COARSE-GRAIN DEFLATION FOR PRECONDITIONED CONJUGATE GRADIENT SOLVERS: APPLICATION TO THE PRESSURE POISSON EQUATION

Fernando Mut, Romain Aubry, Guillaume Pierrot, Jean Roger, Juan Cebral and Rainald Löhner

CFD Center, Dept. of Computational and Data Science
M.S. 6A2, College of Science, George Mason University
Fairfax, VA 22030-4444, USA

Barcelona Supercomputing Center, Barcelona, Spain

ESI Group, Silic 303, 94588 Rungis Cedex, France

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. This procedure belongs to the family of hybrid methods (making use of both direct and iterative solvers) and the preconditioner obtained belongs to the class of optimal ones just as Multigrid and Domain Decomposition Methods. 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.

I. INTRODUCTION

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

\[
\rho v_t + \rho \nabla v + \nabla p = \nabla \mu \nabla v ,
\]

\[
\nabla \cdot v = 0 .
\]  

where \( \rho \), \( v \), \( p \), \( \mu \) 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:

- **Advective-Diffusive Prediction**: \( v^n \rightarrow v^* \)

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

- **Pressure Correction**: \( p^n \rightarrow p^{n+1} \)

\[
\nabla \cdot v^{n+1} = 0 ;
\]
\[ \frac{\rho v^{n+1} - v^*}{\Delta t} + \nabla (p^{n+1} - p^n) = 0 \quad ; \] (6)

which results in

\[ \nabla^2 (p^{n+1} - p^n) = \frac{\rho \nabla \cdot v^*}{\Delta t} \quad ; \] (7)

**Velocity Correction**: \( v^* \rightarrow v^{n+1} \)

\[ v^{n+1} = v^* - \Delta t \nabla (p^{n+1} - p^n) \quad . \] (8)

The solution of the so-called Pressure Poisson equation, given by Eqn.(7), which results in a discrete system of the form:

\[ A \cdot x = b \quad . \] (9)

is typically carried out with preconditioned conjugate gradient (PCG) solvers, 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;
- Linelet preconditioners for highly stretched (e.g. boundary layer) 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).

Several attempts have also been made to use multigrid solvers. 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-à-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.

## II. DEFLATED CONJUGATE GRADIENTS

Nicolaides’ seminal paper 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 impede convergence. As opposed to multigrid techniques, there is neither a need to define a smoother nor any tuning parameters of the cycles. However, no numerical experiments were reported in. The first experiments supporting the theoretical results are found in Mansfield, where the subdomain deflation approach of 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 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 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. 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 for an augmented conjugate gradient, where the Krylov subspace generated by a previous system is recycled for further solves in subsequent systems, and in, where the approximation of the eigenvectors proposed in is used to augment the space of the subsequent system to deflate the lowest eigenvalues. Compared to the approach of 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 the projector relying on the deflation subspace is explicitly built from the structure of the coefficient matrix of the system to be solved. Kotolina demonstrated bound estimates for the effective condition number.
of the deflated system in terms of the gap between the projection space and the set of first $k$ eigenvectors. In the same spirit Vuik and coworkers\textsuperscript{15, 45–47} proved that if grid refinement is performed, keeping the subdomain grid resolution fixed, the condition number is insensitive to the grid size, which establishes the optimality of the preconditioning. Furthermore they applied the deflated technique in various contexts, including problems characterized by layers of large contrast in the porosity\textsuperscript{46, 47}, where the approximated eigenvectors verify the partial differential equation on each subdomain. In\textsuperscript{15} 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\textsuperscript{45}. Deflation is applied to magnetostatic problems in\textsuperscript{9, 10}, where the approximate eigenvectors used to the deflation subspace mimic the flux patterns commonly sketched by engineers intuitively.

