Technical Report

Optimization of the deflated Conjugate Gradient algorithm for the solving of elliptic equations on massively parallel machines

Mathias Malandain* and Nicolas Maheu and Vincent Moureau
Corresponding author: *mathias.malandain@coria.fr*

October 1, 2012

Contents

Abstract

The discretization of Partial Differential Equations often leads to the need of solving large symmetric linear systems. In the case of the Navier-Stokes equations for incompressible flows, solving the elliptic pressure Poisson equation can represent the most important part of the computational time required for the massively parallel simulation of the flow. The need for efficiency that this issue induces is completed with a need for stability, in particular when dealing with unstructured meshes. Here, a stable and efficient variant of the Deflated Preconditioned Conjugate Gradient (DPCG) solver is first presented. This two-level method uses an arbitrary coarse grid to reduce the computational cost of the solving. However, in the massively parallel implementation of this technique for very large linear systems, the coarse grids generated can count up to millions of cells, which makes direct solvings on the coarse level impossible. The solving on the coarse grid, performed with a Preconditioned Conjugate Gradient (PCG) solver for this reason, may involve a large number of communications, which reduces dramatically the performances on massively parallel machines. To this effect, two methods developed in order to reduce the number of iterations on the coarse level are introduced, that is the creation of improved initial guesses and the adaptation of the convergence criterion. The design of these methods make them easy to implement in any already existing DPCG solver. The structural requirements for an efficient massively parallel unstructured solver and the implementation of this solver are described. The novel DPCG method is assessed for a wide range of applications through different 2D and 3D test cases involving turbulence, heat transfers and two-phase flows, with grids ranging from 3.2 million to 17.8 billion elements. Numerical results show a two- to twelve-fold reduction of the number of iterations on the coarse level, which implies a reduction of the computational time of the Poisson solver up to 71 % and a global reduction of the proportion of communication times up to 53 %. As a result, the weak scaling of the LES solver is shown to be clearly improved for massively parallel uses.

Chapter 1

Introduction

1.1 State-of-the-art and motivation

According to J. R. Shewchuk [37], the Conjugate Directions method was introduced by E. Schmidt in 1908 [36], and then independently reinvented by Fox *et al.* in 1948 [11]. A few years later, M. R. Hestenes and E. Stiefel created the Conjugate Gradient method [15, 39], then published in 1952 a joint article considered a seminal reference on CG [16]. Popularized by J. K. Reid [32] as a method for solving large sparse linear systems, it has also been generalized for nonlinear systems in the Sixties [10], and the investigation of the solving of non-symmetric linear systems thanks to methods derived from CG has been carried out from the mid-Seventies, with the BiCG method [9] and variants such as BiCG-STAB and BiCGStab(L) [43, 38].

The idea of preconditioning the CG method dates back to the late Fifties, that is very few years after its creation, with for instance the work of M. Engell *et al.* [5], but the deflation technique came up three decades later, in 1987, thanks to R.A. Nicolaides [29]. This method has since been developed and frequently applied, among others, to linear systems arising from the discretization of Navier-Stokes equations: see for instance [35] about the creation of a Deflated CG method; [12, 42] about the domain-decomposition deflation method; [1, 27] for applications to physical problems. Several comparisons have been made between solvers [28, 21], from which it appears that the DPCG is among the most stable and efficient numerical methods for symmetric systems; moreover, its easy implementation for massively parallel solving on unstructured meshes is a considerable asset. Nevertheless, several stability issues may occur when the method is used for the simulation of fluid flows in complex geometries. These issues may be of particular importance when dealing with Poisson equations in two-phase flows, that feature variable coefficient matrices with strong variations in its diagonal.

In an article written in 2009 by Tang *et al.* [40], the A-DEF2 variant of the Deflated CG is created that is proved fast and robust, even when compared to domain decomposition and multigrid methods, which is why this method is studied here. Some improvements were still to be used: as a matter of fact, reducing the high number of iterations required on the coarse level can become very important, as these iterations will add irreducible communication times to the overall time spent in the solver.

Two novel methods are introduced here, one of them consisting in the computation of improved initial guesses, the other one being an adaptation of the convergence criterion on the coarse grid at every iteration of the solver on the fine grid. For assessment purposes, they have been implemented in the A-DEF2 solver used for the pressure Poisson equation in an unstructured LES solver for incompressible flows. The novel solver created thereby has been tested for different settings, from two-dimensional test cases to real turbulent flow configurations.

This paper is organized as follows. After a quick introduction on the PCG method, the DPCG method and its variant A-DEF2, the creation of initial guesses for the successive systems on the coarse grid based on the former computed solutions is discussed in Chapter 2, and Chapter 3 addresses the

possibility of adjusting the convergence criterion on the coarse grid. Then, Chapter 4 presents the massively parallel implementation of these techniques, thanks to which the numerical results shown in Chapter 5 have been obtained on grids up to billions of cells. Finally, Chapter 6 concludes this report by summing up the developed methods and the resulting gains on computational times.

1.2 Notations and definitions

In what follows, a linear system $Ax = b$ is considered that has to be solved, with $A \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^n$. It is assumed that A is positive semi-definite, that is $\forall x \in \mathbb{R}^n, x^T Ax \geq 0$.

The **A-dot product** is defined as the binary operation $\langle \cdot, \cdot \rangle_A$ such that

$$\forall x, y \in \mathbb{R}^n, \langle x, y \rangle_A = x^T Ay \,. \tag{1.1}$$

Two vectors x and y are then **A-orthogonal** if and only if $\langle x, y \rangle_A = 0$.

One can derive a norm $\| \cdot \|_A$ from this dot product, thus creating the **A-norm** or **energy norm**

$$\forall x \in \mathbb{R}^n, \ \|x\|_A = \sqrt{x^T Ax} \,. \tag{1.2}$$

Finally, the standard infinite norm is defined as

$$\forall x \in \mathbb{R}^n, \ \|x\|_\infty = \max_{k\in\{1;\ldots;n\}} |x_k| \,. \tag{1.3}$$

1.3 Principles and practices of the Conjugate Gradient method

As for every iterative method for the solving of linear systems, the Conjugate Gradient (CG) method consists on a particular way of creating a series of values $\{x_k\}_{k\in\mathbb{N}}$ that converges to $x = A^{-1}b$. The following notes are strongly inspired by [37].

At iteration k, the algorithm computes a direction p_k in which the residual r_k will be decreased, and a given multiplier α_{k+1} for p_k; then

$$x_{k+1} = x_k + \alpha_{k+1} p_k \,. \tag{1.4}$$

The new error $e_{k+1} = x - x_{k+1}$ has to be A-orthogonal to p_k; one will therefore be able to build an A-orthogonal basis $\{p_1; p_2; \ldots; p_n\}$ of $\mathbb{R}^n$.

The choice of p_k is quite simple, as it is the result of the A-orthogonalization of r_k with respect to the previous directions $p_1, p_2, \ldots, p_{k-1}$. It is quite easy to prove that r_k is already orthogonal to p_i for $i \in \{1; 2; \ldots; k-2\}$, so that it is only necessary to find the coefficient β_k such that $p_k = r_k + \beta_k p_{k-1}$ is A-orthogonal to p_{k-1}. Then, as $e_{k+1} = e_k - \alpha_{k+1} p_k$, the A-orthogonality condition and the fact that $Ae_k = r_k$ induce

$$\alpha_{k+1} = \frac{p_k^T r_k}{p_k^T A p_k} \text{ and } \beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}} \,. \tag{1.5}$$

Using a preconditioning matrix K^{-1}, the solving of $Ax = b$ becomes equivalent to the solving of $K^{-1}Ax = K^{-1}b$, thus creating the PCG method [44] given by Algorithm 1.

Algorithm 1 Preconditioned Conjugate Gradient algorithm

Require: A, b, x_0
  $p_0 = r_0 = b - Ax_0$
  $w_0 = K^{-1} r_0$
  for $k = 0, 1, 2 \ldots$ until required convergence **do**
    $\alpha_{k+1} = \frac{r_k^T w_k}{p_k^T A p_k}$
    $x_{k+1} = x_k + \alpha_{k+1} p_k$
    $r_{k+1} = r_k - \alpha_{k+1} A p_k$
    $w_{k+1} = K^{-1} r_{k+1}$
    $\beta_{k+1} = \frac{r_{k+1}^T w_{k+1}}{r_k^T w_k}$
    $p_{k+1} = w_{k+1} + \beta_{k+1} p_k$
  end for

1.4 Deflation of CG: the A-DEF2 algorithm

Numerous variants of preconditioned, deflated and balancing algorithms have been introduced and compared by Tang *et al.* in [40], built thanks to strict combinations of the preconditioners corresponding to each method. The A-DEF2 variant of the DPCG, a novel combination of the deflation and preconditioning techniques, has been chosen here due to its stability and performance, shown in the aforementioned article. It is applied here by combining a standard geometrical deflation with non-overlapping projection vectors, as introduced by Vermolen *et al.* in [42], and a preconditioning by the inverse of the diagonal. These choices are practical, as they reduce the computational time required for initializing the solver. The A-DEF2 algorithm being implemented is described by Algorithm 2, with W the deflation matrix and $M = diag(A)$. In what follows, the right-hand side of the solver on the coarse grid $W^T(AM^{-1} - I)r_i$ is called b_i^C.

Algorithm 2 The A-DEF2 algorithm, to solve $Ax = b$ using the preconditioner M^{-1} and the deflation matrix W

Require: A, b, M^{-1}, W
  $\hat{A} \leftarrow W^T A W$
  Solve $\hat{A} d_{-1} = W^T b$
  $x_0 = W d_{-1}$
  $r_0 = b - Ax_0$
  Solve $\hat{A} d_0 = W^T(AM^{-1} - I) r_0$
  $w_0 = M^{-1} r_0 - W d_0$
  $p_0 = w_0$
  for $k = 0, 1 \ldots$ until required convergence **do**
    $\alpha_{k+1} = \frac{r_k^T w_k}{p_k^T A p_k}$
    $x_{k+1} = x_k + \alpha_{k+1} p_k$
    $r_{k+1} = r_k - \alpha_{k+1} A p_k$
    Solve $\hat{A} d_{k+1} = W^T(AM^{-1} - I) r_{k+1}$
    $w_{k+1} = M^{-1} r_{k+1} - W d_{k+1}$
    $\beta_{k+1} = \frac{r_{k+1}^T w_{k+1}}{r_k^T w_k}$
    $p_{k+1} = w_{k+1} + \beta_{k+1} p_k$
  end for

The coarse grid is created by conglomeration of control volumes from the fine grid, such that all

groups contain approximatively the same number of control volumes, and each group is located on a single processor during parallel solving; the creation of these groups of control volumes is detailed in Chapter 4. Thus, the deflation matrix W counts as many columns as there are groups of control volumes in the grid, and as non-overlapping deflation vectors are created, each of these columns is actually the indicator vector of a group.

The stopping condition is reached when

$$\|r_k\|_\infty \leq \gamma \|b\|_\infty , \tag{1.6}$$

and γ is hereafter called the convergence criterion of the solver on the fine grid. The convergence criterion on the coarse grid usually takes the same value.

1.5 Massively parallel implementation of A-DEF2

The issues that may arise for massively parallel solvings are not addressed by Tang *et al.*, who just mention that the linear systems on the coarse grid are usually solved directly, thanks to a Cholesky decomposition for instance. As the typical number of control volumes per group is between a hundred and a few thousands, the single-processor solving of a linear system on the coarse grid is out of reach for grids counting up to billions of cells, as the corresponding coarse grid can still contain tens or hundreds of millions of groups. For this reason, a PCG solver has to be implemented on the coarse level, which makes a new bottleneck arise.

Although the solving on the coarse grid may not seem very time-consuming, it is in fact an important part of the computational effort for massively parallel solvings, especially because of the extra communications that are needed between the processors; as a matter of fact, pieces of data have to be communicated for every calculation of a dot-product and every evaluation of the residual, so that, even after a clever rearrangement of the algorithm inspired by [2], at least one MPI communication is needed at every iteration on the coarse grid. When the solver deals with large realistic simulations, computational times are still lower than without deflation, because of the important reduction of the number of iterations of the fine grid solving; nevertheless, it seems that going on reducing the computational times requires to focus on these successive systems on the coarse grid.

