INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2011; **87**:2–14
Published online 22 June 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nme.2932

Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation: Extensions and improvements

Rainald Löhner[1,*,†], Fernando Mut[1], Juan Raul Cebral[1], Romain Aubry[2] and Guillaume Houzeaux[2]

[1]*CFD Center, Department of Computational and Data Science, M.S. 6A2, College of Science, George Mason University, Fairfax, VA 22030-4444, U.S.A.*
[2]*Barcelona Supercomputing Center, Barcelona, Spain*

SUMMARY

Extensions and improvements to a deflated preconditioned conjugate gradient technique for the solution of the pressure-Poisson equation within an incompressible flow solver are described. In particular, the use of the technique for embedded grids, for cases where volume of fluid or level set schemes are required and its implementation on parallel machines are considered. Several examples are included that demonstrate a considerable reduction in the number of iterations and a remarkable insensitivity to the number of groups/regions chosen and/or to the way the groups are formed. Copyright © 2010 John Wiley & Sons, Ltd.

Received 11 January 2010; Revised 6 April 2010; Accepted 7 April 2010

KEY WORDS: iterative solvers; conjugate gradients; pressure-Poisson equation; incompressible solvers; finite elements; CFD

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 or fractional step schemes. These split the advancement of the flowfield in time into the following three substeps [1–22]:

- *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*: $p^n \rightarrow p^{n+1}$

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

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

*Correspondence to: Rainald Löhner, CFD Center, Department of Computational and Data Science, M.S. 6A2, College of Science, George Mason University, Fairfax, VA 22030-4444, U.S.A.
†E-mail: rlohner@gmu.edu

which results in

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

- *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 Equation (6), which results in a discrete system [23] of the form

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

is typically carried out with preconditioned conjugate gradient (PCG) solvers [18, 24], 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 [10, 20];
- Linelet preconditioners for highly stretched (e.g. boundary layer) grids [5, 18] (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) [19, 21].

Several attempts have also been made to use multigrid solvers [2, 4, 7, 25–27]. 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 extensions and improvements to a simple deflation technique [28] that has proven remarkably robust, and that works extremely well for those cases where traditional PCGs perform poorly.

2. DEFLATED CONJUGATE GRADIENTS

The discretization of the Pressure-Poisson equation (Equation (6)), as well as many other many field equations (e.g. heat conduction and elasticity), leads to a symmetric positive-definite (SPD) matrix system of the form:

$$\mathbf{A}\cdot\mathbf{x}=\mathbf{b}. \tag{9}$$

Besides multigrid [29] (if applicable), the conjugate gradient [30] is the method of choice for solving SPD systems, such as Equation (9), 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

$$\mathbf{r}=\mathbf{b}-\mathbf{A}\cdot\mathbf{x}, \tag{10}$$

the basic iterative step is given by

$$\Delta\mathbf{x}_k=\alpha_k(\mathbf{r}_{k-1}+e_{k-1}\Delta\mathbf{x}_{k-1})=\alpha_k\mathbf{v}_k, \tag{11}$$

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

$$\Delta\mathbf{x}_{k-1}\cdot\mathbf{A}\cdot\Delta\mathbf{x}_k=0. \tag{12}$$

DOI: 10.1002/nme

This yields

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

which may be simplified for linear systems by observing that

$$\mathbf{r}_{k-1} - \mathbf{r}_{k-2} = -\mathbf{A} \cdot \Delta\mathbf{x}_{k-1}, \tag{14}$$

and, hence,

$$e_{k-1} = -\frac{\mathbf{r}_{k-1} \cdot (\mathbf{r}_{k-1} - \mathbf{r}_{k-2})}{\Delta\mathbf{x}_{k-1} \cdot (\mathbf{r}_{k-1} - \mathbf{r}_{k-2})}. \tag{15}$$

The parameter α_k is obtained by forcing

$$\mathbf{A} \cdot (\mathbf{x}_{k-1} + \Delta\mathbf{x}_k) = \mathbf{b} \tag{16}$$

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

$$\Delta\mathbf{x}_k \cdot \mathbf{A} \cdot \Delta\mathbf{x}_k = \Delta\mathbf{x}_k \cdot \mathbf{r}_{k-1}, \tag{17}$$

yielding

$$\alpha_k = \frac{\mathbf{v}_k \cdot \mathbf{r}_{k-1}}{\mathbf{v}_k \cdot \mathbf{A} \cdot \mathbf{v}_k}. \tag{18}$$

The basic idea behind the deflated conjugate gradient (DCG) is to solve in a separate manner, low frequency, and high frequency errors based on the fact that Krylov iterative solvers are known to be very efficient for reducing high frequency modes whereas quite slow to smooth out low frequency ones [28, 31–48]. At the algorithmic level this may be viewed as adding one more (low-dimensional) search direction $\mathbf{W} \cdot \mathbf{d}$ for the increment:

$$\Delta\mathbf{x}_k = \alpha_k (\mathbf{r}_{k-1} + e_{k-1} \Delta\mathbf{x}_{k-1} - \mathbf{W} \cdot \mathbf{d}_k) = \alpha_k \mathbf{v}_k. \tag{19}$$

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

$$\Delta\mathbf{x}_{k-1} \cdot \mathbf{A} \cdot (\mathbf{r}_{k-1} + e_{k-1} \Delta\mathbf{x}_{k-1} - \mathbf{W} \cdot \mathbf{d}_k) = 0. \tag{20}$$

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

$$\mathbf{W}^{\mathrm{T}} \cdot \mathbf{A} \cdot \mathbf{W} \cdot \mathbf{d}_k = \mathbf{W}^{\mathrm{T}} \cdot \mathbf{A} \cdot \mathbf{r}_{k-1}. \tag{21}$$

As $\mathbf{W}^{\mathrm{T}} \cdot \mathbf{A} \cdot \Delta\mathbf{x}_{k-1} = 0$, e_{k-1} is obtained as before from:

$$e_{k-1} = -\frac{\mathbf{r}_{k-1} \cdot \mathbf{A} \cdot \Delta\mathbf{x}_{k-1}}{\Delta\mathbf{x}_{k-1} \cdot \mathbf{A} \cdot \Delta\mathbf{x}_{k-1}}, \tag{22}$$

$\mathbf{v}_k$ from

$$\mathbf{v}_k = \mathbf{r}_{k-1} + e_{k-1} \Delta\mathbf{x}_{k-1} - \mathbf{W} \cdot \mathbf{d}_k, \tag{23}$$

and α_k from Equation (18).

