Journal of Computational Physics 227 (2008) 10196–10208

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Deflated preconditioned conjugate gradient solvers for the Pressure–Poisson equation

Romain Aubry, Fernando Mut, Rainald Löhner *, Juan R. Cebral

Center for Computational Fluid Dynamics, College of Sciences, M.S. 6A2, George Mason University, Fairfax, VA 2030-4444, USA

ARTICLE INFO

Article history:
Received 12 March 2008
Received in revised form 7 August 2008
Accepted 25 August 2008
Available online 12 September 2008

Keywords:
Conjugate gradients
Pressure–Poisson equation
Incompressible flow solvers
Finite elements
CFD

ABSTRACT

A deflated preconditioned conjugate gradient technique has been developed for the solution of the Pressure–Poisson equation within an incompressible flow solver. The deflation is done using a region-based decomposition of the unknowns, making it extremely simple to implement. The procedure has shown a considerable reduction in the number of iterations. For grids with large graph-depth the savings exceed an order of magnitude. Furthermore, the technique has shown a remarkable insensitivity to the number of groups/regions chosen, and to the way the groups are formed.

1. Introduction

Many solvers for the incompressible Navier–Stokes equations, given by

$$\rho \mathbf{v}_{,t} + \rho \mathbf{v} \nabla \mathbf{v} + \nabla p = \nabla \mu \nabla \mathbf{v}, \tag{1}$$

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

where $\rho, \mathbf{v}, p, \mu$ denote the density, velocity, pressure and viscosity of the fluid, are based on so-called projection techniques, whereby the advancement of the flowfield in time is split into the following three substeps [22,2,23,3,32,15,1,17,37,13,24,8,41,11,19,18,20,40,25–28]:

– *Advective–diffusive prediction*: $\mathbf{v}^n \rightarrow \mathbf{v}^*$

$$\left[\frac{\rho}{\Delta t} - \nabla \mu \nabla\right](\mathbf{v}^* - \mathbf{v}^n) + \rho \mathbf{v}^n \cdot \nabla \mathbf{v}^n + \nabla p^n = \nabla \mu \nabla \mathbf{v}^n. \tag{3}$$

– *Pressure correction*: $p^n \rightarrow p^{n+1}$

$$\nabla \cdot \mathbf{v}^{n+1} = 0; \tag{4}$$

$$\rho \frac{\mathbf{v}^{n+1} - \mathbf{v}^*}{\Delta t} + \nabla (p^{n+1} - p^n) = 0; \tag{5}$$

which results in

* Corresponding author. Tel.: +1 703 993 3807.
E-mail addresses: rlohner@science.gmu.edu, rlohner@gmu.edu (R. Löhner).

0021-9991/$ - see front matter © 2008 Published by Elsevier Inc.
doi:10.1016/j.jcp.2008.08.025

$$\nabla^2(p^{n+1} - p^n) = \frac{\rho \nabla \cdot \mathbf{v}^*}{\Delta t}. \tag{6}$$

– *Velocity correction*: $\mathbf{v}^* \rightarrow \mathbf{v}^{n+1}$

$$\mathbf{v}^{n+1} = \mathbf{v}^* - \Delta t \nabla (p^{n+1} - p^n). \tag{7}$$

The solution of the so-called Pressure–Poisson equation, given by Eq. (6), which results in a discrete system of the form:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}. \tag{8}$$

is typically carried out with preconditioned conjugate gradient (PCG) solvers [38,40], and consumes a considerable percentage of the overall computational effort. Consequently, many attempts have been made to mitigate the impact of the Pressure–Poisson equation on the overall cost of a simulation. Options that have proven useful include:

– Improved prediction of the starting value for the iterative solver [13,26];
– Linelet preconditioners for highly stretched (e.g. boundary layer) grids [32,40] (in the sequel diagonal preconditioning is assumed to be the default for isotropic grids);
– Multistage or implicit treatment of the advective terms (more advective–diffusive work, allowing larger timesteps, nearly the same work for the Pressure–Poisson equation) [25,27].

Several attempts have also been made to use multigrid solvers [52,3,1,46,47,7]. However, for unstructured grids the expected gains have proven elusive to date. Moreover, cases with moving and or adapting meshes place further burdens on multigrid solvers vis a vis conjugate gradient solvers. The present paper describes a simple technique that has proven remarkably robust, and that works extremely well for those cases where traditional PCGs perform poorly.

2. Deflated conjugate gradients

Nicolaides' seminal paper [36] is perhaps the first paper to consider deflation methods for field solvers. The main concept was to remove components of the initial residual which may improved convergence. As opposed to multigrid techniques [33], no need for defining a smoother, a coarse grid or a prolongation/restriction operator is necessary. However, no numerical experiments were reported in [36]. The first experiments supporting the theoretical results are found in Mansfield [29], where the subdomain deflation approach of [36] is successfully applied to the bending of a cantilever beam and to the stationary Stokes problem. Piecewise constant and linear approximations are used to approximate the eigenmodes associated with the low frequencies. In [30] the deflation technique is applied to precondition a Schur complement matrix for second order operators arising from the partition used for parallel computing on each processor. In [31] the deflation technique is coupled to a damped Jacobi preconditioning still relying on a subdomain deflation. For unsymmetric problems, similar ideas were applied for the restart step of the GMRES algorithm in [34,21,12,6]. A few converged eigenvector approximations related to the smallest eigenvalues were saved in order to remove them from the spectrum. More recently, deflation was used for symmetric positive definite (SPD) matrices in [12] for an augmented conjugate gradient, where the Krylov subspace generated by a previous system is recycled for further solves in subsequent systems, and in [39], where the approximation of the eigenvectors proposed in [34] is used to augment the space of the subsequent system to deflate the lowest eigenvalues. Compared to the approach of [29–31] the main difference is that the eigenvectors are explicitly computed and not approximated through a subdomain deflation so that no knowledge of the solution is required for the subsequent solves. In [35,4] the projector relying on the deflation subspace is explicitely build from the structure of the coefficient matrix of the system to be solved. Vuik and coworkers [43,14,44,45] applied the deflated technique in various contexts, including problems characterized by layers of large contrast in the porosity [44,45], where the approximated eigenvectors verify the partial differential equation on each subdomain. In [14] some numerical experiments of deflated methods running on parallel machines are reported, as well as new bounds on the effective condition number of deflated and preconditioned deflated conjugate gradients. It is shown that if grid refinement is performed, keeping the subdomain grid resolution fixed, the condition number is insensitive to the grid size. Other possibilities for choosing the deflation vectors are proposed in [43]. Deflation is applied to magnetostatic problems in [9,10], where the approximate eigenvectors used to the deflation subspace mimic the flux patterns commonly sketched by engineers intuitively.

