Journal of Computational Physics 229 (2010) 5123–5143

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Parallel finite element simulations of incompressible viscous fluid flow by domain decomposition with Lagrange multipliers

Christian A. Rivera [a], Mourad Heniche [a,*], Roland Glowinski [b], Philippe A. Tanguy [a]

[a] Research Center for Industrial Flows Processes (URPEI), Department of Chemical Engineering, École Polytechnique Montreal, P.O. Box 6079, Station Centre-Ville, Montréal, QC, Canada H3C 3A7
[b] Department of Mathematics, University of Houston, 651 PGH, Houston, TX 77204-3008, USA

ARTICLE INFO

Article history:
Received 3 April 2009
Received in revised form 22 January 2010
Accepted 22 March 2010
Available online 27 March 2010

Keywords:
Lagrange multiplier method
Finite element method
Parallel computing
Krylov methods
Domain decomposition method
Mesh partition
ILU preconditioning
Stokes equations

ABSTRACT

A parallel approach to solve three-dimensional viscous incompressible fluid flow problems using discontinuous pressure finite elements and a Lagrange multiplier technique is presented. The strategy is based on non-overlapping domain decomposition methods, and Lagrange multipliers are used to enforce continuity at the boundaries between subdomains. The novelty of the work is the coupled approach for solving the velocity–pressure-Lagrange multiplier algebraic system of the discrete Navier–Stokes equations by a distributed memory parallel ILU (0) preconditioned Krylov method. A penalty function on the interface constraints equations is introduced to avoid the failure of the ILU factorization algorithm. To ensure portability of the code, a message based memory distributed model with MPI is employed. The method has been tested over different benchmark cases such as the lid-driven cavity and pipe flow with unstructured tetrahedral grids. It is found that the partition algorithm and the order of the physical variables are central to parallelization performance. A speed-up in the range of 5–13 is obtained with 16 processors. Finally, the algorithm is tested over an industrial case using up to 128 processors. In considering the literature, the obtained speed-ups on distributed and shared memory computers are found very competitive.

1. Introduction

Computational fluid dynamics (CFD) models are typically based on the solution of the Navier–Stokes equations with help of discretization schemes such as the finite volume or finite element method. In most practical situations, the mesh needs to be highly refined to capture the physics, making the computations highly demanding in memory and time. To benefit from the full computational potential of nowadays multi-processors machines, it is mandatory to develop a parallel solver with distributed memory allowing the use of very refined grids that cannot be handled on a single processor. Some early developments in fluid mechanics were made by Farhat et al. [1] and Johan et al. [2]. These applications demonstrated the possibility to speed-up calculation by the use of several CPU processors.

The most time-consuming part of the simulation is the solution of the algebraic system of equations. Direct solution methods based on the LU-decomposition of the matrix and subsequent forward–backward substitution schemes are often selected as they are very robust especially for ill-conditioned problems. Parallel variants can be found in the case of multi-frontal methods [3] or sparse Cholesky algorithms as in SuperLU software [4]. An evaluation of several solvers as PARDISO, MA57 and BSLIB-EXT for the direct solution of large sparse linear systems can be found in Gould et al. [5]. They concluded

* Corresponding author. Tel.: +1 514 340 4040; fax: +1 514 340 4105.
E-mail addresses: christian.rivera@polymtl.ca (C.A. Rivera), mourad.heniche@polymtl.ca (M. Heniche), roland@math.uh.edu (R. Glowinski), philippe.tanguy@polymtl.ca (P.A. Tanguy).

0021-9991/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2010.03.028

that the parallel efficiency of these methods depends on equations reordering and pivoting strategies. Aggarwal et al. [6] and Henriksen and Keunings [7] have employed these solvers to simulate the flow of viscoelastic fluids. Unfortunately the large memory requirements limit its use to massive parallel computing composed of hundreds of processors. Li and Demmel [4] required 32 processors to solve a system of approximately 10^6 equations (40 K equations/processor).

To overcome the memory limitations, at least partly, an alternative is to use preconditioned iterative Krylov subspace methods. With these methods several algebra operations like sparse matrix–vectors products, inner products, vector updates and forward and backward substitutions need to be performed. Among these operations the preconditioning step is the most critical issue. Its parallelization has been the subject of numerous investigations. Hughes et al. [8] introduced the concept of element-by-element (EBE) preconditioners. It consists of computing the preconditioner from the element matrix avoiding the storage of both the global matrix and preconditioner. Different variants such as the clustered-element-by-element (CEBE), the mixed CEBE and the cluster companion (CC) preconditioners were proposed by Tezduyar and Liou [9] and Tezduyar et al. [10]. However, the effectiveness of the EBE technique is limited because it does not often provide a substantial improvement in CPU time with respect to fast sequential techniques [11]. Another type of techniques include multigrid approaches, physic based preconditioning, structure base preconditioning, matrix free preconditioning which are beyond the scope of this work. A detailed literature review about these preconditioning techniques in the context of Jacobian free Newton Krylov methods can be found in the work of Knoll and Keyes [12].

A preconditioner that is well known for its fast convergence rate and robustness is the incomplete factorization (ILU) preconditioner; however its parallelization is very challenging due to the forward and backward substitutions. One option to parallelize these procedures consists in reordering the variables with help of level scheduling techniques [13]. Nonetheless the results obtained in their work showed that the approach is useful only for 4–6 processors. Ma and Saad [14] reported similar results for multi-coloring techniques. Other approach is to combine iterative methods with domain decomposition methods which consist in the subdivision of the original computational domain into a set of interconnected subdomains. In this manner, internal unknowns are calculated in parallel, while the inter-processor communications is restricted to the unknowns at the subdomain interfaces. For example, ILU preconditioners based on overlapping partitions have been used in the case of viscoelastic fluid flows [15,16] and convection dominated flows [17,18]. Nevertheless, the data structure required by an overlapping partition is always more difficult to handle than the one employed by a non-overlapping one.

An alternative is to use non-overlapping domain decomposition. Wille et al. [19,20] and Staff and Wille [21] used non-overlapping domain decomposition, a priori pivoting and segregation of variables to parallelize the ILU preconditioning of a conjugate gradient solver. Results presented for the Navier–Stokes equations showed that the speed-up is limited due to the sequential characteristics of ILU algorithms. Other implementations based on non-overlapping partitions include the work of Sosonkina et al. [22] who introduce the parallel algebraic recursive multilevel solver (pARMS) applicable to sparse linear systems. The method is based on multilevel ILU techniques such as the ones presented in Saad and Zhang [23]. Henon and Saad [24] also present a parallel hierarchical interface domain decomposition technique for general problems where unknowns are reordered in blocks depending if they are located inside the subdomains or on the interfaces. As a

Table 1
Parallel performance of ILU iterative solvers reported in the literature.

Authors	Mesh structure	ILU parallelization technique	Number of unknowns (M)	(Maximum # processors) / (minimum # processors)	Speed-up for maximum number of processors	Efficiency for maximum number of processors (%)
Dutto and Habashi [13]	Unstructured finite element mesh	Level scheduling technique	0.025	8/1	2.01	25
Staff and Wille [21]	Unstructured finite element mesh	Domain decomposition technique	1	12/1	8.6	71
Henon et al. [48]	Unstructured finite element mesh	Block technique	0.5	16/1	9.8	61
Iwashita et al. [49]	Structured finite differences grid	Multicolor ordering technique	6.5	32/1	9.7	30
Iwashita et al. [49]	Structured finite differences grid	Block red–black ordering technique	6.5	32/1	14.5	45
Shen et al. [50]	Structured finite difference grid	Multilevel block technique	1	32/1	11.26	35
Henon and Saad [24]	Unstructured finite element mesh	Parallel hierarchical interface domain decomposition technique (PHIDAL)	0.5	128/2	18	28
Henon and Saad [24]	Unstructured finite element mesh	Parallel algebraic recursive multigrid method (pARMS)	0.5	128/2	30	47
Ma and Saad [14]	Unstructured mesh	Multicolor ordering technique	0.8	512/32	6.38	39

consequence a block diagonal matrix amenable for parallelization is obtained. Table 1 presents a summary of the speed-ups and parallel efficiencies obtained by state-of-the-art parallel ILU iterative solvers found in the literature. It is noted the difficulty to reach ideal efficiency as the number of processors increases.

Another type of non-overlapping domain decomposition methods are based on Lagrange multipliers constraints. There are different approaches for implementing Lagrange multiplier based methods. One is based on the elimination of the degrees of freedom internal to the subdomains in order to solve an interface problem by a conjugate gradient algorithm [25–27] as in the finite element tearing and interconnecting (FETI) method of Farhat and Roux [26]. It is worth noting that this method has been applied to incompressible flow in the work presented by Vanderstraeten and Keunings [28], Zsaki et al. [29] and Vereecke et al. [30]. A similar approach is presented by Glowinski et al. [31] in combination with fictitious domain methods.

In this work, we present a novel approach to parallelize an ILU iterative finite element solver for the Stokes equations (laminar flow) using Lagrange multipliers constraints. The advantage of this approach with respect to the standard finite element domain decomposition is shown in Fig. 1 for a 2D case. Standard finite element formulation requires communicating all the nodes belonging to the elements located on the subdomain interface. In our approach the communication is restricted only to the nodes that are on the subdomain interfaces allowing the reduction of inter-processor communication. Similar ideas have been applied to solve elasticity problems [32,33]. The finite element parallel solver was implemented in POLY3D (Rheosoft Inc.) software. The organization of the paper is as follows; in Section 2 we describe the proposed numerical model with emphasis on the domain decomposition mathematical formulation. Section 3 presents the details about the parallelization of the method using MPI. Section 4 shows results for three-dimensional benchmark cases as the pipe and cavity flows. The effect of several partitioning algorithms and reordering of variables is described.

2. Parallel numerical model

2.1. One-domain variational formulation

For the sake of brevity the mathematical formulation is presented for the steady Stokes problem in a computational domain Ω with boundary $\partial\Omega$ (Fig. 2(a)).

$$-\mu\nabla^2\mathbf{v} + grad\ p = \mathbf{f},\ \text{in}\ \Omega, \tag{1}$$

$$\text{div}\,\mathbf{v} = 0,\ \text{in}\ \Omega, \tag{2}$$

where $\mathbf{v}$ stands for the velocity, $\mathbf{f}$ the body force, p the pressure and μ the Newtonian fluid viscosity. It is well known that problem (1) and (2) is equivalent to finding the functions $\mathbf{v}$ and p from the following saddle point problem defined for any admissible $\mathbf{w}$ and q functions.

$$\inf_{\mathbf{w}\in[H_0^1(\Omega)]^3}\ \sup_{q\in L^2(\Omega)}\ L(\mathbf{w},q),\ \text{in}\ \Omega, \tag{3}$$

where

$$L(\mathbf{w},q) = \frac{\mu}{2}\int_\Omega |\text{grad}\,\mathbf{w}|^2 d\Omega - \int_\Omega q\,\text{div}\,\mathbf{w} d\Omega - \int_\Omega \mathbf{f}\cdot\mathbf{w} d\Omega,\ \text{in}\ \Omega. \tag{4}$$

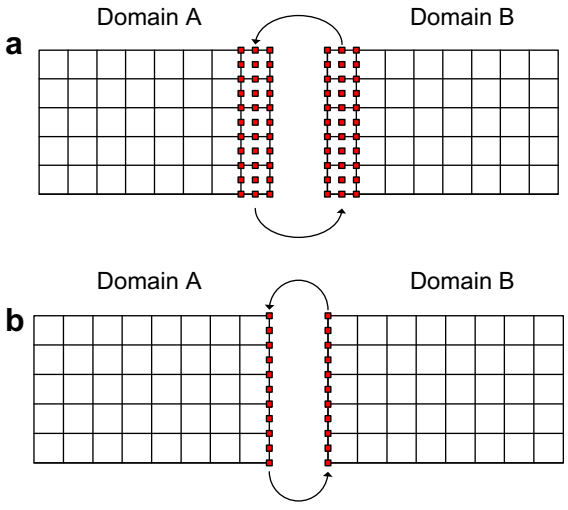


Fig. 1. Reduction in the communication for a two-domain decomposition: (a) standard finite element method and (b) finite element method with Lagrange multipliers.