2.1. Algorithmic implementation

An optimal algorithmic implementation is given in [49]:

- Define a preconditioning matrix: $\mathbf{M}$
- Define: $\hat{\mathbf{A}} = \mathbf{W}^{\mathrm{T}} \cdot \mathbf{A} \cdot \mathbf{W}$

DOI: 10.1002/nme

- Start: Given $\mathbf{x}_{-1}$
  $\mathbf{r}_{-1}=\mathbf{b}-\mathbf{A}\cdot\mathbf{x}_{-1}$
  $\hat{\mathbf{A}}\cdot\mathbf{d}_0=\mathbf{W}^{\mathrm{T}}\cdot\mathbf{r}_{-1}$
  $\mathbf{x}_0=\mathbf{x}_{-1}+\mathbf{W}\cdot\mathbf{d}_0$
  $\mathbf{r}_0=\mathbf{b}-\mathbf{A}\cdot\mathbf{x}_0$
- Compute: $\mathbf{M}\cdot\mathbf{z}_0=\mathbf{r}_0$
- Solve: $\hat{\mathbf{A}}\cdot\mathbf{d}=\mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\mathbf{z}_0$
- Set: $\mathbf{p}_0=-\mathbf{W}\cdot\mathbf{d}+\mathbf{z}_0$
- Do until convergence:
  $\alpha_j=(\mathbf{r}_j\cdot\mathbf{z}_j)/(\mathbf{p}_j\cdot\mathbf{A}\cdot\mathbf{p}_j)$
  $\mathbf{x}_{j+1}=\mathbf{x}_j+\alpha_j\mathbf{p}_j$
  $\mathbf{r}_{j+1}=\mathbf{r}_j-\alpha_j\mathbf{A}\cdot\mathbf{p}_j$
  $\mathbf{M}\cdot\mathbf{z}_{j+1}=\mathbf{r}_{j+1}$
  $\beta_j=(\mathbf{r}_{j+1}\cdot\mathbf{z}_{j+1})/(\mathbf{r}_j\cdot\mathbf{z}_j)$
  $\hat{\mathbf{A}}\cdot\mathbf{d}_j=\mathbf{W}^{\mathrm{T}}\cdot\mathbf{A}\cdot\mathbf{z}_{j+1}$
  $\mathbf{p}_{j+1}=\mathbf{z}_{j+1}+\beta_j\mathbf{p}_j-\mathbf{W}\cdot\mathbf{d}_j$

As the dimensionality of $\mathbf{d}$ is low, the solution/inversion of $\hat{\mathbf{A}}$ 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 in two arrays of $O(N_p)$, where N_p denotes the number of points in the mesh. This is in contrast with eigenvalue deflation [50], where storage increases proportionally to the number of eigenvalues considered.

2.2. *Definition of projection space*

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 $\mathbf{W}$ for a mesh-based system of equations is by agglomerating the nodes of the mesh into subdomains (or 'coarse grains') and then defining a polynomial reconstruction over each of them. The lowest-order reconstruction (and the one most often employed due to its good performance, very low storage requirements and simple interpretation) assumes that a constant value is assigned to all the points of a subdomain. The entries in $\mathbf{W}$ are unity for the points of this region, and zero for all other points. We have implemented several ways of defining these regions [28]. 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 the points have been assigned a region number.

(b) *Advancing front*: Starting from a point where $\mathbf{x}$ is prescribed, neighboring 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. This technique requires a number of refinements in order to work reliably. The main aim of these improvement techniques is to assure that the points belonging to a group are connected in space, so that the approximation obtained from the coarse-grain agglomeration reproduces as faithfully as possible the field of unknowns seen by the mesh. If a desired number of elements or points is prescribed per region, it could happen that at corners or crevices, a few points are added to a new group, prompting the addition of further points that are not connected to these. The best way to proceed is to keep forming groups, and then to add the smaller groups to the neighboring groups in a post-processing step.

3. EMBEDDED AND IMMERSED GRIDS

Embedded and immersed grids have proven invaluable when dealing with dirty or incomplete CAD geometry [51–54], cases where the geometrical input would require too much human intervention

DOI: 10.1002/nme

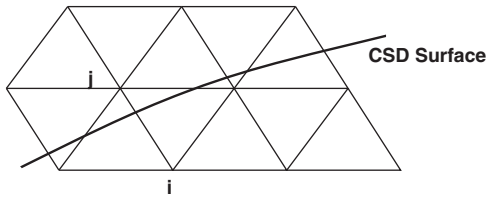

Figure 1. Embedded surface.

or time [55], and fluid–structure problems with changing topology [53, 56]. For the DCG technique to work well, it is important that points on either side of an embedded surface are not allocated to the same group (Figure 1). After all, the unknowns in the upper and lower parts could be completely different. If allocated to the same deflation group, the approximation via the deflation vector **d** would be very poor, yielding no gain. When adding points to a group via the advancing front technique, care is therefore taken not to consider any edges crossed by an embedded surface or immersed body.

