization implementation aspects, the speedup properties of the algorithm will be influenced by:

- The relative weights of the solvers with respect to the overall computation: that is the ratio solvers/element loop computations.
- The relative weight of the GMRES and CG solvers, knowing that the last one is more scalable for a given number of iterations.
- The relative weight of the modified Gram-Schmidt algorithm in the GMRES algorithm, providing that this depends on the Krylov dimension given by the user.

These considerations will be illustrated through the solution of examples in Section 5.1.

One final comment: it is important to remark that, although in this paper the parallelization strategy is applied in this work to the predictor–corrector and incremental projection schemes presented above, it can be easily adapted to other fractional or predictor–corrector step techniques like those of [11].

4.1. Master-Slave technique

The Master reads the mesh, performs its partition and dumps some output files (for example of the convergence residuals). The slaves build the local element matrices (LHS) and right-hand side (RHS) and are in charge of the resulting system solution in parallel.

Two strategies are possible:

- *For small problems (say <15 millions of elements):* The Master creates the mesh partition, sends each of the subdomains and supplementary data to the corresponding slaves and launches the simulation.
- *For large problems:* The Master creates the mesh partition, writes individual restart files for each of the slaves containing its corresponding subdomain and supplementary data and stops the run. Then, a parallel run can start at any time with the Master commanding the Slaves to read the individual restart files. This strategy is particularly well-suited for very large cases whose partitions are expensive to compute in terms of both CPU time and memory. Moreover, it is very likely that the memory available for one CPU in the distributed memory cluster is not enough, and pre-process must be done in a different architecture with shared memory like an Altix cluster.

4.2. Types of communications

In a finite element implementation, only two kinds of communications are necessary between subdomains. The first type of communication consists in exchanging arrays between neighbors with `MPI_Sendrecv`. The strategy is the following:

1. For each slave, compute elemental LHS and RHS for each element.
2. For each slave, assemble (scatter) elemental RHS and LHS into global LHS and RHS.
3. Exchange RHS of boundary nodes, the nodes belonging to more than one subdomain, and sum the contribution.
4. The operations of an iterative solver are matrix-vector multiplications. Then, for each slave, compute matrix-vector multiplication.
5. Exchange the results on the boundary nodes, as was done for the RHS.

The second type of communication is global and of reduce type with `MPI_Reduce`. It is used to compute:

- *The critical time step:* it is the minimum over the slaves.
- *The convergence residual:* the sum over all the nodes of the domain. Residuals are required to check the different convergence tolerances of the scheme.
- *Scalar products:* they take part of iterative solvers.

4.3. Critical points of the parallelization strategy

In order to have an efficient algorithm to run on thousands of processors, some important aspects of the parallelization must be carefully treated: mesh partitioning, node numbering and communication scheduling.

The *mesh partitioning* is performed using METIS [34]. The main input data of METIS are the element graph and the weight of the vertices of the graph. The difficulty of hybrid meshes for implicit schemes stems from the fact that such an algorithm involves both elemental and nodal loops: elemental for the assembly of the LHS and RHS; nodal in the solver. For the element graph, two options are possible, as illustrated in Fig. 1. The straightforward option is to consider, as the adjacent elements to an element e, all the elements that share a node with e. However, this graph could be extremely greedy in terms of memory as shown in the figure for a mesh of hexahedra. In this 50 M hexahedra example, the element graph would occupy 5 Gb using four-byte integers. A second strategy, which shows good load balance results so far, is to take as the adjacent elements to e only the elements sharing a face with e. This strategy requires much less memory and it is the one in the numerical examples. Fig. 2 shows a partition of the volume mesh for an apartment into 100 subdomains.

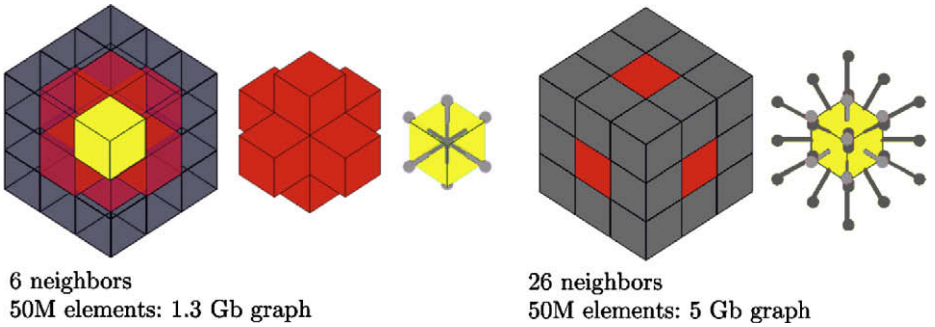


Fig. 1. Two possible element graph strategies. (Left) Adjacency based on face sharing. (Right) Adjacency based on node sharing.

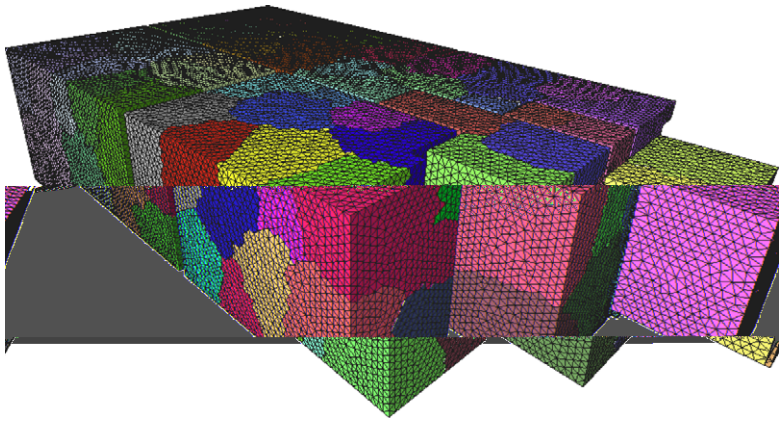


Fig. 2. Partition of an apartment into 100 subdomains.