As an example, one can run the so-called **3D_Cylinder** test case described in Section 5.1 with or without the deflation technique. The Preconditioned CG and the A-DEF2 variant of the DPCG, given by Algorithms 1 and 2 in the present report, have performed the first time steps of the simulation of the isothermal flow in this configuration, on a mesh counting 3.9 million cells, with a required convergence criterion of 10^{-10} on each level. These runs, and every other run studied in this report, have been run with the unstructured LES solver YALES2 developed by Vincent Moureau and coworkers at the CORIA laboratory [25]. A qualitatively representative illustration of the behaviour of both solvers on this case is given by the results on one particular temporal step, so that the results given in what follows focus on the fifth temporal step.

- Without the deflation technique, communications are responsible for 6.57 % of the global computational time, 81.57 % of which is spent in the PCG solver for the pressure Poisson equation.
- With the deflation technique, communications are responsible for 8.22 % of the global computational time, that is 1.25 times as much as in the first case, even though the Pressure solving is quite twice as fast, that is only 54.45 % of the computational time. The global computational time for this iteration, and all the other ones, is still divided by a factor about 2.5, as the thirteen-fold reduction of the number of iterations on the fine grid, from 726 to 54, more than compensates the much higher cost of communications of the pressure Poisson solver, but this communication cost appears as a bottleneck for the parallel simulation of turbulent flows.

From these results, it becomes obvious that the improvement of a DPCG solver built for massively parallel uses has to focus on the number of iterations of the PCG solver on the coarse grid. With this goal in mind, the ideas of creating initial guesses on the coarse grid (Chapter 2) and adapting the convergence criterion of every linear system on the coarse level (Chapter 3) are expected to reduce the number of iterations on the coarse grid "from both sides", as illustrated by Figure 1.1. Even though the CG method is not a smoother in the strict sense, because some components of the solution vector can actually be increased from an iteration to the following one, one can legitimately expect a quite regular decrease in the order of magnitude of the residual; when applying both techniques simultaneously, the number of orders of magnitude by which the residual is required to drop should be consequently decreased, so that one can expect the number of iterations to decrease in a similar way.

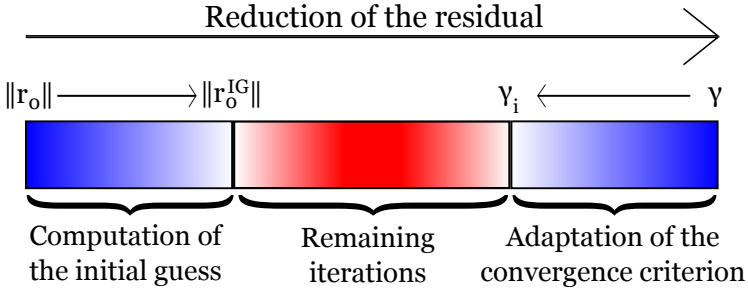

Figure 1.1: *Schematic representation of the expected combined effects of the initial guesses and convergence criterion adaptation techniques: the former aims at replacing the first residual r_0 with a residual r_0^{IG} whose norm is strictly inferior, and the latter raises the default convergence criterion γ, whose value comes from the convergence criterion on the fine grid, to a value γ_i depending on the behaviour of the solver. The quasi-logarithmic decrease of the norm of the residual by the CG method will still be observed, but it will have to be decreased by fewer orders of magnitude, so that less iterations will be required.*

Chapter 2

Improved initial guesses for the DPCG method

2.1 Principle and motivation

The motivation behind this work being explained, the means of improving the convergence of the coarse grid solver are quite limited. The coarse grid has been made by conglomeration of control volumes from the fine grid once and for all, in a seemingly optimal way, by external tools detailed in Chapter 4. Moreover, the numerical method chosen on the coarse level, that is the Conjugate Gradient method, is the most efficient in parallel for the typical sizes of the projected problem. An alternative way of adapting the convergence criterion on the coarse grid, but still keeping it low enough to keep a good overall convergence, is addressed in Chapter 3; however, once the sought precision on the coarse grid is set, the initial guess of the system to be solved seems to be the most important choice left in order to reduce the number of iterations of the successive linear systems on the coarse grid.

A recycling technique has already been developed by Paul F. Fischer in [8], that was not only reused several times for linear systems with multiple right-hand sides [23], but also generalized to the case where there are also slight changes in the matrix of the system [30] and applied to a variety of scientific fields, from medical imaging [3] to model order reduction [7].

The main change here is that the method introduced hereafter does not require the preliminary normalization of the vectors used for the computation of the initial guess; it is not even needed to check the linear independence of these vectors, and the important computational errors happening when the vectors are numerically close to linearly dependent are avoided, as detailed in Section 2.4.

Two advantages are induced by this particular method. First, the costs of the computation of the initial guesses are reduced: as a matter of fact, the orthogonalization step is no longer required, which drops the number of $\hat{A}$-products and communications needed, and the major part of the values computed for the creation of an initial guess are used again for the next one (see Section 2.4). Second, this direct algorithm avoids the propagation and amplification of rounding errors, and therefore the optimality of the initial guess for the given parameters is guaranteed.

2.2 Basic algorithm for the computation of an initial guess

The general idea is creating an initial guess thanks to n_{ig} given vectors. The initial guess for the solving of the system $\hat{A}d_i = b_i^C$ on the coarse level can then be written as $\sum_{j=1}^{n_{ig}} \nu_j^i u_j^i$, with $\{u_j^i\}_{j=1}^{n_{ig}}$ an independent family of vectors, and $\forall j \in \{1, \dots, n_{ig}\}, \nu_j^i \neq 0$ if the vectors are linearly independent. As $d_i = \sum_{j=1}^{n_{ig}} \nu_j^i u_j^i + \tilde{d}_i$, it becomes obvious that the coefficients ν_j^i have to be chosen such that $\tilde{d}_i$ is orthogonal to all vectors u_j^i for j ranging from 1 to n_{ig}.

In other words, $\tilde{d_i}$ can be obtained by orthogonalizing d_i with respect to the set $\{u_j^i\}_{j=1}^{n_ig}$; and as the entire CG algorithm, that is used on both levels, is built on dot products defined thanks to the matrix of the system to be solved, the projected systems will be solved by an algorithm that uses properties derived from the $\hat{A}$-dot product. Therefore, the coefficients ν_j^i have to be chosen such that

$$\forall j \in \{1; \dots; n_{ig}\}, \langle d_i - \sum_{k=1}^{n_{ig}} \nu_k^i u_k^i, u_j^i \rangle_{\hat{A}} = 0\,, \tag{2.1}$$

that is

$$\forall j \in \{1; \dots; n_{ig}\}, \sum_{k=1}^{n_{ig}} \nu_k^i (u_k^i)^T \hat{A} u_j^i = (u_j^i)^T \hat{A} d_i = (u_j^i)^T b_i^C\,. \tag{2.2}$$

This can be written as the linear system $C^i \nu^i = s^i$ with

$$C^i = \left((u_j^i)^T \hat{A} u_k^i\right)_{j,k=1}^{n_{ig}} \quad ; \quad \nu^i = \left(\nu_j^i\right)_{j=1}^{n_{ig}} \quad ; \quad s^i = \left((u_j^i)^T b_i^C\right)_{j=1}^{n_{ig}}\,, \tag{2.3}$$

which is the general form of the linear system solved by YALES2 for the computation of the initial guesses. In order to avoid a high number of distinct communications, all the dot products required for the computation of an initial guess are computed at once, and their values are distributed to all processors, so that every processor can solve the system $C^i \nu^i = s^i$ independently.

An equivalent way of obtaining this system is by choosing to minimize the square of the $\hat{A}$-distance between the initial guess $\sum_{j=1}^{n_{ig}} \nu_j^i u_j^i$ and the solution d_i to the linear system $\hat{A} d_i = b_i^C$, that is

$$\begin{aligned} \left\|\tilde{d_i}\right\|_{\hat{A}}^2 &= \left(d_i - \sum_{j=1}^{n_{ig}} \nu_j^i u_j^i\right)^T \hat{A} \left(d_i - \sum_{j=1}^{n_{ig}} \nu_j^i u_j^i\right) \\ &= d_i^T \hat{A} d_i - 2 \sum_{j=1}^{n_{ig}} \nu_j^i \left((u_j^i)^T \hat{A} d_i\right) + \sum_{j=1}^{n_{ig}} \sum_{k=1}^{n_{ig}} \nu_j^i \nu_k^i \left((u_j^i)^T \hat{A} u_k^i\right) \end{aligned} \tag{2.4}$$

from the definition of the $\hat{A}$-norm given in Equation (1.2). A set of values $\left\{\nu_j^i\right\}_{j=1}^{n_{ig}}$ that minimises $\left\|\tilde{d_i}\right\|_{\hat{A}}$ is such that

$$\forall j \in \{1; \dots; n_{ig}\}, \frac{\partial \left\|\tilde{d_i}\right\|_{\hat{A}}}{\partial \nu_j^i} = 0\,, \tag{2.5}$$

that is, from the direct derivation of Equation (2.4),

$$\forall j \in \{1; \dots; n_{ig}\}, -2(u_j^i)^T \hat{A} d_i + 2 \sum_{k=1}^{n_{ig}} \nu_k^i \left((u_j^i)^T \hat{A} u_k^i\right) = 0\,. \tag{2.6}$$

These equations result in the linear system given by Equation (2.3).

If one wants the time of the solving of this system to keep reasonable with respect to the time required to solve $\hat{A} d_i = b_i^C$, n_{ig} will have to be very small towards the size of the matrix $\hat{A}$. Whatever solver is chosen for this system, the time required for this computation will anyway be negligible towards the global computation time, and one can expect that the solving time gained thanks to the initial guess will be much greater than the time required to compute it.

2.3 Computation of the initial guess from the previous solutions d_k

One can guess that the solutions given by the successive systems on the coarse grid may be quite similar to each other in direction, or at least that the solution d_{i+1} of the system $\hat{A}d_{i+1} = b_{i+1}^C$ will not be $\hat{A}$-orthogonal to the solution d_i of $\hat{A}d_i = b_i^C$; that is, d_{i+1} can be written as $\nu_i d_i + \tilde{d}_{i+1}$, with $\tilde{d}_{i+1}$ $\hat{A}$-orthogonal to d_i and $\nu_i \neq 0$. Pushing this idea further results in using the set of vectors $\{d_{i-j}\}_{j=1}^{n_{ig}}$ for the computation of the initial guess of d_i.

Until iteration $n_{ig} - 1$ on the fine grid, that is until the (n_{ig})-th call of the solver on the coarse grid, less than n_{ig} former solutions are available; there are only $i + 1$ vectors d_k already computed, with k ranging from -1 to $i - 1$. The number of vectors used for the construction of the initial guess of iteration i is therefore

$$n_{vect} = \min\{i + 1, n_{ig}\} \,. \tag{2.7}$$

The matrices used then become

$$C^i = \left((u_j^i)^T b_k^C\right)_{j,k=1}^{n_{vect}} \quad ; \quad \nu^i = \left(\nu_j^i\right)_{j=1}^{n_{vect}} \quad ; \quad s^i = \left((u_j^i)^T b_i^C\right)_{j=1}^{n_{vect}} \tag{2.8}$$

where

$$\forall j \in \{1; \ldots; n_{vect}\} \ , \ u_j^i = d_{i+j-n_{vect}-1} \,. \tag{2.9}$$

Equation (2.9) is merely a formalization of the fact that the vectors d_k are used with k ranging from -1 to $i - 1$ if $i < n_{ig} - 1$, and from $i - n_{ig}$ to $i - 1$ if $i \geq n_{ig} - 1$.

A matrix U^i containing n_{ig} vectors is created as

$$U^i = \left(u_j^i\right)_{j=1}^{n_{ig}} \tag{2.10}$$

where, if $i < n_{ig} - 1$, that is $n_{vect} < n_{ig}$, the vectors u_j^i with $j > n_{vect}$ are left equal to 0.

The way this computation of initial guesses is implemented strongly takes advantage of the fact that most coefficients in matrix C^i are used again in C^{i+1}, actually all of them as long as $n_{vect} < n_{ig}$ and all but one after that.

2.4 Implementational choices

From the first equation in (2.3), matrix C^i is obviously symmetric, and as already mentioned, the system $C^i \nu^i = s^i$ is very small compared to the linear system the solver is conceived for, so that a direct method can be chosen.

However, the practical implementation of this kind of solver may lead to important issues when the number of vectors becomes quite large and/or if some of the vectors d_k become numerically close to linearly dependent: in these cases, matrix C^i can become close to singular, so that a direct solving may become impossible, or very sensitive to perturbations, with a given machine precision. This is why a QR decomposition using a modified Gram-Schmidt process has been chosen here.

