Numerical Performance of Incomplete Factorizations for 3D Transient Convection-Diffusion Problems ⋆

A. Rodríguez-Ferran * and M. L. Sandoval

Laboratori de Càlcul Numèric (LaCàN), Departament de Matemàtica Aplicada III, E.T.S. d'Enginyers de Camins, Universitat Politècnica de Catalunya

Abstract

Many environmental processes can be modelled as transient convection-diffusion-reaction problems. This is the case, for instance, of the operation of activated-carbon filters. For industrial applications there is a growing demand for 3D simulations, so efficient linear solvers are a major concern. We have compared the numerical performance of two families of incomplete Cholesky factorizations as preconditioners of conjugate gradient iterations: drop-tolerance and prescribed-memory strategies. Numerical examples show that the former are computationally more efficient, but the latter may be preferable due to their predictable memory requirements.

Key words: incomplete Cholesky factorizations; preconditioners; conjugate gradients; linear systems; environmental modelling; convection-diffusion; active carbon filters; air pollution

1 Introduction

Many industrial or natural processes of environmental interest are transient convection-diffusion-reaction problems. This is the case, for instance, of the operation of activated-carbon filters and the dispersion of pollutants in the atmosphere. In these technological applications, there is a growing demand for 3D simulations. Due to the transient, 3D nature of the problems, efficient finite element models are needed.

⋆ Grant sponsors: Ministerio de Educación y Ciencia (grant number: CGL2004-06171-C03-01/CLI) and CONACYT, México (grant number: 61078)

* Corresponding author. Departament de Matemàtica Aplicada III, E.T.S. d'Enginyers de Camins, Edifici C2, Campus Nord, Universitat Politècnica de Catalunya, E-08034 Barcelona, Spain.

Email address: antonio.rodriguez-ferran@upc.edu (A. Rodríguez-Ferran).

URL: http://www-lacan.upc.edu (A. Rodríguez-Ferran).

With computational efficiency in mind, we have chosen to stabilize the convective term with a least-squares technique. This results in a symmetric positive definite (SPD) system [1] to be solved at each time-step.

We solve these linear systems by means of the preconditioned conjugate gradient method. Incomplete Cholesky factorizations [2] are used as preconditioners. We have compared the numerical performance of two different strategies: threshold incomplete factorizations based on a drop tolerance [3] and incomplete factorizations with prescribed density [4]. To put the results in context, both a complete Cholesky factorization (i.e. direct solver) and a diagonal Jacobi preconditioner have also been used.

An outline of paper follows. A finite element model for transient convection-diffusion problems is briefly reviewed in Section 2. Incomplete Cholesky factorizations are discussed in Section 3. Nodal ordering and sparse storage schemes are covered in Section 4. The computational performance of the various numerical techniques is assessed in detail by means of numerical examples in Section 5. The concluding remarks of Section 6 close the paper.

2 Finite Element Convection-Diffusion Equations

We start by describing briefly the mathematical model. Section 2.1 discusses the governing equations for transient convection-diffusion-reaction and convection-diffusion problems. The finite element discretization is treated in Section 2.2. More details about both aspects can be found in [1].

2.1 Transient Convection-Diffusion-Reaction Problems

Transient convection-diffusion-reaction problems can be modelled as

$$\frac{\partial c}{\partial t} + \boldsymbol{v} \cdot \nabla c - \nabla \cdot (\nu \nabla c) + \sigma(c)c = f \qquad \text{in } \Omega \times (0,T] \tag{1}$$

$$c = c_{\text{input}} \qquad \text{on } \Gamma_D \times (0,T] \tag{2}$$

$$\nabla c \cdot \boldsymbol{n} = 0 \qquad \text{on } \partial\Omega \backslash \Gamma_D \times (0,T] \tag{3}$$

$$c(\boldsymbol{x},0) = c^0 \qquad \text{in } \Omega \tag{4}$$

In the partial differential equation (1), $c(\boldsymbol{x},t)$ denotes the concentration at point $\boldsymbol{x}$ and instant t, $\boldsymbol{v}(\boldsymbol{x})$ the convective velocity, $\nu > 0$ the diffusivity coefficient, $\sigma(c)$ the reaction coefficient, $f(\boldsymbol{x},t)$ the source term, ∇ the usual nabla operator and T the final time of analysis. This PDE is complemented with Dirichlet and Neumann boundary conditions and initial conditions, Equations (2), (3) and (4) respectively. In these equations, $c_{\text{input}}(\boldsymbol{x},t)$ is the prescribed concentration on the Dirichlet

boundary Γ_D, $\boldsymbol{n}$ is the outward unit normal vector and $c^0(\boldsymbol{x})$ is the prescribed initial concentration.

In the transient equation (1), the term $\partial c/\partial t$ models the time variation of the concentration; $\boldsymbol{v} \cdot \nabla c$ the convection due to the motion of the ambient fluid; $\nabla \cdot (\nu \nabla c)$ the diffusion, $\sigma(c)c$ the (nonlinear) reaction and f the external source. The constant, non-uniform velocity field $\boldsymbol{v}$ is computed a priori by means of an appropriate flow formulation (for instance, porous media flow combined with potential flow for active-carbon filters, see [5]).

2.2 Finite Element Formulation

2.2.1 Fractional-step methods

A very common strategy in this type of problems is to use a fractional-step method, thus treating the various terms separately, see [6,7]. In our applications, reaction is nonlinear, so we have chosen to write the governing PDE as

$$\frac{\partial c}{\partial t} + L_1 c + L_2 c = 0 \tag{5}$$

where $L_1 c := \boldsymbol{v} \cdot \nabla c - \nabla \cdot (\nu \nabla c)$ is the (linear) convection-diffusion differential operator and $L_2 c := \sigma(c)c - f$ is the (nonlinear) reaction operator. During numerical time-integration, each time-step Δt is subdivided into two phases: a convection-diffusion phase, dealing with L_1, and a reaction phase, dealing with L_2. This second phase concentrates all the problem linearity but, since L_2 is not a differential operator, it can be handled efficiently node by node. In the rest of this paper we will focus on the convection-diffusion phase which leads, upon finite element spatial discretization, to large linear systems of equations.

Note that this type of splitting is crucial for large 3D industrial applications: solving the original equation (1) with no splitting would mean (upon space and time discretization) solving many large systems of nonlinear algebraic equations.

2.2.2 The convection-diffusion phase

With the transient nature of the problem and industrial 3D applications in mind, we require the following features on the finite element formulation: (1) robust, standard finite element technology; (2) appropriate stabilization of the convective term; (3) accurate time-integration (of order 2 or higher); (4) efficient solution of resulting linear systems of equations.

Let us consider time-integration first. Using the popular θ scheme [1] for the convec-

tion-diffusion phase leads to

$$\frac{\Delta c}{\Delta t} - \theta \Delta c_t = c_t^n \tag{6}$$

where Δt is the time increment, $\Delta c := c^{n+1} - c^n$ is the concentration increment, c_t is a compact notation for $\partial c / \partial t$ and superscripts denote time stations. For $\theta = 1/2$, the well-known second-order Crank-Nicolson scheme

$$c^{n+1} + \frac{\Delta t}{2} L_1 c^{n+1} = c^n - \frac{\Delta t}{2} L_1 c^n \tag{7}$$

Higher-order schemes, like the ones proposed in [8,9], can also be applied.

We have chosen to stabilize the convective term by means of a standard least-squares (LS) formulation. As discussed in [8,9], this approach introduces more diffusivity that other stabilization techniques, such as SUPG, GLS or SGS.