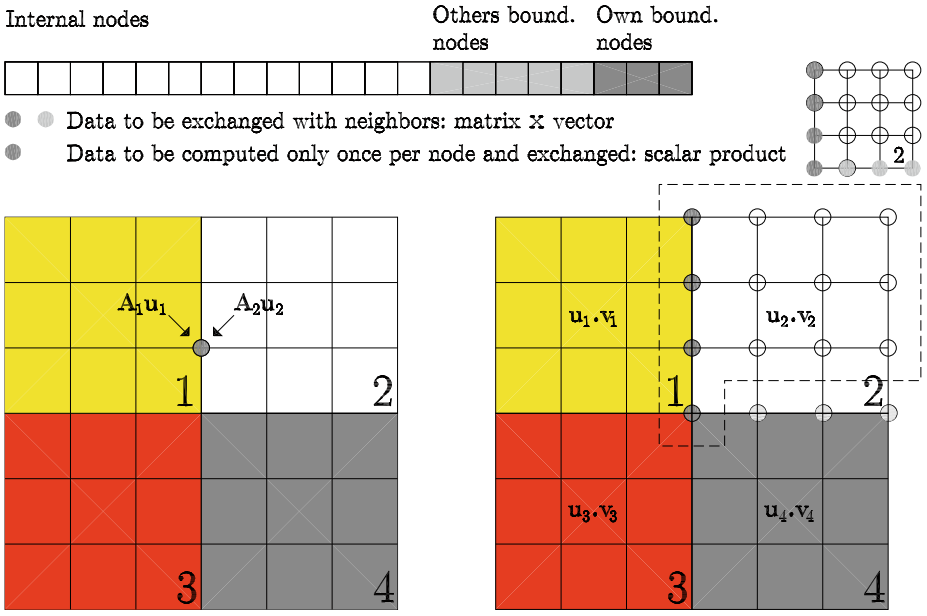


Fig. 3. Local node numbering. From light to dark: internal nodes, other subdomains boundary nodes and own boundary nodes.

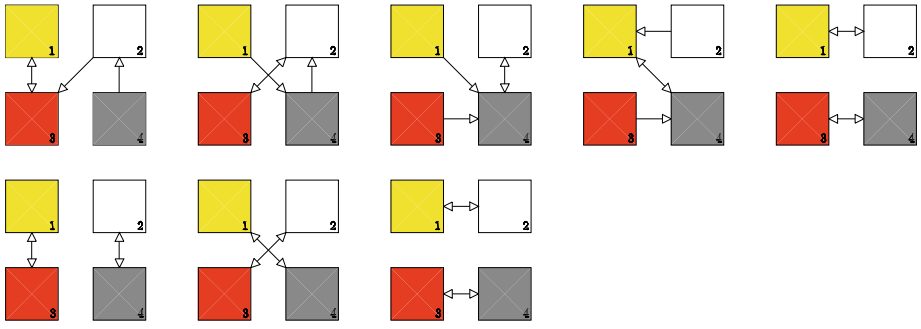
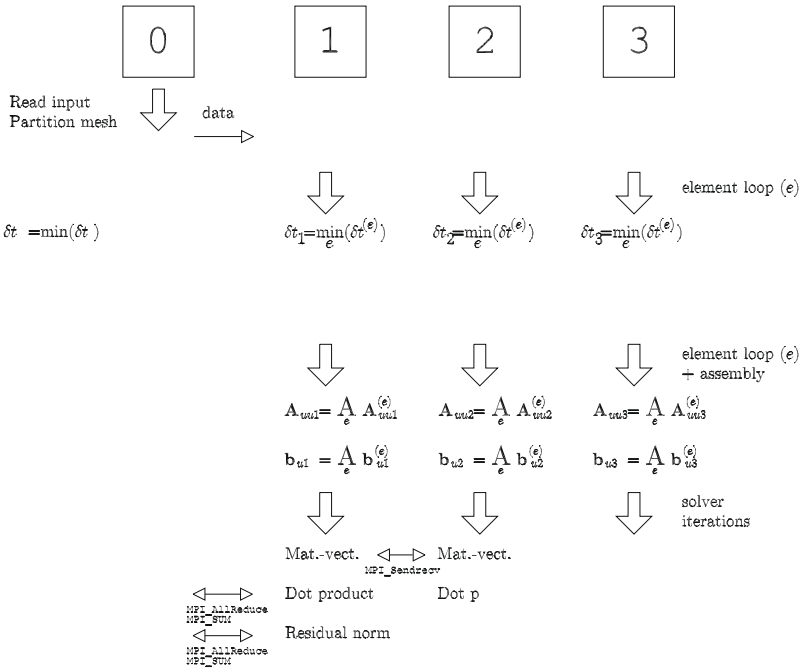


Fig. 4. Scheduling strategy on a simple example. (Top) Bad communication scheduling in five steps. (Bot.) Optimum communication scheduling in three steps.

The second important aspect is the *node numbering* of the local (slave) meshes. In the previous subsection we mentioned that for explicit as well as for implicit solvers, boundary nodal arrays must be exchanged between slaves. In order to perform an efficient data exchange, the local nodes are divided into three categories, as illustrated in Fig. 3. It should be pointed out that in the present implementation, boundary nodes are repeated over the subdomains. The nodes are numbered as followed:

- *Interior nodes.* They are the nodes which are not shared by another subdomain. They are numbered using the METIS function `METIS_NodeND` [34], which "computes fill reducing orderings of sparse matrices".
- *Boundary nodes.* They are the nodes for which subdomains contributions must be summed up when computing a RHS or a matrix-vector product. In turn, each subdomain divides its boundary nodes in Own and Others':



- *Own*: Boundary nodes for which the subdomain is responsible. This is useful for computing the convergence residual so that the residuals of the boundary nodes are accounted for only once before performing the sum over the subdomains with `MPI_Reduce`.
- *Others'*: Boundary nodes belonging to a neighboring subdomain.