A minimum norm is defined under which a vector is considered null, computed as a function of the norm of C^i. Then, at every step of the Gram-Schmidt process, if the norm of a computed vector is below the minimum norm, it is set to zero; else, the normalization of the vector is performed normally. This method can be seen as a particular way of dismissing the vectors d_k that would have caused important numerical errors in a direct solver, but it is also a way of turning a singular problem into a regular one "on the fly", by substracting its degrees of freedom during its solving. All the solutions of this system, in which vectors are not linearly independent, would result in the exact same initial guess for the system $\hat{A}d_i = b_i^C$, so that one only wants to ensure that one of them will be found.

It can also be noticed here that, when using this algorithm in parallel, one extra communication is needed for each iteration of the solver on the coarse grid, in order to compute the sum of the local contributions to the dot-products that appear in the terms of C^i and s^i of Equation (2.8). That being said, the communication costs, as well as the computed ones, are still way lower than those of a Fischer-like projection technique, that would start with a costly orthonormalization step.

The effectiveness of several values of n_{ig} for different test cases is assessed in Section 5.2.

Chapter 3

Adaptation of the convergence criterion on the coarse grid

3.1 Principle

The idea of increasing the convergence criterion of the PCG on the coarse grid arises among others from multigrid methods, where only a few iterations on the coarse grid(s) are usually performed or a quite low error reduction is required [41], instead of reaching a comparable precision as the one sought on the fine grid. The application of a similar method to a deflated solver could be very efficient, as most time spent on the coarse level is incompressible communication time, and as, once again, the successive solvings on this level often account for a major part of the global solving time.

However, fixing a limited number of iterations on the coarse grid may not be appropriate. The first calls for the coarse grid solver have a very important contribution in the definition of the search directions, and they need tens to hundreds of iterations in order to reach the precision usually sought, so that fixing a maximum number of iterations for them will deteriorate the very first steps of descent. On the other side, the solving on the coarse grid takes less iterations as the convergence on the fine grid is approached, so that a fixed number of iterations on the coarse level could end up being a waste of time.

This is where the need of an adaptive convergence criterion arises, that would not be based on empirical results but on a study of the impact of coarse-grid approximations on the fine grid solver.

3.2 Development

Since a PCG solver is used on the coarse grid, the solving of $\hat{A}d_i = b_i^C$ is made by modifying iteratively the value of an estimate d_i^* of d_i, in such a way that a given norm of the residual $r_i^* = b_i^C - \hat{A}d_i^*$, that would be equal to 0 if the exact solution was reached, decreases over the iterations. The stopping condition for this iterative solver is

$$\|r_i^*\|_\infty < \gamma_i \,, \tag{3.1}$$

where γ_i is usually a constant value, often set to the value γ of the convergence criterion on the fine grid. When a value of d_i^* meeting this condition has been reached, one can write the solution d_i as the sum of this value and an absolute error $\Delta d_i = d_i - d_i^*$. By following the next steps of the solver on the fine grid, one may study the impact of Δd_i on the fine grid computations, and then be able to impose a different value of γ_i for each projected system such that a certain deviation criterion is met.

The initial idea behind this work was to study the impact of the approximation of p_i on the research for the next descent direction by following the steps of Algorithm 2, until being able to write explicitly

the direction p_i^* as a sum of the expected exact direction p_i and a computational "error" Δp_i; requiring for a given norm of Δp_i to be negligible compared to the norm of p_i would then have given a maximum value of γ_i. However, as this particular development results in a value of Δp_i whose infinite norm is very hard to estimate as a function of γ_i, a simpler criterion has been developed by focusing only on w_i, that one may call the descent basis, as the descent direction p_i is obtained from the $\hat{A}$-orthogonalization of w_i towards p_{i-1}.

One can consider that all the computations before the solving of $\hat{A}d_i = b_i^C$ are in exact arithmetics, which is only a practical assumption. The solving of this system leads to the approximation d_i^* of d_i, and instead of $w_i = M^{-1}r_i - Wd_i$ is computed $w_i^* = M^{-1}r_i - Wd_i^*$. The approximation error on w_i is then

$$\Delta w_i = w_i^* - w_i = W(d_i - d_i^*) = W\Delta d_i \,, \tag{3.2}$$

which implies $r_i = \hat{A}\,(d_i - d_i^*) = W^T A \Delta w_i$. As $w_i = M^{-1}r_i - Wd_i$, one can write

$$W^T A w_i = W^T A M^{-1} r_i - \hat{A} d_i = W^T A M^{-1} r_i - b_i^C \,. \tag{3.3}$$

If one wants $\left\|W^T A \Delta w_i\right\|_\infty$ to be negligible compared to $\left\|W^T A w_i\right\|_\infty$, which would mean that the deviation on $W^T A w_i$ would be negligible in the sense of the infinite norm, the resulting condition is very difficult to meet, as the infinite norm of $W^T r_i$ tends to be very small. Nevertheless, one can instead require that $W^T A \Delta w_i$ should be negligible compared to each separate term of the expression of $W^T A w_i$ obtained in Equation (3.3), that is

$$\begin{cases} \left\|W^T A \Delta w_i\right\|_\infty \leq C_N \left\|W^T A M^{-1} r_i\right\|_\infty \\ \left\|W^T A \Delta w_i\right\|_\infty \leq C_N \left\|b_i^C\right\|_\infty \end{cases} , \tag{3.4}$$

and this condition can finally be written under the form

$$\gamma_i \leq C_N \min\left(\left\|W^T A M^{-1} r_i\right\|_\infty, \left\|b_i^C\right\|_\infty\right) \,. \tag{3.5}$$

In practice, very low values are avoided by choosing to bring back γ_i to the convergence criterion on the fine grid γ if the value computed from Equation (3.5) is lower, so that the effective computation is

$$\gamma_i = \max\left[\gamma, C_N \min\left(\left\|W^T A M^{-1} r_i\right\|_\infty, \left\|b_i^C\right\|_\infty\right)\right] \,. \tag{3.6}$$

3.3 Choice of C_N

For the automatic use of the convergence criterion adaptation technique, effective values of the convergence criterion C_N must be known and, more importantly, whether or not C_N should have similar values for different configurations, number of cells in the mesh, and so on, must be determined. A series of tests have therefore been performed on different test-cases with various mesh and group sizes.

The very definition of the negligibility constant C_N can make one expect that its most effective values should range somewhere between 10^{-4} and 10^{-2}. Practical experiences confirm this intuitive result: on one hand, values below this interval do not change the global behaviour of the solver on the coarse grid; on the other hand, values above this interval make the solver on the coarse grid compute very imprecise solutions, thus raising the number of iterations required on the fine grid and deteriorating the global quality of the solver.

An extremal case for the first effect is $C_N = 0$, which results from Equation (3.6) in the choice of the default value γ, that is the convergence criterion on the fine grid, for every system on the coarse grid. The second effect is obtained by raising C_N to a value that the initial residual on the coarse grid

will never reach: the greatest possible value for a real number with the computational precision would make it, but values way lower than this would also do the trick; in this particular case, and without any computation of initial guesses on the coarse grid, the solver turns back to a classical PCG. In practical cases, values of C_N ranging from 0.001 to 0.01 seem of particular efficiency, which is shown in Sections 5.3 and 5.4.

The combination of this technique with the computation of initial guesses on the coarse level, described in Chapter 2, creates a variant of the A-DEF2 solver, the efficiency of which depends on the choice of C_N and n_{ig}. This novel solver, aiming at reducing the number of iterations of the PCG solver on the coarse grid, is called RA-DEF2(n_{ig},C_N) in what follows. Its efficiency, in terms of numbers of iterations on the coarse grid, global computational times and proportion of communication over the computation times, is detailed in Chapter 5.

Chapter 4

Massively parallel solving

The primary objective of this study was the solving of linear systems arising from the high-precision simulation of incompressible flows, that is on meshes ranging from a few million to several billions of cells, as it is quite obvious that numerical methods such as the DPCG method aim at solving very large linear systems. An important effort is therefore required for the parallelization of the code, especially regarding the data structures and storage, and one has to choose carefully the parameters used for a given computation on a given supercomputer. The Double Domain Decomposition method used in the YALES2 solver is introduced in this chapter, followed by a short chapter about the choice of parameters for massively parallel solvings.

4.1 Double Domain Decomposition technique

The standard Single Domain Decomposition (SDD) consists of a simple splitting of the domain into a number of subdomains equal to the number of processors used for the computation, as illustrated by Figure 4.1. The variables on the subdomain are stored in the local memory, and communication protocols have to be implemented for the exchange of values at the frontiers between subdomains. Although this is the simplest way of decomposing a grid for parallel solving, it holds several drawbacks for massively parallel solving, especially regarding the issues of load balancing and local mesh refinement. Each of them requires an important computational time, as every modification occurring on a subdomain will lead to a complete reconstruction of the grid on this subdomain.

The Double Domain Decomposition (DDD) methodology, illustrated by Figure 4.2, has been introduced in the YALES2 solver by Moureau *et al.* [25], mostly to avoid the issues of the SDD method and optimize accesses to memory. An additional level of subdomains is introduced, splitting the subdomain assigned to a given processor into groups. The advantages of this approach regarding the issues of load balancing and local mesh refinement are quite obvious. The former can be managed by transferring groups of control volumes from one subdomain to another, which will make the computational effort required for the reconstruction of the subdomains and communicators more reasonable. As for the latter, it is performed on groups, removing the need of a global reconstruction of the subdomain. Moreover, the groups create a coarse grid on the domain, on which preconditioning methods such as deflation can be easily implemented. Last but not least, if the amount of data on one group can be stored in the cache memory, the amount of data passing between the Random Access Memory and the cache memory is reduced.

The implementation of internal communicators, that is the communicators between the groups on the same processor, and external communicators, that is the communicators between the subdomains, is not detailed here. External communicators rely on MPI requests, and internal communicators are either local or based on OpenMP intra-processor requests.

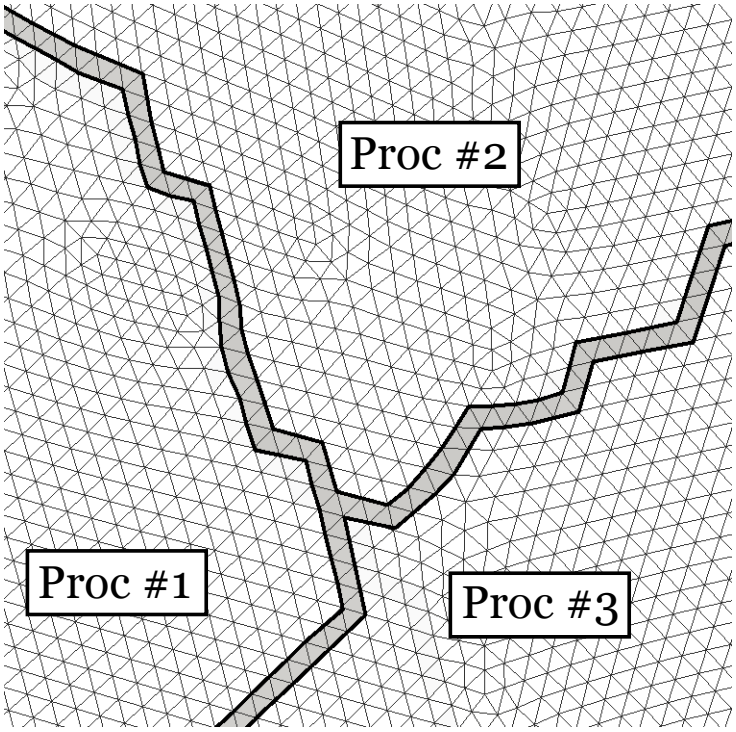

Figure 4.1: *Single Domain Decomposition of a two-dimensional unstructured mesh. The thick edges are boundaries between the subdomains belonging to different processors; the cells between these boundaries are the ones involved in the external communicators.*

4.2 Choice of parameters for massively parallel uses

Given a grid on the whole domain, the number of processors used for the computation and the number of control volumes that should belong to any group, the splitting of the grid is performed by either the METIS [17] or SCOTCH [31] libraries. The algorithms used by both libraries optimize the splitting by minimizing the edgecut, in other words the global weight of the cell edges or faces that belong to the boundaries between subdomains, and the load unbalance between the processors. Therefore, there are two parameters that have a strong influence on the efficiency of the program.