In the LS formulation, we use the differential operator $I + \frac{\Delta t}{2} L_1$ (with I the identity) to write the integral equation

$$\left(v + \frac{\Delta t}{2} L_1 v, c^{n+1} + \frac{\Delta t}{2} L_1 c^{n+1} \right) = \left(v + \frac{\Delta t}{2} L_1 v, c^n - \frac{\Delta t}{2} L_1 c^n \right) \tag{8}$$

where v are test functions from a suitable space and $(\cdot, \cdot)$ is the interior product defined as $(u, v) := \int_\Omega uv \mathrm{d}\Omega$. The left-hand-side term in Equation (8) is symmetric. This contrasts with other stabilization techniques, and leads to symmetrical linear systems of equations, as shown next.

For spatial discretization, we will use standard linear elements (triangles in 2D and tetrahedra in 3D) and a "displacement" formulation. After applying the divergence theorem and the boundary conditions, and noting that $\nabla \cdot (\nu \nabla) \equiv 0$ for linear elements Ω_e, the weak form of the problem is written as

$$\begin{aligned} &\left(v, c^{n+1} \right) + \frac{\Delta t}{2} a \left(v, c^{n+1} \right) + \sum_e \left[\frac{\Delta t}{2} \left(\boldsymbol{v} \cdot \nabla v, c^{n+1} \right)_e + \frac{\Delta t^2}{4} \left(\boldsymbol{v} \cdot \nabla v, \boldsymbol{v} \cdot \nabla c^{n+1} \right)_e \right] \\ &\quad = (v, c^n) - \frac{\Delta t}{2} a \left(v, c^n \right) + \sum_e \left[\frac{\Delta t}{2} \left(\boldsymbol{v} \cdot \nabla v, c^n \right)_e - \frac{\Delta t^2}{4} \left(\boldsymbol{v} \cdot \nabla v, \boldsymbol{v} \cdot \nabla c^n \right)_e \right] \end{aligned} \tag{9}$$

where the bilinear form $a(\cdot, \cdot)$ is defined as

$$a(v, c) := (v, \boldsymbol{v} \cdot \nabla c) + (\nabla v, \nu \nabla c) \tag{10}$$

Finite element discretization of Equation (9) results finally in the linear system of

algebraic equations

$$\begin{aligned}\left[\mathbf{M}+\frac{\Delta t}{2}\left(\mathbf{G}+\mathbf{D}\right)+\frac{\Delta t}{2}\mathbf{G}^T+\frac{\Delta t^2}{4}\widehat{\mathbf{D}}\right]\mathbf{c}^{n+1}=\\ \left[\mathbf{M}-\frac{\Delta t}{2}\left(\mathbf{G}+\mathbf{D}\right)+\frac{\Delta t}{2}\mathbf{G}^T-\frac{\Delta t^2}{4}\widehat{\mathbf{D}}\right]\mathbf{c}^{n}\end{aligned} \tag{11}$$

where $\mathbf{c}$ is the nodal vector of concentrations, and matrices $\mathbf{M}$ (mass matrix), $\mathbf{G}$ (convection matrix), $\mathbf{D}$ (diffusivity matrix) and $\widehat{\mathbf{D}}$ are defined as

$$\begin{aligned} m_{ij}&=\int_\Omega N_i N_j \mathrm{d}\Omega \quad ; \quad g_{ij}=\int_\Omega N_i\left(\boldsymbol{v}\cdot\nabla N_j\right)\mathrm{d}\Omega\\ d_{ij}&=\int_\Omega \nu\nabla N_i\cdot\nabla N_j \mathrm{d}\Omega \quad ; \quad \widehat{d}_{ij}=\int_\Omega \left(\boldsymbol{v}\cdot\nabla N_i\right)\left(\boldsymbol{v}\cdot\nabla N_j\right)\mathrm{d}\Omega\end{aligned} \tag{12}$$

with N_i the shape function of node i. With the compact notation $\mathbf{A}:=\mathbf{M}+\frac{\Delta t}{2}\left(\mathbf{D}+\mathbf{G}+\mathbf{G}^T\right)+\frac{\Delta t^2}{4}\widehat{\mathbf{D}}$ and $\mathbf{B}:=\mathbf{M}+\frac{\Delta t}{2}\mathbf{G}^T$, the linear system (11) can be expressed as

$$\mathbf{A}\mathbf{c}^{n+1}=\left(2\mathbf{B}-\mathbf{A}\right)\mathbf{c}^n \tag{13}$$

Matrices $\mathbf{M}$, $\mathbf{D}$ and $\widehat{\mathbf{D}}$ are symmetric, while $\mathbf{G}$ is non-symmetric. Matrices $\mathbf{G}^T$ and $\widehat{\mathbf{D}}$ are associated to the least-squares stabilization; the former ensures the symmetry of matrix $\mathbf{A}$. Note also that the mass matrix $\mathbf{M}$ is positive definite, $\mathbf{D}$ and $\widehat{\mathbf{D}}$ are positive semi-definite and $\mathbf{G}+\mathbf{G}^T$ is indefinite. For small Δt, matrix $\mathbf{A}$ preserves the positive definiteness of matrix $\mathbf{M}$.

A simple way to estimate the critical time-step below which $\mathbf{A}$ is SPD is to check the positive definiteness of $\mathbf{M}+\frac{\Delta t}{2}\left(\mathbf{D}+\mathbf{G}+\mathbf{G}^T\right)$, because the positive semi-definite term $\frac{\Delta t^2}{4}\widehat{\mathbf{D}}$ cannot have a negative effect. To do so, one can state the generalized eigenvalue problem

$$\mathbf{M}\mathbf{x}=-\frac{\Delta t}{2}\left(\mathbf{D}+\mathbf{G}+\mathbf{G}^T\right)\mathbf{x} \tag{14}$$

and look for the smallest positive eigenvalue Δt. However, this computation is deemed not necessary in practical applications, because time-steps selected according to accuracy and stability constraints are typically below this critical value.

To sum up: thanks to the use of a LS stabilization of the convective term, *the linear system to be solved at each time-step is symmetric positive definite (SPD)*. This is very attractive from a computational viewpoint, especially for 3D industrial applications.

3 Incomplete Cholesky Factorizations

From now we will write our linear system as $\mathbf{Ax} = \mathbf{b}$, the usual notation in numerical linear algebra. Since matrix $\mathbf{A}$ is SPD, the two basic choices are a Cholesky direct solver or a conjugate gradient (CG) iterative solver [10].

The main advantage of the Cholesky method is that the factorization of matrix $\mathbf{A}$ can be amortized over many time-steps. However, for fine finite element meshes in 3D, a significant amount of fill-in occurs, resulting in very large memory and CPU time requirements. To avoid the computation of square roots, we will work with the generalized version ($\mathbf{A} = \mathbf{LDL}^T$), not the standard version ($\mathbf{A} = \mathbf{LL}^T$) of the Cholesky method (mind the notation abuse: $\mathbf{D}$ refers here to the diagonal factor, not the diffusivity matrix defined in Equation (12)).

With the CG method, on the other hand, fill-in is completely precluded. However, two drawbacks can be pointed out: computational work has to be re-started at each time-step, and poor conditioning of matrix $\mathbf{A}$ may lead to slow convergence of the iterative process.

This latter problem can be solved by *preconditioning* the linear system. If left-preconditioning is used, the original linear system is transformed into

$$\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b} \tag{15}$$

where $\mathbf{P}$ is the SPD preconditioner. The non-symmetry of matrix $\mathbf{P}^{-1}\mathbf{A}$ poses no difficulties: the original CG method can be rewritten into a preconditioned conjugate gradient scheme [2] where the two key ideas are that (1) an auxiliary linear system with matrix $\mathbf{P}$ needs to be solved at each iteration, and (2) for appropriate $\mathbf{P}$, the conditioning and hence the convergence of CG greatly improves.