4. VOLUME OF FLUID AND LEVEL SET TECHNIQUES

Volume of fluid (VOF) or level set (LS) techniques have been used frequently to simulate flows with free surfaces [57, 58]. As the pressure is only computed in the region where the liquid is present, the deflated PCG technique has to be modified accordingly. One option would be to form all groups before every update/timestep. This is rather expensive, and as it turns out, unnecessary. A far simpler way to proceed is to see whether all points/edges within a deflation group/region are active. Only if this is the case, the group is considered in the deflation procedure. Conversely, should this not be the case, the usual PCG is used.

5. PARALLEL IMPLEMENTATION

5.1. Distributed memory implementation

Compared with the parallel PCG on distributed memories, the deflated PCG in parallel involves the construction of matrix $\hat{\mathbf{A}}$ at the beginning of each solve, the construction of the small RHS at each iteration by the multiplication with $\mathbf{W}^{\mathrm{T}}$, the resolution of the small linear system, and the extra update for $\mathbf{p}_{j+1}$ through the multiplication by $\mathbf{W}$. In this work, the group decomposition is completely independent of the domain decomposition, so that each processor is allowed to contain an arbitrary number of groups, although roughly the same number is expected as both the domain and the group decomposition are isotropic.

As far as the update for $\mathbf{p}_{j+1}$ is concerned, the multiplication by $\mathbf{W}$ is straightforward due to the fact that it is completely local as each point knows which group it belongs to, so nothing special needs to be done compared with the serial deflated PCG. For the construction of matrix $\hat{\mathbf{A}}$, each processor builds its own part of the matrix and an All_Reduce() is performed so that each processor has a copy of the matrix. The inversion is done in parallel for each processor at the beginning of the solve. Although the small matrix may be quite large for an All_Reduce(), it does not represent a significant CPU part of the whole solve. As in the serial version, it is possible to store matrix $\mathbf{W}^{\mathrm{T}} \cdot \mathbf{A}$ locally in each processor with a point-to-point communication at the beginning of the solve.

The main difficulty lies in the construction of the small RHS. As there are no particular relationship between groups and subdomains, the groups are not local and a processor j that is not a neighbor of processor i will contribute to the small RHS needed by processor i. This involves another All_Reduce() at each iteration. As will be seen in the numerical results, this

DOI: 10.1002/nme

may represent a nonnegligible part of the global CPU consumed by the solver. Two ways to improve the solver in parallel have been followed. First, as a global All_Reduce() is needed by the deflation and a small All_Reduce() is needed by the solver to compute the global β_j coefficient, the local β_j coefficient may be computed first in each processor; then, the local RHS to the small system may be computed next; and then an All_Reduce() on the small RHS extended by the local β_j coefficient may be performed. The vector to be reduced has been expanded in a negligible way, so that negligible time is spent due to the increased size. However, the latency due to the All_Reduce() of the β_j coefficient has been completely eliminated as will be shown in the numerical results.

The second way to improve the parallel performances of the deflated PCG was motivated by the fact that, due to isotropy of the groups and the subdomains, only close (but not necessarily neighbor) subdomains contribute to the same part of the small RHS, hence the All_Reduce() involves many zero additions. Second, the implementation of an All_Reduce() traditionally involves a first phase to reduce the sought vector, and a second phase to gather the sought vector to all the processors, with a global complexity of $2*O(\log(\text{nproc}))$. Another implementation would be to rely on an All_Gatherv() where each processor would gather its contribution to the number of group in its own domain to the rest of the other processors. The complexity of this operation is only half of the All_Reduce(). However, a larger vector is gathered as it has the size of the sum of the number of groups for each processor. The reasoning relies on the fact that this larger vector will be gathered only at the last stages of the All_Gatherv(), whereas an All_Reduce() needs to exchange a vector of the same size at each stage.

The deflated PCG has its roots in the multigrid paradigm. However, compared with a parallel multigrid implementation, the direct solve is the cornerstone of the deflated technique. In a multigrid context, every solve is performed by an iterative approach. The main strength of an iterative approach from a parallel viewpoint is that only point-to-point communications are involved compared with the global communications induced by the RHS of the direct solver. This may constitute a bottleneck for a massively parallel deflated PCG, although multigrid may also degrade in performance in such instances.

5.2. *Shared memory implementation*

As stated before, the deflated PCG requires the construction of the matrix $\hat{\mathbf{A}}$ at the beginning of each solve, the construction of the small RHS at each iteration by the multiplication with $\mathbf{W}^{\mathrm{T}}$, the resolution of the small linear system, and the extra update for $\mathbf{p}_{j+1}$ through the multiplication by $\mathbf{W}$. The multiplication by $\mathbf{W}$ to obtain $\mathbf{p}_{j+1}$ does not involve any form of memory contention or recursion, and is therefore straightforward. The construction of $\mathbf{W}^{\mathrm{T}}\cdot\hat{\mathbf{A}}$ is performed over groups of edges that do not have any memory contention, neither in serial nor in parallel [58] (the same could be done over elements if these are grouped such that memory contention is avoided). The final product to obtain $\mathbf{W}^{\mathrm{T}}\cdot\hat{\mathbf{A}}\cdot\mathbf{W}$ is more difficult to parallelize, and has been carried out in the scalar mode so far. At this point, we have only used the shared memory implementation on less than 64 processors. In such cases this (scalar) part, as well as the (scalar) matrix solve, does not impact the CPU considerably.

6. RESULTS

6.1. *Flow past a sphere, embedded*

The first case, taken from [22], considers laminar incompressible flow past a sphere, and is shown in Figure 2. Owing to symmetry considerations, only a quarter of the sphere is treated. The physical parameters were set as follows: $D=1$, $\mathbf{v}_\infty=(1,0,0)$, $\rho=1.0$, $\mu=0.01$, yielding a Reynolds number of $Re=100$.

