Chapter 3

Sequential Additive Correction Multigrid

3.1 Multigrid Methods for CFD applications

3.1.1 Introduction

Multigrid algorithms, introduced in the 60s by Russian scientists, [16, 17] started to be important in the 80s, mainly after the works by Brandt (e.g., [18, 19, 112, 113]). They are one of the most efficient approaches to solve PDE, at least for sequential computers. Quoting Brandt: "The amount of computational work (in MG algorithms) should be proportional to the amount of real physical changes in the computed system" [112]. This is the so-called Textbook Multigrid Efficiency (TME). In practice, this theoretical optimal can be difficult to obtain, but nevertheless MG algorithms are certainly efficient for large-scale problems (see, for instance, the conclusions of [8]). In this section, an overview of the method is given and Additive Correction Multigrid algorithm is described and benchmarked using different examples. An overview of the coupled ACM algorithms is given. Both approaches are compared.

3.1.2 Overview

Coarse grid correction algorithm

A short description of a generic *geometric*[1] MG algorithm is given to provide a framework to discuss different aspects of the method[2]. For a more detailed exposition, the reader is referred to [114], where an excellent and easy to read introduction to the method is provided.

We consider a general linear PDE equation:

$$Lu\left(\mathbf{x}\right) = f\left(\mathbf{x}\right) \qquad \left(\mathbf{x} \in \Omega\right) \tag{3.1}$$

where f is a known function, u is the unknown and L is a differential operator. A typical (easy) example is the two dimensional Poisson equation:

$$\mathbf{x} = (x_1, x_2) \qquad L = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} \tag{3.2}$$

Our goal is to find u^1, a discrete approximation to u, defined over a grid Ω^1. To do so, after spatial discretization, the following linear algebraic equation has to be solved:

$$L^1 u^1 = f^1 \tag{3.3}$$

[1] The difference between geometric and algebraic MG is discussed in section 3.1.3

[2] Here, the terminology used in *classic* MG papers has been used. Notation in sections 3.2 and 7 is different.

where f^1 and L^1 are respectively discrete approximations of f and L. f^1 is a vector with N^1 components and L^1 is a sparse $N^1 \times N^1$ matrix, where N^1 is the number of unknowns of the grid Ω^1.

Assume that we use an iterative algorithm (any of the classical methods described in section 2.7.3) to solve equation (3.3), with an arbitrary initial guess to start the iterations. To be more precise, here the algorithm is not used as a solver but as a *smoother*: it is not expected to solve equation (3.3) (this would take too long), but just to reduce the high-frequency components of the initial guess (section 2.7.3).

Let $\tilde{u}^1$ be the approximation available after ν_1 iterations (typically, when the algorithm is close to stagnation). We introduce v^1, the *error* of $\tilde{u}^1$, as:

$$v^1 = u^1 - \tilde{u}^1 \tag{3.4}$$

where u^1 is the exact *discrete* solution of equation (3.3). v^1 is the error of the algebraic approximation (or *convergence* error), not to be confused with the *discretization* error $u - u^1$. Applying L^1 operator to equation (3.4), we obtain an expression for it:

$$L^1 v^1 = r^1 \tag{3.5}$$

where

$$r^1 = f^1 - L^1 \tilde{u}^1 \tag{3.6}$$

is the *residual* of $\tilde{u}^1$.

Equation (3.5) can in principle be solved to evaluate exactly v^1 and then correct $\tilde{u}^1$. But this would be as difficult as solving the original equation (3.3), so there is nothing to be gained by doing so.

Instead, we will find v^2, an approximation of v^1, defined in a coarser mesh Ω^2, that we also suppose available. The distance between the nodes in Ω^2 is typically double than in Ω^1, so N^2, the number of unknowns of equation (3.9) is approximately $\frac{N^1}{2^d}$, where N^k is the number of unknowns of Ω^k and $d = 2, 3$ is the dimension of Ω. To do so, we form the equation:

$$L^2 v^2 = f^2 \tag{3.7}$$

where

$$f^2 = I_1^2 r^1 \tag{3.8}$$

L^2 is the approximation to L over the coarser mesh Ω^2 and I_1^2 is a fine-to-coarse transfer operator, called *restriction*.

Imagine that, by any means, we can solve equation (3.7) exactly, to obtain $v^2 = \left(L^2\right)^{-1} f^2$. We use it as a correction to $\tilde{u}^1$.

$$\tilde{u}^1 \leftarrow \tilde{u}^1 + I_2^1 v^2 \tag{3.9}$$

where I_2^1 is a coarse-to-fine *interpolation* or *prolongation* operator. Different types of restriction and prolongation operators can be used. Examples can be found in section 5.

Even if we use the exact solution of equation (3.7), as $I_2^1 v^2$ is an approximation of v^1, we do do not have yet the exact solution u^1. After the correction, starting from the corrected $\tilde{u}_1$, ν_2 iterations of the relaxation method are done. This step is called *post-relaxation*, while the first relaxation (before the correction) is called *pre-relaxation* [3]. Each iteration of the MG algorithm consists of this three-stages process (pre-relaxation, correction and post-relaxation).

[3] Or pre-smoothing and post-smoothing.

If we assume that equation (3.7) can be solved exactly, this two-grid correction algorithm enhances dramatically the behavior of the iterative solver. This enhancement is related to the reasons that cause the stagnation of the iterative solvers. Iterative solvers are effective to reduce the high frequency components of v^1, the initial guess error, as shown in section 2.7.3. After a few sweeps, only the lower frequency components of v^1 remain and the solver becomes inefficient. In this point, the coarse-grid correction algorithm stops the iterations and corrects the u^1 field. The correction, that is based on a extrapolation of v^2, decreases the lower frequency components of v^1. As the information that it contains has been obtained from a lower resolution discretization, it increases again the higher frequency components. However, this is not a problem as these high frequency components will be easily eliminated by the smoother.