The next aspect concerns the *communication scheduling* [35]. Fig. 4 illustrates the importance of scheduling on a simple example. For this example, each subdomain has to communicate with all others. On the top part of the figure, we observe the consequence of a bad scheduling, for which the communications are carried out in five steps. For example, during the first step, subdomain 2 cannot communicate with the subdomain 3 until this last one has communicated with subdomain 1. The way to optimize the scheduling is that at each communication step, every subdomain should communicate. The bottom part of the figure illustrates the optimum scheduling for this example, performed in three communication steps.

Finally, Fig. 5 shows all the ingredients and communications in play in the parallelization strategy. $\mathbf{A}_{uui}$ is the local matrix for the momentum equation of subdomain i, while $\mathbf{A}_{ppi}$ is the local matrix for the continuity equation of subdomain i. That is $\mathbf{A}_{uui}$ should be identified with $\mathbf{A}_{uu}$, $\mathbf{A}_{ppi}$ with $\mathbf{A}_{pp}$ and $\mathbf{P}_i$ with $\mathbf{P}$, in each subdomain i.

5. Numerical examples

In this section, the hard scalability for the proposed algorithm is shown through numerical examples of different complexity. Hard scalability is measured by fixing the size of the problem and run it for different numbers of sub-domains partitioned. The CPU time per time-step for each of the runs is then compared, being normalized by the figure that corresponds to the least partitioned case. When the memory required is not high, the normalizing CPU time is that of the sequential cases. Post-process file dump is excluded from the CPU time computation.

5.1. Example 1: Parallel performances of GMRES, Bi-CGSTAB and CG

Through this example we want to illustrate the different scalability properties of the GMRES, Bi-CGSTAB and CG solvers, as was mentioned at the beginning of Section 4. In particular, we want to show that for a given number of iterations, the scalability of the GMRES solver will decrease with increasing Krylov dimension. In order to do this, we consider a symmetric scalar diffusion problem, which can be converged with both solvers. As long as the GMRES method is concerned, we force the iterative method to perform 100 iterations, simply by setting the convergence tolerance to a negative value. Of course, this example is only illustrative because in real situations, a higher Krylov may help the solver to converge in less iterations, and therefore, the weight of the solver would decrease with respect to the weight of the element assembly. This last comment will be explored below, in Example 5.3. The problem is solved on a 1 M hexahedra mesh. Being a simple scalar diffusion equation, the weight of the solver is much higher than that of the element loop. Fig. 6 shows the scalabilities obtained and confirms the deterioration of the GMRES scalability with increasing Krylov dimensions, due to the increasing number of scalar products involved in the Gram-Schmidt orthogonalization.

5.2. Example 2: Scalability for MareNostrum PPC 970 vs. Blue Gene L/P

In this example we compare the scalability obtained on MareNostrum cluster (http://www.bsc.es) and that on Blue Gene/L and Blue Gene/P clusters with tests using up to 4000 CPUs. These supercomputers will be abbreviated MAR, BGL and BGP respectively. We compare in the following list the main characteristics of the machines:

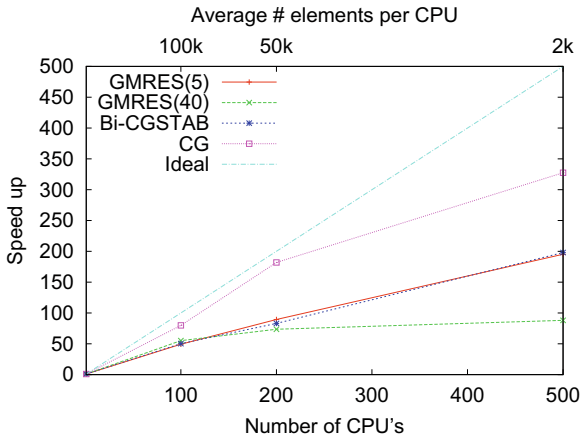


Fig. 6. Example 1. Comparison of the GMRES, Bi-CGSTAB and CG typical scalability for a fixed number of iterations.

- Processors:
  - MAR: IBM Power PC 970MP at 2.3 GHz;
  - BGL: IBM PowerPC 440 at 700 MHz;
  - BGP: IBM PowerPC 450 at 850 MHz.

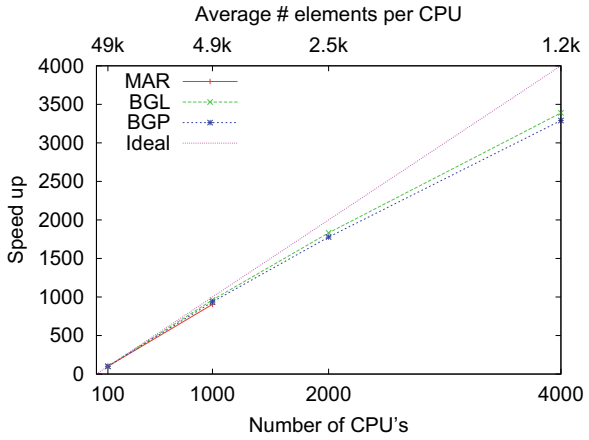


Fig. 7. Example 2. Speedup on MareNostrum, Blue Gene/L and Blue Gene/P.