The grid had an element size of approximately $h=0.0450$ in the region of the sphere. This led to a grid with approximately 140 Kels. The surface grid and the footprint of the embedded

DOI: 10.1002/nme

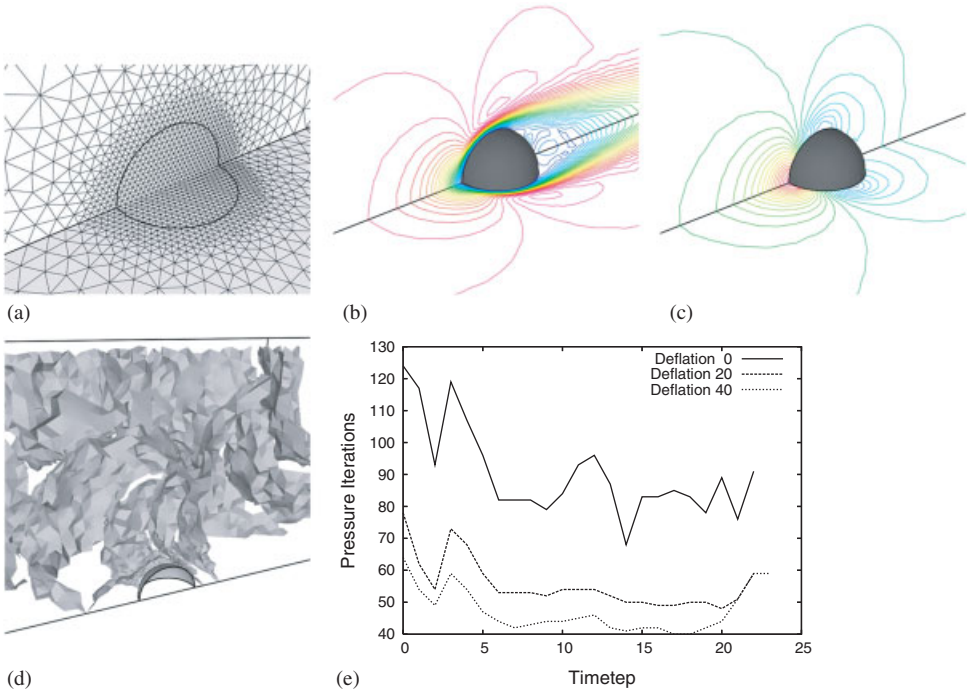

Figure 2. (a)–(c) Sphere: surface grid, Abs(Veloc), Pressure; (d) deflation regions; and (e) pressure iterations required.

surface on the boundary are shown in Figure 2(a). The respective pressures and absolute values of the velocities are shown in Figures 2(b,c). The (20) deflation regions may be discerned from Figure 2(d). The DCG technique was used for the Poisson equation [21]. Figure 2(e) compares the number of pressure iterations required with and without deflation. Note that even on this coarse mesh with rather small graph depth, the savings are considerable.

6.2. Flow in the brain

This case has been chosen to illustrate the efficiency of the deflated PCG in a parallel, distributed memory context. The geometry displayed in Figure 3(a) represents the arteries in the brain. Similar problems are found for complex engineering piping applications. The mesh has 3.4 Mpts and 19 Mels. This example has been run on 512 IBM Power 5 processors. The group number has been set to 500 for this case. The convergence obtained for this example is displayed in Figure 3(b). The convergence is similar to the one obtained in the serial mode, as a speed-up of one order of magnitude is reached in the iteration number. However, various interesting features appear in the parallel version. First, due to the complex topology of the brain, various arteries abut at the same area, as shown in Figure 3(c). One may display the number of surrounding processors for each processor, as illustrated in Figure 3(d,e), where each point represents the barycenter of each processor and its color is related to the number of neighboring processors. The complex topology is reflected in the large variation of surrounding processors per processor. In an artery, the average surrounding processors should be around 2, whereas it is seen that some processors have up to 9 neighbors. In this example, METIS was used to partition the domain. However, it is thought that the variation in the surrounding domains is more due to the complex topology than to a flaw in METIS. An imbalance is also reflected in each subdomain as the number of elements per processor ranges from 2.5 to 3.5 Kels. To further study the parallel implementation, traces of the run are displayed in Figures 3(f–i). Figure 3(f) shows the volume imbalance or the time spent in each processor, where the horizontal axis represents the time and the vertical axis the processor number. Note that some processors spend much more time than others in CPU time and that, for