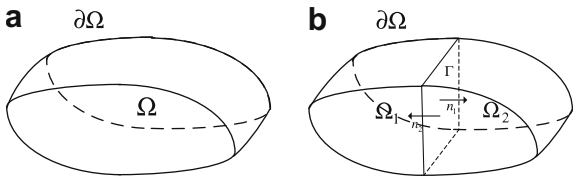


Fig. 2. Three-dimensional domain Ω with boundary $\partial\Omega$: (a) one partition and (b) partitioning into two subdomains Ω_1 and Ω_2.

This last expression is known as the Lagrangian functional. After derivation with respect to each variable, the Euler–Lagrange equations are obtained:

$$a(\mathbf{v},\psi) - b(\psi,p) = (\mathbf{f},\psi) \quad \forall\psi \in [H_0^1(\Omega)]^3, \text{ in } \Omega, \tag{5}$$

$$b(\mathbf{v},\varphi) = 0, \quad \forall\,\varphi \in L^2(\Omega), \text{ in } \Omega, \tag{6}$$

where

$$a(\mathbf{v},\psi) = \mu \int_\Omega \operatorname{grad}\mathbf{v} \cdot \operatorname{grad}\psi \, d\Omega, \tag{7}$$

$$b(\mathbf{v},\varphi) = \int_\Omega \varphi \cdot \operatorname{div}\mathbf{v} \, d\Omega, \tag{8}$$

and $(.,.)_\Omega$ is the scalar product in $L^2(\Omega)$:

$$(\mathbf{u},\mathbf{v})_\Omega = \int_\Omega \mathbf{u}\cdot\mathbf{v} d\Omega, \quad \forall\,\mathbf{u},\mathbf{v} \in L^2(\Omega) \tag{9}$$

in (5)–(8), ψ and φ stand for the shape functions for the velocity and pressure, respectively.

2.2. Two-domain decomposition method with Lagrange multipliers

First, a partition of the domain is introduced. For instance, let us consider two subdomains (Fig. 2(b)). Due the decomposition, the coupled problem defined by Eqs. (1) and (2) is equivalent to (10)–(13).

$$-\mu\Delta\mathbf{v}_i + \operatorname{grad} p_i = \mathbf{f}_i, \text{ in } \Omega_i \text{ for } i = 1,2, \tag{10}$$

$$\operatorname{div}\mathbf{v}_i = 0, \text{ in } \Omega_i, \text{ for } i = 1,2, \tag{11}$$

$$\mathbf{v}_1 = \mathbf{v}_2, \text{ in } \Gamma, \tag{12}$$

$$\frac{\partial\mathbf{v}_1}{\partial\mathbf{n}_1} = -\frac{\partial\mathbf{v}_2}{\partial\mathbf{n}_2}, \text{ in } \Gamma, \tag{13}$$

where $\mathbf{n_i}$ stands for the outward normal to the parallel boundary Γ. Dirichlet boundary conditions are assumed on the domain boundary. Thus, the domain decomposition removes the strong point-wise continuity at the parallel boundary by a weak integral condition generating extra constraints over the subdomains interfaces (Eqs. (12) and (13)). We resort to Lagrange multiplier method and constrained optimization techniques to reformulate the problem in (10)–(13) to find the solutions $\mathbf{v}$, p and λ of the modified saddle point problem defined for any admissible $\mathbf{w}$, q and $\boldsymbol{\mu}$ functions:

$$\inf_{\mathbf{w}\in[H_0^1(\Omega)]^3} \sup_{q\in L^2(\Omega)} \sup_{\boldsymbol{\mu}\in[L^2(\Gamma)]^3} L_{p_i}(\mathbf{w}_i, q_i, \boldsymbol{\mu}) \text{ in } \Omega_i \quad \text{for } i = 1,2, \tag{14}$$

where,

$$L_p(\mathbf{w},q,\boldsymbol{\mu}) = L_i(\mathbf{w},q) - \int_\Gamma \boldsymbol{\mu}\cdot(\mathbf{w}_1 - \mathbf{w}_2)\,\mathrm{d}\Gamma. \tag{15}$$

In (15) we introduce a Lagrange multiplier function $\boldsymbol{\mu}$ associated with the interface boundary condition. The Euler–Lagrange equations corresponding to this constrained problem are given by

$$a(\mathbf{v}_i,\psi) = (\mathbf{f}_i,\psi) + b(\psi,p_i) + (\boldsymbol{\lambda},\psi)_\Gamma, \forall\,\psi \in [H_0^1(\Omega_i)]^3, \text{ in } \Omega_i \text{ for } i = 1,2, \tag{16}$$

$$b(\mathbf{v}_i,\varphi) = 0, \quad \forall\,\varphi \in L^2(\Omega_i), \text{ in } \Omega_i \text{ for } i = 1,2, \tag{17}$$

$$((\mathbf{v}_1 - \mathbf{v}_2),\xi)_\Gamma = 0, \quad \forall\,\xi \in [L^2(\Gamma)]^3 \text{ at } \Gamma, \tag{18}$$

where ξ stands for the shape function of the Lagrange multiplier space. The solution of the Lagrange multiplier function $\boldsymbol{\lambda}$ is nothing but the jump of normal derivates at the subdomain interface [34]. To connect multiple subdomains (as in the simple example of Fig. 3), the only valid constraints to impose are the ones where a face (edge for two-dimensional problems) connecting subdomains exists. Thus in our example, only the following constraints are considered.

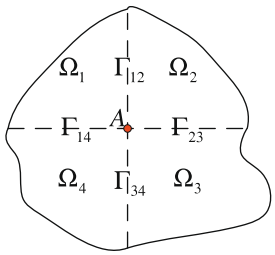


Fig. 3. Multiple domain decomposition of a 2D computational domain Ω in four Ω_i subdomains and respective boundaries Γ_{ij}.

$$\int_{\Gamma 12} \boldsymbol{\mu}_{12} \cdot (\mathbf{w}_1 - \mathbf{w}_2) \mathrm{d}\Gamma_{12}, \tag{19}$$

$$\int_{\Gamma 14} \boldsymbol{\mu}_{14} \cdot (\mathbf{w}_1 - \mathbf{w}_4) \mathrm{d}\Gamma_{14}, \tag{20}$$

$$\int_{\Gamma 23} \boldsymbol{\mu}_{23} \cdot (\mathbf{w}_2 - \mathbf{w}_3) \mathrm{d}\Gamma_{23}, \tag{21}$$

$$\int_{\Gamma 34} \boldsymbol{\mu}_{34} \cdot (\mathbf{w}_3 - \mathbf{w}_4) \mathrm{d}\Gamma_{34}. \tag{22}$$

This is in agreement with the discussion presented by Farhat and Roux [26] for the FETI method. As they pointed out, this particularity helps to reduce communication between processors improving parallel computing performance.

2.3. Finite element discretization

In the present work, P_1^+–P_0 (Fig. 4(a)) [35] and P_2^+–P_1 (Fig. 4(b)) [36] tetrahedral finite elements approximations are used. The former is an enriched version of the P_1–P_0 finite element. Extra degrees of freedom are added at the middle of each face and the pressure is assumed constant at each element. The latter element approximates the velocity by continuous quadratic shape functions, while the pressure and its gradients are computed inside each element. Both elements belong to the class of discontinuous pressure elements that satisfy the Brezzi-Babuska condition ensuring numerical stability.

The Lagrange multiplier space Λ is discretized using Dirac delta functions that are defined by:

$$\delta(\mathbf{x} - \mathbf{x}_i) = \begin{cases} +\infty & \text{if } \mathbf{x} = \mathbf{x}_i, \\ 0 & \text{if } \mathbf{x} \neq \mathbf{x}_i. \end{cases} \tag{23}$$

It must be remarked that when considering the spaces in (18), the Dirac delta function is not in L^2. However, in this work its use to connect multiple subdomains at the discrete level makes sense. The use of Dirac function is inspired by the fictitious domain method [37,38]. This is done with the purpose to simplify the discretization of the constraint in (18) allowing its imposition point-wise, and therefore eliminating the need to compute surface integrals at the parallel boundary. Thus for each constraint distributed over the set of nodes $\{\mathbf{x}_i\}_{i=1}^N$ that lie over the interface between subdomains, we have,

$$\mathbf{v}_1(\mathbf{x}_i) = \mathbf{v}_2(\mathbf{x}_i), \tag{24}$$

where subscripts 1 and 2 stand for subdomain label.

A fully coupled approach is used to solve the set of Eqs. (16)–(18). Meaning that velocity, pressure and multipliers are solved simultaneously. The matrix form of the problem is the following

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1^T & 0 & 0 & -\mathbf{K}_{\Lambda 1}^T \\ \mathbf{B}_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{A}_2 & \mathbf{B}_2^T & \mathbf{K}_{\Lambda 2}^T \\ 0 & 0 & \mathbf{B}_2 & 0 & 0 \\ -\mathbf{K}_{\Lambda 1} & 0 & \mathbf{K}_{\Lambda 2} & 0 & 0 \end{bmatrix} \begin{Bmatrix} \mathbf{U_1} \\ \mathbf{P_1} \\ \mathbf{U_2} \\ \mathbf{P_2} \\ \Lambda \end{Bmatrix} = \begin{Bmatrix} \mathbf{F_1} \\ \mathbf{0} \\ \mathbf{F_2} \\ \mathbf{0} \\ \mathbf{0} \end{Bmatrix}, \tag{25}$$

where $\mathbf{A_i}$ stands for the convection–diffusion matrix, $\mathbf{B_i}$ the matrix obtained from the incompressibility constraint, $\mathbf{B_i^T}$ the transpose of $\mathbf{B_i}$, $\mathbf{K}_{\Lambda i}$ the matrix from the interface constraint, $\mathbf{U_i}$, $\mathbf{P_i}$, Λ and $\mathbf{F_i}$ stand for the velocity, pressure, Lagrange multipliers and body forces, respectively.