A. Theoretical Considerations

The conjugate gradient\textsuperscript{17} is the method of choice for solving SPD systems, such as Eqn.(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 $r$ as:

$$r = b - A \cdot x,$$

(10)

the basic iterative step is given by:

$$\Delta x_k = \alpha_k (r_{k-1} + e_{k-1} \Delta x_{k-1}) = \alpha_k v_k,$$

(11)

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

$$\Delta x_{k-1} \cdot A \cdot \Delta x_k = 0.$$

(12)

This yields

$$e_{k-1} = - \frac{r_{k-1} \cdot A \cdot \Delta x_{k-1}}{\Delta x_{k-1} \cdot A \cdot \Delta x_{k-1}},$$

(13)

which may be simplified for linear systems by observing that:

$$r_{k-1} - r_{k-2} = -A \cdot \Delta x_{k-1},$$

(14)

and, hence,

$$e_{k-1} = - \frac{r_{k-1} \cdot (r_{k-1} - r_{k-2})}{\Delta x_{k-1} \cdot (r_{k-1} - r_{k-2})}. $$

(15)

The parameter $\alpha_k$ is obtained by forcing

$$A \cdot (x_{k-1} + \Delta x_k) = f,$$

in a ‘matrix weighted’ sense by multiplication with $\Delta x_k$:

$$\Delta x_k \cdot A \cdot \Delta x_k = \Delta x_k \cdot r_{k-1},$$

(17)

yielding

$$\alpha_k = \frac{v_k \cdot r_{k-1}}{v_k \cdot A \cdot v_k}.$$

(18)

The CG algorithm generates a sequence $x_1, ..., x_i$ such that:

$$x_i \in x_0 + K_i,$$

(19)

where $K_i = span\{r_0, A \cdot r_0, ..., A_{i-1} \cdot r_0\}$ is the Krylov subspace of dimension $i$ generated by the initial residual $r_0 = b - A \cdot x_0$. At each step, the approximation $x_i$ verifies:
\|x_{ex} - x_i\|_A = \min_{x \in x_0 + K_i} \|x_{ex} - x\|_A , \tag{20}

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

\[
\|x_{ex} - x_k\|_A = \|e_k\|_A \leq 2 \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|e_0\|_A , \tag{21}
\]

where \(\kappa\) is the condition number of matrix \(A\). However, as described in \(^{44}\), 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 idea behind 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. Once a projection space has been defined, the intention is to solve exactly (i.e. by a direct solver) the low frequency error of the coarse system

\[
(W^T \cdot A \cdot W) \cdot x_{coarse} = W^T \cdot b_{fine} ,
\]

whereas the high frequency error is reduced iteratively within the PCG. As the optimal solution in the coarse space is already known, the important point is to keep the residual orthogonal to this coarse space during subsequent iterations. At the algorithmic level this may be viewed as adding one more (low-dimensional) search direction \(W \cdot d\) for the increment:

\[
\Delta x_k = \alpha_k (r_{k-1} + e_{k-1} \Delta x_{k-1} - W \cdot d_k) = \alpha_k v_k . \tag{22}
\]

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

\[
\Delta x_{k-1} \cdot A \cdot (r_{k-1} + e_{k-1} \Delta x_{k-1} - W \cdot d_k) = 0 . \tag{23}
\]

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

\[
W^T \cdot A \cdot W \cdot d_k = W^T \cdot A \cdot r_{k-1} . \tag{24}
\]

As \(W^T \cdot A \cdot \Delta x_{k-1} = 0\), \(e_{k-1}\) is obtained as before from:

\[
e_{k-1} = \frac{r_{k-1} \cdot A \cdot \Delta x_{k-1}}{\Delta x_{k-1} \cdot A \cdot \Delta x_{k-1}} , \tag{25}
\]