2.1. Theoretical considerations

The conjugate gradient [16] is the method of choice for solving SPD systems, such as Eq. (8), in an iterative way. Its memory requirements are minimal, which is particularly attractive for three dimensional problems. It may also be viewed as a direct method, giving the solution in a finite number of steps in exact arithmetic, although it converges much faster in practice. Defining the residual $\mathbf{r}$ as

$$\mathbf{r} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}, \tag{9}$$

the basic iterative step is given by

$$\Delta\mathbf{x}_k = \alpha_k(\mathbf{r}_{k-1} + e_{k-1}\Delta\mathbf{x}_{k-1}) = \alpha_k\mathbf{v}_k, \tag{10}$$

where α_k, e_{k-1} are scaling factors chosen so that successive increments are A-orthogonal to each other:

$$\Delta\mathbf{x}_{k-1}\cdot\mathbf{A}\cdot\Delta\mathbf{x}_k = 0. \tag{11}$$

This yields

$$e_{k-1} = -\frac{\mathbf{r}_{k-1}\cdot\mathbf{A}\cdot\Delta\mathbf{x}_{k-1}}{\Delta\mathbf{x}_{k-1}\cdot\mathbf{A}\cdot\Delta\mathbf{x}_{k-1}} \tag{12}$$

which may be simplified for linear systems by observing that

$$\mathbf{r}_{k-1} - \mathbf{r}_{k-2} = -\mathbf{A}\cdot\Delta\mathbf{x}_{k-1}, \tag{13}$$

and hence,

$$e_{k-1} = -\frac{\mathbf{r}_{k-1}\cdot(\mathbf{r}_{k-1} - \mathbf{r}_{k-2})}{\Delta\mathbf{x}_{k-1}\cdot(\mathbf{r}_{k-1} - \mathbf{r}_{k-2})}. \tag{14}$$

The parameter α_k is obtained by forcing

$$\mathbf{A}\cdot(\mathbf{x}_{k-1} + \Delta\mathbf{x}_k) = \mathbf{f} \tag{15}$$

in a 'matrix weighted' sense by multiplication with $\Delta\mathbf{x}_k$

$$\Delta\mathbf{x}_k\cdot\mathbf{A}\cdot\Delta\mathbf{x}_k = \Delta\mathbf{x}_k\cdot\mathbf{r}_{k-1}, \tag{16}$$

yielding

$$\alpha_k = \frac{\mathbf{v}_k\cdot\mathbf{r}_{k-1}}{\mathbf{v}_k\cdot\mathbf{A}\cdot\mathbf{v}_k}. \tag{17}$$

The CG algorithm generates a sequence $\mathbf{x}_1,\ldots,\mathbf{x}_i$ such that

$$\mathbf{x}_i \in \mathbf{x}_0 + \mathbf{K}_i, \tag{18}$$

where $\mathbf{K}_i = \mathrm{span}\{\mathbf{r}_0, \mathbf{A}\cdot\mathbf{r}_0, \ldots, \mathbf{A}_{i-1}\cdot\mathbf{r}_0\}$ is the Krylov subspace of dimension i generated by the initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\cdot\mathbf{x}_0$. At each step, the approximation $\mathbf{x}_i$ verifies:

$$\|\mathbf{x}_{\mathrm{ex}} - \mathbf{x}_i\|_A = \min_{\mathbf{x}\in\mathbf{x}_0+K_i}\|\mathbf{x}_{\mathrm{ex}} - \mathbf{x}\|_A, \tag{19}$$

where $\|\mathbf{x}\|_A = (\mathbf{A}\cdot\mathbf{x},\mathbf{x})^{1/2}$ and $\mathbf{x}_{\mathrm{ex}}$ denotes the exact solution $\mathbf{A}\cdot\mathbf{x}_{\mathrm{ex}} = \mathbf{b}$. The classical a priori bound for the error in the A-norm is

$$\|\mathbf{x}_{\mathrm{ex}} - \mathbf{x}_k\|_A = \|\mathbf{e}_k\|_A \leqslant 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\|\mathbf{e}_0\|_A, \tag{20}$$

where κ is the condition number of matrix $\mathbf{A}$. However, as decribed in [42], the convergence speeds up as soon as the lowest eigenvalues are 'discovered' by the convergence process, giving rise to a condition number based on the active i.e. the non-discovered eigenvalues. Therefore, if some knowledge of the eigenmodes associated to the lowest eigenvalues is at hand, removing them from the spectrum of A would improve convergence as compared to the classical conjugate gradient process. That is what the deflated conjugate gradient (DCG) tries to achieve.

The basic step of the DCG algorithm simply adds one more (low-dimensional) search direction $\mathbf{W}\cdot\mathbf{d}$ for the increment

$$\Delta\mathbf{x}_k = \alpha_k(\mathbf{r}_{k-1} + e_{k-1}\Delta\mathbf{x}_{k-1} - \mathbf{W}\cdot\mathbf{d}_k) = \alpha_k\mathbf{v}_k. \tag{21}$$

The columns of the matrix $\mathbf{W}$ are a basis for the deflation subspace. $\mathbf{d}$ is of very low dimensionality (<100) as compared to $\mathbf{x}$ (number of points in the mesh). Forcing successive increments to be A-orthogonal now yields:

$$\Delta\mathbf{x}_{k-1}\cdot\mathbf{A}\cdot(\mathbf{r}_{k-1} + e_{k-1}\Delta\mathbf{x}_{k-1} - \mathbf{W}\cdot\mathbf{d}_k) = 0. \tag{22}$$

The additional search direction is obtained by enforcing that all increments be A-orthogonal with $\mathbf{W}$ (i.e. $\mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\Delta\mathbf{x}_k = 0\forall k$):

$$\mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\mathbf{W}\cdot\mathbf{d}_k = \mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\mathbf{r}_{k-1}. \tag{23}$$

With $\mathbf{d}_k$, e_{k-1} is obtained from:

$$e_{k-1} = -\frac{(\mathbf{r}_{k-1} - \mathbf{W}\cdot\mathbf{d}_k)\cdot\mathbf{A}\cdot\Delta\mathbf{x}_{k-1}}{\Delta\mathbf{x}_{k-1}\cdot\mathbf{A}\cdot\Delta\mathbf{x}_{k-1}}, \tag{24}$$

$\mathbf{v}_k$ from

$$\mathbf{v}_k = \mathbf{r}_{k-1} + e_{k-1}\Delta\mathbf{x}_{k-1} - \mathbf{W}\cdot\mathbf{d}_k, \tag{25}$$

and α_k from Eq. (17).

2.2. Algorithmic implementation

An optimal algorithmic implementation is given in [39]:

- Define a preconditioning matrix: $\mathbf{M}$.
- Define: $\widehat{\mathbf{A}} = \mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\mathbf{W}$.
- Start: given $\mathbf{x}_{-1}$:

$$\mathbf{r}_{-1} = \mathbf{b} - \mathbf{A}\cdot\mathbf{x}_{-1}$$
$$\widehat{\mathbf{A}}\cdot\mathbf{d}_0 = \mathbf{W}^{\mathrm{T}}\cdot\mathbf{r}_{-1}$$
$$\mathbf{x}_0 = \mathbf{x}_{-1} + \mathbf{W}\cdot\mathbf{d}_0$$
$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\cdot\mathbf{x}_0.$$

- Compute: $\mathbf{M}\cdot\mathbf{z}_0 = \mathbf{r}_0$.
- Solve: $\widehat{\mathbf{A}}\cdot\mathbf{d} = \mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\mathbf{z}_0$.
- Set: $\mathbf{p}_0 = -\mathbf{W}\cdot\mathbf{d} + \mathbf{z}_0$.
- Do until convergence:

$$\alpha_j = (\mathbf{r}_j\cdot\mathbf{z}_j)/(\mathbf{p}_j\cdot\mathbf{A}\cdot\mathbf{p}_j)$$
$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j\mathbf{p}_j$$
$$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j\mathbf{A}\cdot\mathbf{p}_j$$
$$\mathbf{M}\cdot\mathbf{z}_{j+1} = \mathbf{r}_{j+1}$$
$$\beta_j = (\mathbf{r}_{j+1}\cdot\mathbf{z}_{j+1})/(\mathbf{r}_j\cdot\mathbf{z}_j)$$
$$\widehat{\mathbf{A}}\cdot\mathbf{d}_j = \mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\mathbf{z}_{j+1}$$
$$\mathbf{p}_{j+1} = \mathbf{z}_{j+1} + \beta_j\mathbf{p}_j - \mathbf{W}\cdot\mathbf{d}_j$$

As the dimensionality of $\mathbf{d}$ is low, the solution/inversion of $\widehat{\mathbf{A}}$ is carried out using a skyline solver. The extra storage requirements for the DPCG are very modest: both $\mathbf{W}$ and $\mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}$ may be stored as linked lists [28] in two arrays of $O(N_{\mathrm{p}})$, where N_{p} denotes the number of points in the mesh. This is in contrast with eigenvalue deflation, where storage increases proportionally to the number of eigenvalues considered. The cost in terms of vector operations is also of $O(N_{\mathrm{p}})$, i.e. very modest as compared to the non-deflated PCG.