DOI: 10.1002/nme

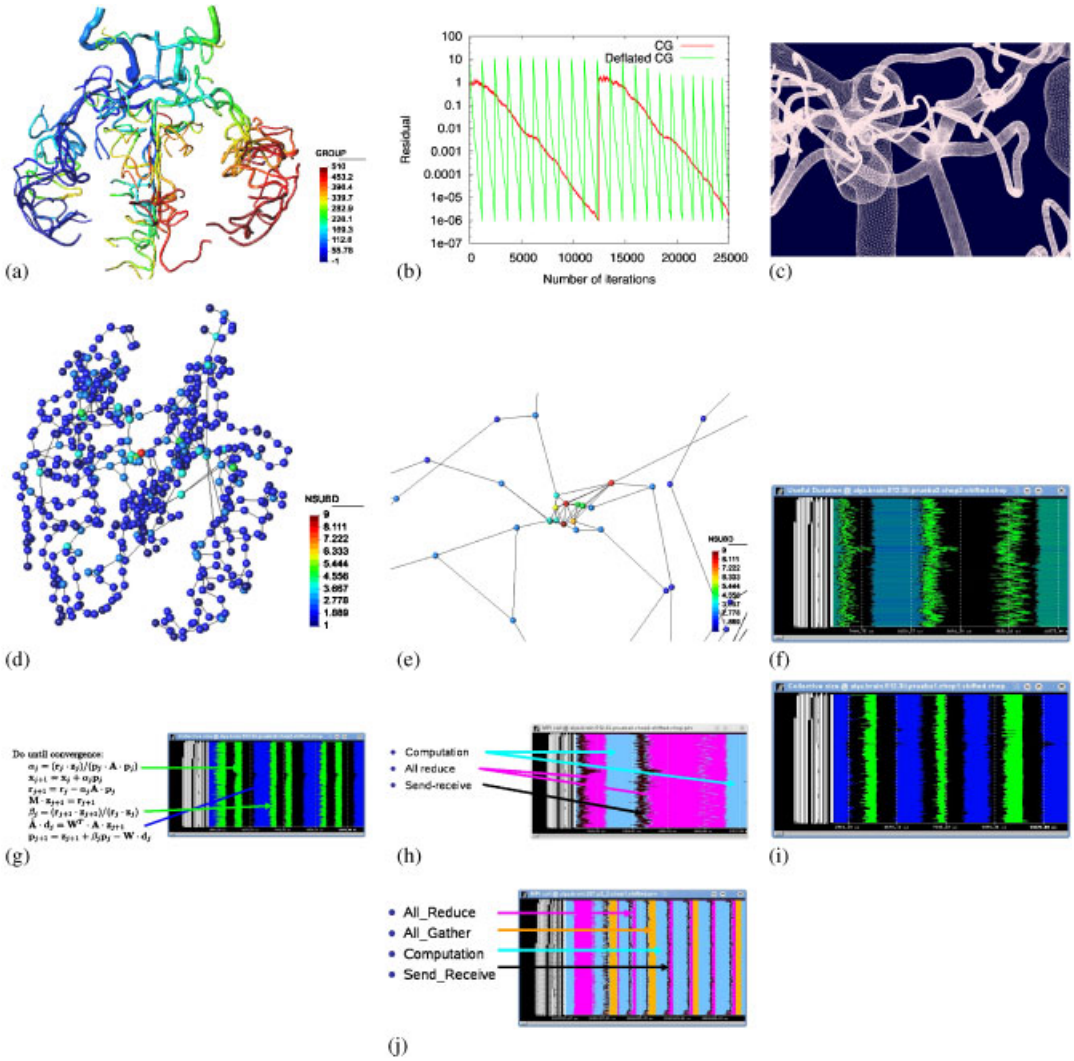

Figure 3. (a) Brain geometry with group distribution; (b) brain convergence of Poisson solver; (c) zoom of a convergent zone (d,e) neighbors per processor and zoom; (f) volume imbalance. Time in horizontal and processor number in vertical. The green color represents the computational time only. Darker color means longer computation time; (g) collective communications related to algorithm; (h) original collective communication compared with computation; (i) collective size improved blue is associated with All_Reduce(), green to Send_Receive(), and black is computation; and (j) implementation with All_Gatherv().

the same processors, the point-to-point communication is much longer, as there are much more neighbors to communicate with. Figure 3(h) presents the mapping between the trace colors and the algorithm for the main loop. In Figure 3(h), the CPU time of the collective communications, computation time, and Send_Receive communications are illustrated by different colors. As may be observed, the time spent in the communication time represents roughly 40% of the total time. In Figure 3(i), the improvement performed by grouping the All_Reduce() related to the scalar product with the one associated with the small RHS is displayed, and may be compared with the original one, Figure 3(h). Roughly, the cost of the scalar product communication has been eliminated. Finally, Figure 3(j) illustrates the implementation with the All_Gatherv(). For this case, the average number of groups per subdomain is three, which means that the gathered vector has a size of 1560. This must be compared with the size of the All_Reduced() vector, which is equal to the number of groups, namely 512. It is observed that the All_Gatherv() does not improve the