Although the mathematical formulation is well-posed, the presence of zeros on the main diagonal makes ILU failing due to the lack of pivoting during the ILU decomposition. To overcome this problem, we introduce penalty parameters in (25) for both the divergence and Lagrange multipliers equations. In this way Eqs. (17) (pressure $\boldsymbol{p}$) and (18) (Lagrange multiplier $\boldsymbol{\lambda}$) are replaced by

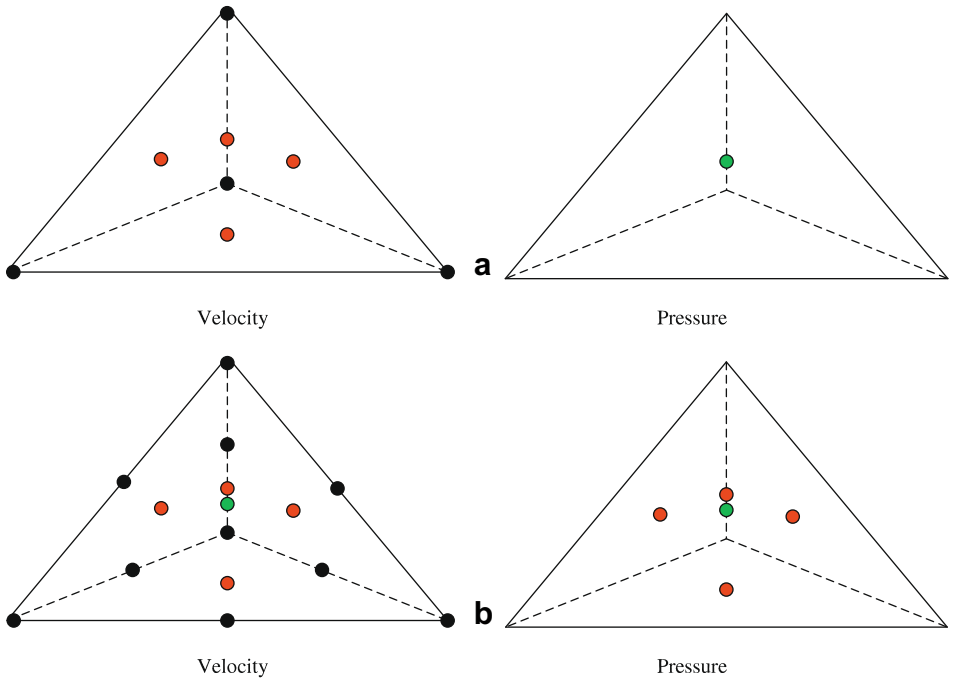


Fig. 4. Tetrahedral finite element approximations: (a) P1$^+$–P0 and (b) P2$^+$–P1.

$$b(\mathbf{v}_i, \varphi) = -(p_i, \varphi)/\varepsilon_p, \quad \forall \phi_h \in P_h, \text{ in } \Omega_i \text{ for } i = 1, 2, \tag{26}$$

$$((\mathbf{v}_1 - \mathbf{v}_2), \xi)_\Gamma = -(\boldsymbol{\lambda}, \xi)_\Gamma/\varepsilon_\lambda, \quad \forall\, \xi_h \in \Lambda_h, \tag{27}$$

where ε_p and ε_λ are penalty parameters. According to (26) and (27), the matrix form (25) is rewritten as follows:

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1^T & 0 & 0 & -\mathbf{K}_{\Lambda 1}^T \\ \mathbf{B}_1 & \varepsilon_p^{-1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{A}_2 & \mathbf{B}_2^T & \mathbf{K}_{\Lambda 2}^T \\ 0 & 0 & \mathbf{B}_2 & \varepsilon_p^{-1} & 0 \\ -\mathbf{K}_{\Lambda 1} & 0 & \mathbf{K}_{\Lambda 2} & 0 & \varepsilon_\lambda^{-1} \end{bmatrix} \begin{Bmatrix} \mathbf{U_1} \\ \mathbf{P_1} \\ \mathbf{U_2} \\ \mathbf{P_2} \\ \Lambda \end{Bmatrix} = \begin{Bmatrix} \mathbf{F_1} \\ \mathbf{0} \\ \mathbf{F_2} \\ \mathbf{0} \\ \mathbf{0} \end{Bmatrix}. \tag{28}$$

In this work, penalty parameters are defined by the expressions given in (29) and (30).

$$\varepsilon_p = 10^{-\alpha}/\rho\mu, \tag{29}$$

$$\varepsilon_\lambda = 10^{-\beta} h/\rho\mu, \tag{30}$$

where α and β has been heuristically determined to be between 6 and 12. In this work, a value of 8 is taken for both parameters. A fixed-point Newton iterative algorithm is chosen to solve the problem in incremental form to give the two-domain solution algorithm showed in Fig. 5. It must be remarked that only one fixed point iteration is required for linear problems.

3. Parallel implementation

The parallel implementation of the algorithm in Fig. 5 is composed of several key elements; in this section we describe the particularities for each one of them.

3.1. Domain partitioning

To partition the computational domain several mesh partitioning schemes such as coordinate recursive bisection, multilevel k-way [39], multilevel-KL [40] and spectral octasection partitioning method [40] were considered. Furthermore Kernigham-Lin local refinement [41] and terminal propagation [42] were also investigated in combination with spectral methods. Kernigham-Lin local refinement is nothing but a greedy local smoothing strategy, moving elements between subdomains in an effort to reduce the amount of interfacial area. Terminal propagation is another technique that yields a partition connectivity suitable for the machine architecture. In our case, we use this feature to simplify as much as possible the connectivity

0. Given $\boldsymbol{U}^{(0)}$, $\boldsymbol{P}^{(0)}$ and $\boldsymbol{\Lambda}^{(0)}$.

1. For n = 1, 2, …, until convergence:

1.1. Solve simultaneously for $\delta\boldsymbol{U}$, $\delta\boldsymbol{P}$ and $\delta\boldsymbol{\Lambda}$:

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1^{\mathrm{T}} & \mathbf{0} & \mathbf{0} & -\mathbf{K}_{\Lambda 1}^{\mathrm{T}} \\ \mathbf{B}_1 & \varepsilon_p^{-1}\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_2 & \mathbf{B}_2^{\mathrm{T}} & \mathbf{K}_{\Lambda 2}^{\mathrm{T}} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_2 & \varepsilon_p^{-1}\mathbf{I} & \mathbf{0} \\ -\mathbf{K}_{\Lambda 1} & \mathbf{0} & \mathbf{K}_{\Lambda 2} & \mathbf{0} & \varepsilon_\lambda^{-1}\mathbf{I} \end{bmatrix} \begin{Bmatrix} \delta\mathbf{U}_1 \\ \delta\mathbf{P}_1 \\ \delta\mathbf{U}_2 \\ \delta\mathbf{P}_2 \\ \delta\boldsymbol{\Lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{R}_{\mathbf{v1}^{(n)}} \\ \mathbf{R}_{\mathbf{p1}^{(n)}} \\ \mathbf{R}_{\mathbf{v2}^{(n)}} \\ \mathbf{R}_{\mathbf{p2}^{(n)}} \\ \mathbf{R}_{\boldsymbol{\lambda}^{(n)}} \end{Bmatrix}$$

$$\begin{Bmatrix} \mathbf{R}_{\mathbf{v1}^{(n)}} \\ \mathbf{R}_{\mathbf{p1}^{(n)}} \\ \mathbf{R}_{\mathbf{v2}^{(n)}} \\ \mathbf{R}_{\mathbf{p2}^{(n)}} \\ \mathbf{R}_{\boldsymbol{\lambda}^{(n)}} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F}_1 \\ \mathbf{0} \\ \mathbf{F}_1 \\ \mathbf{0} \\ \mathbf{0} \end{Bmatrix} - \begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1^{\mathrm{T}} & \mathbf{0} & \mathbf{0} & -\mathbf{K}_{\Lambda 1}^{\mathrm{T}} \\ \mathbf{B}_1 & \varepsilon_p^{-1}\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_2 & \mathbf{B}_2^{\mathrm{T}} & \mathbf{K}_{\Lambda 2}^{\mathrm{T}} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_2 & \varepsilon_p^{-1}\mathbf{I} & \mathbf{0} \\ -\mathbf{K}_{\Lambda 1} & \mathbf{0} & \mathbf{K}_{\Lambda 2} & \mathbf{0} & \varepsilon_\lambda^{-1}\mathbf{I} \end{bmatrix} \begin{Bmatrix} \mathbf{U}_1^{(n)} \\ \mathbf{P}_1^{(n)} \\ \mathbf{U}_2^{(n)} \\ \mathbf{P}_2^{(n)} \\ \boldsymbol{\Lambda}^{(n)} \end{Bmatrix}$$

1.2. Update $\mathbf{U}_i^{(n)}$, $\mathbf{P}_i^{(n)}$ and $\boldsymbol{\Lambda}^{(n)}$ in Ω_i for $i = 1, 2$: $\begin{cases} \mathbf{U}_i^{(n+1)} = \mathbf{U}_i^{(n)} + \delta\mathbf{U}_i \\ \mathbf{P}_i^{(n+1)} = \mathbf{P}_i^{(n)} + \delta\mathbf{P}_i \\ \boldsymbol{\Lambda}^{(n+1)} = \boldsymbol{\Lambda}^{(n)} + \delta\boldsymbol{\Lambda} \end{cases}$

Fig. 5. Two-domain fixed-point solution algorithm.

of the partition generating stripped partitions. The multilevel *k*-way partitioning technique is available from Metis software [39]. Multilevel-KL and all spectral partitioning techniques are available in Chaco software [40].

To characterize how well the work is uniformly distributed among the processors we define the load distribution that can be computed by

$$\text{load distribution} = \min(\text{NNZ})/\max(\text{NNZ}), \tag{31}$$

where NNZ stands for a list that contains the number of non-zero entries in each sub-matrix. The size of each sub-matrix is determined by NNZ instead of number of equations (NEQ) because the demand of CPU and memory is directly related to NNZ, and not to NEQ. In this way, the load distribution takes values between 0 and 1; a value of 1 corresponds to the ideal load distribution and a value of 0 corresponds to the worst case scenario.

3.2. Krylov subspace iterative solvers

System of equations of algorithm in Fig. 5 is solved by iterative Krylov method, which is composed of parallelizable algebra operations like matrix–vector products, inner products and vector updates. This approach is characterized by its low memory requirements allowing the solution of larger number of equations per processor than with direct methods. The choice of iterative methods for the systems arising from the finite element discretization of the Navier–Stokes equations is restricted by the non-symmetry and non-definite positiveness properties of the matrix. Conjugate gradient method, very efficient for symmetric matrices, become inapplicable. One should use variants based on Quasi-Minimal Residuals (QMR) or Biconjugate Gradient Stabilized approach (BiCGSTAB).

Preconditioning is a central issue to ensure convergence. In this work, an incomplete factorization with zero fill-ins (ILU (0)) was selected due to its well known robustness in comparison to other techniques (e.g. diagonal preconditioners), low memory requirements and reduced operations counts with respect to their fill-in counterparts [43]. Matrix factorization and forward–backward substitution operations on a memory distributed parallel environment has been performed based on algorithms presented by Saad [43].

3.3. Ordering of variables

To increase the parallel efficiency the variables were reordered in blocks, depending if the equations correspond to internal, interface, pressure or Lagrange multipliers. In this way, linear algebra operations can be performed in parallel for the internal block variables, while the inter-processor communication is only required for the interface velocity and the

Lagrange multipliers blocks. To reduce the memory required per processor, a piece of the global matrix of (28) is build and stored in each processor. The ordering of variables inside each sub-matrix can play a critical role in the convergence rate behaviour of the Krylov solver as shown in Heniche et al. [44]. It consists in reordering internal velocity (*Ux*, *Uy*, *Uz*), interface velocity (*I*), pressure (*P*) and Lagrange multipliers (*LM*) sub-blocks in each of the sub-systems that form the global system of equations.