2.3. Definition of subdomains

The DCG technique requires the definition of a mapping $\mathbf{W}$ from the lower-dimensional basis $\mathbf{d}$ to the vector of unknowns $\mathbf{x}$. The simplest way of defining this mapping for a mesh-based system of equations is via the points of the mesh: local domains (subdomains) or groups of points are assigned to a variable d_i. The entries in $\mathbf{W}$ are unity for the points of this region, and zero for all other points. This is equivalent to the assumption of a constant shape-function in the subdomains. We have implemented several ways of defining these regions. The two most commonly used are:

(a) Seedpoints:

For this (manual) technique, the user defines an arbitrary set of points, called seedpoints. Given a mesh, the closest mesh points to the seedpoints are found, and a region number is assigned accordingly. Points not assigned to any region are then added one layer at a time until all points have been assigned a region number.

(b) Advancing front:

Starting from a point where $\mathbf{x}$ is prescribed, neighbouring points are added until a specified number of points per region is exceeded. The last set of points added is then used as a starting point for the next group. The procedure is repeated until all points have been assigned a region number.

3. Examples

The deflated PCG solver has been tested on a variety of examples, a few of which are included here. We remark from the outset that the main aim is the comparison of speed. Detailed comparison to experiments, mesh refinement studies, etc. of

the basic scheme may be found in [24,41,11,19,18,20]. In all cases diagonal preconditioning was used in the isotropic mesh regions, while linelet preconditioning [32,40] was used for the highly anisotropic mesh regions (boundary layer grid regions). All examples were run on Intel Xeon 2.77 GHz machines, with either 4 Gbytes or 16 Gbytes of RAM.

3.1. Pipe flow

The first example considered is the classic Poiseuille pipe flow. Given that the pressure is prescribed at the outflow, and that a pressure field has to establish itself along the pipe, the number of iterations required increases with the graph depth (defined as the maximum minimal number of edges connecting any two points) of the finite element mesh. The physical dimensions and parameters were set as follows:

- pipe radius: $r = 1$,
- pipe length: $l = 20, 40, 80$,
- density: $\rho = 1$,
- inflow velocity: $\upsilon = 1$,
- viscosity: $\mu = 0.01$.

The element size was set to $h = 0.1$, implying approximately a graph depth of $gd = 200, 400, 800$ for the cases considered. This resulted in grids of 129 Kels, 260 Kels and 516 Kels respectively. All cases were run for 100 timesteps. In order to assess the sensitivity of the deflated PCG to the number of groups chosen, 15, 30 and 60 groups were chosen. The regions were generated automatically using the advancing front algorithm, starting from the exit (i.e. prescribed pressure) plane. Figs. 1(a)–(d) show the surface mesh, region boundaries for 15 groups, pressure and absolute value of the velocity for $l = 20$. Fig. 1(e) shows the number of iterations required for the PCG. The sudden 'dips' in the number of iterations at some timesteps are due to the fact that a projective prediction of pressure increments with 2 Krylov vectors [26] was used. Note the dramatic decrease in the number of iterations achieved by the deflated PCG solver. This decrease may also be seen in Fig. 1(f), which depicts the average number of iterations for the first 20 steps for the different options chosen. While the number of iterations increases linearly with pipe length for the conventional PCG, the performance of the deflated PCG solver is rather insensitive to the number of the groups chosen. To highlight the importance of a fast Pressure–Poisson solver, the total CPU requirements are compared in Fig. 1(g). Note that for the case $l = 80$ the same solver with deflated PCG runs seven times faster (!).

3.2. NACA0012

The second example considered is the classic NACA0012 wing at $\alpha = 5^\circ$ angle of attack. This is a steady, inviscid case (Euler). Figs. 2(a) and (b) show the surface mesh employed, as well as the surface pressures obtained. The mesh had approximately 370 Kels. This problem was solved using local timesteps to accelerate convergence to steady-state. The seedpoints used are shown in Fig. 2(c). These were input by hand, and the regions for the deflated PCG were grown from them.

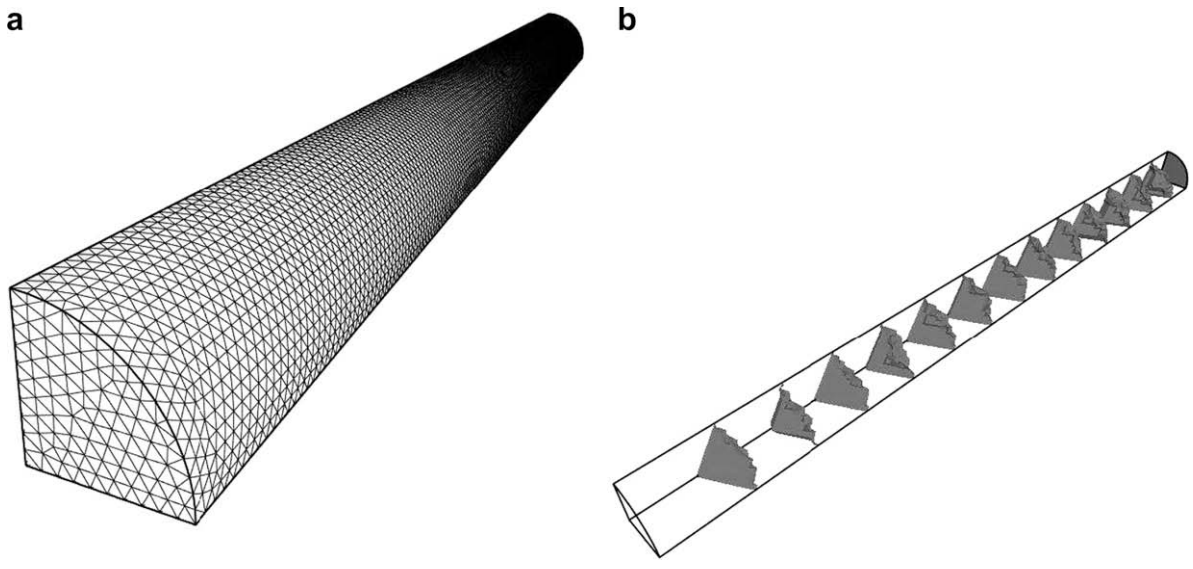

Fig. 1(a,b). Pipe: surface mesh and deflation groups for $l = 20$.

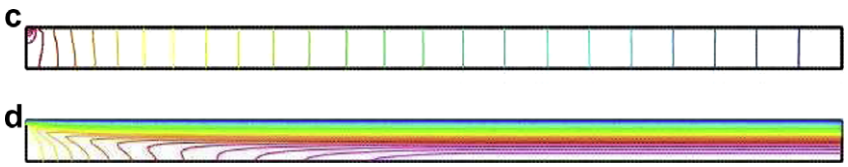

Fig. 1(c,d). Pipe: pressure and Abs(velocity) for plane $z = 0$.