First, the number of processors used for a simulation determines the number of cells for each processor: if there are too few cells for each processor, the computational time becomes negligible towards the communication time, thus worsening the global performances; if there are too many cells for each processor, the computation is slowed down by cache memory overlappings.

Second, the size of a group has to be determined according to several parameters. If a combination of MPI and OpenMP is implemented, one has to consider the distribution of the groups over the cores of every computation node. If pure MPI is used, the number of cells per group is still of great importance, as it has an influence on the efficiency of the DPCG and on the "under-the-surface" management of the cache memory of each processor. Ideally, the piece of data relative to a given group should fit entirely in the cache.

The optimal values of these parameters are defined as the values for which the reduced efficiency, that is the global CPU time per control volume (Section 5.5), reaches a minimum. These values obviously depend on the architecture and properties of the supercomputer used for the computations: number of cores per processor, frequency of each core, storage size, size of the cache memory, and so on. A theoretical computation of the ideal parameters for a given configuration would have to take into account a large set of data, with the risk of neglecting some of greater importance than thought.

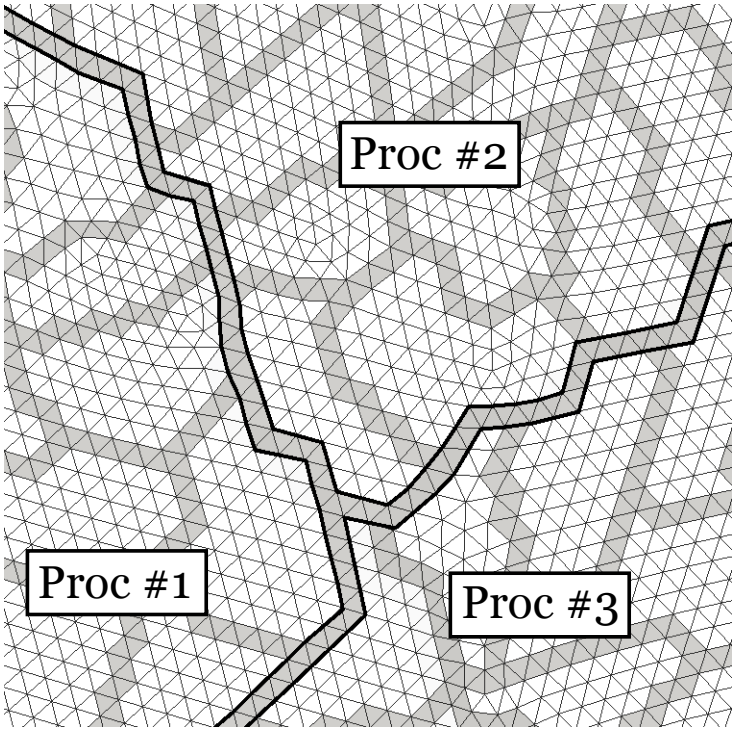

Figure 4.2: *Double Domain Decomposition of a two-dimensional unstructured mesh. The thick edges are boundaries between the subdomains belonging to different processors; the cells between these boundaries are the ones involved in the external communicators. The highlighted cells are at the frontiers between different groups; they are involved in the internal communicators.*

This is why an empirical study should be performed before any large computation. A preliminar study on small test-cases makes it possible to find a nearly optimal number of cells to be assigned to each core, and a good number of control volumes per group.

On the Babel machine for instance (Section 5.1), the number of control volumes on each core should be around ten thousands: communication times tend to become too important if this value is lowered, and memory swapping times become predominant if it is raised too much.

Chapter 5

Numerical results

5.1 Studied configurations

During the simulation of a turbulent low-Mach flow, solving the pressure Poisson equation, that links the pressure to the velocity, can be the most time-consuming step of the whole simulation. The work described in this report was particularly aiming at reducing the cost of the computation of the instantaneous pressure fields. It has been implemented in the unstructured LES solver YALES2, for all numerical experiments and validations described hereafter. The novel solver, built from the A-DEF2 solver but using the improved initial guesses and adaptive convergence criterion techniques with parameters n_{ig} and C_N respectively, is called RA-DEF2(n_{ig},C_N) in what follows.

Several test cases have been studied, ranging from simple validation tests to simulations of turbulent flows in complex geometries. Their names and descriptions are as follows:

2D_Poisson is a simple test case corresponding to the solving of a Poisson equation on a unity square centered on the origin, meshed with triangular elements. The right-hand side of the equation, for this test case, comes from a linear combination of several sinusoidal signals with different frequencies; a function f is defined as

$$\begin{aligned} f(x,y) = 500 \sin(\pi x) \sin(\pi y) + 1,000 \sin\left(\frac{\pi x}{0.2}\right) \sin\left(\frac{\pi y}{0.2}\right) \\ + 6,000 \sin\left(\frac{\pi x}{0.033}\right) \sin\left(\frac{\pi y}{0.041}\right) \end{aligned} \tag{5.1}$$

and the value of the right-hand side on every cell is the product of the value of f on the center of the cell by the volume of the cell. The mesh is refined towards the center, as shown by Figure 5.1, with a resulting RHS as plotted on Figure 5.2,

3D_Cylinder is a test case corresponding to the simulation of an isothermal fluid flow around a cylindrical obstacle placed across a three-dimensional box. The box has a length of 60 cm, a width of 30 cm and a height of 5 cm. The fluid comes at a constant speed of 15 m.s^{-1} from the inlet, that is the whole left section of the box, in the direction orthogonal to the plane of the inlet. The cylinder, whose diameter is 1 cm, is placed 10 cm after the inlet, at mid-width, and goes through the whole height of the box. The coarsest mesh on this configuration, counting 491,000 elements, is illustrated by Figure 5.3.

Preccinsta is a test case corresponding to the simulation of isothermal fluid flows in the PRECCINSTA burner. The name of the so-called PRECCINSTA burner stands for *PREdiction and Control of Combustion INSTAbilities for industrial gas turbines*. This configuration, experimentally investigated by Meier et al. [22], has been thoroughly studied from both experimental and

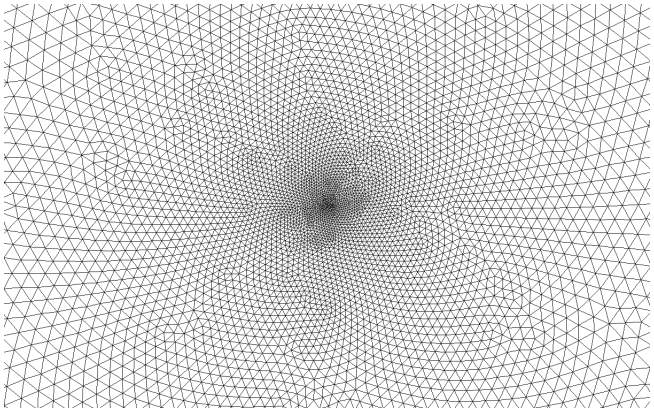

Figure 5.1: *Zoom on the mesh of the* **2D_Poisson** *test case. The volume ratio between the largest and smallest cells of the mesh is about 80,000 to 1.*

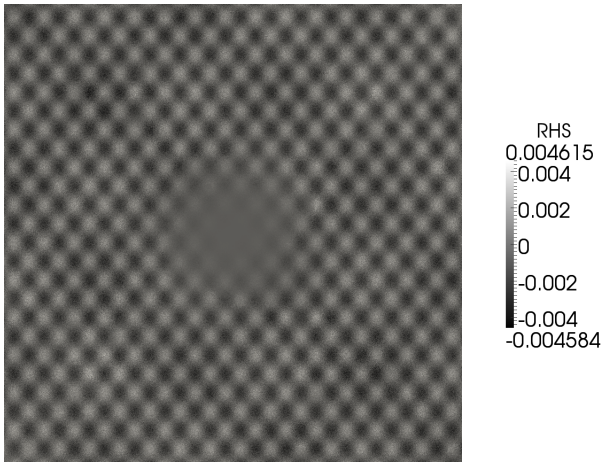

Figure 5.2: *Right-hand side of the Poisson equation of the* **2D_Poisson** *test case.*

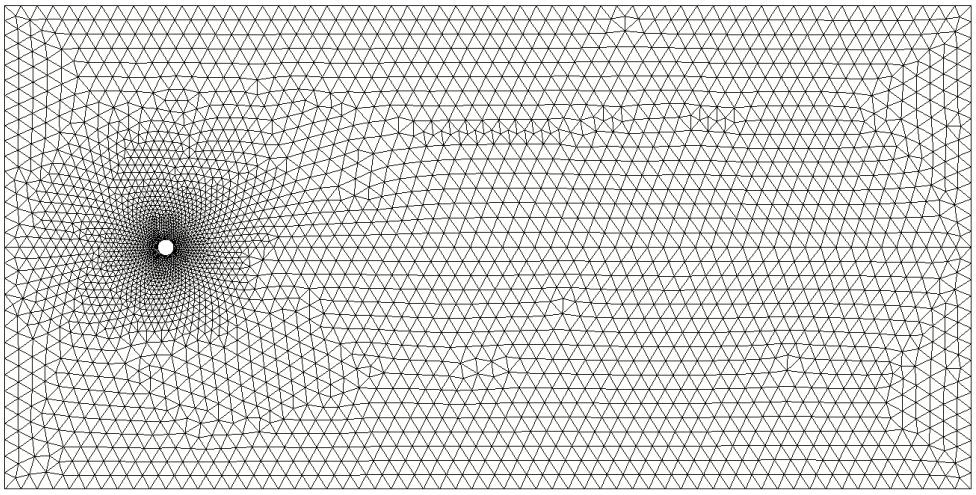

Figure 5.3: *Longitudinal external section of a 491,000 elements mesh of the* **3D_Cylinder** *configuration.*

numerical points of view [19], and widely used as a validation database for combustion models [13, 34, 26] or for numerical methods applied to the solving of the Navier-Stokes equations [24].

The geometry considered in the simulations features a plenum, a swirler, a square combustion chamber and a cylindrical exhaust pipe, as represented in Figure 5.4. Air at ambient temperature is pumped at a speed of 24.5 m.s^{-1} to the 79 mm wide plenum, leading to the injector in which it goes through 12 oblique vanes; in each of them, methane is injected at the same speed by a 1 mm wide pipe orthogonal to the corridor. The nozzle exit has a diameter of 27.85 mm, and leads to the cylindrical combustion chamber, that is 85 mm wide and 114 mm long. Burnt gases go through a cone-shaped exit leading to a short exhaust pipe of diameter 40 mm. The corresponding Reynolds number is approximately 40,000. The whole geometry is meshed in the presented computations.

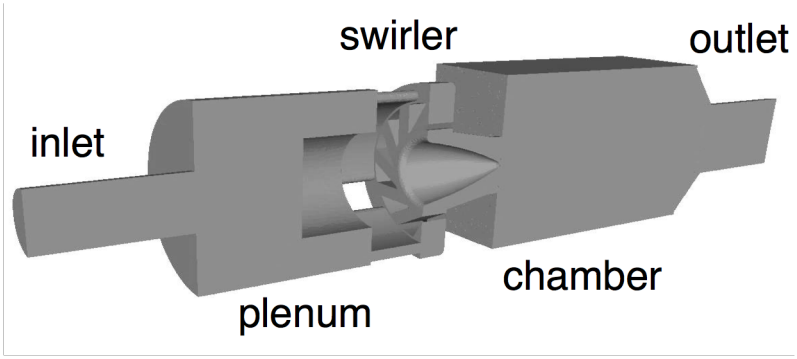

Figure 5.4: *Longitudinal median section of the computed geometry of the PRECCINSTA burner.*

Blade_HT is a test case corresponding to the simulation of a turbulent flow around the T7.2 turbine blade, performed in order to study the effect of turbulence on heat transfers. The T7.2 blade has been designed by MTU, based on a design for a typical cooled low-pressure turbine rotor; the shape of its pressure size induces, for given inlet conditions, a great turbulent intensity under the blade, due to the creation of an important recirculation zone. This blade has been studied experimentally by Ladisch et al. [18], and numerically by Lutum and Cottier [20]; however, the transition to turbulence occurring at the suction side was not captured well by the RANS method used by the latter, with important consequences on the intensity of heat transfers, so that a LES study is currently performed in order to improve the numerical results on this configuration.

The maximum thickness of the blade is 16.53 mm and its chord lenght is 73.93 mm. For the computations, the inlet velocity was set at 29.16 m.s^{-1}; the fluid temperature is 350K and the blade is cooled at 290K, giving Dirichlet boundary conditions, with a no-slip condition on the surface of the blade. The resulting Reynolds number of 150,000 at the exit allows the simulation of a physical time long enough to obtain converged results.