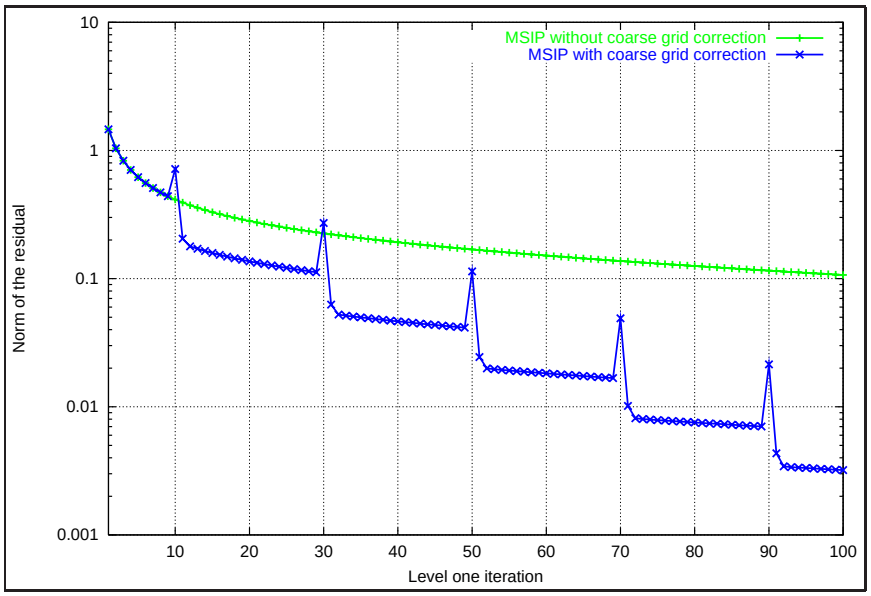

Figure 3.1: Residual evolution with and without coarse grid correction.

As an example, the problem model (section 2.7.3), with $N = 320^2 \approx 10^5$ has been solved with ILU (MSIP) [108] with and without coarse grid correction. Coarse grid correction equation has been obtained using an ACM algorithm (section 3.2.2). An exact LU solver (section 2.7.2) has been used for level 2. The residual evolution of both algorithms has been represented versus the number of iterations in Fig. 3.1[4]. In this example, after the correction the residual increases and then it is reduced effectively by the smoother. After a few sweeps, it is again stalled. For this example, to emphasize the behavior of the algorithm, the number of pre and post-smoothing iterations is 10. However, it can be seen that with less smoothing iterations the convergence ratio per global iteration would be similar at a lower CPU cost.

Multigrid algorithms

Consider now how can we solve equation (3.7). The same iterative method could in principle be used, but if N^1 is too large to solve equation (3.3) with an iterative method, the solution of equation (3.7) is also expensive (see Fig. 2.5).

In multigrid algorithm, to solve level 2, the method used for level 1 is invoked again. After ν_1^2 relaxation sweeps to solve equation $L^2 v^2 = f^2$ the equation $L^3 v^3 = I_2^3 r^2$ is formed, and so on until a level M is reached, in which the last equation

$$L^M v^M = f^M \tag{3.10}$$

[4]The expression "level one iteration" is used in the figures of this section to denote each of the smoothing iterations done on the fine mesh equation.

with N^M unknowns can be easily solved, either directly or iteratively. The number of smoothing sweeps can be different in each level. This recursive definition of the MG algorithm is reformulated in more precise terms in section 3.2.2.

The functions (or components) of the MG algorithm outlined are:

- A function to generate the mesh Ω^k and a function to discretize L to obtain L^k in Ω^k, for k from 1 to M *or* a method to obtain the matrix L^{k+1} from L^k, for k from 1 to $M-1$. Each approach originates a different flavor of MG (section 3.1.3).
- An iterative algorithm (or *smoother*) suitable for the equations of all the levels. Usually, all the matrices L^k have the same structure, differing only in N^k, so the same function can be used for all of them.
- A restriction (fine to coarse) I_k^{k+1} operator.
- A prolongation (coarse to fine) I_{k+1}^k operator.
- A MG engine to call the different components, control the number of iterations to be done by the smoother, etc.

3.1.3 A classification of MG methods for CFD applications

There is such an amount of literature published about MG in general and MG for CFD in particular, that it is a difficult task to keep a general point of view about the subject. With this aim, the following classification is hoped to be useful.

CFD-oriented MG algorithms can be classified according to:

- The method used to obtain the discrete operators L^k: Geometric or Algebraic MG.
- The treatment of the pressure-velocity coupling: Segregated MG or Coupled MG.
- The mesh type: Structured or Unstructured MG.
- The correction equations: Correction Scheme (CS) or Full Approximation Storage (FAS) MG.
- Treatment of the non-linearities: Linear MG or Non-linear MG.
- Control of the smoothing iterations: Fixed or adaptative number of iterations.
- Generation of the initial guess: Use of FMG or not to generate initial guesses.

Additional classification criteria could be introduced such as MG components used, cycle types (V,W), etc. The previous items are discussed in next paragraphs.

Geometric versus Algebraic MG

From the code development point of view, this is probably the most important difference. Both are used in this work. In the overview of the algorithm presented in section 3.1.2 nothing was said about how the family of discrete approximations L^k where obtained. There are two classes of algorithms:

- In the *geometric* or *classic* MG methods, the emphasis is in the solution of equation (3.1) rather than equation (3.3). Discretization and relaxation are seen as part of the same numerical solution process. It is interesting to quote Brandt's paper [18, section 1]: "The purpose of the work reported here is to study how to **intermix discretization and solution processes**, thereby making both of them orders-of-magnitude more effective".

- In *algebraic* multigrid the aim is exactly the opposite: while keeping the efficiency as high of possible, the goal is to separate as clearly as possible the discretization of L and the solution of the discrete equation $L^1 u^1 = f^1$. The discretization process is usually the most complex (or at least the larger) component of the CFD code, as it depends on many details such as heterogeneity of the domain, the use of different physical models and numerical schemes, etc (section 2.3). Thus, it is important to simplify it, generating only a single approximation to the continuous problem, using a single mesh rather than a sequence of grids which might not be available. This is difficult enough for a multipurpose CFD and heat transfer code. Then, MG algorithm generates the correction equations (L^k and f^k) from L^1, f^1 and $\tilde{u}^1$. Doing so, new physical models or discretization techniques can be introduced, without interfering with the MG algorithms.