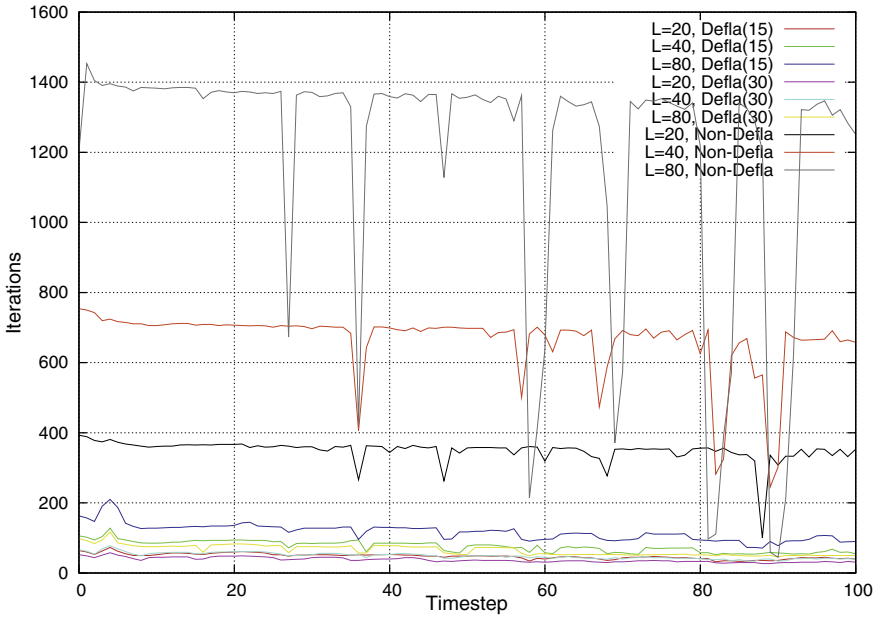

Fig. 1(e). Pipe: number of iterations required.

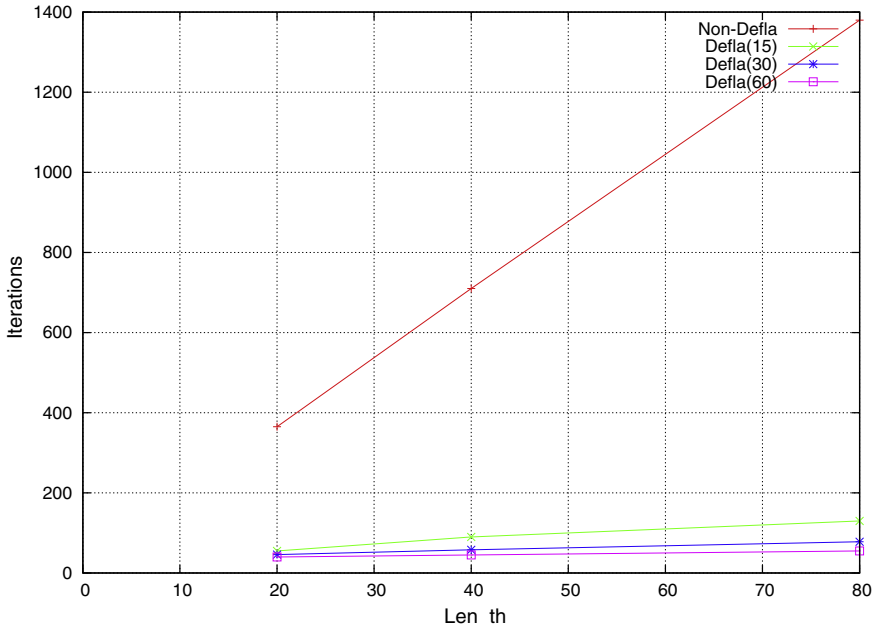

Fig. 1(f). Pipe: number of iterations required.

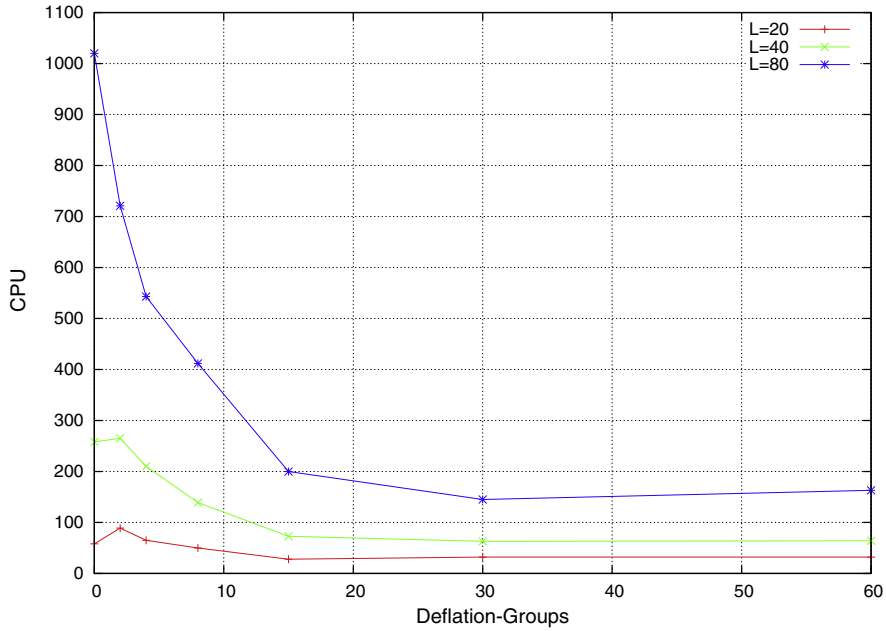

Fig. 1(g). Pipe: CPU required for 100 timesteps.

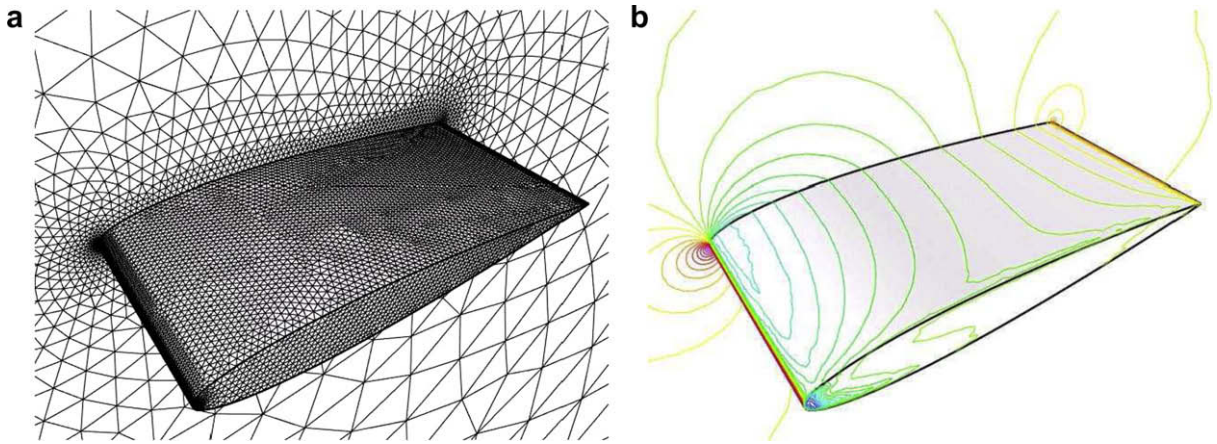

Fig. 2(a,b). NACA 0012: surface mesh and pressure.

Fig. 2(d) shows the number of iterations as the solution is advanced to steady state. One can observe that even on this rather coarse mesh with limited graph depth between the outflow, prescribed pressure boundary and the airfoil, the deflated PCG requires approximately half the iterations of the usual PCG. The overall savings in CPU time were of the order of 10%.