The meshes used for these computations have the same shape as the one illustrated by Figure 5.5; the smallest edge sizes range approximately from 30 microns, in the 34.5 million mesh studied hereafter, to 4 microns in the 2.2 billion mesh, and the largest from 3000 to 400 microns for the same meshes respectively. The dimensions of the bounding box for these grids are 18.1 cm by 9.87 cm by 2.85 cm; periodicity conditions are effective on the y and z directions. A section of an instantaneous velocity field computed on a mesh counting 2.2 billion cells is shown in Figure 5.6.

Triple Disk Injector aka **TDI** is a test case consisting in modeling the primary atomization of a liquid fuel in air at atmospheric conditions with realistic properties for the liquid and gas. This injector is formed by three off-centered disks of different diameters, which create strong

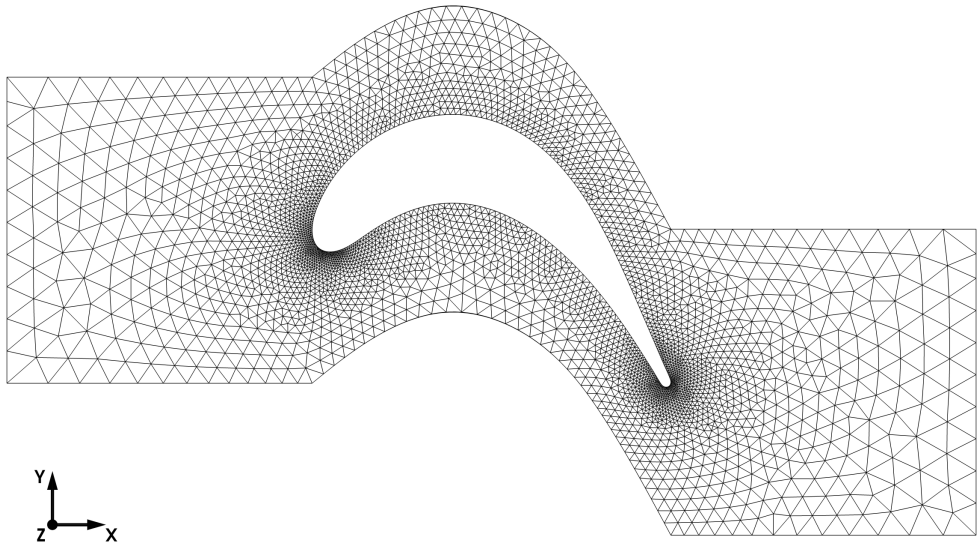

Figure 5.5: *Longitudinal external section of a coarse mesh around the T7.2 blade.*

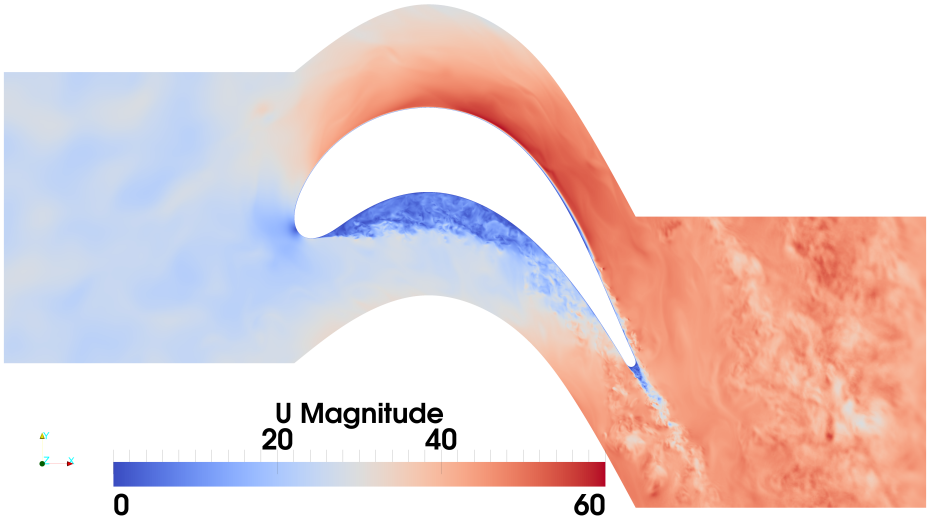

Figure 5.6: *Longitudinal section of the instantaneous velocity field around the T7.2 blade, computed on a mesh of 2.2 billion tetrahedra on 16,384 cores of Babel.*

recirculation zones in the fuel in order to form a liquid sheet from the outlet of the injector. This sheet disintegrates quickly to form small droplets even for moderate pressure losses. Several measurement campaigns were conducted by Grout et al [14] to assess advanced optical diagnostics. In the present computations of this experiment, both the inside of the injector and the basis of the liquid sheet are simulated. The geometry of the injector, which has a circular outlet with a diameter of 180 μm, and the simulated liquid sheet are shown in Figure 5.7. The Reynolds number in the liquid is approximately $3,700$, and the liquid Weber number is $We_L = 1,061$.

This two-phase flow is modeled with a Ghost-Fluid Method [6] and the tracking of the interface is based on a Conservative Level Set approach [4]. This type of flow is particularly challenging

for linear solvers, because the Poisson equation resulting of the governing equations is

$$\nabla \cdot \left(\frac{1}{\rho} \nabla P \right) = RHS \,, \tag{5.2}$$

where the variation of the density ρ reaches three orders of magnitude across the interface.

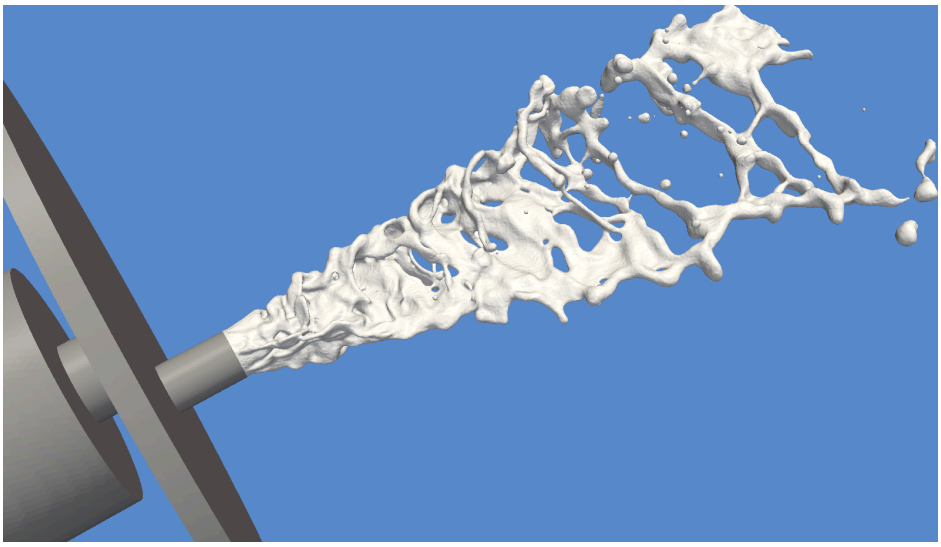

Figure 5.7: *Geometry of the injector and simulated liquid sheet for the* **Triple Disk Injector** *on a grid counting 1.6 billion tetrahedral cells.*

Depending on the studied cases and the sizes of the meshes, several machines have been used:

A 12-core Intel Xeon X5680 CPU at 3.33GHz has been used for the smallest test-cases, in which only the numbers of iterations are significant.

The **2D_Poisson** test case, on a mesh of 3.23 million triangular elements, with 200 elements per group, has been run on 4 cores of this machine, and the same test-case on a 12.9 million mesh, with 400 elements per group, on 8 cores of the same machine.

The **3D_Cylinder** test case, on a mesh counting 491 thousand tetrahedral elements, with 500 elements per group, has been run on 4 cores of this machine.

The **Preccinsta** test case, on a mesh counting 1.7 million tetrahedral elements, with 2000 elements per group, has been run on 4 cores of this machine.

Babel, an IBM Blue Gene/P machine , has been used for the major part of test-cases, in which the numbers of iterations and the statistics on computational times are significant. This machine, owned by a French institute for the development of scientific computation called IDRIS, counts 10 racks of 1,024 computing nodes, each one consisting of 4 cores running at 850MHz.

The **Preccinsta** test case, on a mesh of 110 million tetrahedral elements, with 2,000 elements per group, has been run on 512 cores (128 nodes) of this machine.

The **Blade_HT** test case on a mesh of 34.5 million tetrahedral elements, with 500 elements per group, has been run on 512 cores (128 nodes) of this machine. The same test case, on a 2.2 billion mesh with 5,000 elements per group, has been run on 16,384 cores (4,096 nodes) of this machine.

The **TDI** test case, on a mesh of 1.6 billion tetrahedral elements, with 1,000 elements per group, has been run on 16,384 cores (4,096 nodes) of this machine.

Curie, a BullX machine , is a 2 petaflop Tier-0 supercomputer installed in France at the TGCC, operated by the CEA (French atomic energy agency) and started on March 1st, 2012. It is split into three different types of resources aiming at different kinds of computations, from hybrid OpenMP/MPI to pure MPI codes. The so-called "thin nodes", dedicated to MPI codes, are 10080 Intel Xeon new generation octocore nodes.

The **Blade_HT** test case, on meshes of 35 million, 280 million, 2.2 billion and 17.9 billion tetrahedra, every one of them with 5000 elements per group, has been run on respectively 128; 1,024; 8,192 and 8,192 cores (16; 128; 1,024 and 1,024 nodes) of this machine. To distinguish these runs from the runs of the **Blade_HT** test case on Babel, the label **Blade_HT_Curie** has been adopted in what follows.

It also has to be noticed that the average values presented in Sections 5.4 and 5.5 have been computed for different numbers of temporal steps, depending on the case studied.

The first 100 temporal steps are considered for the **3D_Cylinder** test case and for all the simulations in the PRECCINSTA burner. As for the **Blade_HT**, the first 400 steps have been taken into account on the 34.5 million mesh on 256 cores, the first 750 steps for the same simulations on 512 cores; 45 temporal steps have been run on the **Blade_HT** 2.2 billion mesh, and only 23 on the **TDI** test case, as the simulations without initial guesses and adaptive convergence criterion did not go past these numbers of iterations within the given time limits. All the results for the **Blade_HT_Curie** test cases take into account the first 200 temporal steps, except in Section 5.7.

Finally, the convergence criterion γ on the fine grid is 10^{-12} for the **2D_Poisson** test case, 10^{-10} for all the simulations of the **3D_Cylinder** test case and the PRECCINSTA burner, 10^{-9} for all the simulations of the **Triple Disk Injector**, and 5×10^{-9} for all the simulations of the T7.2 blade.

5.2 Effect of the computation of initial guesses on the number of iterations

The first parameter studied has been the number of former solutions used for the computation of the initial guesses on the coarse grid, that is n_{ig}. Table 5.1 shows the optimal gains, for given choices of n_{ig}, on the number of iterations on the coarse grid, for test cases whose sizes range from 3.23 to 110 million cells. For the **2D_Poisson** case, the **3D_Cylinder** and the 1.7 million elements on the **Preccinsta** burner, values in the set $\{5; 10; 25; 50\}$ have been tested; for the 110 million mesh for the **Preccinsta** test case, they were in the set $\{1; 2; 5; 10; 20\}$.

Test case	2D_Poisson		3D_Cylinder	Preccinsta	
Elements ($\times 10^6$)	3.23	12.9	0.49	1.7	110
Cores	4	8	4	4	512
Percentage of gain	2.67 %	2.19 %	11.8 %	8.11 %	9.67 %

Table 5.1: *Gain on the average number of iterations on the coarse grid for the* **2D_Poisson**, **3D_Cylinder** *and* **Preccinsta** *test cases when computing initial guesses.*

It can be pointed out that the highest value of n_{ig} has not always given the least number of iterations; for the 110 million **Preccinsta** test case, for instance the best results were obtained for $n_{ig} = 1$, although the other non-zero values for n_{ig} gave results that only differed from 0.5 to 2.6 %.

These first results seem to show that the computation of initial guesses as described in Chapter 2 can be of small importance for simple systems, but may be more useful for simulations in more complex

geometries. The choice of an effective value for n_{ig} may be strongly case-dependent and related to many different factors, which is why this method alone does not seem to be of great interest. The adjunction of this method to the adaptation of the convergence criterion on the coarse grid introduced in Chapter 3, however, has been proved very efficient as shown in Sections 5.4 and 5.5.