Algebraic MG correction equations

A possible way to generate correction equations from the original equation begins with the expression (3.9) of the correction for $\tilde{u}^1$:

$$\tilde{u}^1 \leftarrow \tilde{u}^1 + I_2^1 v^2 \tag{3.11}$$

Where here we do not assume that v^2 is the exact solution of expression (3.7), but only a generic vector with N^2 components (while u^1 has N^1). Our goal now is to find a suitable expression for v^2. After the correction we would like $\tilde{u}^1$ to exactly satisfy:

$$L^1 \tilde{u}^1 = f^1 \tag{3.12}$$

Thus, to find an expression for v^2 it seems reasonable to impose:

$$L^1 \left(\tilde{u}^1 + I_2^1 v^2 \right) = f^1 \tag{3.13}$$

Let $\tilde{r}^1$ be the residual after the correction. From the previous expression we obtain:

$$\tilde{r}^1 = f^1 - L^1 I_2^1 v^2 - L^1 \tilde{u}^1 \tag{3.14}$$

The condition that we would to like to impose to evaluate v^2 is,

$$\tilde{r}^1 = 0 \tag{3.15}$$

However, this is not possible (in general), as expression (3.15) has N^1 equations and v^2 only $N^2 < N^1$ unknowns. All we can do is impose only N^2 conditions. To do so, we impose that the *restriction* of $\tilde{r}^1$ has to be null:

$$I_1^2 \tilde{r}^1 = 0 \tag{3.16}$$

By simple manipulation of equation (3.16) we obtain:

$$I_1^2 \left[f^1 - L^1 I_2^1 v^2 - L^1 \tilde{u}^1 \right] = 0 \tag{3.17}$$

$$I_1^2 L^1 I_2^1 \tilde{v}^2 = I_1^2 \left(f^1 - L^1 \tilde{u}^1 \right) \tag{3.18}$$

$$L^2 \tilde{v}^2 = I_1^2 r^1 \tag{3.19}$$

This is equivalent to the geometric MG correction equation (3.7) if

$$L^2 = I_1^2 L^1 I_2^1 \tag{3.20}$$

Expression (3.20), that can also be obtained using other methods, is called *Coarse grid Galerkin Approximation* or CGA. If equation (3.20) is used, MG can be used as an algorithm for the solution of *linear* equations rather than PDEs. This is the *Algebraic MG* (AMG) approach.

In spite of the formal similitude, both approaches can generate different algorithms, as CGA operator is not necessarily equal to the discretization of L in the mesh Ω^2. Depending on the operators I_1^2 and I_2^1, the matrix obtained with equation (3.20) can have a wider stencil. This could be acceptable in algorithms with $M = 2$, but not in general, as then the stencil would increase progressively in the different levels.

Strictly speaking, an AMG method should be usable to solve *any* linear equation. In practice, for the sake of efficiency, many AMG implementations assume an algebraic structure for the matrix, like that it arises from the discretization of a PDE in determinate type of mesh.

The difference between algebraic and geometric MG is important: Geometric MG is a method for solving PDEs while AMG is a method for solving linear equations. For instance, it can be used to solve a linear equation with no direct physical interpretation or as a subroutine of a more complex method (e.g., Schur complement, section 6). AMG is totally independent of the discretization details, as long as they do not change the algebraic structure of the equation. This important aspect is not hallways stressed. For instance [54], includes ADI and geometric MG in its classification of methods for the solution of linear equations. In both cases, the exposition begins with a PDE that is to be discretized.

As knowledge of CFD grows, it is becoming harder to master all the stages from the formulation of governing continuous equations to the solution of the linear discrete equations, and then to apply the algorithms to engineering problems. It seems convenient to prefer solution methods that allow to split the global process in different stages, allowing the different stages to be treated, if necessary, as black boxes with well-defined inputs and outputs. Thus, for people working to enhance a certain stage of the algorithm (or the computer code) it would be enough with a superficial knowledge of the other stages. This is an argument in favor of algebraic multigrid, as it can be considered *only* as a solution algorithm for the linear systems of equations.

In an AMG-based CFD code, MG is just a function to solve the equation systems, that can be replaced by any other linear solver. However, in a geometric-MG code, it has to be in a higher hierarchical level, calling all the other components, including the discretization function.

Segregated versus Coupled MG

MG can be used to solve a single PDE or a system of coupled PDEs. For the case of NS equations, the coupling is usually restricted to continuity and momentum equations (section 2.6) but MG algorithms for the coupled solution of all the unknowns have also been devised [97]. If a segregated approach is used, MG is the solver for each of the scalar fields. The segregated approach is usually better for time accurate problems (section 3.4.1).

Structured versus Unstructured mesh

For the case of structured meshes, the operator L^k is a penta or hepta-diagonal matrix. It is reasonable to keep the 5 or 7 point stencil constant for the next matrix L^{k+1}, as otherwise the size of the stencil would be increasing. Additionally, if the algebraic structure of the matrix is kept constant, the same smoother can be used for all the levels.

For the case of unstructured meshes, there is no equation pattern to be preserved to the next level. The coarse grid mesh is usually generated joining adjacent control volumes. This process is called *agglomeration* [115, 116, 117]. An example, using ACM, is given in section 3.2.