Incomplete Cholesky factorizations (ICF) are a popular choice for $\mathbf{P}$ [2,11,12]. The large fill-in of the complete (i.e. standard) Cholesky factorization is completely or partially avoided by discarding coefficients along the factorization process. The ICF with no fill-in prescribes the incomplete factor $\mathbf{L}$ to have the same sparsity pattern as the lower triangle of matrix $\mathbf{A}$. A better (but also denser) preconditioner is obtained by allowing some degree of fill-in according to various strategies. In either case, it is worth noting that the incomplete factorization ($\mathbf{P} = \mathbf{LL}^T$ or $\mathbf{P} = \mathbf{LDL}^T$) can be amortized over many time-steps.

More recently, sparse approximate inverses have become an interesting approach to precondition conjugate gradient iterations [13,14]. Contrary to incomplete factorizations, the computation of approximate inverses can be parallelized in a straightforward way. In a sequential environment like ours, however, incomplete factorizations can still be considered as a more efficient preconditioning strategy.

3.1 Drop-Tolerance Strategies

In drop-tolerance strategies, off-diagonal coefficients are dropped during the factorization process if they are below a certain threshold. Various elimination criteria may be used. Munksgaard [3], for instance, drops element $a_{ij}^{(k+1)}$ during stage k if

$$|a_{ij}^{(k+1)}| \leq \tau \sqrt{|a_{ii}^{(k)} a_{jj}^{(k)}|} \tag{16}$$

where τ is the dropping tolerance or threshold. Parameter τ controls the fill-in of the factorization: the smaller the τ, the denser the ICF.

Let us analyze the two extreme cases. For $\tau \rightarrow 0$, no entries are dropped and a complete Cholesky factorization is obtained. For $\tau \rightarrow \infty$, on the other hand, all non-diagonal entries are discarded, and the diagonal (Jacobi) preconditioner results. Note also that we cannot get the incomplete factorization with no fill-in (same sparsity pattern) as a particular case, because drop-tolerance approaches drop elements according to their size, not to their location in the matrix.

Since the factorization is incomplete, its existence is not guaranteed by $\mathbf{A}$ being SPD. The Munksgaard algorithm [3] contains two additional features to avoid breakdowns or instabilities caused by negative, null or very small pivots [15].

Pivots can be shifted in a dynamic local manner, according the rule:

$$\text{If} \quad d_{kk} \leq u \left(\sum_{j \neq k} |a_{kj}| \right) \quad \text{then} \quad \begin{cases} d_{kk} = \sum_{j \neq k} |a_{kj}| & \text{if} \quad \sum_{j \neq k} |a_{kj}| \neq 0 \\ d_{kk} = 1 & \text{if} \quad \sum_{j \neq k} |a_{kj}| = 0 \end{cases} \tag{17}$$

Note that setting $u = 1$ ensures the diagonal dominance of the preconditioner. However, this rule would cause significant changes in many pivots, and a poor preconditioner would result. In practice, a far less demanding rule is preferred. In our numerical examples, we have used rule (17) with the (typical) value $u = 0.01$.

Even with $u = 0.01$, many pivot changes may be necessary. If this affects the quality of the preconditioner, it may be better to perform a global diagonal shift before starting the factorization. Matrix $\mathbf{A}$ is perturbed into $\widehat{\mathbf{A}} = \mathbf{A} + \alpha \mathrm{diag}\mathbf{A}$. Choosing α large enough to ensure the diagonal dominance of $\widehat{\mathbf{A}}$ (and, hence, the existence of the incomplete factorization) again leads to a poor preconditioner, so much lower values of α are preferred.

In practical implementations [16], the two shifting strategies (local and global) are combined. Parameter α is chosen in an iterative, trial-and-error basis (starting with $\alpha = 0$) and, if still needed, pivots are shifted during the factorization. In our numerical experiments, $\alpha = 0$ provided satisfactory results and only local shifts were required.

3.2 Prescribed-Memory Strategies

The major drawback of drop-tolerance strategies is that the memory needed to store the incomplete factor $\mathbf{L}$ cannot be predicted in advance. In general, one cannot accurately predict the amount of fill-in for a given threshold parameter τ.

Various prescribed-memory strategies have been proposed to overcome this difficulty. The algorithm of Lin and Moré [4], for instance, retains the `col_len` $+ p$ largest elements in each factorization step, where `col_len` is the original number of non-zeros in the column of matrix $\mathbf{A}$ and p is a parameter that controls fill-in (pN extra entries are allowed, with N the system dimension). Diagonal elements are always retained.

Like in drop-tolerance strategies, the incomplete factorization without fill-in cannot be obtained as a particular case. For $p = 0$, the `col_len` largest non-zero elements in column of $\mathbf{L}$ will not, in general, coincide in location with the `col_len` non-zero elements in column of $\mathbf{A}$. This means that, for $p = 0$, the sparsity pattern of $\mathbf{A}$ is not maintained.

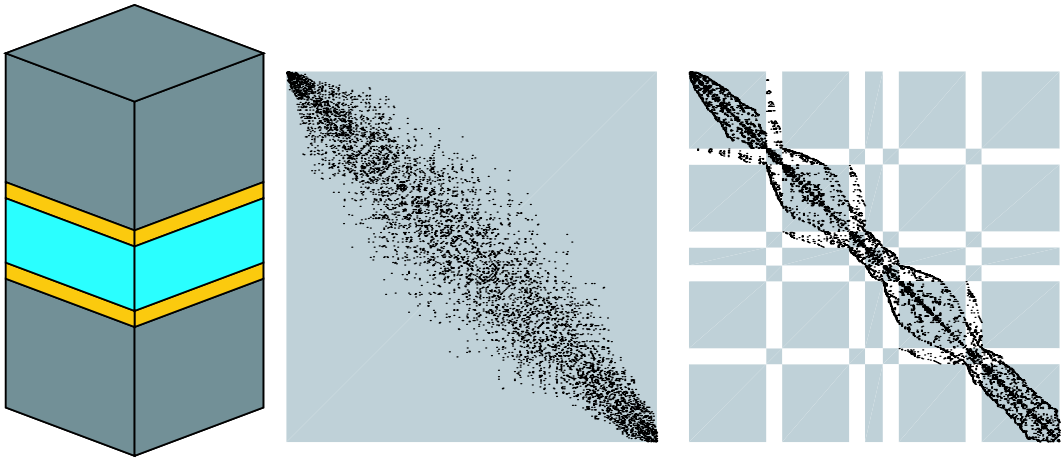
Two additional features of the Lin-Moré algorithm that ensure the existence of the factorization are the scaling of the matrix and the perturbation $\mathbf{A} + \alpha\mathbf{I}$, see [4].

4 Nodal Ordering and Storage Schemes

We have used a reverse Cuthill-McKee [2] nodal reordering to reduce the fill-in in the direct solver. With domain decomposition techniques (not discussed in this work) in mind, we have constrained the reordering to respect the typical block-interface structure of many active carbon filters (our industrial application of interest). Consider for instance the domain of Figure 1(a), consisting of five blocks and four interfaces. The sparsity patterns before and after reordering are shown in Figures 1(b) and 1(c) respectively. Note that the reordered matrix is block-tridiagonal, clearly showing that each block/interface interacts with two interfaces/blocks.

Nodal reordering also affects preconditioned iterations. As discussed in [17], reverse Cuthill-McKee is a good reordering strategy for conjugate gradient iterations preconditioned by means of incomplete factorizations, especially in the case of non-structured finite element meshes.