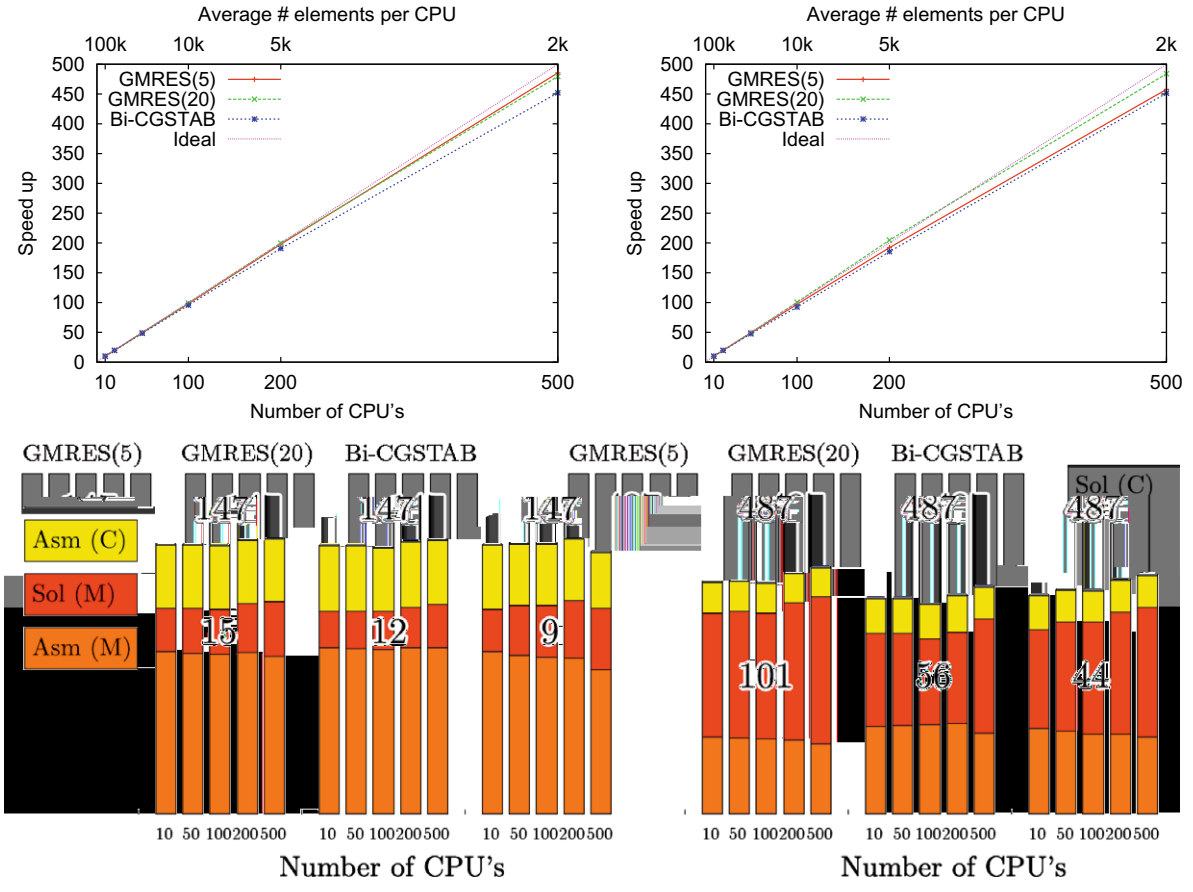


Fig. 8. Example 3. 1M mesh. (Top) (Left) Speedup, tolerance: 10^{-3}. (Top) (Right) Speedup, tolerance: 10^{-6}. (Bot.) Relative CPU time of the different tasks.

- Interconnection network:
  - MAR: Myrinet with 10 elements Clos256+256;
  - BGL/P: Three parallel communications networks: a 3D toroidal network for peer-to-peer communication between compute nodes, a collective network for collective communication, and a global interrupt network for fast barriers.

The example solves a three dimensional cavity flow using the IPS. The mesh is composed of 5 M hexahedra. All the element integrals are computed using 8 Gauss points. The GMRES solver is used with a Krylov dimension of 10 and a tolerance of 10^{-7}. The same tolerance is used for the CG solver. As an average, the number of iterations to converge the GMRES solver is around 200, and 800 iterations are necessary for the CG solver. Fig. 7 compares the scalabilities obtained on the three supercomputers. The speedup is computed using the first 10 iterations and normalized with respect to the CPU time obtained on 100 CPU's. On the top x-axis, we added the average number of elements per processor. As an example, we have only 1200 elements per processor when using 4000 CPU's. Despite this very low number, speed up is around 90% at 4000 CPU's with respect to 100 CPU's. We note that the speed up is decreasing with increasing clock frequency (MAR > BGP > BGL), as the weight of the communications increases with respect to that of the computation. We do not show the figures here, but the ratios of CPU times follow exactly the ratios of the processor clock frequencies.

5.3. Example 3: scalability Bi-CGSTAB vs. GMRES

The three-dimensional cavity flow is solved at Reynolds number 200. The method used to converge to a steady state is the first order in time IPS. We want to compare through this example the performances of the Bi-CGSTAB and GMRES (with Krylov dimension of 5) iterative solvers to solve the momentum equations. Two tolerances of the solvers are considered, 10^{-3} and 10^{-6}. The maximum number of iterations is chosen sufficiently high so that the tolerance is achieved at each time step. Two meshes are considered, of 1 M and 5 M hexahedra elements, and we use an 8 Gauss points integration rule.