5.3 Effect of the adaptation of the convergence criterion on the number of iterations

As mentioned in Section 3.3, the value of C_N has to be chosen so that two opposite effects are avoided: a very low value would result in no effective change in the behaviour of the solver on the coarse level, and a value too high would worsen the behaviour of the solver on the fine grid.

These effects are illustrated by Figures 5.8 and 5.9, that show the behaviour of the solver on both levels for different values of C_N on the 1.7 million mesh of the **Preccinsta** test case. The values of C_N are not the same for both figures, for lisibility reasons. For instance, for $C_N = 0.02$, the number of iterations on the coarse grid is quite the same as for $C_N = 0.01$, whereas the number of iterations on the fine grid raises dramatically. On the other hand, activating the adaptation of the convergence criterion on the coarse grid with $C_N = 0.0001$ leaves the number of iterations on the fine grid nearly unchanged whereas the number of iterations on the coarse grid drops. These figures are not means to be exhaustive, but to show a general trend followed by the numbers of iterations on both grids when C_N varies: the average number of iterations on the coarse grid is reduced by higher values of C_N, but there seems to be a threshold close to 0.01, with the values of C_N above this threshold raising the average number of iterations on the fine grid by more than 10 %.

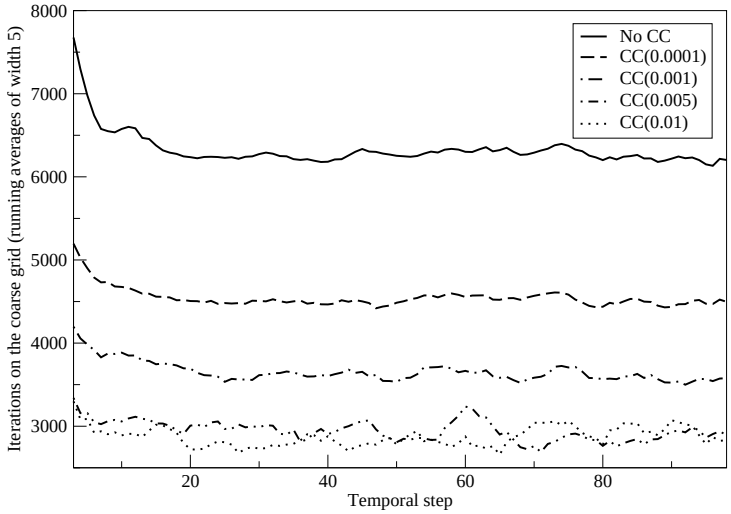

Figure 5.8: *Number of iterations at each temporal step, for different values of C_N, on the coarse grid for the* **Preccinsta** *test case on a mesh counting 1.75 million elements on 4 processors.*

The behaviour of the solver when changing the value of C_N is qualitatively the same for every configuration and mesh size, but the threshold value of C_N and the corresponding gains may differ from one case to another. This is why series of tests have been performed, for a wide range of values of C_N, namely values from 0.0001 to 0.05, on the same test cases as those in Section 5.2. The only test case for which the number of iterations on the fine grid stays low even for high values of C_N is the **3D_Cylinder** test case; for the tested values, the reduction factor on the number of iterations on the coarse grid ranges from 1.29 for $C_N = 0.0001$ to 2.43 for $C_N = 0.02$. For every other test case, Table 5.2 gives the highest value of C_N, among the values that have been tested, for which the average

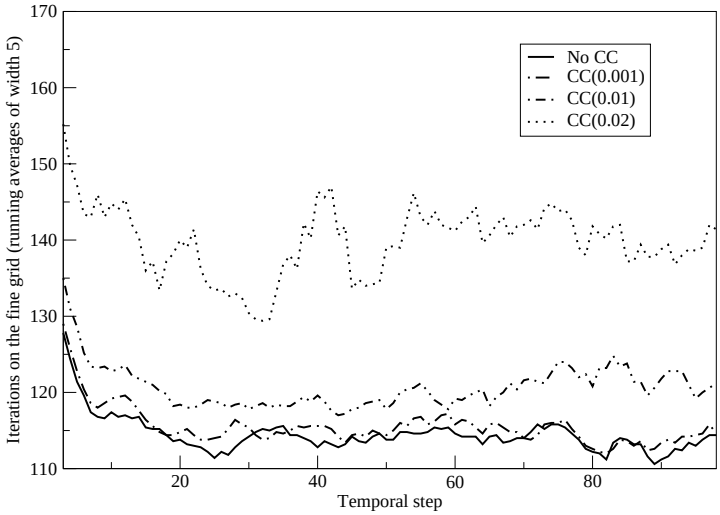

Figure 5.9: *Number of iterations at each temporal step, for different values of C_N, on the fine grid for the* **Preccinsta** *test case on a mesh counting 1.75 million elements on 4 processors.*

number of iterations on the fine grid is raised by less than 10 %, as well as the corresponding reduction factor on the number of iterations on the coarse grid.

Test case	2D_Poisson		Preccinsta	
Elements ($\times 10^6$)	3.23	12.9	1.7	110
Cores	4	8	4	512
Threshold value of C_N	0.001	0.001	0.01	0.005
Reduction factor	3.76	4.27	2.2	1.95

Table 5.2: *Threshold value of C_N and corresponding reduction factor on the number of iterations on the coarse grid for the* **2D_Poisson** *and* **Preccinsta** *test cases.*

These results give a good idea of the typical values of C_N to be used, that is values from 0.001 to 0.01. Further investigation seems to confirm that the value 0.005 is a good candidate.

5.4 Effect of the combined techniques on the numbers of iterations

From the beginning, it has been expected that the initial guess computation and the adaptive convergence criterion method would work better when used together in the same solver, as explained in Section 1.5. In order to confirm this intuition, all possibilities have been tested, with or without the use of each method, on the test cases described above.

Examples of the behaviour of the solver on the coarse grid, when changing the values of C_N and n_{ig}, are plotted over the temporal steps for two test cases in Figures 5.10 and 5.11; three curves only are plotted on each graph so that the curves can easily be distinguished, but the behaviour of the solver when changing these parameters is the same, qualitatively speaking, in every simulation.

The combination of the initial guesses computation and the use of the convergence criterion adaptation on the coarse grid has two positive effects: first, it helps getting the number of iterations on the coarse grid still lower than with the adaptive convergence criterion alone; second, it has been observed that the initial guess can also stabilize the solver when a high convergence criterion adaptation parameter is taken. This effect is illustrated by Figure 5.12, that shows the numbers of iterations on both grids on the 1.75 million mesh of the **Preccinsta** test case, for different values of n_{ig} and a constant

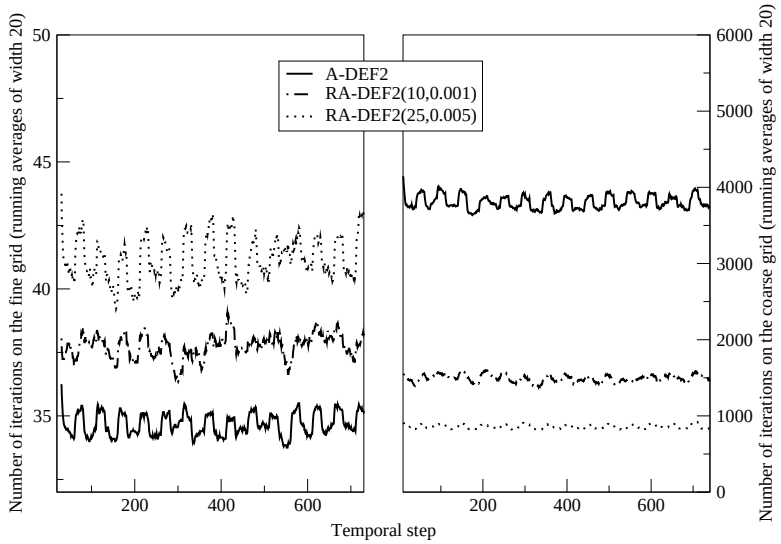

Figure 5.10: *Number of iterations at each temporal step, with different sets of parameters, on the coarse grid for the* **Blade_HT** *test case, on a mesh counting 35 million elements on 512 processors.*

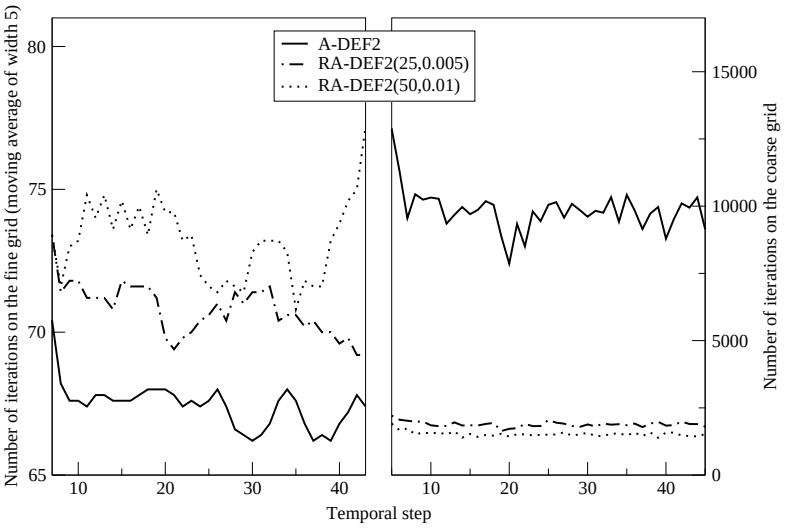

Figure 5.11: *Number of iterations at each temporal step, with different sets of parameters, on the coarse grid for the* **Blade_HT** *test case, on a mesh counting 2.2 billion elements on 16,384 processors.*

high value for C_N, namely $C_N = 0.02$. The number of iterations on the fine grid, raised by the choice of the negligibility constant, is reduced by the computation of initial guesses on the coarse grid; this reduction is roughly the same for any value of n_{ig} greater or equal to 5, although Figure 5.12 only shows the impact of the values $n_{ig} = 5$ and $n_{ig} = 50$ for commodity reasons.

The number of iterations on the coarse grid goes on dropping when one raises the value of C_N, but it has been observed that, in every case, $C_N = 0.005$ is close to a threshold; if the value of C_N is chosen above this threshold, the global computational time rises again in several cases. What has to be noticed, however, is that $C_N = 0.005$ usually does not prevent the solver on the fine grid to attain the required convergence within a quite reasonable number of iterations, that is a number close to the number of iterations required without any adaptation of the convergence criterion on the coarse grid. In all test cases studied here, the only exception is the **Triple Disk Injector** case, for which even small values of C_N result in important variations of the behaviour of the solver on the fine grid; this

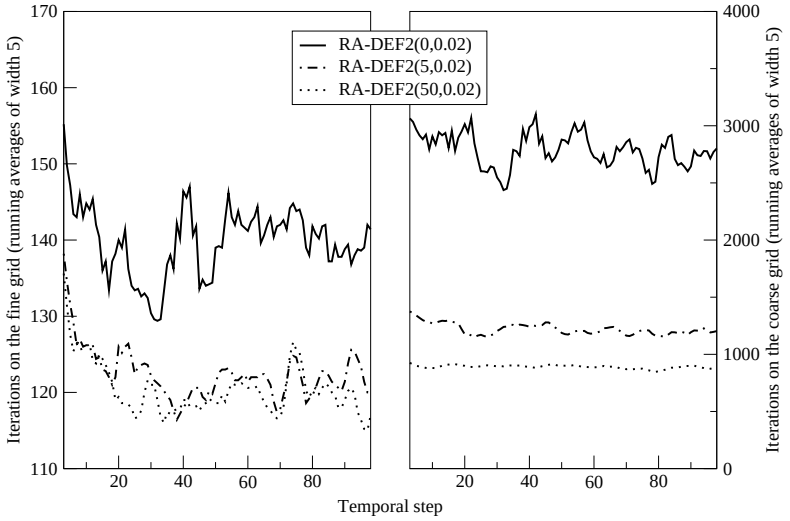

Figure 5.12: *Numbers of iterations on both grids for the RA-DEF2 solver, with* $C_N = 0.02$ *and different values for* n_{ig}*, for the* ***Preccinsta*** *test case, on a mesh counting 1.75 million elements on 4 processors.*

can be explained by the steepness of the linear problem to be solved for this particular simulation. However, the iterations on the coarse grid are still decreased by an important factor, as can be seen of Figure 5.13.