3.3. von Karman vortex street

The third example considered is also a well known benchmark case. A circular cylinder is suspended in a uniform stream of incompressible fluid. The separation at the back of the cylinder generates the so-called von Karman vortex street, whose characteristics depend on the Reynolds number $Re = \rho V_\infty D/\mu$, where D denotes the diameter of the cylinder. This is essentially a 2-D example, but was run with the 3-D solver. A mesh of 113 Kels was used for the simulation, with special placement of points in the vicinity of the cylinder. The parameters were chosen such that the resulting Reynolds number was $Re = 190$.

Figs. 3(a), 3(b), 3(c) show the surface grid and the absolute value of the velocity in a cut plane. The run was started impulsively and continued until the vortex street was fully developed. Starting from this (restart) state, the solution was advanced 50 steps using a 3-stage Runge–Kutta scheme for the advection, and the different deflation options were excercised and compared to one another. The regions of a typical run with deflated PCG are shown in Fig. 3(d). The iterations required per time-

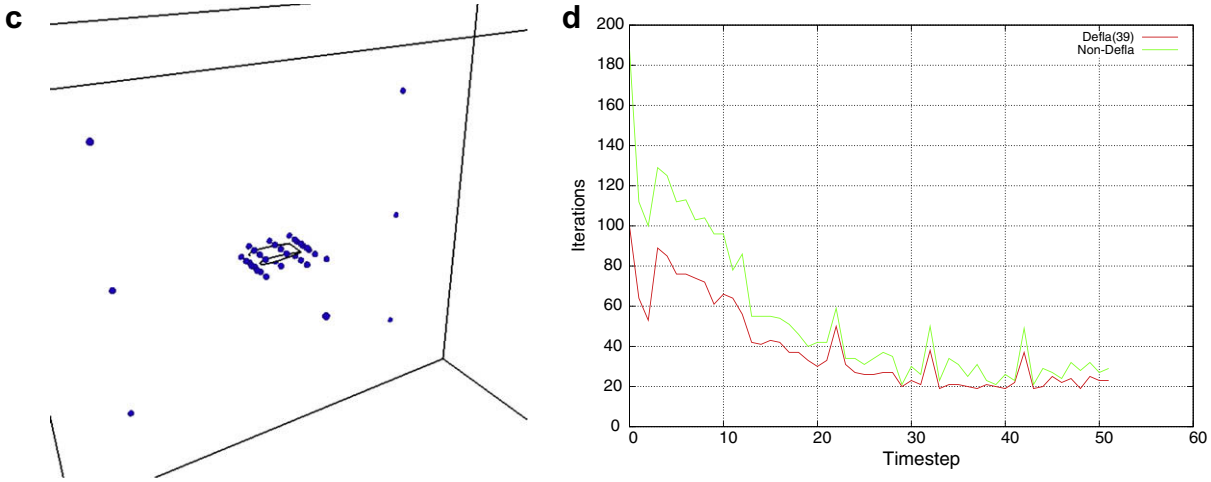

Fig. 2(c,d). NACA 0012: seed points and iterations.

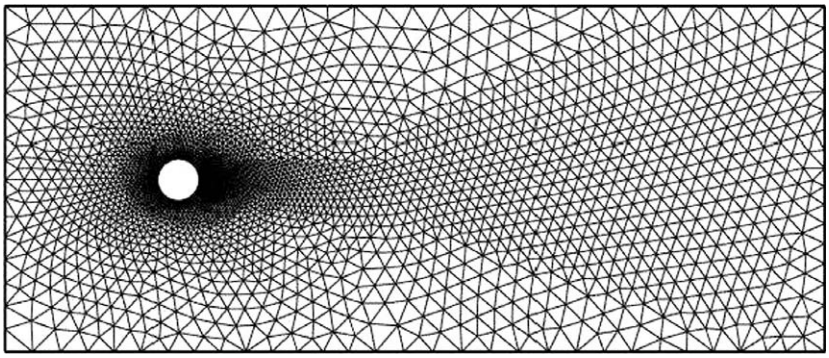

Fig. 3(a). von Karman vortex street: surface mesh.

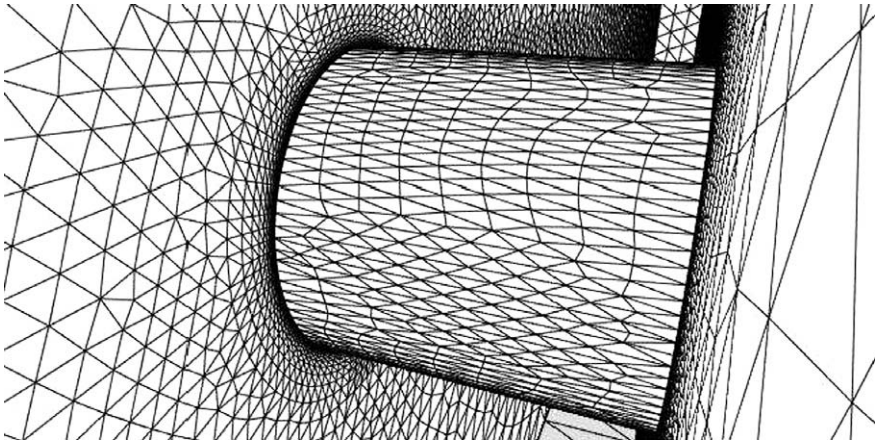

Fig. 3(b). von Karman vortex street: surface mesh (detail).

step are displayed in Fig. 3(e). One can observe that the deflated PCG requires substantially less iterations, and is rather insensitive to the number of domains chosen.

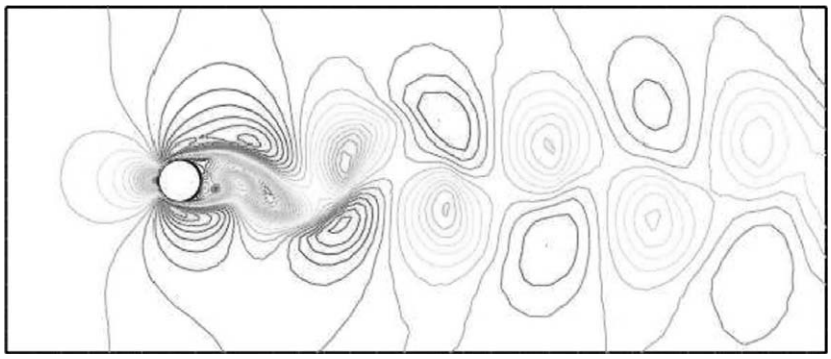

Fig. 3(c). von Karman vortex street: Abs(Veloc).

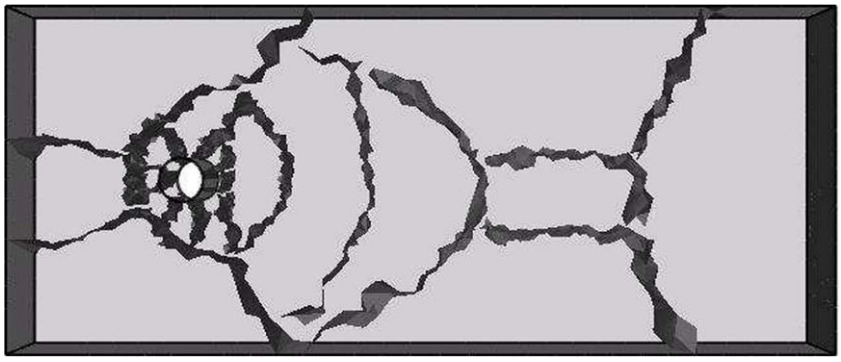

Fig. 3(d). von Karman vortex street: deflated PCG regions.