\(v_k\) from

\[
v_k = r_{k-1} + e_{k-1} \Delta x_{k-1} - W \cdot d_k , \tag{26}
\]

and \(\alpha_k\) from Eqn.(18).
B. Algorithmic Implementation

An optimal algorithmic implementation is given in\(^4^1\):
- Define a preconditioning matrix: \(M\)
- Define: \(\hat{A} = W^T \cdot A \cdot W\)
- Start: Given \(x_{-1} \cdot r_{-1} = b - A \cdot x_{-1}\)
  \(\bar{A} \cdot d_0 = W^T \cdot r_{-1}\)
  \(x_0 = x_{-1} + W \cdot d_0\)
  \(r_0 = b - A \cdot x_0\)
- Compute: \(M \cdot z_0 = r_0\)
- Solve: \(\hat{A} \cdot d = W^T \cdot A \cdot z_0\)
- Set: \(p_0 = -W \cdot d + z_0\)
- Do until convergence:
  \(\alpha_j = (r_j \cdot z_j)/(p_j \cdot A \cdot p_j)\)
  \(x_{j+1} = x_j + \alpha_j p_j\)
  \(r_{j+1} = r_j - \alpha_j \bar{A} \cdot p_j\)
  \(M \cdot z_{j+1} = r_{j+1}\)
  \(\beta_j = (r_{j+1} \cdot z_{j+1})/(r_j \cdot z_j)\)
  \(\hat{A} \cdot d_j = W^T \cdot A \cdot z_{j+1}\)
  \(p_{j+1} = z_{j+1} + \beta_j p_j - W \cdot d_j\)

As the dimensionality of \(d\) is low, the solution/inversion of \(\hat{A}\) is carried out using a skyline solver. The extra storage requirements for the DPCG are very modest: both \(W\) and \(W^T \cdot A\) 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, where storage increases proportionally to the number of eigenvalues considered.

C. Definition of Projection Space

The DCG technique requires the definition of a mapping \(W\) from the lower-dimensional basis \(d\) to the vector of unknowns \(x\). The simplest way of defining this mapping \(W\) for a mesh-based system of equations is by agglomerating the nodes of the mesh into subdomains (or ‘coarse grains’) and then define a polynomial reconstruction over each of them. The easiest way is to adopt a constant reconstruction: local domains (subdomains) or groups of points are then assigned to a variable \(d_i\). The entries in \(W\) 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 \(x\) 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. 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 pressure field 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 neighbouring groups in a post-processing step.

III. 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\textsuperscript{11,19–21,26,43}. In all cases diagonal preconditioning was used in the isotropic mesh regions, while linelet preconditioning\textsuperscript{34,42} 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 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: $v = 1$
- 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 layers algorithm, starting from the exit (i.e. prescribed pressure) plane. Figures 1a-d show the surface mesh, region boundaries for 15 groups, pressure and absolute value of the velocity for $l = 20$. Figure 1e 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\textsuperscript{29} was used. Note the dramatic decrease in the number of iterations achieved by the deflated PCG solver. This decrease may also be seen in Figure 1f, 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 Figure 1g. Note that for the case $l = 80$ the same solver with deflated PCG runs seven times faster (!).

![Figures 1a,b Pipe: Surface Mesh and Deflation Groups for $L = 20$](image)
Figures 1c,d  Pipe: Pressure and $\text{Abs(Velocity)}$ for Plane $z = 0$

Figure 1e  Pipe: Number of Iterations Required
Figure 1f  Pipe: Number of Iterations Required

Figure 1g  Pipe: CPU Required for 100 Timesteps
3.2 NACA0012: The second example considered is the classic NACA0012 wing at an angle of attack of $\alpha = 5^\circ$. This is a steady, inviscid case (Euler). Figures 2a,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 Figure 2c. These were input by hand, and the regions for the deflated PCG were grown from them. Figure 2d 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.