Correction Scheme (CS) versus Full Approximation Scheme (FAS)

The algorithm described in section 3.1.2 uses the Correction Scheme (CS). The name means that the coarse mesh unknown v^{k+1} is intended to approximate the correction to be added to the currently

available approximation $\tilde{u}^k$. It is not an approximation of the solution but an approximation of its correction. Instead, Full Approximation Scheme (FAS) uses an approximation to the solution as the coarse mesh unknown, $\tilde{u}^{k+1}$:

$$\tilde{u}^{k+1} = \hat{I}_k^{k+1}\tilde{u}^k + v^{k+1} \tag{3.21}$$

where $\hat{I}_k^{k+1}$ is a fine-to-coarse transfer operator (not necessarily equal to I_k^{k+1}). This coarse-grid variable $\tilde{u}^{k+1}$ approximates the restriction of the solution to the coarse mesh, $\hat{I}_k^{k+1}$. The FAS coarse grid equations (that can be obtained from equation (3.7) and (3.21) are

$$L^{k+1}\tilde{u}^{k+1} = \hat{f}^{k+1} \tag{3.22}$$

where

$$\hat{f}^{k+1} = L^{k+1}\left(\hat{I}_k^{k+1}\tilde{u}^k\right) + I_k^{k+1}r^k \tag{3.23}$$

Equation (3.22) is solved, starting with $\hat{I}_k^{k+1}\tilde{u}^k$ (instead of 0) as initial approximation. Its approximate solution, $\tilde{u}^{k+1}$ is used to correct the currently available approximate solution $\tilde{u}^k$:

$$\tilde{u}^k \leftarrow \tilde{u}^k + I_{k+1}^k\left(\tilde{u}^{k+1} - \hat{I}_k^{k+1}\tilde{u}^k\right) \tag{3.24}$$

This expression is the equivalent to equation (3.9).

Linear versus Non-linear MG

If a non-linear PDE is to be solved using MG, there are two possibilities.

1. Use a linearisation technique (section 2.5) and then a linear MG algorithm to solve each of the linear iterations. For instance, a Newton linearization of L^k around the current approximation $\tilde{u}^k$ can be done, and then CS can be applied to the linear problem. As another example, DPC does a standard linearization of the equations [57] and then uses ACM as a linear solver.

2. Use a non-linear MG algorithm [112, section 8.3]. FAS has to be used to solve non-linear equations with multigrid. For linear equations, CS and FAS are totally equivalent. However, for non-linear equations, L^1 is a function of u^1. As the coarse grid unknown (v^2) is not an approximation to the fine grid unknown (u^1) but to its correction (v^1), the CS correction equation matrix $L^2\left(v^2\right)$ would not be an approximation to the fine grid matrix $L^1\left(u^1\right)$. FAS, which uses an approximation to the fine grid solution as unknown, can be directly applied to non-linear problems.

Fixed versus adaptative number of iterations

The number of pre-smoothing or post-smoothing iterations to be done in each level can be fixed or it can be modified according to the residual and the rate of residual reduction. The optimal choice is strongly dependent of the case considered. This is discussed in section 3.2.2. For parallel MG algorithms, the evaluation of the norm of the residual implies a global communication, and this is an additional argument in favor of a fixed number of iterations.

Use of FMG to generate initial guesses

In the Full-Multigrid (FMG) variants of the algorithm, an approximate initial guess to the solution of the finest level equation is first obtained from a coarser mesh. This is, we start by forming and solving the coarsest level equation, $L^M v^M = f^M$. Its solution is prolongated to level $M-1$ and then improved using MG. This process goes on until level 1 is reached. When the (costly) iterations on level 1 begin, a good initial guess is already available.

3.1.4 Examples of MG applied to CFD

Examples of all the possible combinations of MG algorithms for CFD would probably be found within the vast amount of papers devoted to the subject. With no aim of being exhaustive, a few examples of MG algorithms applied to CFD and heat transfer are given in Tables 3.1 and 3.2.

	Comments
[118]	Conventional SIMPLE algorithm, enhanced with a classic multigrid algorithm (V cycle) for the solution of the pressure correction equation, with Red-Black Gauss-Seidel as smoother.
[119]	Classic MG algorithm, with FAS, used to solve the heat conduction equation with different smoothers.
[120]	FAS-FMG classic multigrid solver, with a segregated smoother, is used. Impressive convergence ratios are obtained.
[121]	A mixed conduction - convection problem is solved using a classic FAS-FMG multigrid algorithm. The paper is the only example known to the author of the application of a classic MG algorithm to a domain with a solid-fluid discontinuity. This is not a problem for ACM but it is an important difficulty to the restriction-prolongation operators needed by classic MG. In the paper, this difficulty is overcome avoiding interpolations near the discontinuity.
[122] [123]	Both (separately) solved NS equations using a curvilinear collocated mesh, with a segregated FAS-FMG algorithm.
[124]	A classic multigrid algorithm, with a CS for a time-accurate NS solver. It is pointed out that, for time accurate solutions, the advantage of MG is the solution of Poisson equation. This is confirmed by the results obtained in sections 3.3.3 and 3.4 of this work.
[125] [126]	FAS-FMG classic multigrid solver, with a coupled (SCGS) smoother, used for different laminar flow problems.
[127]	Streamfunction - vorticity formulation used to solve the NS equations with a classic coupled multigrid, using a ILU smoother.
[98]	A comparison between a conventional pressure-correction algorithm and a coupled FAS algorithm, using a 2nd order discretization scheme.

Table 3.1: Examples of geometric multigrid algorithms for CFD applications.

3.2 Additive Correction Multigrid

ACM algorithm was introduced by Hutchinson and Raithby [137]. It is an algebraic MG algorithm, CS-based, with simplified restriction/prolongation operators. The algebraic structure of the correction equations (penta- or hepta- diagonal for two or three dimensional problems) is maintained equal to the structure of the original equations. It has been shown that it is equivalent to a cell-centered geometric MG algorithm, using restriction and prolongation operators based on piecewise constant interpolation [138].

If these operators are introduced in expression (3.20), the ACM algorithm can be obtained. However, the description of the algorithm given in [137] is totally different (similar to [57, 59]), so at first glance ACM might seem a quite different algorithm.