Five configurations were tested based on modifying the structure of the internal velocity and pressure sub-blocks. Lagrange multiplier and interface velocity sub-blocks were not modified since parallel efficiency depends on them. One option is to keep contiguous each component of the velocity for each node in the mesh while the pressure is keep in a separate block; this generated two orderings depending if the pressure sub-block is located after or before the interface velocity, represented as follows:

$$UxUyUz{-}I{-}P{-}LM \rightarrow \left\langle \begin{matrix} u_{intx1}, u_{inty1}, u_{intz1}, \ldots, u_{intxn}, u_{intyn}, u_{intzn}, \\ u_{ifacex1}, u_{ifacey1}, u_{ifacez1}, \ldots, u_{ifacexm}, u_{ifaceym}, u_{ifacezm}, \\ p_1, \ldots, p_n, \\ \lambda_{x1}, \lambda_{y1}, \lambda_{z1}, \ldots, \lambda_{xm}, \lambda_{ym}, \lambda_{zm}. \end{matrix} \right\rangle, \tag{32}$$

$$UxUyUz - P - I - LM \rightarrow \left\langle \begin{matrix} u_{intx1}, u_{inty1}, u_{intz1}, \ldots, u_{intxn}, u_{intyn}, u_{intzn}, \\ p_1, \ldots, p_n, \\ u_{ifacex1}, u_{ifacey1}, u_{ifacez1}, \ldots, u_{ifacexm}, u_{ifaceym}, u_{ifacezm}, \\ \lambda_{x1}, \lambda_{y1}, \lambda_{z1}, \ldots, \lambda_{xm}, \lambda_{ym}, \lambda_{zm}. \end{matrix} \right\rangle. \tag{33}$$

Another variant was to consider a sub-block for each component of the velocity and pressure; this was denoted as *Ux–Uy–Uz–I–P–LM*.

$$Ux{-}Uy{-}Uz{-}I{-}P{-}LM \rightarrow \left\langle \begin{matrix} u_{intx1}, \ldots, u_{intxn}, u_{inty1}, \ldots, u_{intyn}, u_{intz1}, \ldots, u_{intzn}, \\ p_1, \ldots, p_n, \\ u_{ifacex1}, u_{ifacey1}, u_{ifacez1}, \ldots, u_{ifacexm}, u_{ifaceym}, u_{ifacezm}, \\ \lambda_{x1}, \lambda_{y1}, \lambda_{z1}, \ldots, \lambda_{xm}, \lambda_{ym}, \lambda_{zm}. \end{matrix} \right\rangle. \tag{34}$$

The opposite was to consider a single sub-block for both velocity and pressure where each component of the velocity and pressure equations are keep adjacent for each node in the mesh; this was referred as *UxUyUzP–I–LM*.

$$UxUyUzP{-}I{-}LM \rightarrow \left\langle \begin{matrix} u_{intx1}, u_{inty1}, u_{intz1}, p_1, \ldots, u_{intxn}, u_{intyn}, u_{intzn}, p_n, \\ u_{ifacex1}, u_{ifacey1}, u_{ifacez1}, \ldots, u_{ifacexm}, u_{ifaceym}, u_{ifacezm}, \\ \lambda_{x1}, \lambda_{y1}, \lambda_{z1}, \ldots, \lambda_{xm}, \lambda_{ym}, \lambda_{zm}. \end{matrix} \right\rangle. \tag{35}$$

Finally, the pressure sub-block was positioned before the internal velocity in the ordering named *P–UxUyUz–I–LM*.

$$P{-}UxUyUz{-}I{-}LM \rightarrow \left\langle \begin{matrix} p_1, \ldots, p_n, u_{intx1}, u_{inty1}, u_{intz1}, \ldots, u_{intxn}, u_{intyn}, u_{intzn}, \\ u_{ifacex1}, u_{ifacey1}, u_{ifacez1}, \ldots, u_{ifacexm}, u_{ifaceym}, u_{ifacezm}, \\ \lambda_{x1}, \lambda_{y1}, \lambda_{z1}, \ldots, \lambda_{xm}, \lambda_{ym}, \lambda_{zm}. \end{matrix} \right\rangle. \tag{36}$$

3.4. Communication

The communication pattern is governed by the forward/backward parallel substitution algorithms presented in Saad [43] that were adapted to the linear system in Fig. 5. They consist in two steps, the first one consists in sending the interface velocity from subdomain i to subdomains j where $j > i$ (forward communication). The second step requires to send the Lagrange multiplier values from subdomain i to subdomains j where $j < i$ (backward communication). Fig. 6 presents a schematic representation of this process for the case of two partitions.

In the case of multiple partitions as shown in Fig. 7, it is necessary to communicate data to only certain partitions. One way is to communicate from the sender partition to the receivers (the ones that are related with the sender) making a

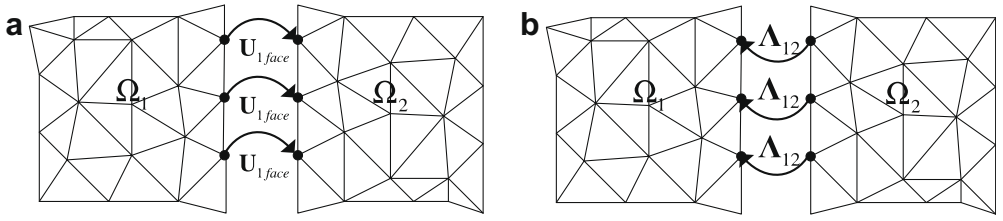


Fig. 6. Schematic representation of the communication between two subdomains: (a) forward communication for the interface velocities and (b) backward communication for the Lagrange multipliers.

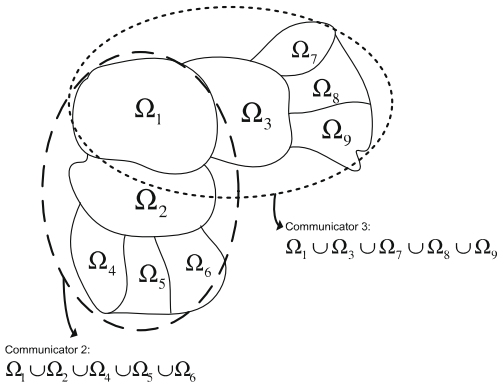


Fig. 7. Schematic representation of a domain partitioned into nine subdomains, that exemplifies the set of subdomains that define communicators for subdomain 2 (broken line) and 3 (dotted line).

send-receive call at each time a partition communicates to other partitions. By experience this is very inefficient procedure in terms of communication and overhead time. A better option is to use MPI_BROADCAST routine, which allows sending data in an optimized way from a 'sender' process to the rest of partitions included in the communicator. The communicator here is defined as a collection of processes that can send messages to each other. By default MPI uses MPI_COMM_WORLD communicator where all the processes in the computation are included. For our application the use of a unique MPI_COMM_WORLD communicator in the broadcast of information would generate overhead time for the processes that are not related to the sender. This is caused by the fact that the broadcast function would not return until all the data is send to all receiver processes included in MPI_COMM_WORLD. To avoid this, we have defined several sets of communicators, one per each process, which is composed of the partitions that are related between them. Fig. 7 exemplifies the set of subdomains that compose the communicator for subdomains 2 and 3. In this way, in the case of several subdomains, broadcast operations make use of these particular communicators instead of MPI_COMM_WORLD. Thus, processes that are not related to the sender can continue their work while processes that are connected can receive their information.

4. Three-dimensional benchmark cases

The presented numerical methodology was tested on two benchmark problems, namely the flow in a pipe and the lid-driven cavity flow, with the purpose to analyze the effect of the shape of the geometry over the parallel performance. Since the convective term was not considered at this point, Stokes flows (linear problems) were solved by means of the Lagrange multiplier based parallel Conjugate Gradient Krylov solver developed in this work. Furthermore, the solutions obtained by the CG Krylov solver were considered converged when the norm ratio between the last residual (r^i) and the first residual (r^0) satisfies the following:

$$\|r^i\|/\|r^0\| < 10^{-6}, \tag{37}$$

where $\|\cdot\|$ stands for the Euclidean norm. The initial solution for all cases was set to zero. The simulations were run on a 16 processors IBM-P690 with 64 GB of shared memory. In Section 4.1 and 4.2 Metis was used for partitioning and the variables were *UxUyUz–I–P–LM* ordered.

4.1. Flow in a pipe

The partitioning for the pipe flow consisted in single connected strips where each interface connects only two subdomains avoiding the multiple subdomains issue described in Section 2.2. The total number of elements, number of equations, number of non-zero entries in the matrix and time required to solve the problem in sequential mode for each of the computational grids can be found in Table 2. A characteristic of the method is the growth in size of the global system of equations

Table 2
Number of equations and size of the matrix for the pipe flow test case.

Nomination	Finite element	# Elements	# Nodes	**v**–**p** equations ($\times 10^6$)	NNZ ($\times 10^6$)	CPU time (min)
Mesh1	$P1^+$–P0	100 k	338.5 k	0.7	28.7	3.30
Mesh2		200 k	646.8 k	1.4	256.8	43.28
Mesh3		400 k	1.32 M	2.9	545.9	43.87
Mesh1	$P2^+$–P1	100 k	474.6 k	1.6	128.5	30.85
Mesh2		200 k	903.7 k	3.1	256.5	176.56
Mesh3		400 k	1.84 M	6.5	545.9	255.34

due to the addition of Lagrange multipliers equations with respect to the number of subdomains as can be seen on Fig. 8. As was already discussed, the partitioning was performed by means of Metis software that distributes in an almost uniform fashion the number of elements in the subdomains. This characteristic can be better observed with help of the load balance ratio as shown in Table 3.

Fig. 9 presents the speed-up per CG iteration with respect to the CPU time to solve the problem by a sequential ILU (0)-CG solver. A value close to 12 was obtained with $P2^+$–P1 element, while the maximum speed-up/iteration with $P1^+$–P0 was around 11.3.

It can be noted that the speed-up is not linear due to: (i) the loss of efficiency due to the matrix–vector and preconditioning operations because processors need to wait for data to be sent from contiguous processors to pursue the calculations; this loss of synchronization between processors is called "overhead"; (ii) the increase in the number of conjugate gradient iterations required to reach convergence with respect to the number of partitions (Table 4). At this stage, let us mention that in Table 4 in going from mesh 2 to mesh 3, one may notice the strong reduction of the number of iterations to converge the problem so that the corresponding CPU times reported in Table 2 not increase significantly although if the number of equations doubles. Also, in some cases one observes a decrease of the number of iterations as the number of partitions increases (mesh3/$P1^+$–P0 and mesh2/$P2^+$–P1), but in overall, a slight increase (around 1.5 times for 16 partitions) in the number of iterations is observed as the number of partitions increases.

The parallelization is based on the addition of Lagrange multiplier equations to the original algebraic system of equations to connect the multiple subdomains. As a consequence, if the number of partitions increases, the problem size increases as shown in Fig. 8. Then, one can expect the number of iterations to grow with increasing the number of partitions. The dependence of the iteration number with respect to the partition number is a typical behaviour of ILU based Krylov parallel solvers when the parallelization is made on a global reordering of the equations [1–5]. One heuristic explanation is that the different global equations ordering produce different ILU preconditioner non-zero entries matrix profiles that affect the convergence rate [6,7]. We will get back to this issue in Section 4.2.2.

Finally, we present in Table 5 the overall speed-up that combines both the convergence rate deterioration due to overhead and the increment in conjugate gradient iterations with respect to the lowest CPU time we could obtain for an ILU

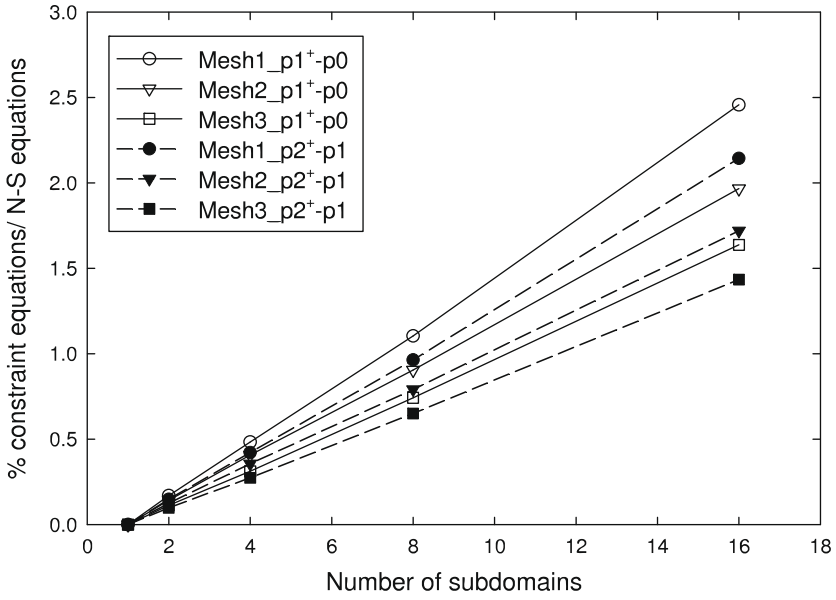


Fig. 8. Increase of the number of equations due to the introduction of Lagrange multiplier constraints at the interface with respect to the number of subdomains for $P1^+$–P0 and $P2^+$–P1 finite element for pipe flow test case.