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 $Re = 190$. 
Figures 3a-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 exercised and compared to one another. The regions of a typical run with deflated PCG are shown in Figure 3d. The iterations required per timestep are displayed in Figure 3e. One can observe that the deflated PCG requires substantially less iterations, and is rather insensitive to the number of domains chosen.
3.4 Cerebral Artery Tree: The fourth example is a patient-specific vascular model of the circle of Willis together with the arterial network of the brain constructed from an MRA of a normal subject\textsuperscript{5}. The blood was considered as a Newtonian fluid, which is modeled by the \textit{unsteady} incompressible Navier-Stokes equations. The volume mesh had 3.8 Mpts and 19 Mels. The groups were constructed using both alternatives. For the seedpoints option, 512 and 978 equally spaced points were generated along the skeletons of the model. For the advancing front option, 250, 500, 1000 and 2000 groups were generated. Figure 4a shows the overall domain, as well as the group boundaries for the case with 1000 groups. The simulation was performed using implicit time-stepping, solving a pseudo-steady problem at each timestep\textsuperscript{30}. Within each pseudo-timestep the advective terms were integrated implicitly using 5 LU-SGS passes (local Courant number $C = 5.0$), followed by the pressure projection. The timestep was set to $\Delta t = 0.01$ sec. The material properties of the fluid were taken to be $\rho = 1.0g/cm^3$ and $\mu = 0.04Poise$. The time-dependent velocity profile was prescribed at the 62 outflows, while pressure was prescribed at the 3 inflows (bottom part). The velocity profile was computed using a Womersley model with measured inflow rates of a normal subject\textsuperscript{5}. No-slip boundary conditions were applied at the vessel walls. This case was run for 200 timesteps (two cardiac cycles). Figures 4b,c depict the pressure and wall shear stress at time $T = 1.4$ sec (second cardiac cycle, just after the outflow rate peak). The differences in pressure between the left and the right side of the model are due to a high bend in the vessel which produces a sudden pressure drop in that region. Figures 4d,e show a close-up view of this feature. Figure 4f shows the average number of iterations for the Pressure-Poisson solver at each timestep. The seedpoint option is represented by uppercase, while the advancing front alternative is represented by lowercase. Note that the number of iterations follows the outflow profile, having the highest
values where the outflow rate is maximum. The deep peaks found in the non-deflated case are due to a good approximation of the initial guess, which is based in the solution of the previous timestep. This behavior is accentuated around $T = 1 \text{ sec}$, at the end of the cardiac cycle, when the outflow rate changes very slowly. Figure 4g shows the average number of iterations for the deflated cases only. Note that as the number of groups increases, the number of iterations always decreases. Moreover, the group selection method does not seem to produce a significant change. Figures h,i show the CPU-time required at each timestep for all the cases and for the deflated cases only. Note that again, as the number of groups increases, the CPU-time per timestep always decreases. This shows that within this range ($250 \sim 2000$ groups) the extra-cost of solving a direct system $O(ngroup)$ each iteration does not contribute significantly to the total time of the iteration. Figure 4j shows the timing for all the cases that were run. The best speed-up obtained is about 3.6, with 2000 groups.

Figure 4a  Brain: Patient-Specific Model Circle of Willis With Arterial Network

Figures 4b,c  Brain: Wall Pressure and Shear Stress at $T = 1.4 \text{ sec}$
Figures 4d,e  Brain: Wall Pressure and Shear Stress at $T = 1.4$ sec (Closeup)

Figures 4f,g  Brain: Average Number of Iterations

Figures 4h,i  Brain: Times Required
3.5 **Generic Pickup Truck**: The fifth example considers flow past a generic pickup truck. The geometry is displayed in Figure 5a. The surface of the mesh, which had approximately 9 Mels, is shown in Figures 5b,c. Note the presence of several layers of elements in the boundary layer region. The number of groups chosen was of the order of 30,000. The shape of the domains may be inferred from the surface colouring shown in Figure 5d. For a tolerance of pressure residual of $10^{-6}$, the usual and deflated PCG solvers took an average of 1260 and 90 iterations per timestep respectively, a reduction of more than 1:14. Figure 5e shows pressures in the early stages of the run.
Figure 5b  Pickup Truck: Surface Mesh

Figure 5c  Pickup Truck: Surface Mesh (Zoom)
Figure 5d  Pickup Truck: Subdomains for Deflation Groups

Figure 5e  Pickup Truck: Pressure
IV. CONCLUSIONS AND OUTLOOK

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 and ad hoc preconditioning of the remaining part of the system accounting for the high frequency error.

References