Its main features are its robustness and its ease of implementation in pre-existent codes: it can be used for a variety of problems (independently of the discretization details), obtaining convergent solutions without any modification and improving dramatically the performances of an original code

	Comments
[128]	A segregated algebraic MG algorithm, with CGA, for the NS equations on non-structured meshes.
[129]	An algebraic (ACM) algorithm is used to solve heat conduction equation in compact heat exchangers, using a blocking-off [57] technique.
[130]	An algebraic (ACM) algorithm is used to solve the convection-diffusion equation on a non-structured grid. GS and PCGS ([131]) are used as smoothers. In both cases, the use of ACM provides better convergence ratios, while PCGS used as a solver is only slightly better that GS.
[132] [133] [134] [135]	Coupled, algebraic (ACM) multigrid algorithms, with SCGS as smoother.
[46]	Streamfunction - vorticity formulation used to solve the NS equations with a ACM multigrid, with a coupled ILU smoother.
[136]	Coupled algebraic MG used in a staggered curvilinear mesh. The following quote from their paper is relevant to the role of algebraic multigrid algorithms, ".. from our point of view of software development it is very attractive to separate the discretization and solution phases. Therefore we use linear multigrid method inside an outer Newton linearization, together with Galerkin coarse grid approximation (CGA), so that changes in the discretization leave the solver completely unaffected". As a smoother, a variant of SCGS [120] is used, called CILU, that allows a better treatment of the linear coefficients arising from mixed derivatives.
[81]	A coupled algebraic (CGA) MG algorithm is used to solve NS equations with unstructured grids.

Table 3.2: Examples of algebraic multigrid algorithms for CFD applications.

based on classic iterative solvers.

A derivation of the ACM correction equations will be given using a notation similar to [137] and [134] and then the MG algorithm will be presented.

3.2.1 Additive Correction Equations

Consider a conventional iterative linear smoother (section 2.7.3) and a two-dimensional[5] linear equation $A\phi = b$, expressed using the notation of [137]:

$$a^p_{i,j}\phi_{i,j} = a^w_{i,j}\phi_{i-1,j} + a^e_{i,j}\phi_{i+1,j} + a^s_{i,j}\phi_{i,j-1} + a^n_{i,j}\phi_{i,j+1} + b_{i,j} \tag{3.25}$$

The only property needed for equation (3.25) is its "compatibility" with the iterative solver (i.e., diagonal dominant for GS or Line-GS), so the iterative algorithm could in principle solve it, but with an exceedingly large number of iterations.

Starting with any initial guess, after a number of iterations, the algorithm is stalled. Let ϕ be the approximation available[6].

Then, a *block* partition of the original equations is defined. Usually (but not necessarily) the blocks are formed joining adjacent equations. Assume that groups of (2x2) equations are joined. The integer vectors (i, j) and (I, J) are used respectively to index the original equations and the

[5] The extension to three dimensions is straightforward.

[6] Note that here, unlike in section 3.1.2, ϕ is used to refer to the unknown as well as to the approximation.

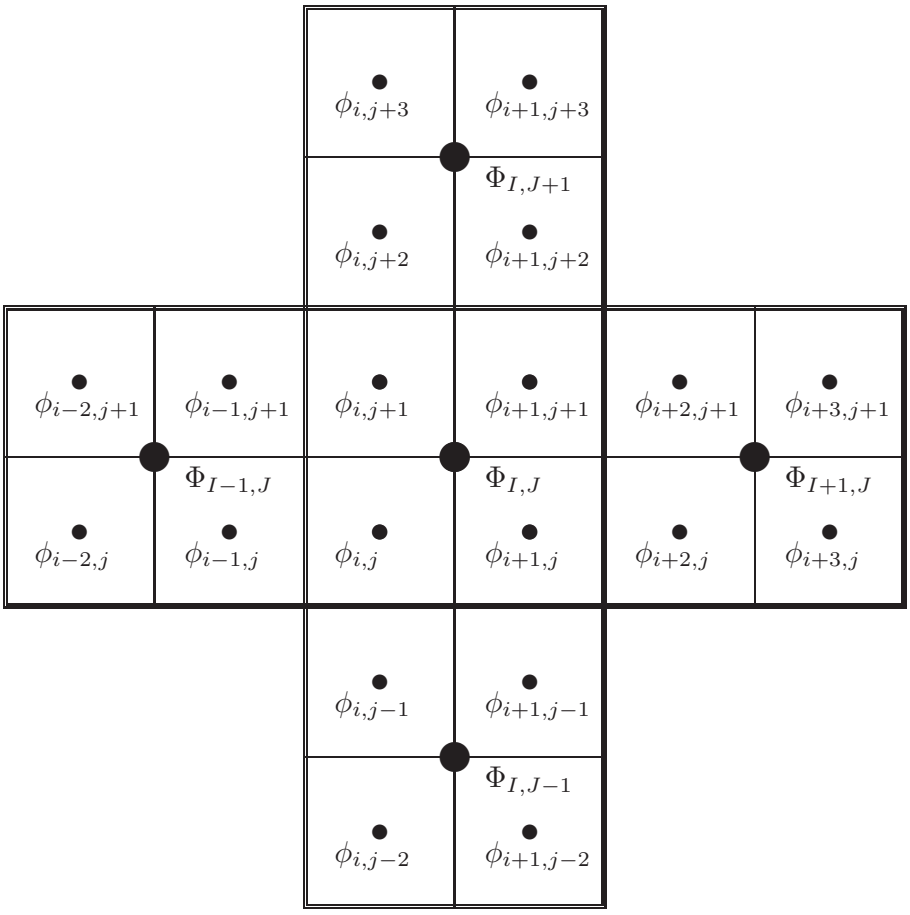

Figure 3.2: Disposition of blocks of control volumes in ACM.

blocks, respectively. Fig. 3.2 represents the algebraic structure of a portion of equation (3.25), as well as the blocks.