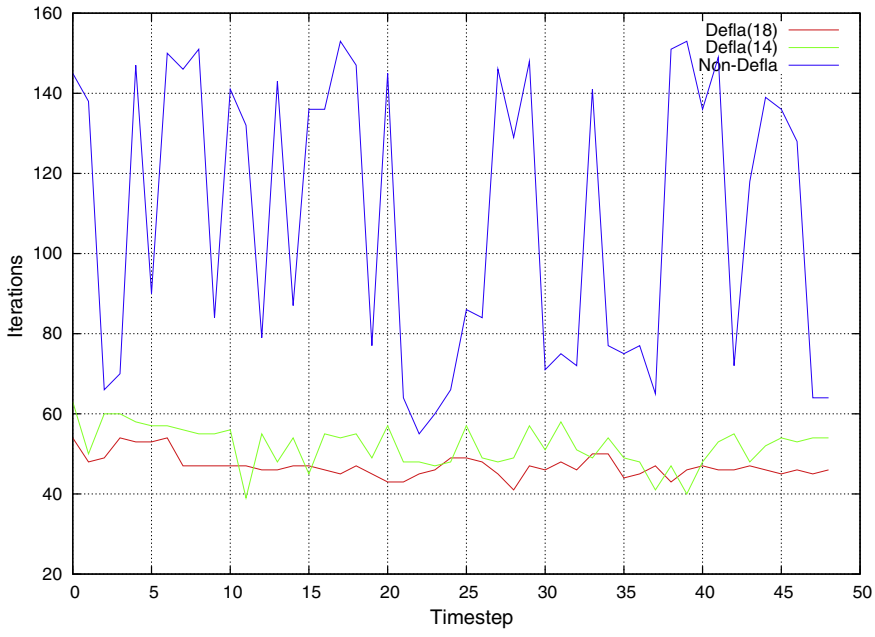

Fig. 3(e). von Karman vortex street: iteration count.

3.4. Cerebral aneurysm

The fourth example is an image-based patient-specific model of a cerebral aneurysm with its parent vessel. The model was constructed using patient-specific three-dimensional rotational angiography (3DRA) images [5]. The blood was considered as a Newtonian fluid, which is modeled by the *unsteady* incompressible Navier–Stokes equations. Fig. 4(a) shows the surface mesh of the complete model (left), a close-up view of the aneurysm (right-top) and a closer-up view of the surface elements (right-bottom). The volume mesh has 646 K points and 3.6 M elements.

The groups were constructed using the seedpoints alternative, where 45 points were selected manually from the surface model (see Fig. 4(b)). The simulation was performed using implicit timestepping, solving a pseudo-steady problem at each timestep [27]. Within each pseudo-timestep the advective terms were integrated implicitly using 5 LU-SGS passes (local

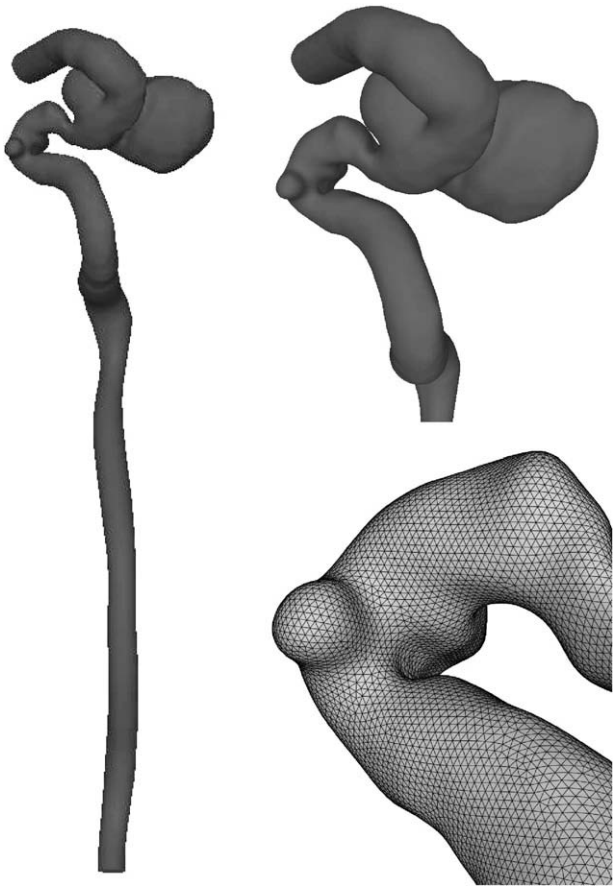

Fig. 4(a). Patient-specific model of a cerebral aneurysm.

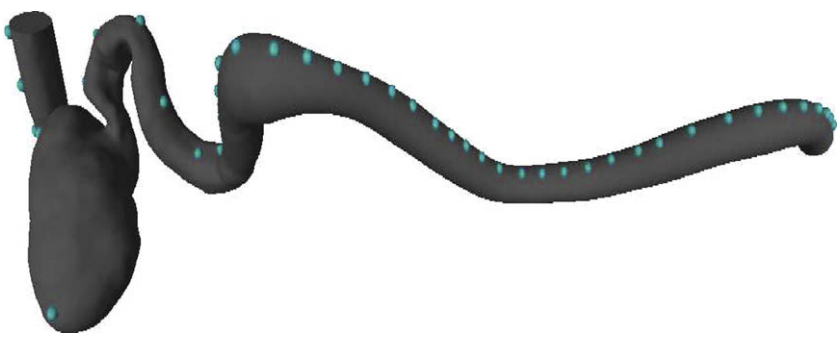

Fig. 4(b). Seed-points used.

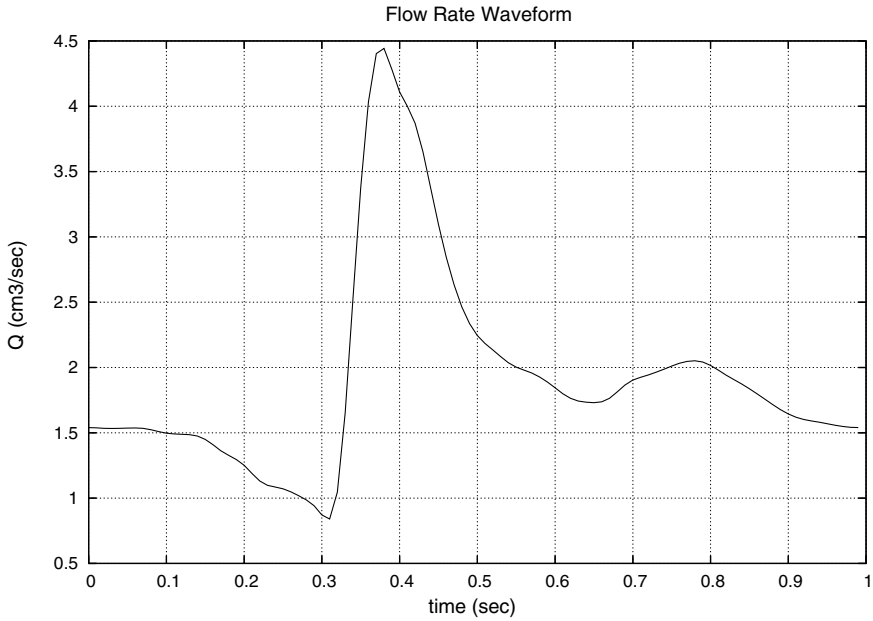

Fig. 4(c). Flow rate imposed at inflow.