Table 3
Ratio between minimum and maximum number of NNZ in the subdomains for pipe flow test case.

Processors	Mesh1	Mesh2	Mesh3
2	0.985	0.998	0.997
4	0.973	0.990	0.979
8	0.951	0.940	0.964
16	0.941	0.910	0.939

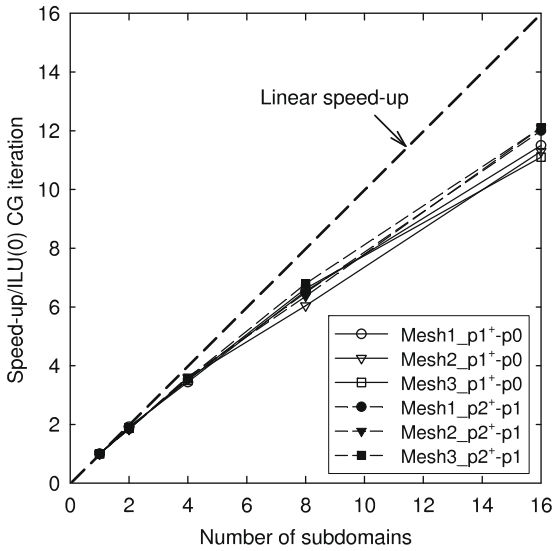


Fig. 9. Speed-up per ILU (0) preconditioned conjugate gradient iteration for the pipe flow test case with $P1^+$–P0 and $P2^+$–P1 finite elements.

Table 4
Number of conjugate gradient iterations to solve the pipe flow test case.

Processors	Mesh1	Mesh2	Mesh3	Mesh1	Mesh2	Mesh3
	$P1^+-P0$			$P2^+-P1$		
1	284	1894	826	567	1530	1084
2	333	2913	596	615	1615	1440
4	341	2697	629	718	1793	1100
8	351	2230	623	669	1345	1146
16	372	2793	643	683	1249	1503

Table 5
Overall speed-up for pipe flow test case (CPU time in seconds in parenthesis).

Processors	Mesh1	Mesh2	Mesh3	Mesh1	Mesh2	Mesh3
	$P1^+-P0$			$P2^+-P1$		
2	1.6 (118)	1.2 (2116)	2.6 (1012)	1.7 (1079)	1.7 (6034)	1.4 (10,779)
4	2.9 (67)	2.5 (1030)	4.6 (572)	2.8 (651)	3.1 (3414)	3.5 (4333)
8	5.3 (36)	5.1 (504)	8.8 (298)	5.6 (330)	7.2 (1461)	6.5 (2366)
16	8.8 (21)	7.7 (336)	14.3 (182)	10.1 (182)	14.9 (709)	8.8 (1734)

(0)-CG solver running in sequential mode. The speed-up for the maximum number of partitions employed (16) varies in the range of 8–14 leading to a parallel efficiency of 50–88%.

4.2. Lid driven cavity flow

The objective of this second example is to apply the parallel computational method developed in this work to the lid-driven cavity flow case where a more challenging hydrodynamics complexity takes place. Three unstructured meshes were employed to conduct the analysis. The Metis based partitions into 2, 4, 8, 16 subdomains are presented in Fig. 10. In comparison to the pipe flow, this problem has the particularity that the subdomains are highly interconnected yielding several nodes that connects multiple subdomains requiring a more complicated communication pattern. Table 6 presents the total number of elements, number of equations, the number of non-zero entries in the matrix and time required to solve the problem in sequential mode for each of the computational grids. Fig. 11 presents the percentage of Lagrange multiplier equations ob-

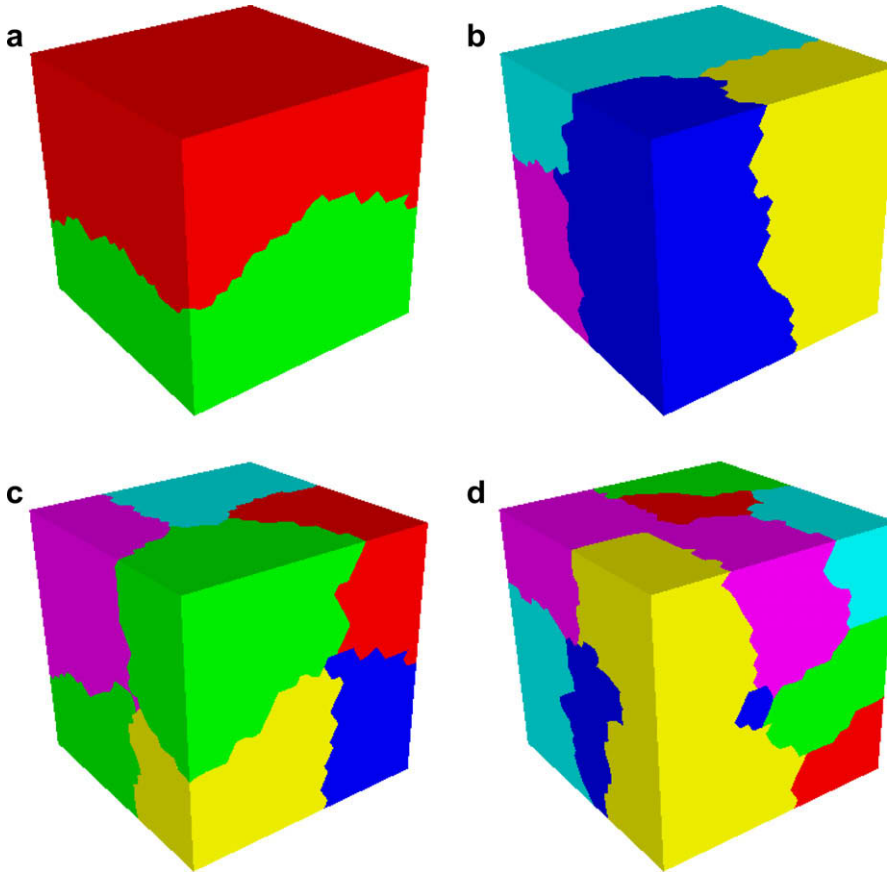


Fig. 10. Partitions generated by METIS: (a) 2 partitions, (b) 4 partitions, (c) 8 partitions, and (d) 16 partitions.

Table 6
Number of equations per mesh used and corresponding CPU time required to solve with a single processor the cavity flow test case.

Nomination	Finite element	# Elements	# Nodes	Equations ($\times 10^6$)	NNZ ($\times 10^6$)	CPU time (min)
Mesh1	$P1^+$–P0	100 k	360.1 k	0.8	36.4	4.83
Mesh2		200 k	685.9 k	1.5	71.0	13.66
Mesh3		400 k	1.303 M	3.0	137.2	171.47
Mesh1	$P2^+$–P1	100 k	498.6 k	1.8	159.7	48.63
Mesh2		200 k	947.4 k	3.5	310.6	136.49
Mesh3		400 k	1.79 M	6.7	599.4	541.60
Mesh4		800 k	3.69 M	13.9	625.9	912.35

tained for both type of finite element discretization used in this work. It can be observed that the amount of constraints required for this case is larger than for the pipe flow since the interfacial area is considerably superior. Table 7 provides the sub-matrices balance defined in the same way as in previous section. It is worth to underline that it is hard to obtain a good load distribution ratio as the partitions increases varying from 0.97 for two subdomains to around 0.778–0.84 for 16 subdomains.

Fig. 12 presents the speed-up per conjugate gradient iteration with respect to the sequential ILU (0)-CG solver. As should be expected, the speed-up is lower than for the pipe flow problem due the larger connectivity between sub-matrices taking values in the range of 9.5–11 with 16 processors. It can also be observed that the expected increase in parallel efficiency as the problem becomes larger (increasing NEQ values). Table 8 presents the number of iterations to reach convergence. Contrary to the pipe flow case, the number of iterations tends to increase in a more regular fashion with respect to the partitions. At this stage, it must be remarked that by looking at Table 8 one may notice that NNZ increases by a factor of 2 in going from mesh 2 to mesh 3 and, in the mean time, the CPU time required to solve increases by a factor of 10. The reason of this lack of proportionality is that when using Krylov iterative methods for solving the required number of iterations to converge is difficult to predict. However the CPU time per iteration better scales with NNZ; for example for these two meshes, the CPU time per iteration almost double, from 1.86s to 3.33s; then the proportionality is more balanced.

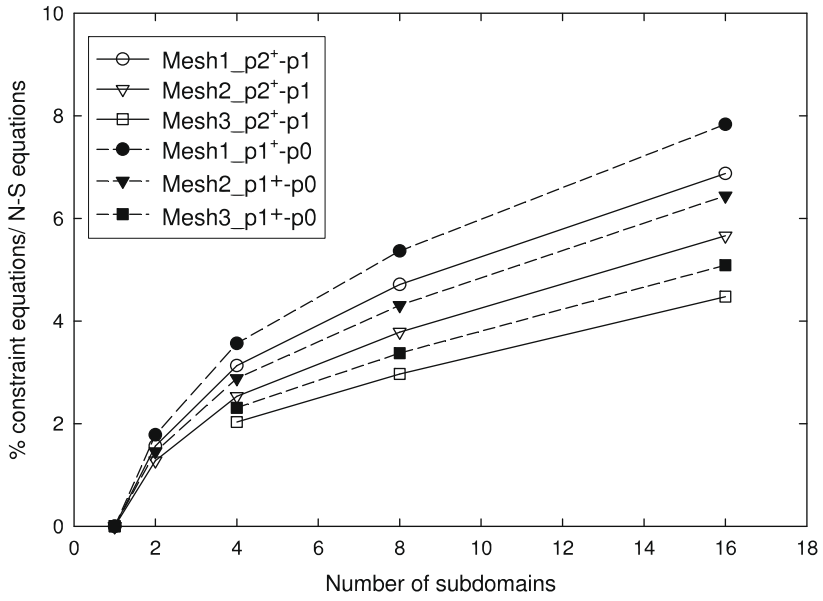


Fig. 11. Increase of the number of equations due Lagrange multiplier constraints at the interface for P1$^+$–P0 and P2$^+$–P1 finite element for cavity flow test case.

Table 7
Ratio between minimum and maximum number of NNZ in the subdomains for cavity flow test case.

Processors	Mesh1	Mesh2	Mesh3	Mesh1	Mesh2	Mesh3
	$P1^+-P0$			$P2^+-P1$		
2	0.985	0.974	–	0.986	0.976	–
4	0.913	0.976	0.963	0.917	0.978	0.964
8	0.869	0.945	0.902	0.878	0.951	0.908
16	0.779	0.821	0.800	0.804	0.840	0.820

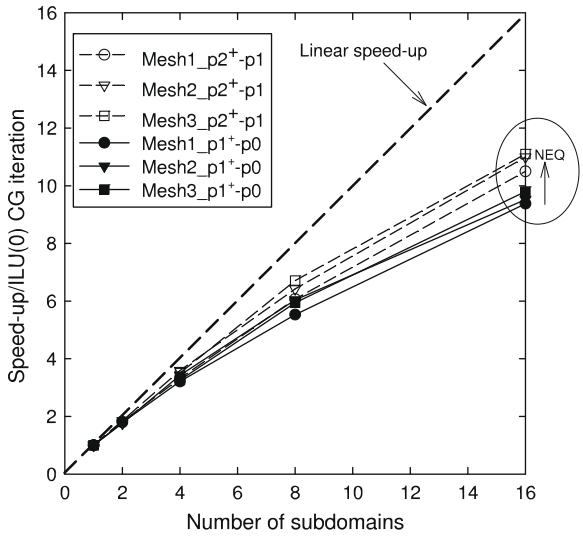


Fig. 12. Speed-up per ILU (0) preconditioned conjugate gradient iteration for the cavity flow test case with P1$^+$–P0 and P2$^+$–P1 finite elements.