The residual vector r, defined as in equation (3.6) is:

$$r_{i,j} = b_{i,j} - a^p_{i,j}\phi_{i,j} + a^w_{i,j}\phi_{i-1,j} + a^e_{i,j}\phi_{i+1,j} + a^s_{i,j}\phi_{i,j-1} + a^n_{i,j}\phi_{i,j+1} \tag{3.26}$$

The sum of all the residuals of the block yields a *block residual* $R_{I,J}$,

$$\begin{aligned} &R_{I,J} = \sum r_{i,j} = \\ &\sum \left(b_{i,j} - a^p_{i,j}\phi_{i,j}\right) + \\ &a^w_{i,j}\phi_{i-1,j} + a^e_{i,j}\phi_{i+1,j} + a^s_{i,j}\phi_{i,j-1} + a^n_{i,j}\phi_{i,j+1} + \\ &a^w_{i+1,j}\phi_{i,j} + a^e_{i+1,j}\phi_{i+2,j} + a^s_{i+1,j}\phi_{i+1,j-1} + a^n_{i+1,j}\phi_{i+1,j+1} + \\ &a^w_{i,j+1}\phi_{i-1,j+1} + a^e_{i,j+1}\phi_{i+1,j+1} + a^s_{i,j+1}\phi_{i,j} + a^n_{i,j+1}\phi_{i,j+2} + \\ &a^w_{i+1,j+1}\phi_{i,j+1} + a^e_{i+1,j+1}\phi_{i+2,j+1} + a^s_{i+1,j+1}\phi_{i+1,j} + a^n_{i+1,j+1}\phi_{i+1,j+2} \end{aligned} \tag{3.27}$$

where the sum is extended to all the equations of the block.

An *additive correction* Φ, restricted to be constant in each block, will be used to improve the current approximation ϕ,

$$\phi \leftarrow \phi + \Phi \tag{3.28}$$

this is,

$$\phi_{i,j} \leftarrow \phi_{i,j} + \Phi_{I,J} \tag{3.29}$$

$$\phi_{i+1,j} \leftarrow \phi_{i+1,j} + \Phi_{I,J}$$
$$\phi_{i,j+1} \leftarrow \phi_{i,j+1} + \Phi_{I,J}$$
$$\phi_{i+1,j+1} \leftarrow \phi_{i+1,j+1} + \Phi_{I,J}$$

The same holds for the other blocks. For example,

$$\begin{aligned}
&\phi_{i-2,j} \leftarrow \phi_{i-2,j} + \Phi_{I-1,J} \\
&\phi_{i-1,j} \leftarrow \phi_{i-1,j} + \Phi_{I-1,J} \\
&\phi_{i-2,j+1} \leftarrow \phi_{i-2,j+1} + \Phi_{I-1,J} \\
&\phi_{i-1,j+1} \leftarrow \phi_{i-1,j+1} + \Phi_{I-1,J}
\end{aligned} \tag{3.30}$$

To evaluate Φ, with the aim to improve ϕ, we will impose the block residual to be zero after the correction,

$$R_{I,J} = 0 \tag{3.31}$$

This equation is expanded using equations (3.27) and (3.28),

$$\begin{aligned}
&a^w_{i,j}\left(\phi_{i-1,j} + \Phi_{I-1,J}\right) + a^e_{i,j}\left(\phi_{i+1,j} + \Phi_{I,J}\right) + \\
&a^s_{i,j}\left(\phi_{i,j-1} + \Phi_{I,J-1}\right) + a^n_{i,j}\left(\phi_{i,j+1} + \Phi_{I,J}\right) + b_{i,j} - a^p_{i,j}\left(\phi_{i,j} + \Phi_{I,J}\right) + \\
&a^w_{i+1,j}\left(\phi_{i,j} + \Phi_{I,J}\right) + a^e_{i+1,j}\left(\phi_{i+2,j} + \Phi_{I+1,J}\right) + \\
&a^s_{i+1,j}\left(\phi_{i+1,j-1} + \Phi_{I,J-1}\right) + a^n_{i+1,j}\left(\phi_{i+1,j+1} + \Phi_{I,J}\right) + b_{i+1,j} - a^p_{i+1,j}\left(\phi_{i+1,j} + \Phi_{I,J}\right) \\
&a^w_{i,j+1}\left(\phi_{i-1,j+1} + \Phi_{I-1,J}\right) + a^e_{i,j+1}\left(\phi_{i+1,j+1} + \Phi_{I,J}\right) + \\
&a^s_{i,j+1}\left(\phi_{i,j} + \Phi_{I,J}\right) + a^n_{i,j+1}\left(\phi_{i,j+2} + \Phi_{I,J+1}\right) + b_{i,j+1} - a^p_{i,j+1}\left(\phi_{i,j+1} + \Phi_{I,J}\right) + \\
&a^w_{i+1,j+1}\left(\phi_{i,j+1} + \Phi_{I,J}\right) + a^e_{i+1,j+1}\left(\phi_{i+2,j+1} + \Phi_{I+1,J}\right) + \\
&a^s_{i+1,j+1}\left(\phi_{i+1,j} + \Phi_{I,J}\right) + a^n_{i+1,j+1}\left(\phi_{i+1,j+2} + \Phi_{I,J+1}\right) + b_{i+1,j+1} - \\
&a^p_{i+1,j+1}\left(\phi_{i+1,j+1} + \Phi_{I,J}\right) = 0
\end{aligned}$$

Rearranging , we obtain the correction equation $A\Phi = B$,

$$A^p_{I,J}\Phi_{I,J} = A^w_{I,J}\Phi_{I-1,J} + A^e_{I,J}\Phi_{I+1,J} + A^s_{I,J}\Phi_{I,J-1} + A^n_{I,J}\Phi_{I,J+1} + B_{I,J} \tag{3.32}$$

where the diagonals of the matrix A are:

$$\begin{aligned}
A^p_{I,J} &= \sum\left(a^{p_{i,j}}\right) \\
&- \left(a^e_{i,j} + a^n_{i,j} + a^w_{i+1,j} + a^n_{i+1,j} + a^e_{i,j+1} + a^s_{i,j+1} + a^w_{i+1,j+1} + a^s_{i+1,j+1}\right) \\
A^w_{I,J} &= a^w_{i,j} + a^w_{i,j+1} \\
A^e_{I,J} &= a^e_{i+1,j} + a^e_{i+1,j+1} \\
A^s_{I,J} &= a^s_{i,j} + a^s_{i+1,j} \\
A^n_{I,J} &= a^n_{i,j+1} + a^n_{i+1,j+1}
\end{aligned} \tag{3.33}$$

and the right-hand side term is:

$$B_{I,J} = \sum r_{i,j} \tag{3.34}$$

where the sums are extended to all the equations in the block.