Many storage schemes for sparse matrices are available, see [2,12]. The Cholesky direct solver is usually combined with a skyline storage (SKS). Back in 1980, Munksgaard used a symmetric coordinate storage (SCS) [3]. Much more recently, Lin and Moré preferred the more efficient modified column storage (MCS).



(a) (b) (c)

Fig. 1. Nodal reordering: (**a**) typical block-interface domain, (**b**) matrix before reordering and (**c**) matrix after reordering

5 Numerical Examples

The main goal of this section is to compare the performance of the two incomplete Cholesky factorizations (drop-tolerance and prescribed-memory) as preconditioners of conjugate gradient iterations. For the sake of completeness and as a reference, we will also include in the analysis the very simple diagonal (Jacobi) preconditioner and the standard (non-preconditioned) CG scheme. More importantly, we will also test the Cholesky direct solver: from the point of view of industrial applications, iterative solvers (preconditioned or not) are to be preferred to direct solvers only if they are indeed more efficient. Comparisons of computational cost will comprise both memory requirements (measured in non-zero entries in incomplete factor $\mathbf{L}$) and CPU time.

For iterative solvers, the solution at the end of the previous time-step is taken as the initial approximation. The stopping criteria are

$$\left|x_i^{(k)} - x_i^{(k+1)}\right| \leq tol_x\Big(\left|x_i^{(k+1)}\right| + 1\Big) \qquad \text{for each component } i, \tag{18}$$

$$\left\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)}\right\| \leq tol_r \left\|\mathbf{b}\right\| \tag{19}$$

with $tol_x = 0.5 \cdot 10^{-10}$ and $tol_r = 0.5 \cdot 10^{-9}$.

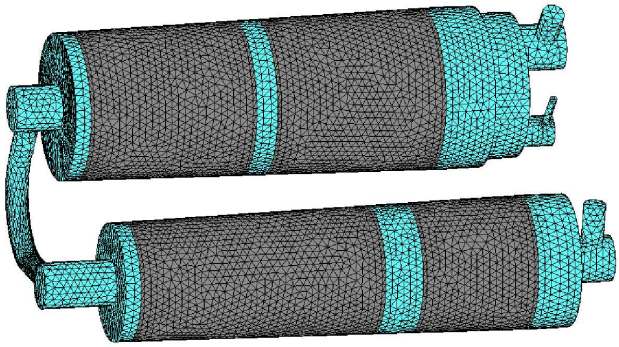
All the algorithms are coded in Fortran 90 with double precision arithmetics and with a compiler optimization option that maximizes performance for the target platform processor. Tests were run on one processor of a SGI Origin 3000 with eight 600 MHz IP35 processors and 8 Gb RAM.

5.1 Applications: active carbon filters

We will work with the active carbon filters of Figure 2. For each of these filters, we have used two finite element meshes, denoted as “coarse” and “fine”. For filter A, we also work with a third, “coarser” mesh.

Filter A (fine mesh)

Filter B (fine mesh)



Filter C (coarse mesh)

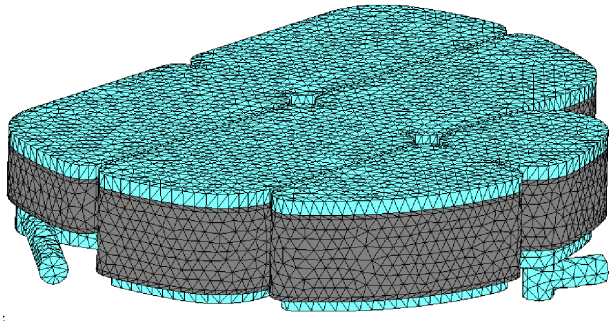


Fig. 2. Active carbon filters

Table 1 summarizes the main numerical parameters. For each filter and mesh, it shows (1) the number of nodes, (2) the number of finite elements, (3) the size N of matrix $\mathbf{A}$ (slightly smaller than the number of nodes because it does not include Dirichlet nodes), (4) the number of non-zero entries $\mathrm{nnz}(\mathbf{A})$ in the triangle of matrix $\mathbf{A}$, (5) the time-step Δt and (6) the number of time-steps.

	Filter A			**Filter B**		**Filter C**	
	Coarser	Coarse	Fine	Coarse	Fine	Coarse	Fine
# nodes	5 186	17 943	87 502	48 138	84 087	33 041	65 941
# elem.	19 081	74 139	460 765	249 645	454 815	167 558	346 907
N	5 171	17 914	87 473	48 125	84 072	33 028	65 925
$\mathrm{nnz}(\mathbf{A})$	32 928	117 181	653 435	357 251	636 669	242 793	493 278
Δt	5	8	4.31	4	3.6	8.59	6.82
# steps	23 490	17 961	32 867	20 719	23 152	11 135	13 974

Table 1
Main numerical parameters in filter simulations

The time-step is selected in accordance with element size by prescribing a Courant number of 1 [8,9] and, hence, is different from mesh to mesh. To verify the positive definiteness of the resulting matrices $\mathbf{A}$, we have computed the minimum eigenvalue, see Table 2, which turns out to be positive in all the examples. In fact, our numerical experience indicates that the Courant condition is more restrictive in the selection of Δt than the SPD requirement. Table 2 also shows the maximum eigenvalue and the condition number of the unpreconditioned matrix.

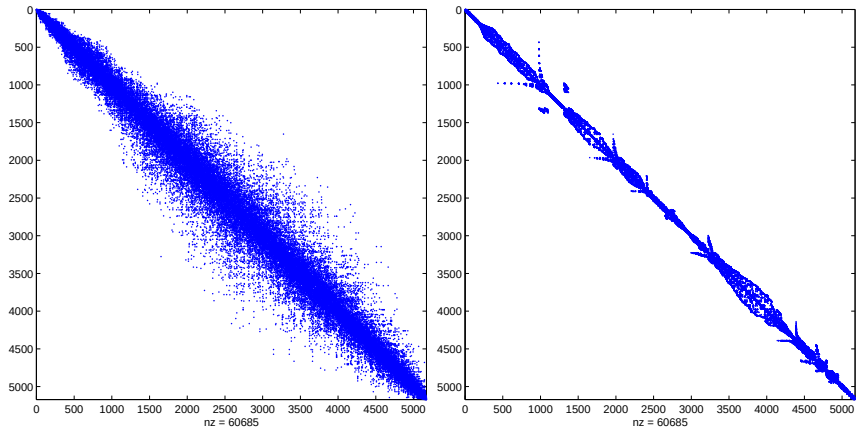
	Filter A		**Filter B**		**Filter C**	
	Coarse	Fine	Coarse	Fine	Coarse	Fine
$\lambda_{\max}$	$2.68 \cdot 10^{-4}$	$5.11 \cdot 10^{-5}$	$1.02 \cdot 10^{-4}$	$4.39 \cdot 10^{-5}$	$2.42 \cdot 10^{-4}$	$1.15 \cdot 10^{-4}$
$\lambda_{\min}$	$8.87 \cdot 10^{-10}$	$5.65 \cdot 10^{-10}$	$4.11 \cdot 10^{-9}$	$4.08 \cdot 10^{-9}$	$1.30 \cdot 10^{-9}$	$1.35 \cdot 10^{-9}$
$\kappa(\mathbf{A})$	$3.02 \cdot 10^{5}$	$9.04 \cdot 10^{4}$	$2.48 \cdot 10^{4}$	$1.08 \cdot 10^{4}$	$1.86 \cdot 10^{5}$	$8.52 \cdot 10^{4}$

Table 2
Positive definiteness of matrices: maximum eigenvalue $\lambda_{\max}$, minimum eigenvalue $\lambda_{\min}$ and condition number $\kappa(\mathbf{A}) = \lambda_{\max}/\lambda_{\min}$

5.2 Effect of Nodal Ordering

The effect of nodal ordering is assessed with the numerical simulation of filter A. Figure 3 shows the sparsity pattern of matrix $\mathbf{A}$ for the coarser mesh, see Table 1, before and after reordering. The numerical performance of the Cholesky direct solver and the CG iterative solver with various ICF preconditioners (drop-tolerance

with $\tau = 10^{-2}$ and $\tau = 10^{-4}$; prescribed-memory with $p = 2$ and 7) is summarized in Table 3.