The overall speed-up with respect to the sequential run is in the range of 5–5.5 for 16 processors (Table 9). The differences between the iteration speed-up and the overall speed-up are discussed in Section 4.2.2.

Table 8
Number of conjugate gradient iterations to solve the cavity flow test case.

Processors	Mesh1	Mesh2	Mesh3	Mesh1	Mesh2	Mesh3
	$P1^+-P0$			$P2^+-P1$		
1	316	432	3076	685	961	1908
2	422	537	Not run	867	1146	Not run
4	487	628	4091	1064	1585	2577
8	523	715	4160	1138	1528	2713
16	587	754	5001	1440	1831	3823

Table 9
Overall speed-up for cavity flow test case (CPU time in seconds in parenthesis).

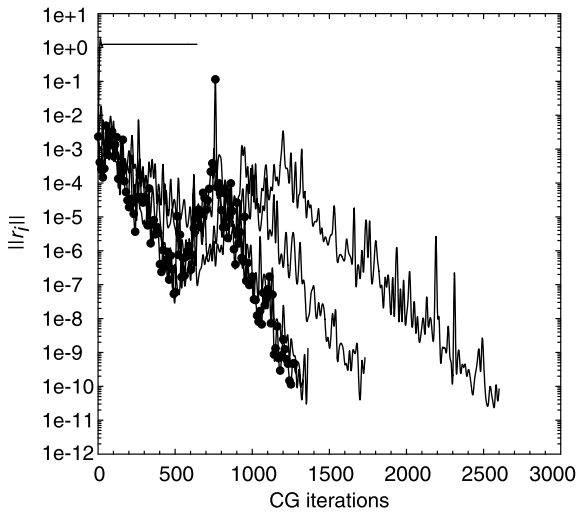
Processors	Mesh1	Mesh2	Mesh3	Mesh1	Mesh2	Mesh3
	$P1^+$–P0			$P2^+$–P1		
2	1.3 (202)	1.4 (553)	Not run	1.4 (1950)	1.6 (5134)	Not run
4	2.1 (131)	2.4 (332)	2.5 (4172)	2.1 (1308)	2.1 (3661)	2.5 (12343)
8	3.3 (82)	3.7 (215)	4.4 (2329)	3.7 (767)	4.0 (1971)	4.7 (6782)
16	5.0 (54)	5.5 (144)	5.5 (1850)	5.0 (560)	5.8 (1374)	5.5 (5772)

4.2.1. Effect of ordering of the variables

As preliminary test it was necessary to known the optimal way to order the unknowns in the system of equations. We present in Fig. 13 the obtained convergence rates of ILU (0)-CG when 16 partitions are employed for the studied configurations. It is observed that *UxUyUz–I–P–LM* produces the best results. Similar trends were observed when the number of partitions was varied. This is the ordering that is used in next section.

4.2.2. Effect of partitioning schemes

Two factors limit the overall speed-up of the method. The first factor is dealing with the parallel implementation of the linear algebra where the main parameter to consider is the speed-up per iteration. It is dominated by the parallel efficiency of the employed parallel algebra operations, where parallel matrix–vector products and ILU preconditioning operations are the more time consuming as evidenced in Fig. 14. It must be remarked that the ideal linear speed-up is very difficult to obtain due to the unbalanced charge loading when using unstructured meshes.



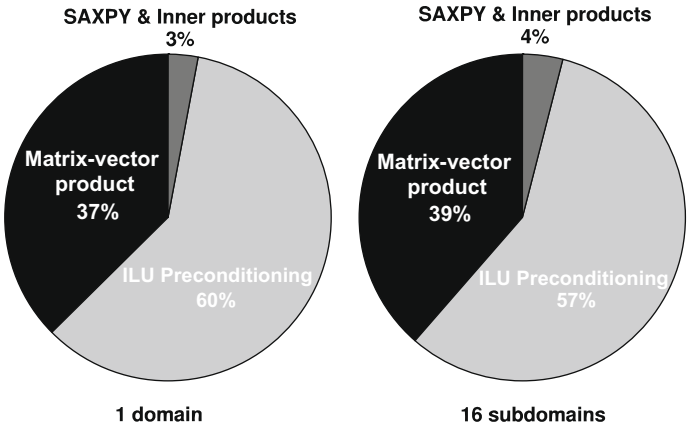


Fig. 14. CPU time distribution at each conjugate gradient iteration.

The second factor is dealing with the numerical scalability governed by the number of iterations required to reach convergence. The general trend is that the number of CG iterations increases with respect to partitioning. One cause of this trend is attributed to the total number of equations growth. It is also expected that the global matrix structure plays a role in the numerical performance since it affects the quality of ILU preconditioner [45,46]. Figs. 15 and 16 illustrate how the matrix structure evolves as the number of partitions increases having an impact on the quality of ILU preconditioner. Another factor that affects the parallel scalability is the smoothness of the interfaces. To observe its effect over the speed-up, a 100 K element block structured mesh was discretized by $P2^{+}$–P1 finite elements. Coordinate recursive bisection was employed producing smooth partitions as evidenced in Fig. 17(a) in opposite to the irregular ones presented in Fig. 17(b). This latter configuration was attained by the coordinate recursive bisection method applying a small perturbation that produces an irregular subdomain interface shape. It was important to use the same partitioning method to keep the subdomains connectivity unchanged. As can be observed in Fig. 18, the alteration of interface smoothness causes a considerable increase of the number of iterations. As the interface becomes more irregular, the number of Lagrange multiplier equations increases inducing a decay of the convergence rate.

The convergence rate is also associated with the structure of the global matrix for the already discussed reasons. To demonstrate that we have run some experiments using different partition schemes to solve the cavity flow case. The purpose of

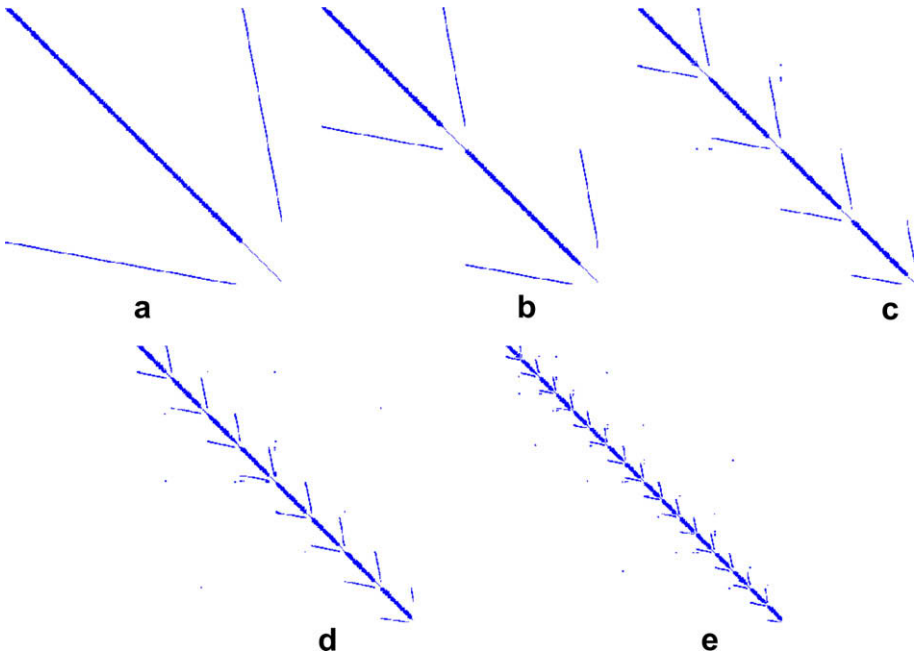


Fig. 15. Matrix non-zero entries for pipe flow test case: (a) 1 domain; (b) 2 domains; (c) 4 domains; (d) 8 domains; (e) 16 domains.

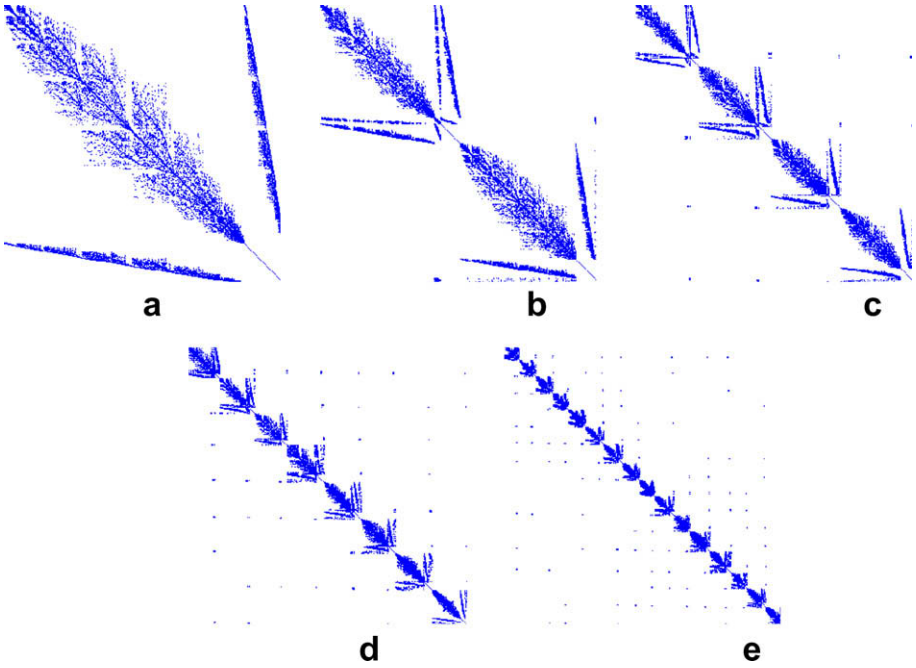


Fig. 16. Matrix non-zero entries for cavity flow test case: (a) 1 domain; (b) 2 domains; (c) 4 domains; (d) 8 domains; (e) 16 domains.

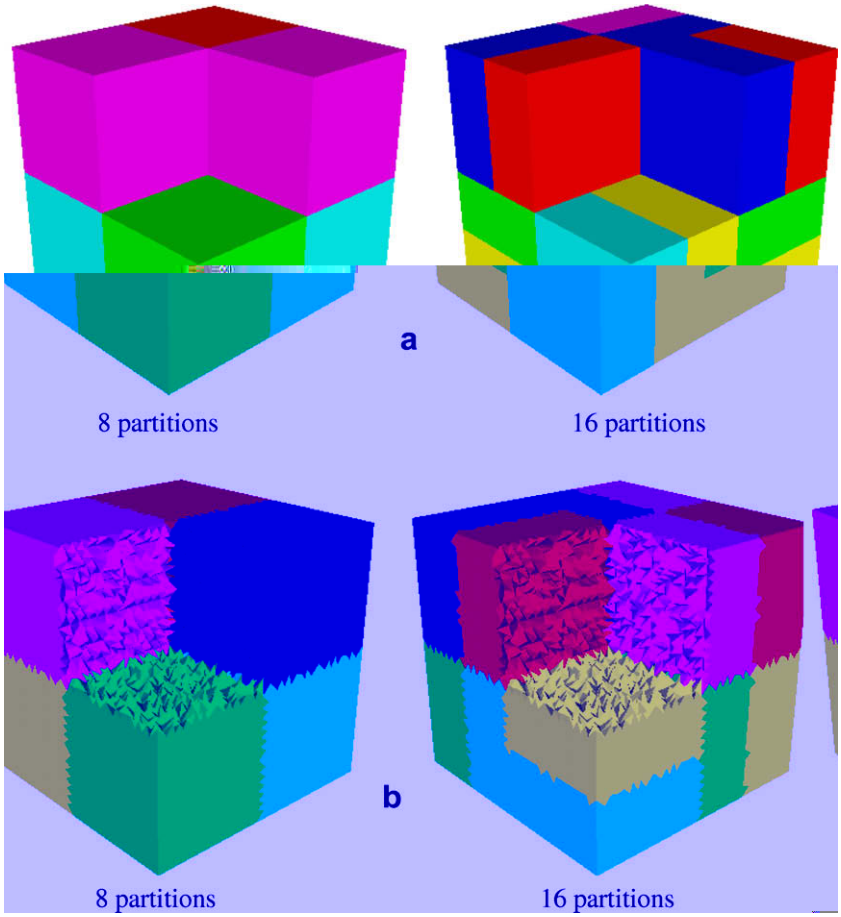


Fig. 17. Partitions generated by recursive bisection technique: (a) smooth interfaces and (b) irregular interfaces.