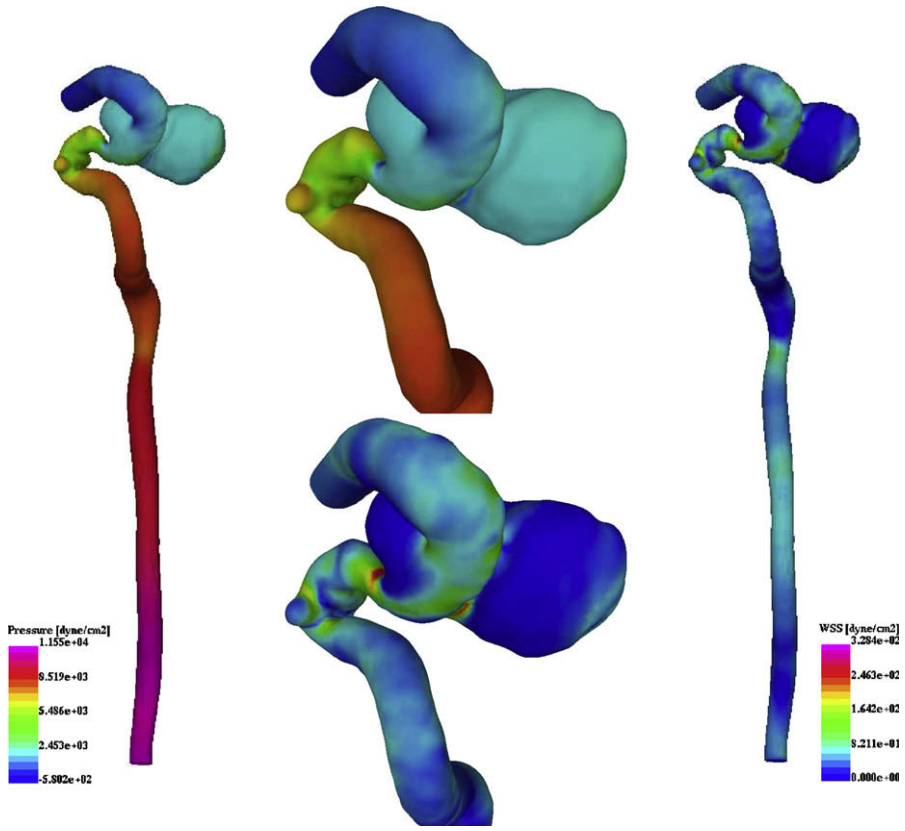

Fig. 4(d). Pressure and wall shear stress (WSS) at time $T = 1.4$ s.

Courant number $C = 5.0$), followed by the pressure projection. The timestep was set to $\Delta t = 0.01$ s. The material properties of the fluid were taken to be $\rho = 1.0\,\mathrm{g/cm^3}$ and $\mu = 0.04$*Poise*. The pressure was prescribed (homogeneous bc) at the outflow (top part), while a time-dependent velocity profile was prescribed at the inflow (bottom part). The velocity profile was com-

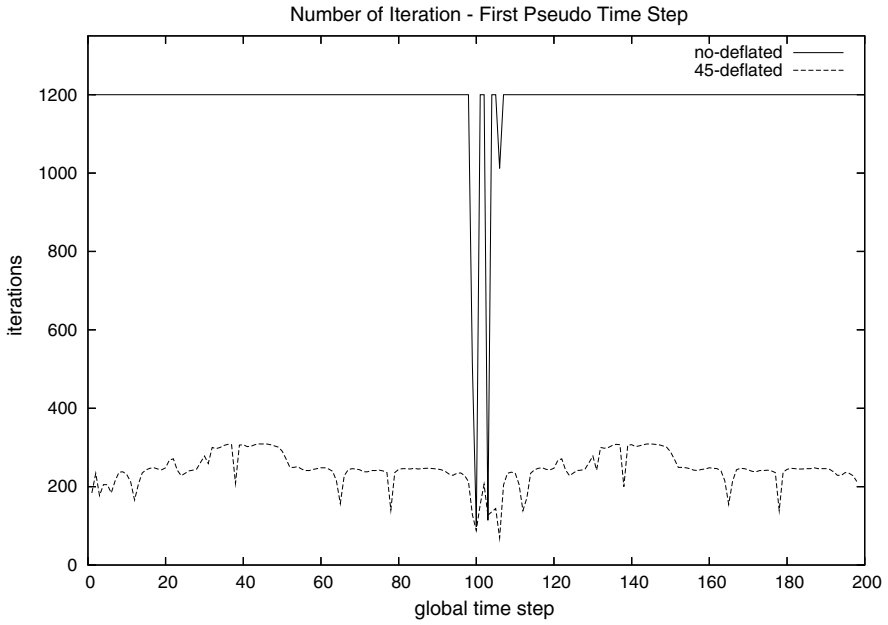

Fig. 4(e). Iterations required to solve the Pressure–Poisson system.

puted using a Womersley model with measured inflow rates as described in [5]. Fig. 4(c) shows the inflow rate curve for one cardiac cycle. No-slip boundary conditions were applied at the vessel wall.

This case was run for 200 time steps (two cardiac cycles). Fig. 4(d) depicts the pressure (left and top) and wall shear stress (right and bottom) at time $T = 1.4$ s (second cardiac cycle, just after the inflow rate peak).

Fig. 4(e) shows the number of iterations required for the Pressure–Poisson solver during the first pseudo time step of each implicit timestep with and without deflation. Note that, without deflation, the number of iterations reaches the pre-set limit of 1200 without having converged. This is expected since it corresponds to the first pseudo time step, where the inflow boundary conditions have just changed. However, the periodicity (in time) of the boundary conditions is reflected in the number of iterations for the deflated case, showing convergence even though we are in the first pseudo time step. The deep peaks found in the non-deflated case around time $t = 1$ s are due to the slow variation in inflow rate at the end of the cardiac cycle. This slow variation makes the solution of the previous time step a very good initial guess for the next time step. The total CPU time required to complete the simulation without deflation was 51 h 47 m, while with deflation was 19 h 10 m, about 2.7 times faster.

4. Conclusions

A deflated preconditioned conjugate gradient technique has been developed for the solution of the Pressure–Poisson equation within an incompressible flow solver. The deflation is done using a region-based decomposition of the unknowns, making it extremely simple to implement.

For all cases tried to date, this procedure has shown a considerable reduction in the number of iterations. For grids with large graph-depth the savings may exceed an order of magnitude. Moreover, the technique has shown a remarkable insensitivity to the number of groups/regions chosen, and to the way the groups are formed.

Future work will explore more complex deflation matrices **W**, allowing for linear or distance-based shape functions.

Acknowledgments

It is a pleasure to acknowledge the input of Guillaume Pierrot and Jean Roger from ESI-Group, Rungis, France, who first informed us on the potential of DPCG methods for the Pressure–Poisson equation.

This work was partially supported by DTRA. The technical monitors were Drs. Michael Giltrud, Young Sohn and Ali Amini.

References

[1] B. Alessandrini, G. Delhommeau, A multigrid velocity–pressure-free surface elevation fully coupled solver for calculation of turbulent incompressible flow around a hull, in: Proc. 21st Symp. on Naval Hydrodynamics, Trondheim, Norway, June, 1996.

[2] J.B. Bell, P. Colella, H. Glaz, A second-order projection method for the Navier–Stokes equations, J. Comp. Phys. 85 (1989) 257–283.