(a) (b)

Fig. 3. Sparsity pattern of matrix with coarser mesh of filter A: (**a**) before reordering and (**b**) after reordering

For each solver, Table 3 shows the CPU time needed to (1) obtain the (complete or incomplete) factorization and (2) solve the linear systems in all the time-steps. Note that the reordering leads to a very significant reduction (over 75%) in the CPU time for the direct solver, while it has little effect on the performance of the ICF preconditioners. The reordering is applied in all subsequent computations.

5.3 Effect of Storage Scheme

We analyze next the effect of the storage scheme. The drop-tolerance algorithm and filter B are considered. In Figure 4 we compare the computational cost of the SCS scheme originally used by Munksgaard [3] and the more modern MCS scheme. In this and subsequent figures, computational cost is represented by CPU time (in seconds) in the x-axis and number of non-zero entries in factor $\mathbf{L}$ in the y-axis.

Figure 4 clearly shows that there is a significant reduction in the CPU time if the MCS scheme is used, because it is more suited for the required substitutions at each time-step. For this reason, in all subsequent analyses we will use the MCS format for both the drop-tolerance and prescribed-memory ICF preconditioners.

5.4 Computational Performance of Direct and Iterative Solvers

After the two preliminary analyses of Sections 5.2 and 5.3, we will assess now the relative numerical performance of the various direct and iterative solvers considered in this work.

	Cholesky		
	Before (s)	After (s)	Reduction (%)
Factorization	2.3	0.6	73.9
Solution	896.6	213.8	76.2
TOTAL	898.9	214.4	76.2

	Drop-tolerance ICF preconditioner					
	$\tau = 10^{-2}$			$\tau = 10^{-4}$		
	Before (s)	After (s)	Red. (%)	Before (s)	After (s)	Red. (%)
Preconditioner	0.4	0.4	0.0	1.5	1.6	-6.7
Solution	723.5	726.1	-3.6	568.2	608.6	-7.1
TOTAL	723.9	726.5	-3.6	569.7	610.2	-7.1

	Prescribed-memory ICF preconditioner					
	$p = 2$			$p = 7$		
	Before (s)	After (s)	Red. (%)	Before (s)	After (s)	Red. (%)
Preconditioner	0.0	0.0	0.0	0.0	0.0	0.0
Solution	919.9	920.1	0.0	776.1	796.8	-2.7
TOTAL	919.9	920.1	0.0	776.1	796.8	-2.7

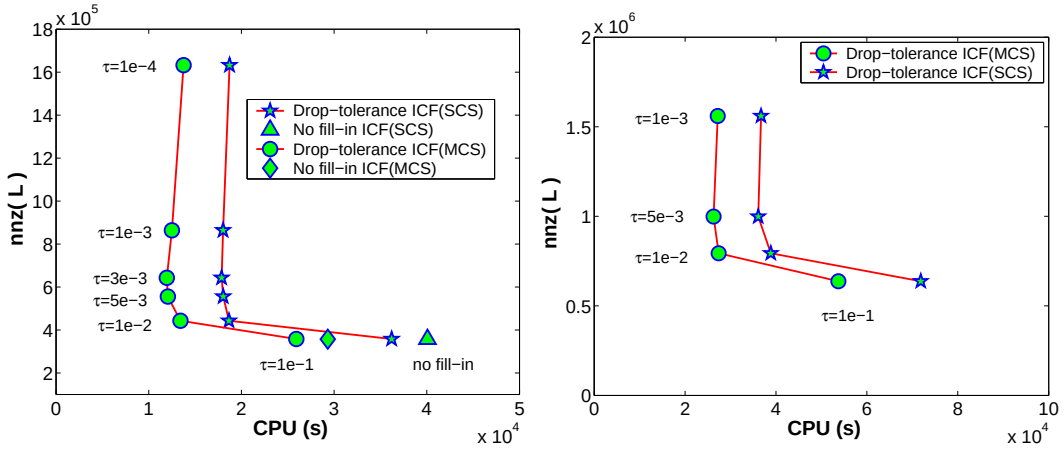
Table 3
Effect of reordering on simulation of filter A with coarser mesh. CPU time (in seconds) before reordering, after reordering and percentage reduction

5.4.1 Standard vs. Preconditioned Conjugate Gradients

For a first, broad view, we analyze filter B with all the solvers discussed: Cholesky direct solver, standard CG and preconditioned CG with three different strategies: diagonal, drop-tolerance ICF and prescribed-memory ICF.

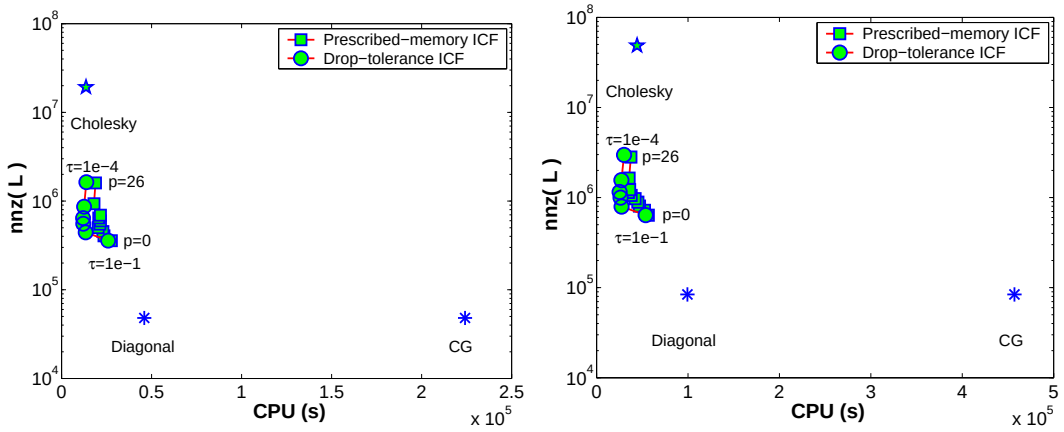
The results are shown in Figure 5 and clearly illustrate the need to precondition CG iterations. Indeed, even the very simple diagonal preconditioner leads to a very significant reduction (by a factor of more than 4) in the CPU time with respect to the standard CG with no additional storage requirements. For this reason, standard CG will not be used in the remaining analyses.

Figure 5 also indicates that ICFs represent a substantial saving in memory requirements with respect to a complete Cholesky factorization. This aspect is addressed in more detail next.