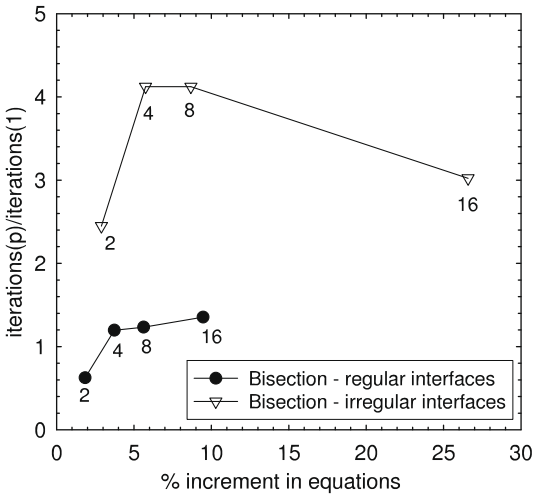


Fig. 18. Normalized variation in iterations for cavity flow test case (numbers close to symbols indicate the number of partitions used for each point in the graph).

the experiments is that different partition algorithms would produce different amount of multiplier equations and non-zero matrix entries patterns allowing to check their respective influence. Table 10 shows the obtained results, spectral octasection with terminal propagation and Kerninghan-Lin smoothing being the one that gives the best results in terms of number of iterations and overall speed-up (8.9), even faster than the case where smooth interfaces where employed (8.4). When considering spectral-octasection parallel performance it is interesting to mention that a good speed-up per iteration does not imply a good overall speed-up. The results confirm that both the amount of Lagrange multipliers and the matrix structure (average number of adjacent subdomains) influence the overall performance of the method. The data suggests that irregular subdomains connectivity with large non-smooth interfacial area induces a deterioration of the convergence rate. When comparing the performance of spectral methods it appears that the interface smoothing, by Kernighan-Lin heuristics, is critical to reduce the number of CG iterations. The smoothing action of Kerninghan-Lin rule can be observed in Fig. 19. Furthermore, the speed-up per iteration remains almost constant for the different schemes studied. Finally, we have run again the cases presented in Table 6 with this new partitioning strategy (spectral method-Kernighan-Lin refinement- terminal propagation) to verify the validity of our findings. Table 11 summarizes the new speed-ups for these cases using 16 partitions. They show a significant improvement with respect to the ones in Table 9 confirming our preliminary observations. It is worth to mention the results of Klawonn and Widlund [33] who used a similar method applied to elasticity problems over structured grids. They observed an increase in iterations too and concluded that there is a strong indication that this problem would be cured by the use of a better block preconditioner as algebraic multigrid instead of ILU. We have demonstrated in this work that this

Table 10
Summary of the speed-ups obtained for the different partitioning schemes investigated of the cavity flow test case when running on 16 processors.

Partition method	Refinement	Increase in equations (%)	Average adjacent subdomains	Iterations (p)/ iteration (1)	Speed-up/ iteration	Overall speed-up	CPU time (s)
Spectral-octasection	Kernighan-Lin + Terminal propagation	19	2.375	1.179	10.5	8.9	427
Coordinate recursive bisection	Regular interfaces	9	4.375	1.355	11.3	8.4	456
Spectral-octasection	Terminal propagation	20	2.500	1.404	10.9	7.8	484
Spectral-octasection	Kernighan-Lin	11	4.250	1.422	10.2	7.2	531
Multilevel-k-way	None	7	6.250	1.518	10.8	7.1	536
Spectral-octasection	None	11	4.125	2.314	11.3	5.1	754
Coordinate recursive bisection	Irregular interfaces	27	4.375	3.022	10.1	3.4	1143
Multilevel-octasection	Kernighan-Lin	10	7.875	No convergence	10.0	No convergence	No convergence

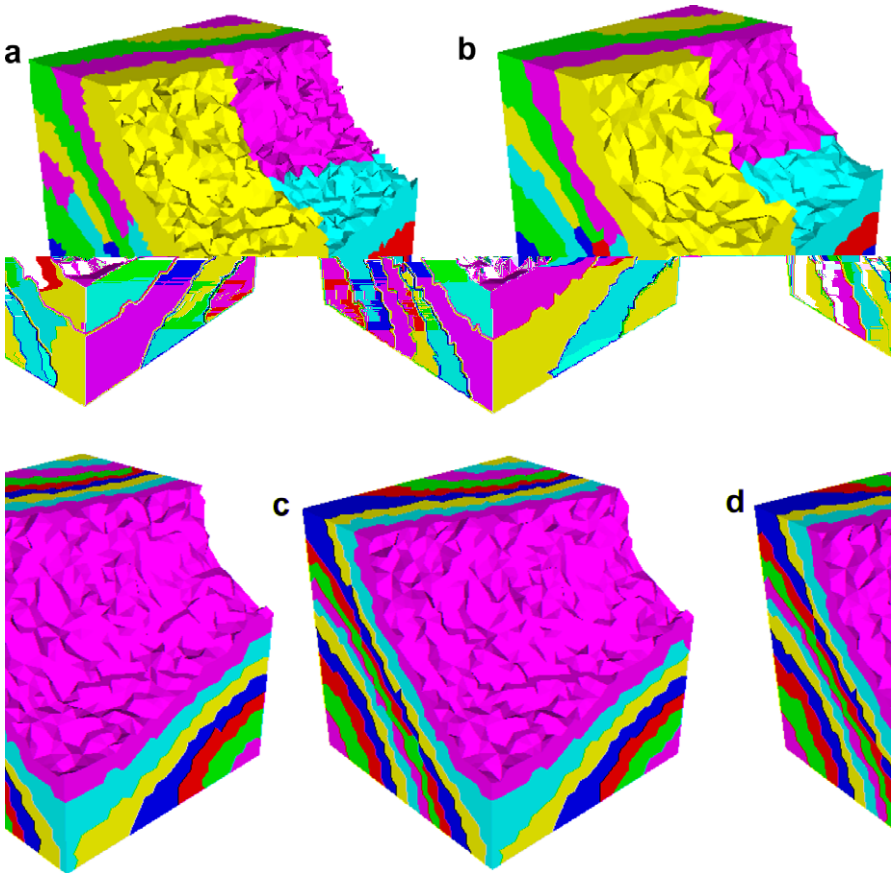


Fig. 19. Smoothing effect over the interfaces due Kernighan-Lin heuristics for 16 partitions in cavity flow test case: (a) spectral octasection without Kernighan-Lin; (b) spectral octasection with Kernighan-Lin; (c) spectral octasection and terminal propagation without Kernighan-Lin; and (d) spectral octasection and terminal propagation with Kernighan-Lin.

Table 11
Overall speed-up obtained for the 16 partition cases described in Table 5.

Mesh	Finite element	**v**–**p** equations ($\times 10^6$)	Increase in equations (%)	Average adjacent subdomains	iterations (p)/ iterations (1)	Overall speed-up	CPU time (s)
mesh1	$P1^+ - P0$	0.8	21	2.125	1.8	5.5	49
mesh2		1.5	17	1.875	1.5	7.3	108
mesh3		3.0	14	1.875	1.0	10.0	1020
mesh1	$P2^+ - P1$	1.8	19	2.125	1.5	7.5	373
mesh2		3.5	15	1.875	1.4	8.8	900
mesh3		6.7	12	1.875	1.1	11.0	2888
mesh4		13.9	10	1.875	1.4	8.9	6158

situation can also be controlled by the use of suitable partitioning strategies in the case of unstructured meshes where the smoothness of the interface and partition connectivity are key parameters to obtain good performance even at the cost of larger interfacial area.

5. Parallel performance for up to 128 processors

Let us demonstrate herein the potential of the parallel solver on a significantly larger partition and on a different platform. To assess the parallel performance with a larger number of processors, a flow simulation in a Kenics static mixer is solved on an unstructured mesh made of as much as 1.25 M tetrahedral finite elements. The details about the flow parameters and boundary conditions can be found in the work of Heniche and Tanguy [47]. For that purpose, up to 128 processors of a parallel cluster with a distributed memory interconnected by an infiniband 4X were employed. The velocity and pressure were

Table 12
Number of iterations, CPU time and relative speed up for the Kenics problem using P1⁺–P0 and P2⁺–P1 finite element approximations.

Processors	Iterations	CPU time (s)	Speed-up	Iterations	CPU time (s)	Speed-up
	P1⁺–P0			P2⁺–P1		
2	7034	50,432	1.00	–	–	–
4	7166	26,599	1.90	–	–	–
8	7248	13,711	3.68	–	–	–
16	7089	6851	7.36	3650	13,073	1.00
32	6801	3549	14.21	3643	6738	1.94
64	6999	2166	23.28	3747	3810	3.43
128	6520	1519	33.20	3587	2410	5.42

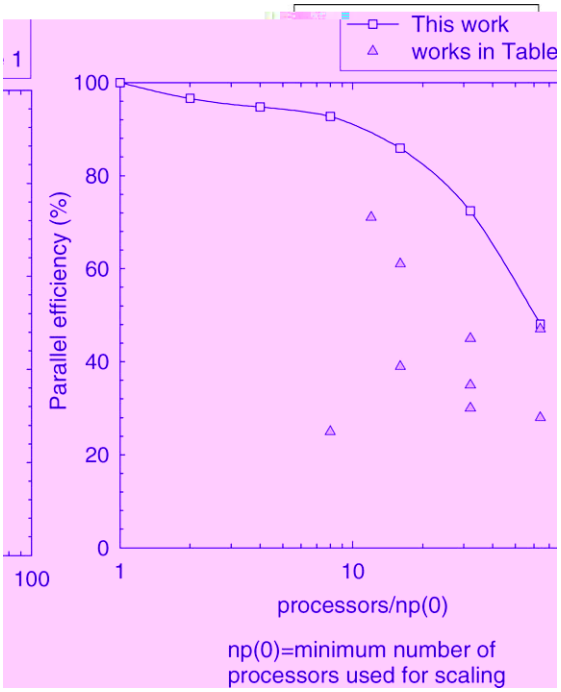


Fig. 20. Comparison of the parallel efficiency obtained in this work with respect to the ones reported in others works (Table 1).

approximated with both the P1⁺–P0 and P2⁺–P1 finite elements, yielding 9.3 M and 21.5 M of velocity–pressure equations, respectively. For this case the partitioning strategy developed in the last section was used. The CPU times and speed-ups are summarized in Table 12. The results demonstrate that the method can be used to speed-up the solution of large scale problems on large number of processors. It can be remarked that the parallel performance obtained with the distributed memory computer is better than the one obtained with the shared memory computer used in Section 4. The explanation lies in the higher traffic in the shared memory computer bus when several processors attempt to access the main memory. The algorithm shows a very good parallel efficiency with respect to the efficiencies reported in the literature as shown in Fig. 20. However, as the number of processors increases the parallel efficiency decays a typical behaviour of parallel ILU based iterative solvers.