The simulations were carried out on 10, 100, 200 and 500 CPU's. The speedup results are shown in Figs. 8 and 9 (Top) for the 1 M mesh and 5 M mesh, respectively. The corresponding bottom figures show the histograms representing the relative

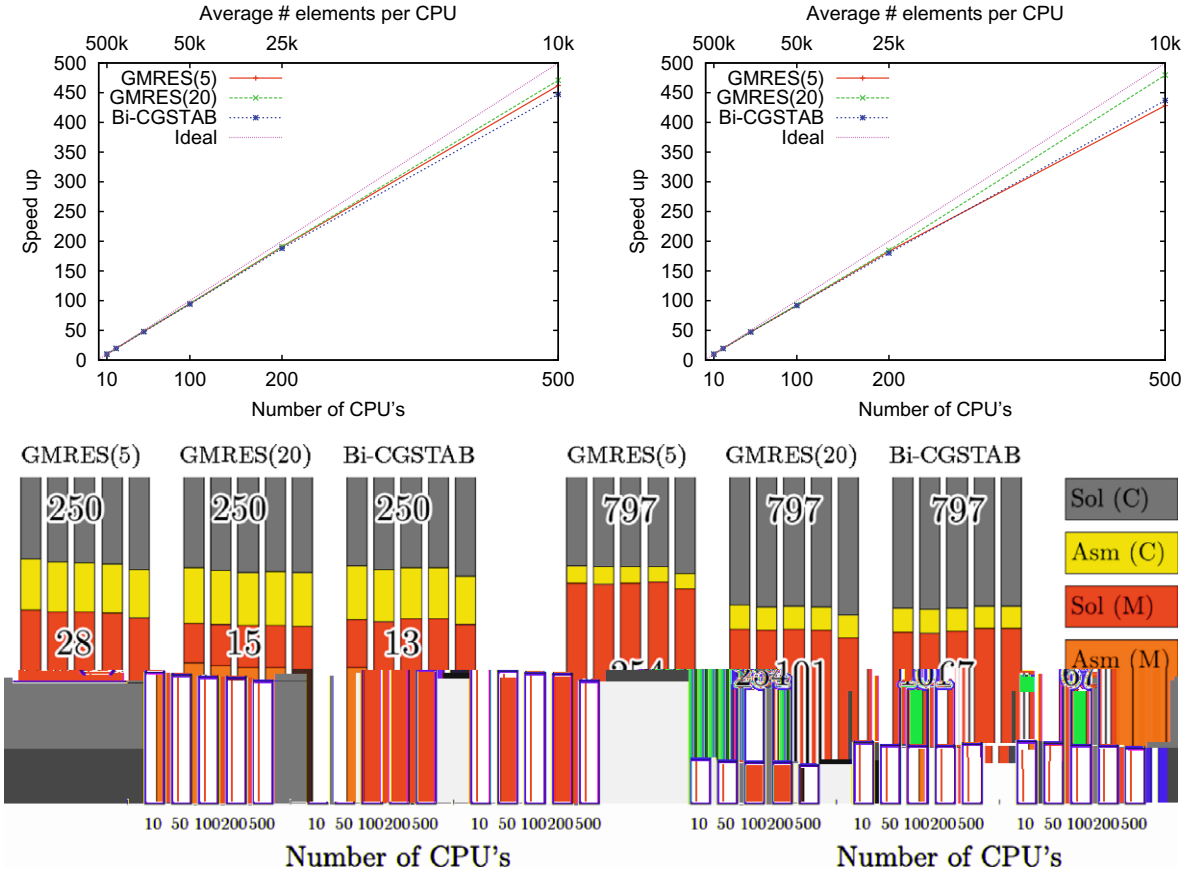


Fig. 9. Example 4. 5 M mesh. (Top) (Left) Speedup, tolerance: 10^{-3}. (Top) (Right) Speedup, tolerance: 10^{-6}. (Bot.) Relative CPU time of the different tasks.

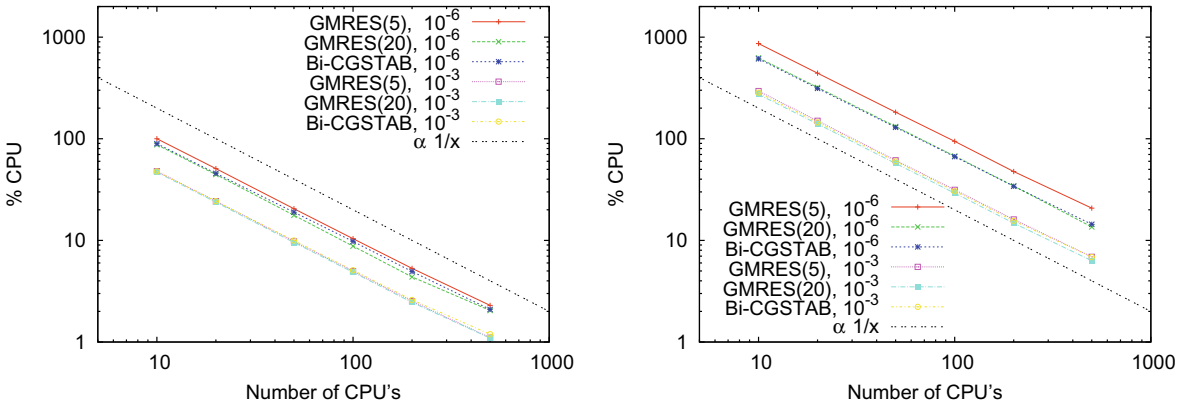


Fig. 10. Example 3. % CPU with respect to CPU time of GMRES(5), 1 M, 10^{-6} on 10 CPU's. (Top) 1 M mesh. (Bot.) 5 M mesh.

contributions in CPU time of the momentum assembly, momentum solver, continuity assembly and continuity solver. They are referred to as Asm (M), Sol (M), Asm (C) and Sol (C) respectively. The histograms also include the number of solver iterations required to reach the solver tolerances. Finally, Fig. 10 shows the CPU percentage with respect to CPU time of GMRES(5), using the 1 M mesh and tolerance of 10^{-6} on 10 CPU's. The following conclusions can be drawn:

- We know from the first example that, for a given number of GMRES iterations, the weight of the communication increases with the Krylov dimension. However, when the Krylov dimension increases, the convergence of the solver is achieved in less iterations. In fact, we observe that for both meshes and both solver tolerances, GMRES(20) has less contribution to the total CPU time that GMRES(5) (Figs. 8 and 9(Bot.);
- For a tolerance of 10^{-3}, the assemblies dominate over the solvers. For a tolerance of 10^{-6}, we have the opposite situation.
- We observe a slight decrease of parallel efficiency when the solver contribution dominates.