DOI: 10.1002/nme

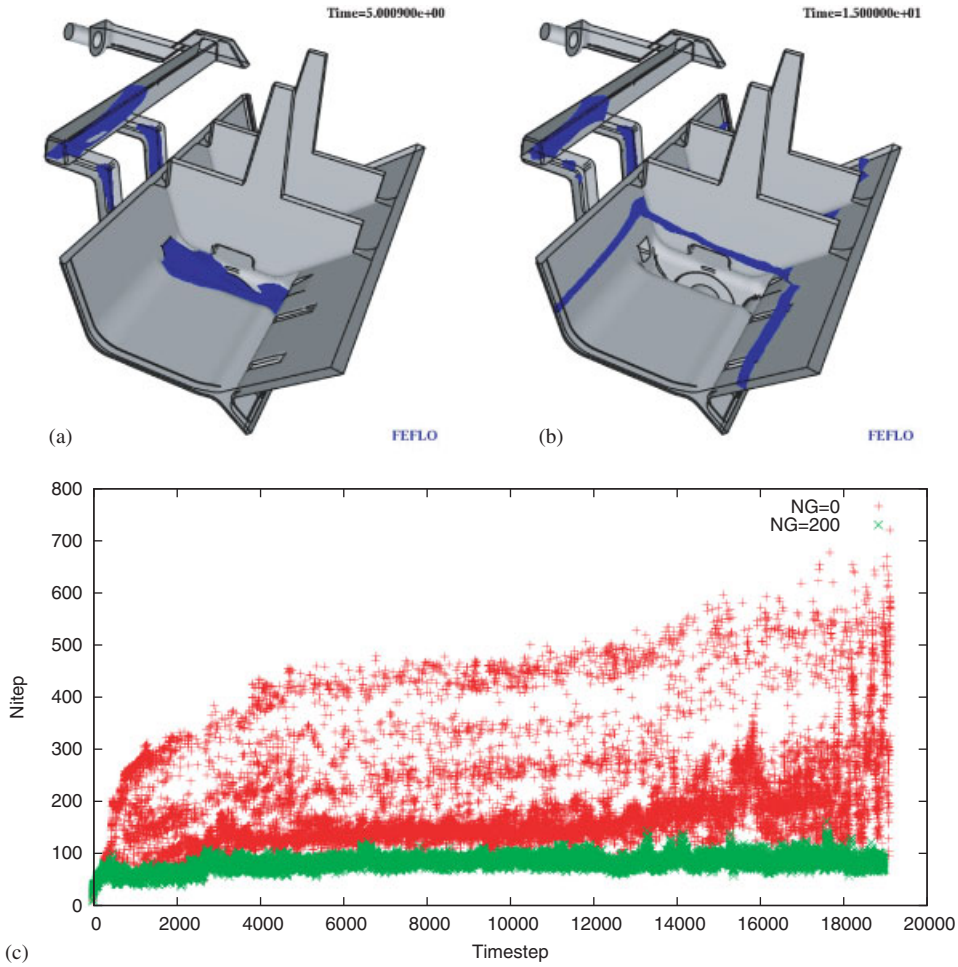

Figure 4. (a, b) Moldfilling: free surface; and (c) moldfilling: number of iterations required.

CPU time. This may be related to the fact that the final vector size has been multiplied by a factor of three.

To sum up, the convergence has been increased by an order of magnitude, but the communication represents 40% of the run-time. The final parallel speed up between the PCG and its deflated version is therefore around four. It is not as impressive as the serial case but is still considerable, as some cases may not even converge without deflation.

6.3. Moldfilling

The third case considers the filling of a piece (in this case a large earthmover shovel) with molten metal. The flow is described by the incompressible Navier–Stokes equations with a Smagorinsky turbulence model. Two stages during the mold-filling process are shown in Figures 4(a, b).

The mesh had approximately 1.42 Mels. The DCG technique was used for the Poisson equation [21] that enforces the incompressibility of the fluid. Figure 4(c) compares the number of pressure iterations required with and without deflation. As the number of elements with fluid increases continuously as the piece is filled, so does the graph distance between the inflow (prescribed velocity) and the free surface (prescribed pressure). This translates into an increasing number of pressure iterations required per step as the filling process takes place. Using the deflated groups, this number is almost constant.

DOI: 10.1002/nme

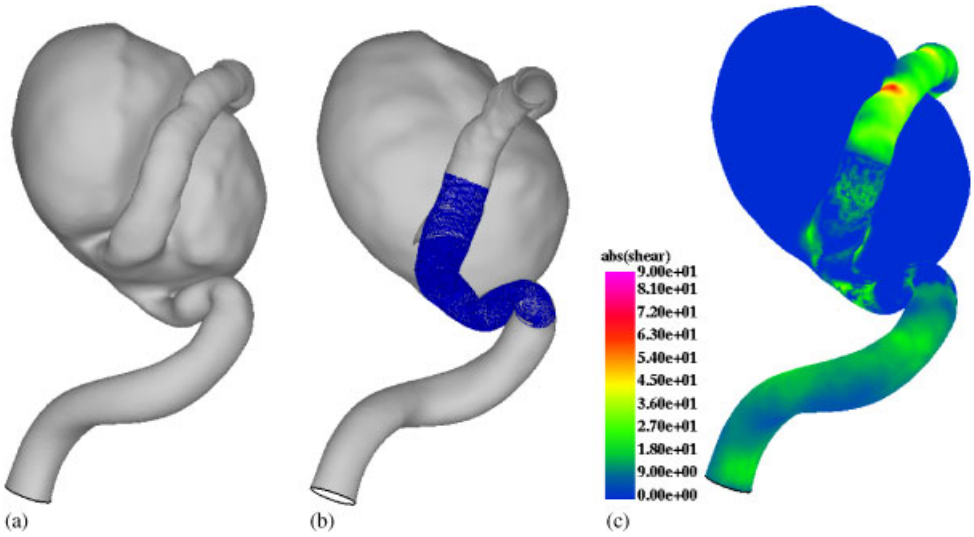

Figure 5. (a–c) Aneurysm: geometry and wall shear stress.

Table I. Statistics for aneurysm.

nelem (M)	nproc	CPU (s)	Speedup	Speedup(Ideal)	nproc*time/elem/step
14	8(S)	31 853	1.00	1.00	1.76e−5
14	8(S1)	15 362	1.00	1.00	0.88e−5
64	4(S)	199 433	1.00	1.00	1.25e−5
64	64	18 176	1.00	1.00	1.82e−5
64	128	8901	2.04	2.00	1.78e−5
64	256	5330	3.41	4.00	2.13e−5
128	8(S1)	319 592	1.00	1.00	2.00e−5
128	256	13 822	1.00	1.00	2.76e−5

6.4. Aneurysm

The fourth example is the incompressible flow in an aneurysm with a stent. The geometry, together with the stent, is shown in Figure 5(a,b). The shear stress obtained is shown in Figure 5(c). This case was run on an SGI ICE machine with 640 Intel processors running at 2.3 Ghz. Each node contained two processors and the nodes were grouped in to 80 blades or nodes of 8 cores at each node (2 Quad-core Xeon processors Hapertown E5440 series). The interconnection network was composed of two InfinibBand interconnects (10/20 Gb/s), with one interconnect dedicated to I/O and the other to MPI traffic. The operating system was Suse Linux Enterprise 10 tuned for SGI.

The code (FEFLO) was compiled using the Intel compiler, version 10.1.017, and the following compiler options were used:

The advective terms were integrated implicitly using an LU-SGS technique [59–61]. The pressure-Poisson equation was solved using a diagonally PCG algorithm with projective pressure prediction [20] and approximately 200 deflated groups. The (approximately 120 000) spheres that describe the stent are treated using the immersed body approach [22, 56, 62].

This case was run to steady state in 1000 timesteps. Different meshes and different numbers of processors were used to obtain the timings, which are summarized in Table I. The code was also run in shared memory mode on eight processors on the SGI ICE (denoted as 8(S)), as well as some Harpertown shared memory machines (denoted as 8(S1)). Note that the 'figure of merit' nproc*time/nelem/nstep remains largely unaffected by the grid size or by the number of processors, up to approximately 300 Kels per processor.