[3] J.B. Bell, D.L. Marcus, A second-order projection method for variable-density flows, J. Comp. Phys. 101 (1992) 2.
[4] S.S. Bielawski, S.G. Mulyarchik, A.V. Popov, The construction of an algebraically reduced system for the acceleration of preconditioned conjugate gradients, J. Comput. Appl. Math. 70 (2) (1996) 189–200.
[5] J.R. Cebral, M.A. Castro, S. Appanaboyina, C.M. Putman, D. Millan, A.F. Frangi, Efficient pipeline for image-based patient-specific analysis of cerebral aneurysm hemodynamics: technique and sensitivity, IEEE Trans. Med. Imaging 24 (1) (2005) 457–467.
[6] A. Chapman, Y. Saad, Deflated and augmented Krylov subspace with eigenvectors, Numer. Linear Algebra Appl. 4 (1997) 43–66.
[7] N.H. Chen, Multigrid methods for the incompressible Navier–Stokes problem on three-dimensional ynstructured meshes, Ph.D. thesis, George Mason University, 2004.
[8] R. Codina, Pressure stability in fractional step finite element methods for incompressible flows, J. Comp. Phys. 170 (2001) 112–140.
[9] H. De Gersem, K. Hameyer, Convergence improvement of the conjugate gradient iterative method for finite element simulations, COMPEL (2000) 20-90-97.
[10] H. De Gersem, K. Hameyer, Deflated iterative solver for magnetostatic finite element models with large differences in permeability, Eur. Phys. J. Appl. Phys. 13 (2000) 45–49.
[11] E. Eaton, Aero-Acoustics in an Automotive HVAC Module American PAM User Conf, 24–25, Birmingham, Michigan, 2001. October.
[12] J. Erhel, K. Burrage, B. Pohl, Restarted GMRES preconditioned by deflation, J. Comput. Appl. Math. 69 (1996) 303–318.
[13] P.F. Fischer, Projection techniques for iterative solution of $Ax = b$ with successive right-hand sides, Comp. Meth. Appl. Mech. Eng. 163 (1998) 193–204.
[14] J. Frank, C. Vuik, On the construction of deflation-based preconditioners, SIAM J. Sci. Comput (2001) 23-442-462.
[15] M.D. Gunzburger, R. Nicolaides (Eds.), Incompressible Computational Fluid Dynamics: Trends and Advances, Cambridge University Press, 1993.
[16] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Stand (1952) 49-409-436.
[17] Y. Kallinderis, A. Chen, An Incompressible 3-D Navier–Stokes Method with Adaptive Hybrid Grids, 1996. AIAA-96-0293.
[18] Y. Li, T. Kamioka, T. Nouzawa, T. Nakamura, Y. Okada, N. Ichikawa, Verification of Aerodynamic Noise Simulation by Modifying Automobile Front-Pillar Shape; JSAE 20025351, JSAE Annual Conf., Tokyo, July, 2002.
[19] K.J. Karbon, S. Kumarasamy, Computational aeroacoustics in automotive design, computational fluid and solid mechanics, in: Proc. First MIT Conference on Computational Fluid and Solid Mechanics, 871-875, Boston, June, 2001.
[20] K.J. Karbon, R. Singh, Simulation and design of automobile sunroof buffeting noise control, in: 8th AIAA-CEAS Aero-Acoustics Conf., Brenckridge, June, 2002.
[21] S.A. Kharchenko, A.Yu. Yeremin, Eigenvalue translation based preconditioners for the GMRES (k) method, Numer. Linear Algebra Appl. 2 (1995) 51–70.
[22] J. Kim, P. Moin, Application of a fractional-step method to incompressible Navier–Stokes equations, J. Comp. Phys. 59 (1985) 308–323.
[23] R. Löhner, A Fast Finite Element Solver for Incompressible Flows, 1990. AIAA-90-0398.
[24] R. Löhner, C. Yang, E. Oñate, S. Idelssohn, An unstructured grid-based, parallel free surface solver, Appl. Num. Math. 31 (1999) 271–293.
[25] R. Löhner, Multistage explicit advective prediction for projection-type incompressible flow solvers, J. Comp. Phys. 195 (2004) 143–152.
[26] R. Löhner, Projective prediction of pressure increments, Comm. Num. Meth. Eng. 21 (4) (2005) 201–207.
[27] R. Löhner, Chi Yang, J.R. Cebral, F. Camelli, O. Soto, J. Waltz, Improving the speed and accuracy of projection-type incompressible flow solvers, Comp. Meth. Appl. Mech. Eng. 195 (23-24) (2006) 3087–3109.
[28] R. Löhner, Applied CFD Techniques, J. Wiley & Sons, London, 2008.
[29] L. Mansfield, On the use of deflation to improve the convergence of the conjugate gradient iteration, Comm. Appl. Num. Meth. 4 (1988) 151–156.
[30] L. Mansfield, On the conjugate gradient solution of the Schur complement system obtained from domain decomposition, SIAM J. Numer. Anal. 27 (1990) 1612–1620.
[31] L. Mansfield, Damped Jacobi preconditioning and coarse grid deflation for conjugate gradient iteration on parallel computers, SIAM J. Numer. Anal. 12 (1991) 1314–1323.
[32] D. Martin, R. Löhner, An Implicit Linelet-Based Solver for Incompressible Flows, 1992. AIAA-92-0668.
[33] S.F. McCormick, Multigrid methods, SIAM, 1987.
[34] R.B. Morgan, A restarted GMRES method augmented with eigenvectors, SIAM J. Matrix Anal. Appl. 16 (1995) 1154–1171.
[35] S.G. Mulyarchik, S.S. Bielawski, A.V. Popov, Efficient computational method for solving linear systems, J. Comput. Phys. 110 (1994) 201–211.
[36] R.A. Nicolaides, Deflation of conjugate gradients with applications to boundary value problems, SIAM J. Numer. Anal. 24 (1987) 355–365.
[37] R. Ramamurti, R. Löhner, A parallel implicit incompressible flow solver using unstructured meshes, Comput. Fluids 5 (1996) 119–132.
[38] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing, Boston, 1996.
[39] Y. Saad, J. Yeung, J. Erhel, F. Guyomarc'h, A deflated version of the conjugate gradient algorithm, SIAM J. Sci. Comput 21 (2000) 1909–1926.
[40] O. Soto, R. Löhner, F. Camelli, A linelet preconditioner for incompressible flows, Int. J. Numer. Method Heat Fluid Flow 13 (1) (2003) 133–147.
[41] A. Takamura, M. Zhu, D. Vinteler, Numerical Simulation of Pass-by Maneuver Using ALE Technique, JSAE Annual Conf. (Spring), Tokyo, May, 2001.
[42] A. van der Sluis, H.A. van der Vorst, The rate of convergence of the conjugate gradients, Numerische Mathematik 48 (1986) 543–560.
[43] F. Vermolen, C. Vuik, A. Segal, Deflation in preconditioned conjugate gradient methods for finite element problems, in: M. Křižek, P. Neittaanmäki, R. Glowinski, S. Korotov (Eds.), Conjugate Gradient and Finite Element Methods, Springer, Berlin, 2004, pp. 103–129.
[44] C. Vuik, A. Segal, J.A. Meijerink, An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients, J. Comp. Phys. 152 (1999) 385–403.
[45] C. Vuik, A. Segal, J.A. Meijerink, G.T. Wijma, The construction of projection vectors for a deflated ICCG method applied to problems with extreme contrasts in the coefficients, J. Comp. Phys. 172 (2001) 426–450.
[46] J. Waltz, R. Löhner, A Grid Coarsening Algorithm for Unstructured Multigrid Applications, 2000. AIAA-00-0925.
[47] J. Waltz, Unstructured Multigrid Methods, Ph.D. thesis, George Mason University, 2000.