5.4. Example 4: Nasal airflow

We now present a real application case, the simulation of airflow within a human nose. This work has been carried out in collaboration with Imperial College London [36], who kindly provided the mesh and the boundary conditions of the problem. Their study aims to

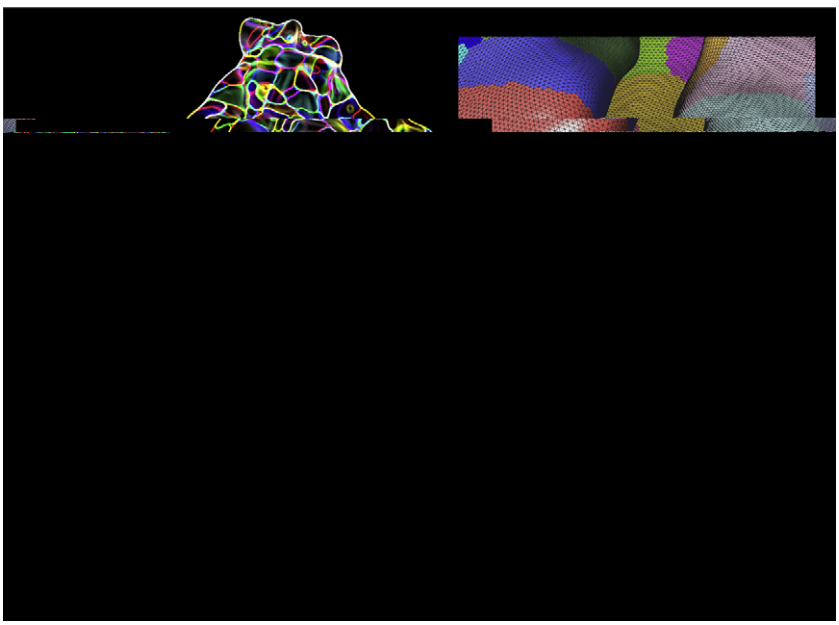


Fig. 11. Example 4. (Top) (Left) Partition in 500 subdomains. (Top) (Right) Detail of the surface mesh and partition. (Bot.) (Left) Streamwise cut and partition. (Bot.) (Right) Streamlines.

demonstrate that the intimate relation between nasal form and flow can be explored in greater detail than hitherto possible.

The mesh is composed of 6,352,819 elements with a three layer prism boundary layer adjacent to the wall (1,183,488 prisms) and a tetrahedron core (5,169,331 tetrahedra). The objective of this example is first to compare the results to those obtained by [36,37] and to explore the scalability. The speedup is obtained on MareNostrum supercomputer averaging the CPU time of the first 10 time steps, using the CPU time obtained on one CPU for the normalization. Fig. 11(Top) shows the partition of the mesh into 500 subdomains and a detail of the surface mesh. This test is particularly challenging for the speedup test for the following reasons:

- The mesh is hybrid. No special strategy is adopted here. The weight of the vertices of the element graph are the Gauss points of the elements: 4 for the tetrahedra and 6 for the prisms.
- We have used a fast element assembly for the continuity and momentum equations. For example, only the Laplacian part of the viscous term is taken into account. This was not the case in the previous examples.
- The mesh is heavily composed of tetrahedra. Therefore, the Hessian matrix is null so that no Laplacian term has to be included in the element matrices and right-hand sides. In addition the number of integration points is 4. In the previous examples, 8 Gauss points were used.
- The elongated geometry (high aspect ratio) combined with boundary layer elements lead to a large condition number. We therefore expect the CG algorithm to dominate the computation, and thus to strongly affect the speedup. This is illustrated by Fig. 11(Bot.) (Left) which shows a streamwise cut of the geometry.

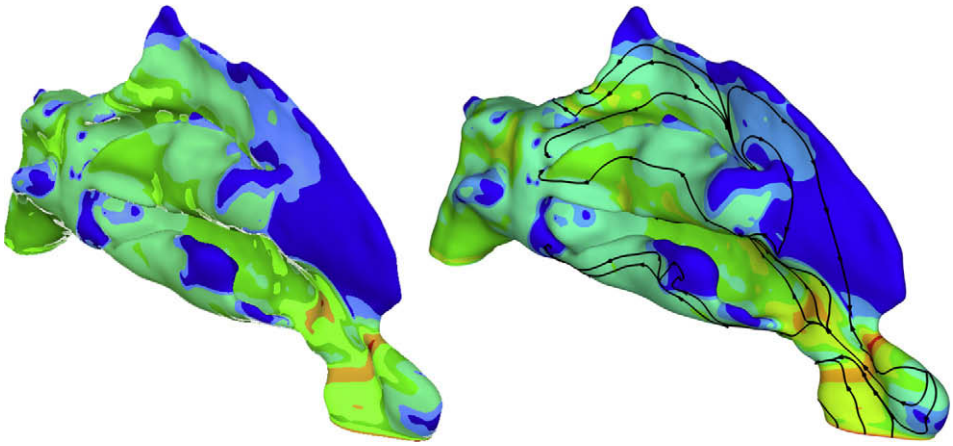


Fig. 12. Example 4. Wall shear stress. (Left) Present work. (Right) Fluent 6.2.16[36].