(a) (b)

Fig. 4. Effect of storage scheme on the simulation of filter B: (**a**) coarse mesh and (**b**) fine mesh



(a) (b)

Fig. 5. Computational cost of all solvers for the simulation of filter B: (**a**) coarse mesh and (**b**) fine mesh

5.4.2 Direct vs. Iterative Solvers

We compare now the Cholesky direct solver and the CG solver preconditioned with various strategies: diagonal preconditioner, ICF with no fill-in, drop-tolerance ICF and prescribed-memory ICF. Figure 6 shows the results for the three filters. The following aspects should be noted:

- If memory constraints allow its use, the direct Cholesky solver is faster that the CG method with a simple preconditioner (i.e. diagonal or ICF without fill-in). A better preconditioner is required to beat the direct solver.
- Allowing some fill-in in the ICF considerably reduces the CPU time of the ICF without fill-in in exchange for a modest increase in non-zero entries.
- *The best ICFs outperform the direct solver both in terms of memory requirements and CPU time.*

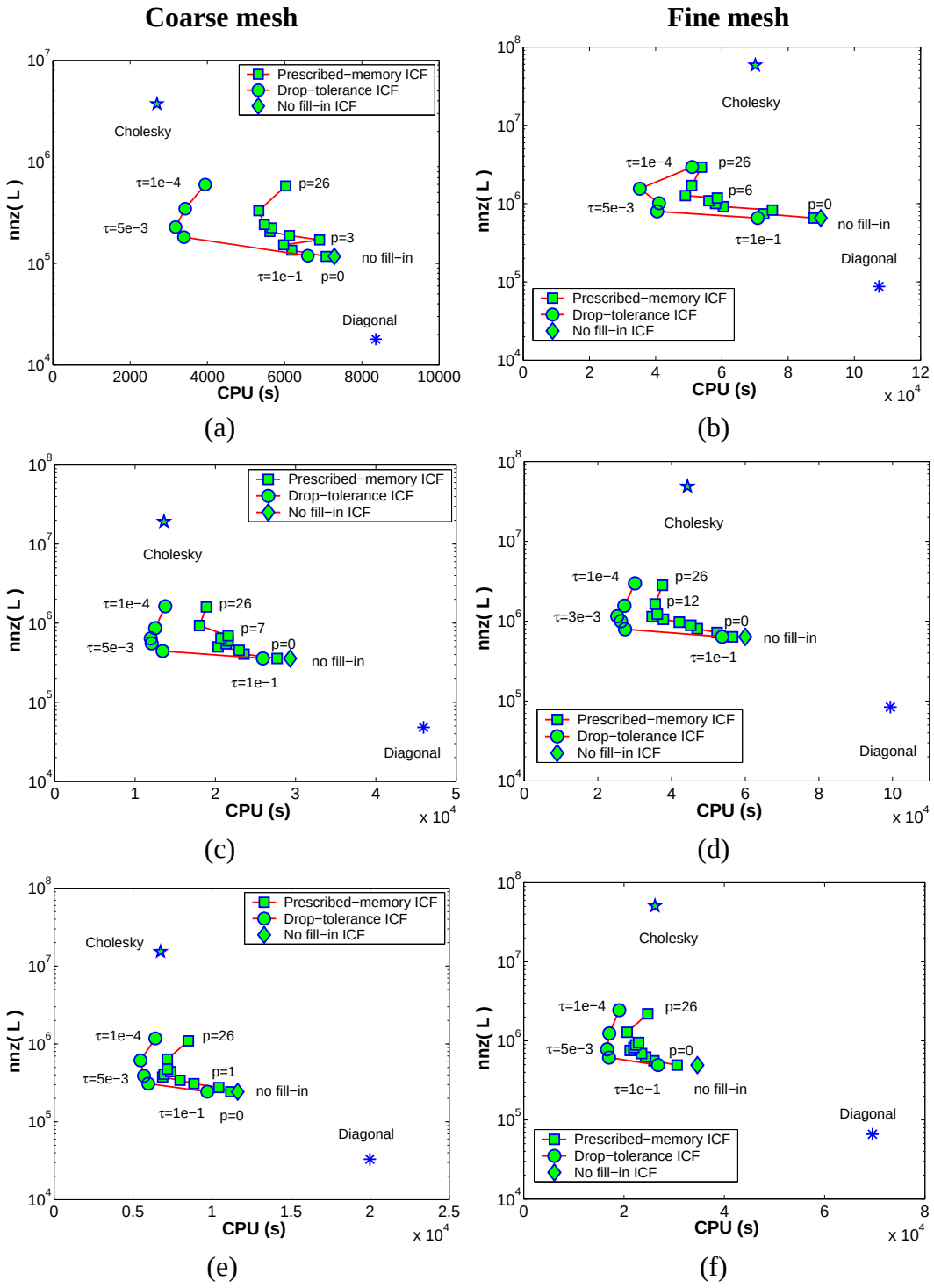


Fig. 6. Computational cost of Cholesky and preconditioned CG solvers for the simulation of the three filters: (**a**) filter A, coarse mesh, (**b**) filter A, fine mesh, (**c**) filter B, coarse mesh, (**d**) filter B, fine mesh, (**e**) filter C, coarse mesh, and (**f**) filter C, fine mesh

5.4.3 Drop-Tolerance vs. Limited-Memory Strategies

To conclude our numerical study of the active carbon filters, we compare here the two strategies for ICF. According to Figure 7, drop-tolerance strategies are more ef-

ficient than prescribed-memory strategies, in the sense that less CPU time is needed for a given memory storage. This result contrasts the one reported by Lin and Moré [4]. In must be noted, however, that the memory requirements of a drop-tolerance strategy cannot be predicted in advance, so one may prefer a safer (although less efficient) prescribed-memory ICF if memory is a critical constraint.