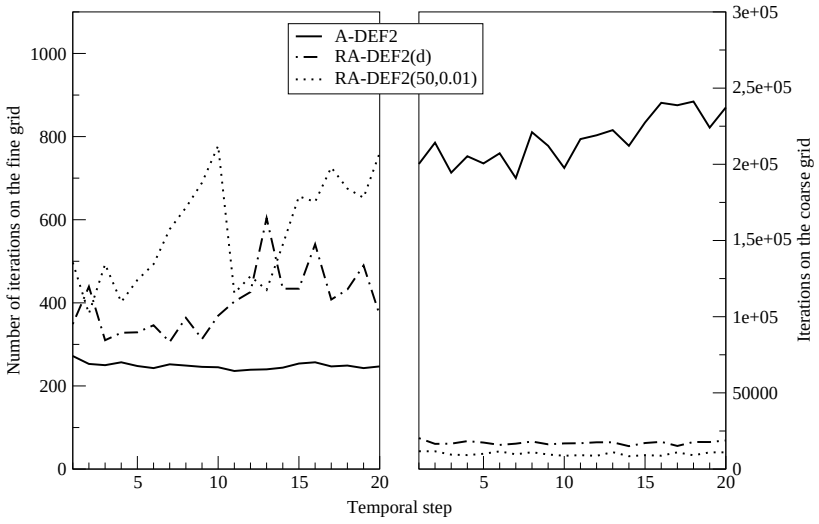

Figure 5.13: *Number of iterations on both grids at each temporal step, for the A-DEF2, RA-DEF2(d) and RA-DEF2(50,0.01) solvers, for the* ***TDI*** *test case, on a mesh counting 1.6 billion elements, on 16,384 processors.*

A close study of the results obtained with this value of C_N, on test cases running on less than a thousand processors, has shown that, in some cases, recycling more than 25 former solutions on the coarse grid can represent a waste of resources, as the number of iterations will not drop anymore. This is why the values ($n_{ig} = 25; C_N = 0.005$) have been chosen as default input values for the solver; for commodity reasons, RA-DEF2(25;0.005) is called RA-DEF2(d) in what follows, as well as in Figure 5.13.

Table 5.3 gives an extensive comparison of the average number of iterations on the coarse grid per temporal step, for every case studied for this report expect the **2D_Poisson** test case, for the

A-DEF2 and RA-DEF2(d) solvers. These results show that the efficiency of the developed method is even greater for more complex simulations, with a reduction factor on the number of iterations on the coarse grid ranging here from 2.62 for the **3D_Cylinder** test case to a striking 12.5 for the **Triple Disk Injector**.

Test case	3D_Cylinder	Preccinsta	Blade_HT		TDI
Elements ($\times 10^6$)	0.49	110	34.5	2,200	1,600
Cores	4	512	512	16,384	16,384
A-DEF2	2,468	13,203	3,813	10,364	216,153
RA-DEF2(d)	941	3,044	859	1,948	17,227
Reduction factor	**2.62**	**4.34**	**4.44**	**5.32**	**12.5**

Test case	Blade_HT_Curie		
Elements ($\times 10^6$)	34.5	276	2,200
Cores	128	1,024	8,192
A-DEF2	4,002	6,233	9,593
RA-DEF2(d)	1,059	1,670	2,293
Reduction factor	**3.78**	**3.73**	**4.18**

Table 5.3: *Average number of iterations on the coarse grid per temporal step for the A-DEF2 and RA-DEF2(d) solvers, followed by the reduction factor of the number of iterations, for all test cases except the* **2D_Poisson** *test case.*

5.5 Effect on computational and communication times

The time spent on the coarse level mainly consists of communications between cores, so that the drastic reduction of the number of iterations on the coarse grid, clearly seen in Table 5.3, is bound to reduce both the computational time and the proportion of communications towards global computational time. Both of them can be given by the YALES2 solver at every temporal step, but the former will be studied via the reduced efficiency of a temporal step, that is the product of the time spent in the solver by the number of cores, divided by the number of control volumes in the mesh; it stands for the average computational time spent by the computing machine for every control volume in the mesh, and would be constant for a given configuration on a given machine, when changing the number of cores dedicated to the solving, if communication times were negligible.

The average reduced efficiencies for several test cases, that is the ones running on several hundreds or thousands of processors, for both the A-DEF2 and RA-DEF2(d) solvers, are given by Table 5.4.

Quite obviously, as the whole simulation time is not spent in the Poisson solver, the reduction of computational times is not directly proportional to the reduction of the number of iterations; nevertheless, these results clearly show the efficiency of the novel RA-DEF2 method, especially for massively parallel simulations on complex geometries. Once again, the results on the **TDI** test case are striking, with a nearly four-fold reduction of computational times. This particular result is matching the reduction of the number of iterations on the coarse grid, as the solving of the pressure Poisson equation by the A-DEF2 solver accounts for 58.9 % of the global computational time for this simulation.

Figure 5.14 shows the reduced efficiencies for the first twenty temporal steps of the **TDI** test case, for the same three solvers as in Figure 5.13, so that the efficiency of the RA-DEF2 method, both in terms of number of iterations and computational times, can be linked. It is obvious that the reduction of the number of iterations on the coarse grid more than makes up for the important increase of the number of iterations on the fine grid in this case.

As the reduction of computational times is strongly linked to the reduction of communications,

Test case	Preccinsta	Blade_HT		TDI
Elements ($\times 10^6$)	110	34.5	2,200	1,600
Cores	512	512	16,384	16,384
A-DEF2	646.7	776.6	862.5	9,023
RA-DEF2(d)	527.7	685.2	660	2,533
Gain	**18.4 %**	**11.8 %**	**23.5 %**	**71.9 %**

Test case	Blade_HT_Curie		
Elements ($\times 10^6$)	34.5	276	2,200
Cores	128	1024	8192
A-DEF2	77.2	96.21	243.7
RA-DEF2(d)	70.4	76.51	128
Gain	**8.81 %**	**20.5 %**	**47.5 %**

Table 5.4: *Average reduced efficiency per temporal step for the A-DEF2 and RA-DEF2(d) solvers, in microseconds-cores per control volume and per iteration, for massively parallel test cases.*

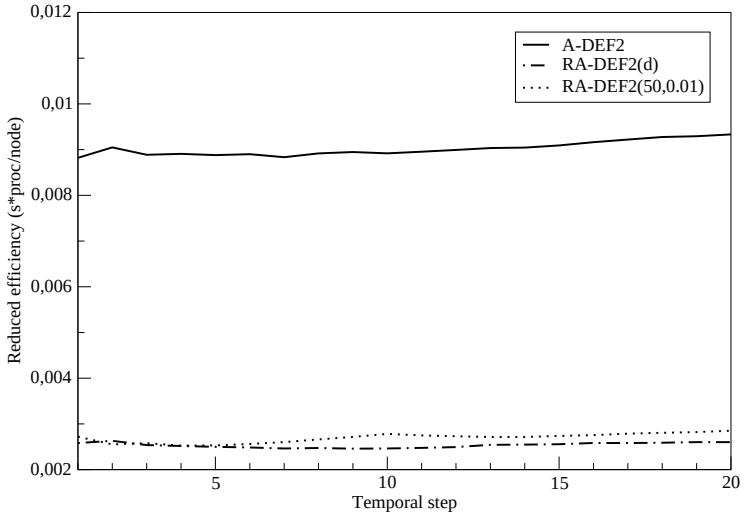

Figure 5.14: *Reduced efficiencies for each temporal step, for the A-DEF2, RA-DEF2(d) and RA-DEF2(50,0.01) solvers, for the **TDI** test case on a mesh counting 1.6 billion elements on 16,384 processors.*

the proportion of the global computational time spent in communications is bound to be decreased by the RA-DEF2 method. The average proportions of communications over global computational times, in the massively parallel solving of several test cases by the A-DEF2 and RA-DEF2(d) solvers, are given by Table 5.5. The important decrease of these proportions, ranging from a 20 % reduction to a two-fold drop, has been expected from the beginning, as the iterations of the coarse grid tend to account for the major part of interprocessor communications in massively parallel simulations.

5.6 Effect on the weak scaling

The weak scaling of a solver is a measurement of the variation of the computational times when the number of processors varies, with a fixed problem size per processor. If communications were instantaneous, one could expect that multiplying the number of processors by x would divide the computational times by x; the resulting speed-up would then be equal to the number of processors

Test case	Preccinsta	Blade_HT		TDI
Elements ($\times 10^6$)	110	34.5	2,200	1,600
Cores	512	512	16,384	16,384
A-DEF2	11.35 %	16.15 %	27.35 %	58.9 %
RA-DEF2(d)	9.015 %	9.496 %	12.71 %	36.28 %
Gain	**20.6 %**	**41.2 %**	**53.5 %**	**38.4 %**

Test case	Blade_HT_Curie		
Elements ($\times 10^6$)	34.5	276	2,200
Cores	128	1,024	8,192
A-DEF2	14.24 %	19.46 %	18.92 %
RA-DEF2(d)	10.96 %	13.16 %	13.83 %
Gain	**23 %**	**32.4 %**	**26.9 %**

Table 5.5: *Average proportions of communications for the initial DPCG and the optimized DPCG, for test cases running on several hundreds or thousands of processors.*

involved in the computations, resulting in a linear scaling. However, these ideal values are never reached in practice, as the ratio between the actual speed-up and the number of processors tends to decrease when the number of processors is raised.

The number of iterations on the coarse grid, and thus the number of interprocessor communications, are particularly decreased on massively parallel computations using the RA-DEF2(d) solver. In order to measure the impact of this reduction on the speed-up of the Yales2 solver, its weak scaling has been studied using both the A-DEF2 and RA-DEF2(d) solvers, on three meshes on the **Blade_HT** test case. Simulations on the 34.5 million mesh have run on 256 cores of the Babel machine, and the 276 million and 2.2 billion meshes obtained by successive non-degenerative homogeneous mesh refinements [33] were used for the 2,048 and 16,384 cores simulations respectively, on the same machine.

The computational times for 40 successive iterations were averaged. The first 10 iterations have been skipped, in order to focus on the computations occurring after the regime has become close to stationary from a statistical point of view. The resulting weak scaling, with the results of the simulations on 256 cores chosen as reference points, is plotted in Figure 5.15. One can see that the modifications of the A-DEF2 algorithm, that have created the RA-DEF2(d) solver, result in a tremendous improvement of the weak scaling of the YALES2 solver. The speed-up between the simulations on 256 and 16,384 cores reaches 90.57 % of its ideal value when using the RA-DEF2(d) solver, versus 75.44 % with the A-DEF2 solver.

5.7 Effect on the extreme scaling

Compared to the nodes of Babel used hereinbefore, the fine nodes of Curie are more powerful and communications between them are fast, enabling one to perform computations on very large meshes. An additional run has been performed on 8,192 processors (1,024 nodes) of this machine, on a mesh of 17.8 billion tetrahedral elements. Although the efficiency statistics have only been computed every ten temporal steps, one can consider that the averages on the whole run, consisting of 1,600 steps, are nearly exact. The results of this run are given, and compared with data from the 2.2 billion elements on the same number of processors, in Table 5.6.

The average reduced efficiency is lower by nearly 15 % than the one on a 2.2 billion element mesh on the same number of processors, that is 110 microseconds-cores per iteration and per control volume, instead of 129. It is probably a direct consequence of the reduction of communications due to the higher amount of data on each core, as the proportion of communications drops from 13.8 % to 9.55 %

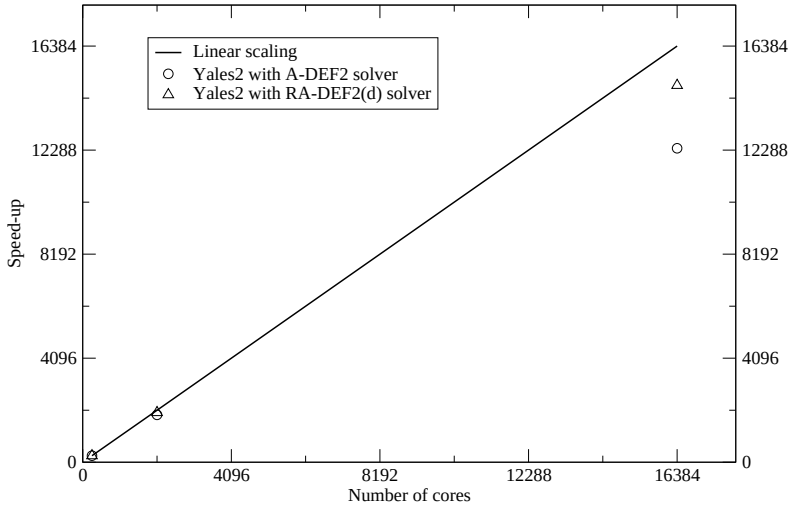

Figure 5.15: *Weak scaling of YALES2 on Babel, up to 16,384 processors and 2.2 billion elements, for the* ***Blade_HT*** *test case.*

Test case	Blade_HT_Curie	
Elements	2.2 billion	17.8 billion
Average reduced efficiencies	128	110.3
Average proportions of communications	13.83 %	9.55 %
Average numbers of iterations on the coarse grid	2293	4436

Table 5.6: *Average reduced efficiencies per temporal step, in microseconds-cores per control volume and per iteration; average proportions of communications over the global computational times; and average numbers of iterations on the coarse grid per iteration on the fine grid; for the* **Blade_HT_Curie** *test case, on 2.2 and 17.8 billion elements, running on 8,192 cores.*

by refining the mesh, from 2.2 to 17.8 billion cells, without raising the number of processors. Still, this result indicates that the YALES2 solver with the RA-DEF2(d) linear solver is able to handle such large meshes and to produce the corresponding simulations with satisfying results in terms of computational times.