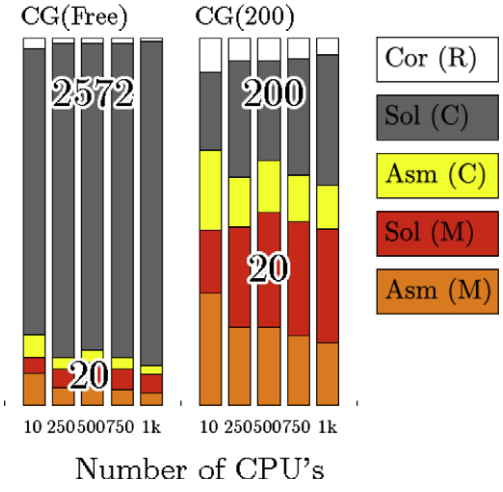


Fig. 13. Example 4. Percentage CPU time.

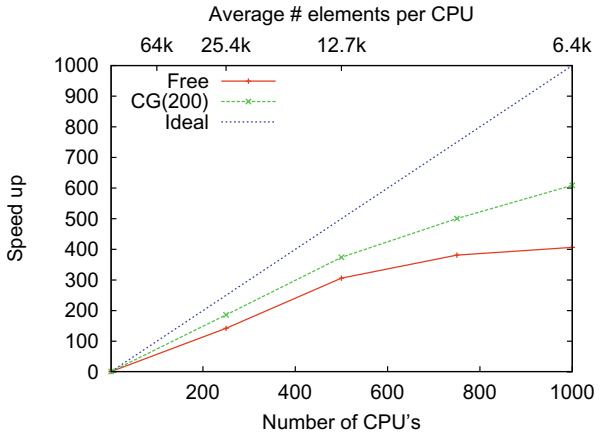


Fig. 14. Example 4. Speedup.

Fig. 12 shows the comparisons of the wall shear stress on the walls obtained with the present algorithm and those obtained by [36] using the commercial code [38]: very similar wall shear stress is obtained.

Two tests are carried out in order to see the impact of the CG solver on the speedup. In the first one, labeled CG(Free), we let both the CG and the GMRES (with Krylov 20) solvers converge to a tolerance of 10^{-5}. In the second test, referred to as CG(200), we perform a fixed number of CG iterations (200). As expected for the CG(Free) test, the CG solver convergence is very slow (2572 iterations in average) which represents 75% of the total CPU time for the sequential run as shown in Fig. 13. The speedup for this test is therefore closely related to that of the CG, as was shown in the first example described in Section 5.1. This is confirmed by Fig. 14, which shows that the speedup is not as good as in the cavity flow presented in Section 5.3. This is in fact due to the enormous weight of the CG solver in the computation, compared to the cavity flow. Nevertheless, reducing the number of CG iterations, the efficiency increases.

6. Conclusion and future work

We have presented the parallel implementation of both incremental projection and predictor–corrector schemes. To assess the massively parallel performance we studied two examples with different features. The first one is the cavity flow: the mesh is uniform, the geometry is a cube and the elements are hexahedra. For all these reasons, the solvers' relative contributions (in particular CG) are relatively low with respect to the element calculations. The speedup is almost linear up to 4000 CPUs on Blue Gene. The second example is the nasal airflow which exhibits the opposite behavior. The elemental calculations are very low with respect to the iterative solvers (in particular CG). The parallel efficiency is therefore lower than in the other example. It should be pointed out that we used a diagonal preconditioning.

Regarding future lines, we are presently testing the deflated CG method [39,40] together with a linelet preconditioner [41]. The preliminary speedup results for this solver are very encouraging. The mid-term developments will be based on the following points. First, the algorithm will be tested using a hybrid strategy, using MPI to carry out the communication between nodes and OpenMP for shared memory parallelization on the CPU's located on the same node. This should have beneficial effects on the scalability of the code. The other future line will consist in developing a strategy for treating hybrid meshes. In fact, the load balancing is a-priori difficult to estimate and cannot be simply based on the weight of the elemental work, like it would be possible for explicit solvers [42]. In this last case, the weight given to the elemental graph can be the number of integration points.

Acknowledgments

This work has been carried out in the framework of the Spanish Project OPTIDIS (ENE2005-05274) and the research of Dr. Houzeaux has been partly done under a *Ramon y Cajal* contract with the Spanish *Ministerio de Educación y Ciencia*. The authors would like to thank: P. Vezolle from IBM France for the run on Blue Gene Supercomputers at Montpellier and Watson IBM Centers and for his fruitful advise, D. Doorly, D. Taylor and J. Peiró of Imperial College London for the nasal airway mesh and to the Computer Science and Operations Departments of BSC-CNS for all the aspects related to the parallel performance analysis of the code.

References

[1] G. Strang, On the construction and comparison of difference schemes, SIAM J. Numer. Anal. 5 (3) (1968) 506–517.
[2] A. Chorin, A numerical method for solving incompressible viscous problems, J. Comput. Phys. 2 (1967) 12–26.