Assume now that equation (3.32) is solved by any means. This aspect is discussed later. Then the additive correction (3.28) is done and the iterative solution is continued (as in the algorithm of section 3.1.2).

Remarks:

- Relation between ACM and CGA correction equations. As aforementioned, ACM can be considered as a particular case of classic MG [138]. It is also a particular case of a CGA

algebraic multigrid. The process outlined here is essentially like the derivation of the CGA expression in section 3.1.3. The matricial expressions for I_1^2 and I_2^1 that lead to the ACM correction equation (3.32) can easily be obtained. To do so it is useful to renumber the unknowns block by block.

- The ACM algorithm is based on the additive correction strategy [139], used as a MG algorithm. The method proposed in [139] is in summary a one-dimensional additive correction for each of the axis. This approach is by far less robust and efficient. Its main advantage is that the correction equations have a tri-diagonal structure so they can be solved efficiently with a TDMA algorithm: there is no need to use a hierarchy of correction equations.

- Equation (3.32) has to be formed and solved many times. However, note that the matrix coefficients (A^p, A^{nb}) in equation (3.32) depend *only* on the coefficients of the original equation (3.25), while the right hand side B depends *only* on the residuals of the approximation currently available[7]. Thus, the left-hand-side of equation can be evaluated just once at the beginning while the right-hand-side has to be updated for each iteration.

- Diagonal dominance is in principle not essential for ACM (provided that the iterative algorithm can solve non-diagonal dominant equations). However, if equation (3.25) is diagonal dominant, the same holds for expression (3.32). As it can easily be seen, if $a^p = a^w + a^e + a^n + a^s$ then $A^p = A^w + A^e + A^n + A^s$.

- Physical interpretation. ACM method is not based on a physic or geometric interpretation of equation (3.25). The only restriction for the method as here presented is its algebraic structure. However, it is interesting to point out that [137] if equation (3.25) is a balance of the fluxes of a certain quantity, then the equation (3.31), represents essentially an addition of all the balances in the block, i.e., a global balance over the block.

- As expected, the additive correction vector of the exact solution is null, as the residual vector is zero and so the term $B_{I,J} = 0$ (so the solution is not corrected).

- Equation (3.32), if solved exactly, forces the sum of the residuals of each block to be zero after the correction. However, the residual of each of the control volumes in the block is not, in general, null. In fact, it can even be larger than before the correction. But this residual is easily reduced with a few sweeps of an iterative smoother (as shown in the example of Fig. 3.1).

- Block sizes. In principle, any block size could be used. The deduction of the correction equations for blocks of generic size is straightforward. However, 2x2 is probably the optimal (and 2x2x2 in three dimensions). Even with 2x2 blocks, there are reasons to use other block sizes in certain parts of the system:

  1. If the number of control volumes in any direction in the mesh is odd, 2x1 and/or 1x2 control volume blocks are to be used. This is only a small inconvenience. It could be argued that the number of control volumes can be forced to be even in all the directions. But this condition would be too restrictive as a hierarchy of equations has to be used.
  2. If a parallel smoother is to be used, it is necessary to force the equations assigned to different processors not to be merged in a single block, as in DDACM (section 7).
  3. Blocks can be forced not to mix equations for boundary conditions and equations for inner control volumes. However, this technique has little effect over the performance of the solver.
  4. If the number of nodes in the different axis of the domain is very different, after a few levels in one of them only one or two nodes remain while in the other there may be still many nodes so it is not possible to generate a correction equation using 2x2 blocks. Then, 1×2 or 2×1 blocks are used to go on with the generation of correction equations.

[7] As L^2 CGA operator equation (3.20) only depends on L^1 and the right-hand-side of equation (3.19) only depends on r^1.

- Structure of the correction equations. Inspection of equation (3.32) reveals that the coefficients that relate $\Phi_{I,J}$ to its neighbours are the sum of the coefficients of the equations in block (I, J) that are related to equations belonging to the neighbouring blocks. As an example, $A^w_{I,J} = a^w_{i,j} + a^w_{i,j+1}$. This is because $a^w_{i,j}$ and $a^w_{i,j+1}$ are the only coefficients in the block (I, J) that affect the unknowns in block $(I-1, J)$.

- ACM for non-structured grids. The pattern of the correction equations discussed in the previous paragraph is maintained for equations with more complex algebraic structures, such as those arising from non-structured meshes. In non-structured meshes, as there is no algebraic structure to be preserved, the control volumes can be grouped arbitrarily to form blocks. It has been shown that the efficiency of the algorithm can be increased if the nodes with stronger links are grouped. Different algorithms have been proposed to do so, e.g., [115, 130, 140].

  An example of the application of ACM for the solution of scalar problem with unstructured meshes is presented in Fig. 3.3. A variant of the method described in [130] has been used. The problem solved is the transport of a passive scalar ϕ, using the velocity field obtained from the solution of a driven cavity problem with a structured mesh. In one of the nodes $\phi = 1$ is imposed (filled in red, at the bottom left of the domain), while $\phi = 0$ is used as a boundary condition. The velocity field used can be seen at the bottom of the figure.

  From the original structured mesh (not shown in Fig. 3.3), the control volumes with stronger links are grouped to obtain the first correction equation (top left). The process is repeated (top right, bottom left and bottom right) until only three equations remain. As an example, the correction equations in the area filled with yellow in the first correction are assigned to the yellow correction equation of the third correction map. The same holds for the blue area in levels 3 and 4. The red control volume, where the equation is $\phi = 1$ (independent of the neighbours values), remains alone. The velocity field (not directly used in the agglomeration process, but through the discretization coefficients) is at the bottom.

  Even using a structured mesh for the original equation, if the agglomeration is done to keep the nodes with a closer relationship in the same block, the correction equation (level 2) is no longer structured. The blocks follow the pattern of the streamlines. The same process goes on in next levels. The node with the trivial equation $\phi = 1$ is never joined with other nodes. In the last level, there are only three equations: the source node, the boundary conditions and the domain to be solved. The interesting point of this algorithm is that all this agglomeration process is done *without using the velocity field*, but just the discretization coefficients of the first level. The same algorithm can be directly applied to an initially non-structured mesh.