Chapter 6

Summary

Simulations of incompressible flows require the solving of the pressure Poisson equation at every temporal step, and this particular solving can account for the most important part of the computational cost of the solver. The use of a deflation method is an efficient way of reducing the cost of a Poisson solver and, unlike multigrid methods, its implementation is easy in a solver on unstructured meshes and can be adapted to any iterative solver; nevertheless, as an important part of the computational time of a deflated solver is spent in communications on the coarse grid, the reduction of the number of iterations on this grid is a critical step towards the optimization of the solver.

Two methods have been developed in order to accelerate the convergence of a DPCG solver for the pressure Poisson equation. The first one consists in determining, for every system to solve on the coarse grid, an initial guess computed from the solutions of the previous systems on this grid. The method developed in this report is inspired by the projection method introduced by Fischer in [8], but does not require the orthogonality, nor even the linear independence, of the recycled vectors, and ensures the optimality of the computed initial guess for the given input.

The second method consists in adapting the convergence criterion of the system on the coarse grid, in a way that still ensures that the approximate solution is precise enough not to deteriorate the global convergence behavior of the solver. The analytical introduction of the error on the solution of a system on the coarse grid, and the study of its impact on the following operations in the current iteration of the fine grid solver, are used. As a result, at every iteration of the solver on the fine grid, a convergence criterion on the coarse grid is computed that ensures that a requested tolerance on the descent basis on the fine grid is not reached.

These two methods have been implemented in the pressure Poisson solver of the unstructured LES solver YALES2 developed at the CORIA laboratory. Their efficiency has been tested on several test cases, ranging from a model two-dimensional Poisson problem solved on four processors to the simulation of realistic three-dimensional turbulent flows in complex geometries on up to 16384 processors. Smaller test cases have provided indicative results concerning the choice of parameters for both methods and the possible reduction on the number of iterations on the coarse grid; massively parallel simulations of realistic turbulent flows in complex geometries have confirmed the validity of the chosen default parameters and provided reliable results on the reduction of computational times thanks to this novel DPCG method.

With the default parameters chosen for the method on massively parallel computations, the number of iterations on the coarse level can reach a twelve-fold drop, which in term implies a reduction of the computational time spent in the pressure Poisson solver of up to 70 %, accompanied by a reduction of the global proportion of communication times of the LES solver by 20 to more than 50 %. Moreover, the novel solver has proven stable and capable of handling meshes up to 17.8 billion tetrahedral elements.

This progress is of course an important asset for the massively parallel simulation of incompressible flows, but the efficiency of a Poisson solver is vital in many other problems derived from physics, such

as electrostatics, quantum mechanics, and so on. New multi-level deflation methods can already be considered, inspired by the multigrid methodology, but way more flexible and easy to implement in a solver on unstructured meshes.

Acknowledgements

We acknowledge that the results in this report have been achieved using the PRACE Research Infrastructure resource Curie based in France at the CEA-TGCC center under the allocation x2012026880. This work was also granted access to the HPC resources of IDRIS under the allocation 2012-6880 made by GENCI (Grand Equipement National de Calcul Intensif). We would like to thank Ghislain Lartigue, David Taieb and Luc Vervisch for their helpful support.

Bibliography

[1] Aubry, R., Mut, F., Löhner, R., and Cebral, J. R. Deflated preconditioned conjugate gradient solvers for the pressure-poisson equation. *Journal of Computational Physics 227* (2008), 10196–10208.

[2] D'Azevedo, E. F., Eijkhout, V., and Romine, C. H. Lapack Working Note 56 - Conjugate Gradient algorithms with reduced synchronization overhead on distributed memory multiprocessors. Tech. rep., Mathematical Sciences Section, Oak Ridge National Laboratory, December 1999.

[3] de Sturler, E., and Kilmer, M. Recycling subspace information for diffuse optical tomography. *SIAM J. Sci. Comput 27* (2004), 2140–2166.

[4] Desjardins, O., Moureau, V., and Pitsch, H. An accurate conservative level set/ghost fluid method for simulating turbulent atomization. *J. Comput. Phys. 227*, 18 (2008), 8395–8416.

[5] Engell, M., Ginsburg, T., Rutishauser, H., and Stiefel, E. Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems. *Birkhauser Verlag* (1959).

[6] Fedkiw, R., Aslam, T., Merriman, B., and Osher, S. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys. 152* (1999), 457–492.

[7] Feng, L., Benner, P., and Korvink, J. G. Parametric model order reduction accelerated by subspace recycling.

[8] Fischer, P. F. Projection techniques for iterative solution of $A\underline{x} = \underline{b}$ with successive right-hand sides. *Computer methods in applied mechanics and engineering 163* (1998), 193–24.

[9] Fletcher, R. Conjugate gradient methods for indefinite systems. In *Proc. of the Dundee Biennial Conference on Numerical Analysis* (New-York, 1975), G. Watson, Ed., Springer-Verlag.

[10] Fletcher, R., and Reeves, C. M. Function minimization by conjugate gradients. *Computer Journal 7* (1964), 149–154.

[11] Fox, L., Huskey, H. D., and Wilkinson, J. H. Notes on the solution of algebraic linear simultaneous equations. *Quarterly Journal of Mechanics and Applied Mathematics 1* (1948), 149–173.

[12] Frank, J., and Vuik, C. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing 23*, 2 (2001), 442–462.

[13] Galpin, J., Naudin, A., Vervisch, L., Angelberger, C., Colin, O., and Domingo, P. Large-Eddy Simulation of a fuel lean premixed turbulent swirl burner. *Combustion and Flame 155*, 1 (2008), 247–266.

[14] Grout, S., Dumouchel, C., Cousin, J., and Nuglisch, H. Fractal analysis of atomizing liquid flows. *International Journal of Multiphase Flow 33*, 9 (2007), 1023–1044.

[15] Hestenes, M. R. Iterative methods for solving linear equations (originally published in 1951 as NAML Report No. 52-9, National Bureau of Standards, Washington D.C.). *Journal of Optimization Theory and Applications 11*, 4 (1973), 323–334.

[16] Hestenes, M. R., and Stiefel, E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards 49* (1952), 409–436.

[17] Karypis, G., and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing 20*, 1 (1999), 359–392.

[18] Ladisch, H., Schulz, A., and Bauer, H. Heat transfer measurements on a turbine airfoil with pressure side separation. *ASME Conference Proceedings 2009*, 48845 (2009), 783–793.

[19] Lartigue, G., Meier, U., and Bérat, C. Experimental and numerical investigation of self-excited combustion oscillations in a scaled gas turbine combustor. *Applied Thermal Engineering 24*, 11-12 (2004), 1583–1592.

[20] Lutum, E., and Cottier, F. Aerothermal predictions on a highly loaded turbine blade including effects of flow separation. In *9th European Turbomachinery Conference* (Istanbul, 2011), I. F. of Mechanical Engineering, Ed.

[21] MacLachlan, S. P., Tang, J. M., and Vuik, C. Fast and robust solvers for pressure-correction in bubbly flow problems. *Journal of Computational Physics 227* (2008), 9742–9761.

[22] Meier, W., Weigand, P., Duan, X. R., and Giezendanner-Thoben, R. Detailed characterization of the dynamics of thermoacoustic pulsations in a lean premixed swirl flame. *Combustion and Flame 150*, 1-2 (July 2007), 2–26.

[23] Morgan, R., and Wilcox, W. Deflated iterative methods for linear equations with multiple right-hand sides.

[24] Moureau, V., Bérat, C., and Pitsch, H. An efficient semi-implicit compressible solver for Large-Eddy Simulations. *Journal of Computational Physics 226* (2007), 1256–1270.

[25] Moureau, V., Domingo, P., and Vervisch, L. Design of a massively parallel CFD code for complex geometries – Une algorithmique optimisée pour le supercalcul appliquée à la mécanique des fluides numérique. *Comptes-Rendus de Mécanique* (2009).

[26] Moureau, V., Minot, P., Bérat, C., and Pitsch, H. A ghost-fluid method for Large-Eddy Simulations of premixed combustion in complex geometries. *Journal of Computational Physics 211* (2007), 600–614.

[27] Nabben, R., and Vuik, C. A comparison of deflation and coarse grid correction applied to porous media flow. *SIAM Journal on Numerical Analysis* (2004).

[28] Nabben, R., and Vuik, C. A comparison of deflation and the balancing preconditioner. *SIAM Journal on Scientific Computing 27*, 5 (2006), 1742–1759.

[29] Nicolaides, R. A. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis 24*, 2 (April 1987).

[30] Parks, M. L., Sturler, E. D., Mackey, G., Johnson, D. D., and Maiti, S. Recycling krylov subspaces for sequences of linear systems. Tech. rep., SIAM J. Sci. Comput, 2004.

[31] Pellegrini, F. Distillating knowledge about SCOTCH. In *Combinatorial Scientific Computing* (Dagstuhl, Germany, 2009), U. Naumann, O. Schenk, H. D. Simon, and S. Toledo, Eds., no. 09061 in Dagstuhl Seminar Proceedings, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[32] Reid, J. K. On the method of conjugate gradients for the solution of large sparse systems of linear equations. *Large Sparse Sets of Linear Equations (London and New York)* (1971), 231–254.

[33] Rivara, M. C. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis 21* (1984).

[34] Roux, S., Lartigue, G., Poinsot, T., Meier, U., and Bérat, C. Studies of mean and unsteady flow in a swirled combustion using experiments, acoustic analysis and large-eddy simulations. *Combustion and Flame 141* (2005), 40–54.

[35] Saad, Y., Yeung, M., Erhel, J., and Guyomarc'h, F. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing 21* (2000), 1909–1926.

[36] Schmidt, E. Title unknown. *Rendiconti del Circolo Mathematico di Palermo 25* (1908), 53–77.

[37] Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain. Available at http://www.cs.cmu.edu/q̃uake-papers/painless-conjugate-gradient.pdf, 1994.

[38] Sleijpen, G. L., and Fokkema, D. R. BiCGStab(L) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions on Numerical Analysis 1* (September 1993), 11–32.

[39] Stiefel, E. Über einige methoden der relaxationsrechnung. *Zeitschrift für Angewandte Mathematik und Physik 3*, 1 (1952), 1–33.

[40] Tang, J., Nabben, R., Vuik, C., and Erlangga, Y. Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods. *SIAM Journal on Scientific Computing 39* (2009), 340–370.

[41] Trottenberg, U., Oosterlee, C. W., and Schuller, A. *Multigrid*. Academic Press, 2001.

[42] Vermolen, F. J., Vuik, C., and Segal, A. Deflation in preconditioned conjugate gradient methods for finite element problems. In *DIAM Annual Reports, available at http://ta.twi.tudelft.nl/TWA_Reports/02/02-10new.pdf* (2002).

[43] Vorst, H. A. V. D. Bi-CGSTAB : A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific Computing 13* (1992), 631–644.

[44] Vorst, H. A. V. D. Parallel iterative solution methods for linear systems arising from discretized PDE's. In *Lecture Notes on Parallel Iterative Methods for discretized PDE's. AGARD Special Course on Parallel Computing in CFD, available from http://www.math.ruu.nl/people/vorst/agard.ps.gz* (1995).