[3] R. Temam, Sur l'approximation de la solution des équations de Navier–Stokes par la méthode des pas fractionaires (I), Arch. Rat. Mech. Anal. 32 (1969) 135–153.
[4] O. Zienkiewicz, J. Szmelter, J. Peraire, Compressible and incompressible flow; an algorithm for all seasons, Comput. Meth. Appl. Mech. Eng. 78 (1990) 105–121.
[5] R. Codina, M. Vázquez, O. Zienkiewicz, A general algorithm for compressible and incompressible flow – Part III. The semi-implicit form, Int. J. Numer. Meth. Fluids 27 (1998) 13–32.
[6] J. Guermond, Some implementations of projection methods for Navier–Stokes equations, Math. Mod. Numer. Anal. 30 (5) (1996) 637–667.
[7] J. Kim, P. Moin, Application of a fractional-step method to incompressible Navier–Stokes, J. Comput. Phys. 33 (1985) 308–323.
[8] J. Perot, An analysis of the fractional step method, J. Comput. Phys. 108 (1993) 51–58.
[9] H. Elman, V. Howle, J. Shadid, R. Shuttleworth, R. Tuminaro, A taxonomy and comparison of parallel block multi-level preconditioners for ther incompressible Navier–Stokes equations, J. Comput. Phys. (2008) 1790–1808.
[10] A. Quarteroni, F. Saleri, A. Veneziani, Factorization methods for the numerical approximation of Navier–Stokes equations, Comput. Meth. Appl. Mech. Eng. 188 (2000) 505–526.
[11] S. Badia, R. Codina, Algebraic pressure segregation methods for the incompressible Navier–Stokes equations, Arch. Comput. Meth. Eng. 15 (2008) 343–369.
[12] G. Houzeaux, J. Principe, A variational subgrid scale model for transient incompressible flows, Int. J. Comput. Fluid Dyn. 22 (3) (2008) 135–152.
[13] G. Batchelor, An Introduction to Fluid Dynamics, Cambridge University Press, 1970.
[14] T.J. Hughes, Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods, Comput. Meth. Appl. Mech. Eng. 127 (1995) 387–401.
[15] R. Codina, Stabilized finite element approximation of transient incompressible flows using orthogonal subscales, Comput. Meth. Appl. Mech. Eng. 191 (2002) 4295–4321.
[16] R. Codina, Pressure stability in fractional step finite element methods for incompressible flows, J. Comput. Phys. 190 (2000) 1579–1599.
[17] H. Vander Vorst, Iterative Krylov Methods for Large Linear Systems, Cambridge University Press, 2003.
[18] Y. Saad, Iterative methods for sparse linear systems, SIAM (2003).
[19] A. Chorin, Numerical solution of the Navier–Stokes equations, Math. Comput. 22 (1968) 745–762.
[20] R. Temam, Sur l'approximation de la solution des équation de Navier–Stokes par la méthode des pas fractionaires (i), Arch. Rat. Mech. Anal. 32 (1969) 135–153.
[21] J. van Kan, A second-order accurate pressure correction scheme for viscous incompressible flow, SIAM J. Sci. Stat. Comput. Arch. 7 (1986) 870–891.
[22] R. Löhner, C. Yang, J. Cebral, F. Camelli, O. Soto, J. Waltz, Improving the speed and accuracy of projection-type incompressible flow solvers, Comput. Meth. Appl. Mech. Eng. 195 (2006) 3087–3109.
[23] R. Codina, O. Soto, Approximation of the incompressible Navier–Stokes equations using orthogonal subscale stabilization and pressure segregation on anisotropic finite element meshes, Comput. Meth. Appl. Mech. Eng. (2004) 1403–1419.
[24] R. Verfürth, Multilevel algorithms for mixed problems, SIAM J. Numer. Anal. 21 (2) (1984) 264–271.
[25] M. Griebel, T. Neunhoeffer, H. Regler, Algebraic multigrid methods for the solution of the Navier–Stokes equations in complicated geometries, Int. J. Numer. Meth. Fluids 26 (1998) 281–301.
[26] C. Iwamura, F.S. Costa, I. Sbarski, A. Easton, N. Li, An efficient algebraic multigrid preconditioned conjugate gradient solver, Comput. Meth. Appl. Mech. Engrg. 192 (2003) 2299–2318.
[27] D.J. Mavriplis, M.J. Aftosmis, M. Berger, High resolution aerospace applications using the NASA Columbia supercomputer, Int. J. High Perform. Comput. Appl. 21 (1) (2007) 106–126.
[28] D.J. Mavriplis, S. Pirzadeh, Large-scale parallel unstructured mesh computations for 3D high-lift analysis, AIAA Paper 990537.
[29] G. Golub, C.V. Loan, Matrix Computations, The Johns Hopkins University Press, 1996.
[30] E. Dean, R. Glowinski, On some finite element methods for the numerical simulation of incompressible viscous flow, in: M. Gunzburger, R. Nicolaides (Eds.), Incompressible Computational Fluid Dynamics, Cambridge University Press, Cambridge, 1993, pp. 17–65.
[31] J. Cahouet, J. Chabard, Some fast 3D finite element solvers for the generalized Stokes problem, Int. J. Numer. Meth. Fluids 8 (1988) 869–895.
[32] G. Houzeaux, R. Aubry, M. Vázquez, Extension of fractional step techniques for incompressible flows: The preconditioned Orthomin (1) for the pressure Schur complement, in preparation.
[33] M. Arioli, D. Loghin, A. Wathen, Stopping criteria for iterations in finite element methods, Numer. Math. 99 (2005) 381–410.
[34] Metis, family of multilevel partitioning algorithms: <http://glaros.dtc.umn.edu/gkhome/views/metis>.
[35] P. Brucker, Scheduling Algorithms, fourth ed., Springer, 2003.
[36] D. Doorly, D. Taylor, A. Gambaruto, R. Schroter, N. Tolley, Nasal architecture: form and flow, Phil. Trans. R. Soc. A, submitted for publication.
[37] D. Taylor, D. Doorly, R. Schroter, J. Peiro, R. Almeyda, N. Tolley, G. Houzeaux, M. Vázquez, Computational modelling of nasal airflow, in: Grand Challenges in Computational Biology. Joint BSC-IRB Barcelona Conference, Barcelona (Spain), communication. No proceeding, 2008.
[38] Fluent 6.2.16, ansys, inc., usa: <http://www.ansys.com>.
[39] R.A. Nicolaides, Deflation of conjugate gradients with applications to boundary value problems, SIAM J. Numer. Anal. 24 (1987) 355–365.
[40] R. Aubry, F. Mut, R. Löhner, Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation, J. Comput. Phys. 227 (24) (2008) 10196–10208.
[41] D. Martin, R. Löhner, An implicit linelet-based solver for incompressible flows, AIAA Paper(AIAA-92-0668).
[42] M. Vázquez, G. Houzeaux, R. Grima, J. Cela, Applications of parallel computational fluid mechanics in Mare Nostrum supercomputer: low-Mach compressible flows, in: PARCFD2007, Antalya, Turkey, 2007.