6. Conclusions

The objective of this article was to present a method to parallelize a finite element solver based on domain decomposition for the simulation of viscous fluid flows. The novelty of the technique is the use of Lagrange multipliers constraints and domain decomposition to parallelize the ILU preconditioned CG solver. The advantage of this technique with respect to a standard ILU parallelization is a reduction on the communication between subdomains. The method was tested over two benchmark cases (pipe and cavity flow) employing discontinuous pressure finite element approximations (P1⁺–P0/P2⁺–P1) on unstructured tetrahedral grids. It was found that the proposed approach can solve problems around 5–13 times faster than the sequential ILU-iterative solver running on 16 processors. The speed-up is mainly affected by the load distribution balance and convergence rate which decays as the number of partitions increases. The ordering of physical variables and

partitioning strategy are critical to obtain good convergence rate. It was found that *UxUyUz–I–P–LM* ordering combined to the spectral octasection with terminal propagation and Kerninghan-Lin smoothing partitioning scheme produced the most promising speed-up performance. Eventually, this work has shown that the Lagrange multipliers technique can help in the parallelization of an ILU preconditioned Krylov type solver providing a way to reduce inter-processor communication and consequently improving the overall parallel efficiency demonstrated for up to 128 processors and on two different platforms namely distributed and shared memory computers.

Acknowledgments

The authors would like to acknowledge the financial contribution of the National Science and Engineering Research Council of Canada (NSERC) and the oil company TOTAL. We also thank the Réseau Québécois en Calcul de Haute Performance (RQCHP) for its assistance in granting access to high performance computing facilities.

References

[1] C. Farhat, L. Fezoui, S. Lanteri, Two-dimensional viscous flow computations on the connection machine: unstructured meshes, upwind schemes and massively parallel computations, Comput. Methods Appl. Mech. Eng. 102 (1993) 61–88.
[2] Z. Johan, K.K. Mathur, S.L. Johnsson, T.J.R. Hughes, A case study in parallel computation: viscous flow around an ONERA M6 wing, Int. J. Numer. Methods Fluids 21 (1995) 877–884.
[3] J.A. Scott, Parallel frontal solvers for large sparse linear systems, ACM Trans. Math. Softw. 29 (2003) 395–417.
[4] X.S. Li, J.W. Demmel, SuperLU_DIST: a scalable distributed-memory sparse direct solver for unsymmetric linear systems, ACM Trans. Math. Softw. 29 (2003) 110–140.
[5] N.I.M. Gould, Y. Hu, J.A. Scott, A Numerical Evaluation of Sparse Direct Solvers for the Solution of Large Sparse, Symmetric Linear Systems of Equations, CCLRC Rutherford Appleton Laboratory, Oxfordshire, 2005.
[6] R. Aggarwal, R. Keunings, F.X. Roux, Simulation of the flow of integral viscoelastic fluids on a distributed memory parallel computer, J. Rheol. 38 (1994) 405–419.
[7] P. Henriksen, R. Keunings, Parallel computation of the flow of integral viscoelastic fluids on a heterogeneous network of workstations, Int. J. Numer. Methods Fluids 18 (1994) 1167–1183.
[8] T.J.R. Hughes, I. Levit, J. Winget, An element-by-element solution algorithm for problem of structural and solid mechanics, Comput. Methods Appl. Mech. Eng. 36 (1983) 241–254.
[9] T.E. Tezduyar, J. Liou, Grouped element-by-element iteration schemes for incompressible flow computations, Comput. Phys. Commun. 53 (1989) 441–453.
[10] T.E. Tezduyar, M. Behr, S.K. Aliabadi, S. Mittal, S.E. Ray, New mixed preconditioning method for finite element computations, Comput. Methods Appl. Mech. Eng. 99 (1992) 27–42.
[11] I.S. Duff, H.A. Van Der Vorst, Developments and trends in the parallel solution of linear systems, Parallel Comput. 25 (1999) 1931–1970.
[12] D.A. Knoll, D.E. Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, J. Comput. Phys. 193 (2004) 357–397.
[13] L.C. Dutto, W.G. Habashi, Parallelization of the ILU(0) preconditioner for CFD problems on shared-memory computers, Int. J. Numer. Methods Fluids 30 (1999) 995–1008.
[14] S. Ma, Y. Saad, Distributed ILU(0) and SOR Preconditioners for Unstructured Sparse Linear Systems, Technical Report 94-027, Army High Performance Computing Research Center, University of Minnesota, Minnesota, 1994.
[15] A.E. Caola, R.A. Brown, Robust iterative methods for solution of transport problems with flow: a block two-level preconditioned Schwarz-domain decomposition method for solution of nonlinear viscous flow problems, Chem. Eng. Sci. 57 (2002) 4583–4594.
[16] A.E. Caola, Y.L. Joo, R.C. Armstrong, R.A. Brown, Highly parallel time integration of viscoelastic flows, J. Non-Newtonian Fluid Mech. 100 (2001) 191–216.
[17] X.-C. Cai, D.E. Keyes, L. Marcinkowski, Non-linear additive Schwarz preconditioners and application in computational fluid dynamics, Int. J. Numer. Methods Fluids 40 (2002) 1463–1470.
[18] F.-N. Hwang, X.-C. Cai, A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier–Stokes equations, J. Comput. Phys. 204 (2005) 666–691.
[19] S.O. Wille, O. Staff, A.F.D. Loula, Parallel ILU preconditioning, a priori pivoting and segregation of variables for iterative solution of the mixed finite element formulation of the Navier–Stokes equations, Int. J. Numer. Methods Fluids 41 (2003) 977–996.
[20] S.O. Wille, O. Staff, A.F.D. Loula, Block and full matrix ILU preconditioners for parallel finite element solvers, Comput. Methods Appl. Mech. Eng. 191 (2002) 1381–1394.
[21] O. Staff, S. Wille, Parallel ILU preconditioning and parallel mesh adaptation with load balancing for general domain decompositions for the Navier–Stokes equations, Int. J. Numer. Methods Fluids 47 (2005) 1301–1306.
[22] M. Sosonkina, Y. Saad, X. Cai, Using the parallel algebraic recursive multilevel solver in modern physical applications, Future Generat. Comput. Syst. 20 (2004) 489–500.
[23] Y. Saad, J. Zhang, Bilum: block versions of multi elimination and multilevel ILU preconditioner for general sparse linear systems, SIAM J. Sci. Comput. 20 (1999) 2103–2121.
[24] P. Henon, Y. Saad, A parallel multistage ILU factorization based on a hierarchical graph decomposition, SIAM J. Sci. Comput. 28 (2006) 2266–2293.
[25] Y. Achdou, Y.A. Kuztetsov, O. Pironneau, Substructuring preconditioners for the Q1 mortar element method, Numer. Math. 71 (1995) 419–449.
[26] C. Farhat, F.-X. Roux, Method of finite element tearing and interconnecting and its parallel solution algorithm, Int. J. Numer. Methods Eng. 32 (1991) 1205–1227.
[27] Q. Hu, Z. Shi, D. Yu, Efficient solvers for saddle-point problems arising from domain decompositions with Lagrange multipliers, SIAM J. Numer. Anal. 42 (2004) 905–933.
[28] D. Vanderstraeten, R. Keunings, A parallel solver based on the dual Schur decomposition of general finite element matrices, Int. J. Numer. Methods Fluids 28 (1998) 23–46.
[29] A. Zsaki, D. Rixen, M. Paraschivoiu, A substructure-based iterative inner solver coupled with Uzawa's algorithm for the Stokes problem, Int. J. Numer. Methods Fluids 43 (2003) 215–230.
[30] B. Vereecke, H. Bavestrello, D. Dureisseix, An extension of the FETI domain decomposition method for incompressible and nearly incompressible problems, Comput. Methods Appl. Mech. Eng. 192 (2003) 3409–3429.
[31] R. Glowinski, T.W. Pan, J. Periaux, Fictitious domain/domain decomposition methods for partial differential equations, in: Domain-based parallelism and problem decomposition method in computational science and engineering, Philadelphia, 1995, pp. 177–192.
[32] A. Klawonn, O. Rheinbach, Inexact FETI-DP methods, Int. J. Numer. Methods Eng. 69 (2007) 284–307.
[33] A. Klawonn, O.B. Widlund, A domain decomposition method with Lagrange multipliers and inexact solvers for linear elasticity, SIAM J. Sci. Comput. 22 (2000) 1199–1219.

[34] R. Glowinski, T.W. Pan, A.J. Kearsley, J. Periaux, Fictitious Domain Methods for Viscous Flow Simulation, Rice University, Houston, 1995.
[35] F.H. Bertrand, M.R. Gadbois, P.A. Tanguy, Tetrahedral elements for fluid flow, Int. J. Numer. Methods Eng. 33 (1992) 1251–1267.
[36] M. Crouzeix, P. Raviart, Conforming and non-conforming finite element methods for solving the stationary Stokes equations, RAIRO Anal. Numer. 7 (1973) 33–76.
[37] B. Coesnon, M. Heniche, C. Devals, F. Bertrand, P.A. Tanguy, A fast and robust fictitious domain method for modelling viscous flows in complex mixers: the example of propellant make-down, Int. J. Numer. Methods Fluids 58 (2008) 427–449.
[38] F. Bertrand, P.A. Tanguy, F. Thibault, A three-dimensional fictitious domain method for incompressible fluid flow problems, Int. J. Numer. Methods Fluids 25 (1997) 719–736.
[39] G. Karypis, V. Kumar, METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices, University of Minnesota, Minnesota, 1998.
[40] B. Hendrikson, R. Leland, The Chaco User Guide, version 1.0, Sandia Laboratories, Alburquerque, 1993.
[41] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Labs. Technol. J. 49 (1970) 291–308.
[42] A.E. Dunlop, B.W. Kernighan, A procedure for placement of standard-cell VLSI circuits, IEEE TCAD CAD-4 (1985) 92–98.
[43] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, 2003.
[44] M. Heniche, Y. Secretan, M. Leclerc, Efficient ILU preconditioning and inexact-Newton-GMRES to solve the 2D steady shallow equations, Commun. Numer. Methods Eng. 17 (2001) 69–75.
[45] M. Benzi, D.B. Szyld, A. Van Duin, Orderings for incomplete factorization preconditioning of nonsymmetric problems, SIAM J. Sci. Comput. 20 (1999) 1652–1670.
[46] L. Oliker, L. Xiaoye, P. Husbands, R. Biswas, Effects of ordering strategies and programming paradigms on sparse matrix computations, SIAM Rev. 44 (2002) 373–393.
[47] M. Heniche, P.A. Tanguy, A predictor–corrector shooting scheme for tracer trajectory calculations, in: The Proceedings of the Fourth International Conference on CFD in Oil and Gas, Metallurgical & Process Industries, Trondheim, Norway, 2005.
[48] P. Henon, P. Ramet, J. Roman, On finding approximate supernodes for an efficient block-ILU(k) factorization, Parallel Comput. 34 (2008) 345–362.
[49] T. Iwashita, Y. Nakanishi, M. Shimasaki, Comparison criteria for parallel orderings in ILU preconditioning, SIAM J. Sci. Comput. 26 (2005) 27.
[50] C. Shen, J. Zhang, K. Wang, Distributed block independent set algorithms and parallel multilevel ILU preconditioners, J. Parallel Distribut. Comput. 65 (2005) 331–346.