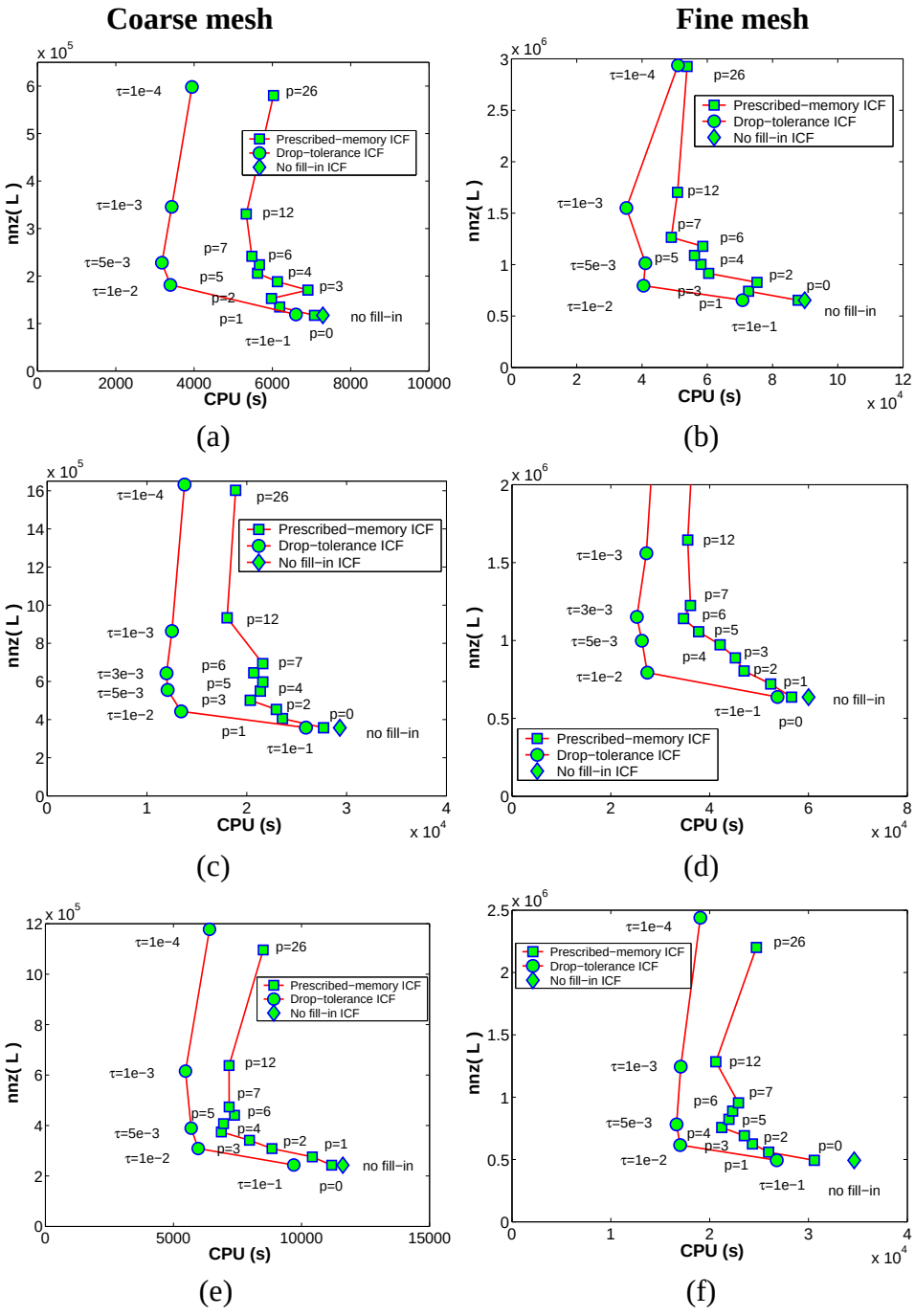


Fig. 7. Computational cost of drop-tolerance and prescribed-memory ICF preconditioners for the three filters: (**a**) filter A, coarse mesh, (**b**) filter A, fine mesh, (**c**) filter B, coarse mesh, (**d**) filter B, fine mesh, (**e**) filter C, coarse mesh, and (**f**) filter C, fine mesh

A more quantitative look is offered by Table 4. For filter A and a selection of the ICF depicted in Figure 7, it shows (1) the accumulated number of iterations, (2) the number of time-steps, (3) the average number of iterations per step, (4) the number of non-zero entries in factor $\mathbf{L}$, and the CPU time required (5) to compute $\mathbf{L}$, (6) to solve the linear systems in all time-steps, and (7) in total.

	No fill-in ICF		**Drop-tolerance ICF**			
			$\tau = 5 \times 10^{-3}$		$\tau = 10^{-4}$	
	Coarse	Fine	Coarse	Fine	Coarse	Fine
# iterations	1 079 406	1 060 343	238 244	327 287	103 675	176 583
# steps	17 961	32 867	17 961	32 867	17 961	32 867
iter/step	60.10	32.26	13.26	9.96	5.77	5.37
$\mathrm{nnz}(\mathbf{L})$	117 181	653 435	228 041	1 013 875	598 245	2 937 167
Precond. (s)	0.2	1.8	0.5	4.0	2.4	18.0
Solution (s)	6 918.7	86 523.9	2 871.1	37 679.6	3 639.0	47 558.9
TOTAL (s)	6 918.9	86 525.7	2 871.6	37 683.6	3 641.4	47 576.9

	Prescribed-memory ICF					
	$p = 0$		$p = 3$		$p = 7$	
	Coarse	Fine	Coarse	Fine	Coarse	Fine
# iterations	1 024 771	1 025 930	662 734	605 856	404 001	377 358
# steps	17 961	32 867	17 961	32 867	17 961	32 867
iter/step	57.05	31.21	36.90	18.43	22.49	11.48
$\mathrm{nnz}(\mathbf{L})$	117 181	653 435	170 364	915 092	241 621	1 264 893
Precond. (s)	0.1	0.4	0.1	0.6	0.1	0.7
Solution (s)	6 779.3	84 561.5	6 587.6	57 167.5	5 173.8	45 746.4
TOTAL (s)	6 779.4	84 561.9	6 587.7	57 168.1	5 173.9	45 747.1

Table 4
Computational cost of ICF preconditioners for filter A

The following aspects should be remarked in Table 4:

- For a given preconditioning strategy the number of iterations is controlled by the quality (i.e. density) of the preconditioner: more non-zero entries in $\mathbf{L}$ means less iterations.
- The CPU time, however, may decrease or increase. With a denser preconditioner less iterations are required, but each of these iterations is more costly (cf. the two

drop-tolerance ICF).

- For transient convection-diffusion problems, the time needed to compute the incomplete factor $\mathbf{L}$ is *negligible* in comparison with the time needed to solve the linear systems in all the time-steps. How much time it takes to *compute* the preconditioner is an irrelevant factor, because it is amortized over many time-steps. The two key issues are (1) the number of iterations and (2) the time needed to *apply* the preconditioner at each iteration.
- This is even the case for the direct Cholesky solver. For the range of problem sizes considered, the critical factor is not the time needed for a complete factorization of matrix $\mathbf{A}$, but the time needed for substitutions with a very dense $\mathbf{L}$ at each iteration.

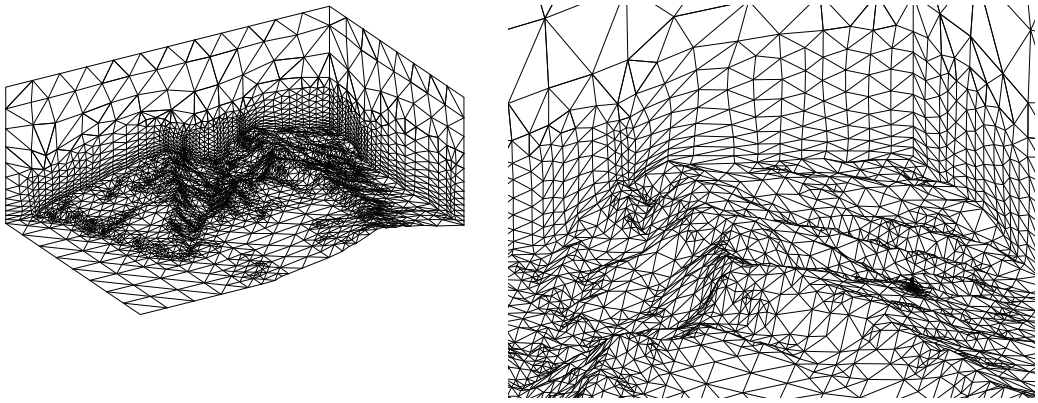
5.5 Application: air pollution

In all the active carbon filter simulations discussed above, the complete Cholesky factorization is feasible. This has enabled a detailed comparison, which clearly shows that a good ICF-preconditioned conjugate gradient method beats the direct solver both in terms of CPU time and memory requirement. However, it can be argued that these examples do not illustrate the real need for iterative methods, since the complete factorization is (more expensive but) possible anyway.

With this idea in mind, we consider now a larger example, dealing with air pollution modelling. A four-species linear dispersion model is used to describe the oxidation and hydrolysis of sulphur and nitrogen oxides [18]. This process is modelled by a vectorial version of Equation (1), where the unknowns are the four concentration fields, $\boldsymbol{v}$ is the wind velocity, and the reaction term represents the chemical reactions and the wet deposition. Splitting is not applied in this case. Boundary conditions are also adapted, mainly to take into account the dry deposition effects in the terrain and the emission source (stack).