DOI: 10.1002/nme

7. CONCLUSIONS AND OUTLOOK

The deflated PCG technique has been extended to include:

- Embedded grid techniques (i.e. treatment of cut edges and deactive zones);
- Cases where VOF or LS schemes are required; and
- The implementation on parallel machines (in particular distributed memory machines).

Several examples are included that demonstrate a considerable reduction in the number of iterations and a remarkable insensitivity to the number of groups/regions chosen, and/or to the way the groups are formed.

Future developments will consider extensions to cases where embedded surfaces or immersed bodies are moving. For these, the techniques used to date to assign points to deflation groups are inadequate.

REFERENCES

1. Kim J, Moin P. Application of a fractional-step method to incompressible Navier–Stokes equations. *Journal of Computational Physics* 1985; **59**:308–323.
2. Bell JB, Colella P, Glaz H. A second-order projection method for the Navier–Stokes equations. *Journal of Computational Physics* 1989; **85**:257–283.
3. Löhner R. A fast finite element solver for incompressible flows. *AIAA-90-0398*, 1990.
4. Bell JB, Marcus DL. A second-order projection method for variable-density flows. *Journal of Computational Physics* 1992; **101**:2.
5. Martin D, Löhner R. An implicit linelet-based solver for incompressible flows. *AIAA-92-0668*, 1992.
6. Gunzburger MD, Nicolaides R. *Incompressible Computational Fluid Dynamics*: *Trends and Advances*. Cambridge University Press: Cambridge, 1993.
7. Alessandrini B, Delhommeau G. A multigrid velocity-pressure-free surface elevation fully coupled solver for calculation of turbulent incompressible flow around a hull. *Proceedings of the 21st Symposium on Naval Hydrodynamics*, Trondheim, Norway, June 1996.
8. Kallinderis Y, Chen A. An incompressible 3-D Navier–Stokes method with adaptive hybrid grids. *AIAA-96-0293*, 1996.
9. Ramamurti R, Löhner R. A parallel implicit incompressible flow solver using unstructured meshes. *Computers and Fluids* 1996; **5**:119–132.
10. Fischer PF. Projection techniques for iterative solution of $Ax=b$ with successive right-hand sides. *Computer Methods in Applied Mechanics and Engineering* 1998; **163**:193–204.
11. Löhner R, Yang C, Onate E, Idelssohn S. An unstructured grid-based, parallel free surface solver. *Applied Numerical Mathematics* 1999; **31**:271–293.
12. Codina R. Pressure stability in fractional step finite element methods for incompressible flows. *Journal of Computational Physics* 2001; **170**:112–140.
13. Takamura A, Zhu M, Vinteler D. Numerical simulation of pass-by maneuver using ALE technique. *JSAE Annual Conference* (Spring), Tokyo, May 2001.
14. Eaton E. Aero-acoustics in an automotive HVAC module. *American PAM User Conference*, Birmingham, Michigan, 24–25 October 2001.
15. Karbon KJ, Kumarasamy S. Computational aeroacoustics in automotive design, computational fluid and solid mechanics. *Proceedings of the First MIT Conference on Computational Fluid and Solid Mechanics*, Boston, June 2001; 871–875.
16. Li Y, Kamioka T, Nouzawa T, Nakamura T, Okada Y, Ichikawa N. Verification of aerodynamic noise simulation by modifying automobile front-pillar shape. *JSAE 20025351*, *JSAE Annual Conference*, Tokyo, July 2002.
17. Karbon KJ, Singh R. Simulation and design of automobile sunroof buffeting noise control. *8th AIAA-CEAS Aero-Acoustics Conference*, Brenckridge, June 2002.
18. Soto O, Löhner R, Camelli F. A linelet preconditioner for incompressible flows. *International Journal of Numerical Methods for Heat and Fluid Flow* 2003; **13**(1):133–147.
19. Löhner R. Multistage explicit advective prediction for projection-type incompressible flow solvers. *Journal of Computational Physics* 2004; **195**:143–152.
20. Löhner R. Projective prediction of pressure increments. *Communications in Numerical Methods in Engineering* 2005; **21**(4):201–207.
21. Löhner R, Yang C, Cebral JR, Camelli F, Soto O, Waltz J. Improving the speed and accuracy of projection-type incompressible flow solvers. *Computer Methods in Applied Mechanics and Engineering* 2006; **195**(23–24):3087–3109.
22. Löhner R, Appanaboyina S, Cebral J. Comparison of body-fitted, embedded and immersed solutions for low Reynolds-number flows. *International Journal for Numerical Methods in Fluids* 2008; **57**(1):13–30.