The problem is defined in a rectangular region, $15\,600$ m $\times 22\,803$ m, in the south of La Palma island (Canary Islands). The upper boundary of the 3-D domain is a horizontal plane placed at a height of $9\,000$ m. The adaptive discretization of this domain has been developed by Montenegro et al. [19], see Figure 8. The untangling and smoothing procedure introduced by Escobar et al. [20] has been applied in the generation of this mesh, which contains $15\,3085$ tetrahedra and $28\,387$ nodes. The wind velocity field has been simulated by Montero et al. [21]. Both the mesh and the wind field have been used in this paper with the authors' kind permission. The simulation takes $4\,000$ steps of $\Delta t = 0.13$ (selected with the Courant criterion).

The problem size is $N = 107\,548$ (four unknowns per non-Dirichlet node) and the system matrix has $\mathrm{nnz}(\mathbf{A}) = 3\,094\,002$ non-zero entries. A memory check indicates that 4.25 Gb (i.e. more than half of the total RAM of the multi-user server)



(a) (b)

Fig. 8. Finite element mesh for air pollution problem: (**a**) surface mesh and (**b**) detail showing the complex orography and the stack

would be required for a complete factorization. This renders the direct method unfeasible in practice, and an iterative method is needed.

Since memory is the limiting factor, a prescribed-memory ICF is chosen as preconditioner. With $p = 0$, 510 Mb are needed; each unit in p results in 1.64 additional Mb. This means that we can go to up to a (very large) value of $p = 100$ with 674 Mb in total.

Figure 9 shows the performance of the preconditioned conjugate gradients. A prescribed-memory ICF clearly beats the diagonal preconditioner in terms of CPU time while keeping memory requirements under control. It is also worth noting how too large values of p result in an increase of both the memory and the CPU time.

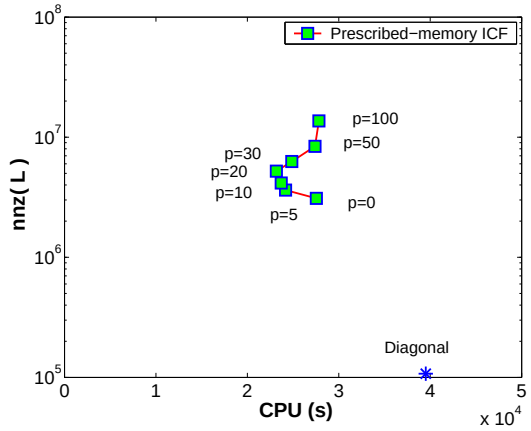


Fig. 9. Computational cost of prescribed-memory ICF preconditioners for air pollution problem

6 Concluding Remarks and Future Research

We have analyzed the numerical performance of two families of incomplete Cholesky factorizations (ICF) to precondition conjugate gradient (CG) iterations: drop-tolerance and limited-memory approaches. With an appropriate value for the corresponding numerical parameter (tolerance τ and fill-in density p, respectively), ICFs outperform both the diagonally preconditioned CG and the Cholesky direct solver.

According to our numerical experiments, the drop-tolerance τ should be selected in the range $[0.005, 0.01]$ for linear systems of order larger than 30 000. Similar results are reported in [22] for elasticity problems. Regarding the fill-in density p, our experiments suggest taking $p = 4$ or $p = 5$. This latter value is also recommended in [4].

Our numerical experiments also show that, for transient convection-diffusion problems with a constant velocity field (and, hence, the same matrix in all the time-marching process), the numerical efficiency of a preconditioner is completely controlled by the *application* stage at each time-step. Since the computational cost of the *computation* stage of the preconditioner can be amortized over many time-steps (tens of thousands in our applications), how much it costs to *obtain* the factorization turns out to be an irrelevant factor.

This conclusion motivates the search for more efficient preconditioners, even if they are more expensive to obtain. We are currently working in a symmetric sparse approximate inverse that exploits the block-tridiagonality of the system matrix, see Figure 1.

Another line also in progress is the use of domain decomposition with overlapping subdomains. This is a promising approach especially in convection-diffusion problems with abrupt fronts, where the solution only varies over the time-step in one or two subdomains. We believe that a combination of domain decomposition and appropriate preconditioning of each subdomain can result in a very efficient solution of 3D transient convection-diffusion problems.

References

[1] J. Donea, A. Huerta, Finite Element Methods for Flow Problems, John Wiley & Sons, Chichester, 2003.

[2] Y. Saad, Iterative Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[3] N. Munksgaard, Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients, ACM Trans. Math. Softw. 6 (2) (1980) 206–219.

[4] C.-J. Lin, J. J. Moré, Incomplete Cholesky factorizations with limited memory, SIAM J. Sci. Comput. 21 (1) (1999) 24–45.

[5] A. Rodríguez-Ferran, J. Sarrate, A. Huerta, Numerical modelling of void inclusions in porous media, Int. J. Numer. Methods Eng. 59 (4) (2004) 577–596.

[6] Z. Zlatev, Computer Treatment of Large Air Pollution Models, Kluwer Academic Publishers, Dordrecht, 1995.

[7] A. Quarteroni, A. Valli, Numerical Approximation of Partial Differential Equations, Vol. 23 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 1994.

[8] A. Huerta, J. Donea, Time-accurate solution of stabilized convection-diffusion-reaction equations: I — Time and space discretization, Commun. Numer. Methods Eng. 18 (8) (2002) 565–573.

[9] A. Huerta, B. Roig, J. Donea, Time-accurate solution of stabilized convection-diffusion-reaction equations: II — Accuracy analysis and examples, Commun. Numer. Methods Eng. 18 (8) (2002) 575–584.

[10] L. N. Trefethen, D. Bau III, Numerical Linear Algebra, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[11] A. Greenbaum, Iterative Methods for Solving Linear Systems, Vol. 17 of Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[12] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[13] M. Benzi, M. Tůma, A comparative study of sparse approximate inverse preconditioners, Appl. Numer. Math. 30 (2-3) (1999) 305–340.

[14] M. Benzi, J. K. Cullum, M. Tůma, Robust approximate inverse preconditioning for the conjugate gradient method, SIAM J. Sci. Comput. 22 (4) (2000) 1318–1332.

[15] M. Benzi, Preconditioning techniques for large linear systems: a survey, J. Comput. Phys. 182 (2) (2002) 418–477.

[16] S. A. Salvini, G. J. Shaw, An evaluation of new NAG Library solvers for large sparse symmetric linear systems, Tech. Rep. TR1/95, NAG Ltd, Oxford (1995).

[17] I. S. Duff, G. A. Meurant, The effect of ordering on preconditioned conjugate gradients, BIT 29 (4) (1989) 635–657.

[18] N. Sanín, G. Montero, Air pollution modelling using finite differences in a terrain conformal coordinate system, in: B. Topping, C. M. Soares (Eds.), Proceedings of The Fourth International Conference on Engineering Computational Technology, Civil-Comp Press, Stirling, Scotland, 2004.

[19] R. Montenegro, G. Montero, J. M. Escobar, E. Rodríguez, Efficient strategies for adaptive 3-D mesh generation over complex orography, Neural, Parallel & Scientific Comp. 10 (1) (2002) 57–76.

[20] J. M. Escobar, E. Rodríguez, R. Montenegro, G. Montero, J. M. González-Yuste, Simultaneous untangling and smoothing of tetrahedral meshes, Comput. Methods Appl. Mech. Eng. 192 (25) (2003) 2775–2787.

[21] G. Montero, E. Rodríguez, R. Montenegro, J. M. Escobar, J. M. González-Yuste, Genetic algorithms for an improved parameter estimation with local refinement of tetrahedral meshes in a wind model, Adv. Eng. Softw. 36 (1) (2005) 3–10.

[22] J. K. Dickinson, P. A. Forsyth, Preconditioned conjugate gradient methods for three-dimensional linear elasticity, Int. J. Numer. Methods Eng. 37 (13) (1994) 2211–2234.