DOI: 10.1002/nme

23. Zienkiewicz OC. *The Finite Element Method*. McGraw-Hill: New York, 1991.
24. Saad Y. *Iterative Methods for Sparse Linear Systems*. PWS Publishing: Boston, 1996.
25. Waltz J, Löhner R. A grid coarsening algorithm for unstructured multigrid applications. *AIAA-00-0925*, 2000.
26. Waltz J. Unstructured multigrid methods. *Ph.D. Thesis*, George Mason University, 2000.
27. Chen NH. Multigrid methods for the incompressible Navier–Stokes problem on three-dimensional unstructured meshes. *Ph.D. Thesis*, George Mason University, 2004.
28. Aubry R, Mut F, Löhner R, Cebral JR. Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation. *Journal of Computational Physics* 2008; **227**(24):10196–10208.
29. McCormick SF. *Multigrid Methods*. SIAM: Philadelphia, PA, 1987.
30. Hestenes MR, Stiefel E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards* 1952; **49**:409–436.
31. van der Sluis A, van der Vorst HA. The rate of convergence of the conjugate gradients. *Numerische Mathematik* 1986; **48**:543–560.
32. Nicolaides RA. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis* 1987; **24**:355–365.
33. Mansfield L. On the use of deflation to improve the convergence of the conjugate gradient iteration. *Communications in Applied Numerical Methods* 1988; **4**:151–156.
34. Mansfield L. On the conjugate gradient solution of the schur complement system obtained from domain decomposition. *SIAM Journal on Numerical Analysis* 1990; **27**:1612–1620.
35. Mansfield L. Damped Jacobi preconditioning and coarse grid deflation for conjugate gradient iteration on parallel computers. *SIAM Journal on Numerical Analysis* 1991; **12**:1314–1323.
36. Mulyarchik SG, Bielawski SS, Popov AV. Efficient computational method for solving linear systems. *Journal of Computational Physics* 1994; **110**:201–211.
37. Morgan RB. A restarted GMRES method augmented with eigenvectors. *SIAM Journal on Matrix Analysis and Applications* 1995; **16**:1154–1171.
38. Bielawski SS, Mulyarchik SG, Popov AV. The construction of an algebraically reduced system for the acceleration of preconditioned conjugate gradients. *Journal of Computational and Applied Mathematics* 1996; **70**(2):189–200.
39. Erhel J, Burrage K, Pohl B. Restarted GMRES preconditioned by deflation. *Journal of Computational and Applied Mathematics* 1996; **69**:303–318.
40. Chapman A, Saad Y. Deflated and augmented Krylov subspace with eigenvectors. *Numerical Linear Algebra with Applications* 1997; **4**:43–66.
41. Erhel J, Guyomarc'h F. An augmented subspace conjugate gradient. *Research Report INRIA*, 1997.
42. Kotolina LY. Twofold deflation preconditioning of linear algebraic systems I. Theory. *Journal of Mathematical Sciences* 1998; **89**:6.
43. Vuik C, Segal A, Meijerink JA. An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients. *Journal of Computational Physics* 1999; **152**:385–403.
44. De Gersem H, Hameyer K. Convergence improvement of the conjugate gradient iterative method for finite element simulations. *COMPEL*: *The International Journal for Computation and Mathematics in Electrical and Electronic Engineering* 2000; **20**:90–97.
45. De Gersem H, Hameyer K. A deflated iterative solver for magnetostatic finite element models with large differences in permeability. *The European Physical Journal Applied Physics* 2000; **13**:45–49.
46. Frank J, Vuik C. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing* 2001; **23**:442–462.
47. Vuik C, Segal A, Meijerink JA, Wijma GT. The construction of projection vectors for a deflated ICCG method applied to problems with extreme contrasts in the coefficients. *Journal of Computational Physics* 2001; **172**:426–450.
48. Vermolen F, Vuik C, Segal A. Deflation in preconditioned conjugate gradient methods for finite element problems. In *Conjugate Gradient and Finite Element Methods*, Křižek M, Neittaanmäki P, Glowinski R, Korotov S (eds). Springer: Berlin, 2004; 103–129.
49. Saad Y, Yeung J, Erhel J, Guyomarc'h F. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing* 2000; **21**:1909–1926.
50. Kharchenko SA, Yu Yeremin A. Eigenvalue translation based preconditioners for the GMRES(k) method. *Numerical Linear Algebra with Applications* 1995; **2**:51–70.
51. Landsberg AM, Boris JP. The virtual cell embedding method: a simple approach for gridding complex geometries. *AIAA-97-1982*, 1997.
52. Aftosmis MJ, Berger MJ, Adomavicius G. A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries. *AIAA-00-0808*, 2000.
53. Baum JD, Mestreau E, Luo H, Löhner R, Pelessone D, Charman Ch. Modeling structural response to blast loading using a coupled CFD/CSD methodology. *Proceedings of the Design and Analysis of Protective Structures against Impact/Impulsive/Shock Loads* (*DAPSIL*), Tokyo, Japan, December 2003.
54. Camelli FF, Löhner R. VLES study of flow and dispersion patterns in heterogeneous urban areas. *AIAA-06-1419*, 2006.
55. Appanaboyina S, Mut F, Löhner R, Putman CM, Cebral JR. Computational fluid dynamics of stented intracranial aneurysms using adaptive embedded unstructured grids. *International Journal for Numerical Methods in Fluids* 2008; **57**(5):475–493.

DOI: 10.1002/nme

56. Löhner R, Cebral JR, Camelli FF, Appanaboyina S, Baum JD, Mestreau EL, Soto O. Adaptive embedded and immersed unstructured grid techniques. *Computer Methods in Applied Mechanics and Engineering* 2008; **197**:2173–2197.
57. Löhner R, Yang C, Onate E. Simulation of flows with violent free surface motion and moving objects using unstructured grids. *International Journal for Numerical and Methods in Fluids* 2007; **53**:1315–1338.
58. Löhner R. *Applied CFD Techniques*. Wiley: New York, 2008.
59. Luo H, Baum JD, Löhner R. A fast, matrix-free implicit method for compressible flows on unstructured grids. *Journal of Computational Physics* 1998; **146**:664–690.
60. Sharov D, Luo H, Baum JD, Löhner R. Implementation of unstructured grid GMRES+LU−SGS method on shared-memory, Cache-based parallel computers. *AIAA-00-0927*, 2000.
61. Luo H, Baum JD, Löhner R. Parallel unstructured grid GMRES+LU−SGS method for turbulent flows. *AIAA-03-0273*, 2003.
62. Cebral JR, Castro MA, Appanaboyina S, Putman CM, Millan D, Frangi AF. Efficient pipeline for image-based patient-specific analysis of cerebral aneurysm hemodynamics: technique and sensitivity. *IEEE Transactions in Medical Imaging* 2005; **24**(1):457–467.

DOI: 10.1002/nme