  Due to the non-structured nature of the equations, conventional ILU smoothing (e.g., MSIP) can not be directly used in this algorithm. In the example, Gauss-Seidel is used. In spite of the lower efficiency of the smoother, for high Pe numbers the unstructured approach has better convergence properties than the conventional structured ACM.

  One of the key points of unstructured ACM is the cost of the agglomeration algorithm. For equations with constant left hand side, it is worth spending CPU time in a good hierarchy of correction equations, but for the equations that change at each iteration it is important to use a fast agglomeration algorithm.

3.2.2 ACM Algorithm

To solve the equation (3.32), ACM algorithm is invoqued recursively.

Iterative solver. The iterative solver (Alg. 3.1) calls a basic smoother (such as Gauss-Seidel or ILU) until certain conditions are satisfied. The general idea is that this function will sweep repeatedly the domain until stagnation condition or a maximum number of iterations has been reached.

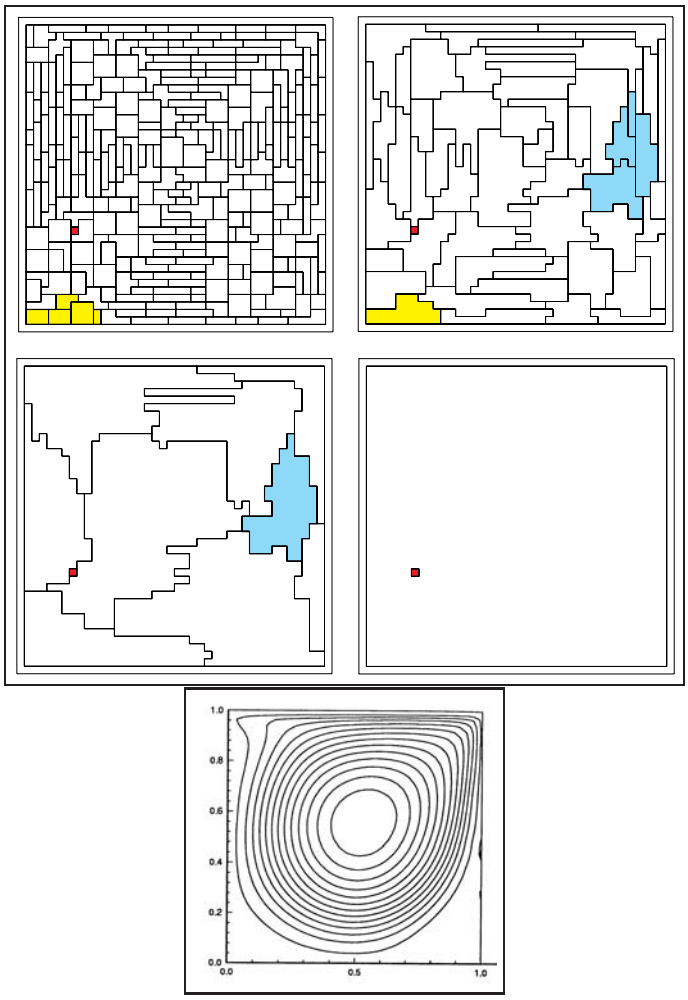

Figure 3.3: Non-structured agglomeration of control volumes.

```
iterate(A,φ,b,ϵ,n2,algorithm) {
        i=1 → n2 {
                algorithm(A,φ,b)
                r = b − Aφ
                nri = |r|
                if (nri < ϵ) break
                if (i ≥ 2 and nri−1/nri < β ) break
        }
        return(φ,r,nr)
}
```

Algorithm 3.1: ACM algorithm. Iterate.

Here, *algorithm* is the basic iterative algorithm (or smoother) to be called, such as Gauss-Seidel or ILU, ϵ is the precision to stop the iterations, n_2 is a maximum number of iterations, and β is control parameter that has to be set to the ratio of residual reduction where it is considered that the smoother has reached stagnation state. Typical β values are $(1.1 \cdots 3.0)$. After calling the smoother, the residual vector r and its norm nr are evaluated. The algorithm can be used to do an adaptative or a fixed or number of iterations:

- $n_2 = \infty, \epsilon = 0$. Adaptative, controlled with the ratio of residual reduction. The ratio between

the norms of the current and the previous residuals, $\frac{nr_{i-1}}{nr_i}$, is a measure of the efficiency of the algorithm. When it is smaller than β, the smoother is considered to be stalled and the iterations are stoped.

- $n_2 = \infty, \beta = 0$. Adaptative, controlled with the norm of the residual. If $\beta = 0$ the iterations are stopped only when the residual norm is smaller than ϵ, independently of the how poor the ratio of residual reduction might be.
- $n_2 > 0, \epsilon = 0, \beta = 0$. n_2 iterations are done independently of the residual evolution.

Single level solver. The core of the algorithm is the recursive solution function (Alg. 3.2), used to solve level l up to a precision ϵ.

```
solve_level(A,φ,b,l,ε) {
        correction=F(A,l)
        if (correction) A2=correction_lhs(A)
        i=1 → n1 {
                (φ,r,nr)=iterate(A,φ,b,ε,n2a,algorithma)
                if (nr < ε) break
                if (correction) {
                        b2=correction_rhs(r)
                        Φ=0
                        εl+1 = α nr
                        (Φ,nr2)=level(A2,Φ,b2,εl+1,l+1)
                        φ=additive_correction(φ,Φ)
                        (φ,r,nr)=iterate(A,φ,b,ε,n2b,algorithmb)
                        if (nr < ε) break
                }
        }
        return(φ,nr)
}
```

Algorithm 3.2: ACM algorithm. Recursive solution of one level.

The parameters of the algorithm are:

- $algorithm_a$, $algorithm_b$ are the pre and post-smoothing algorithms.
- α is the precision imposed for the solution Φ of the correction equation, relative to the residual of the current level.
- n_{2a}, n_{2b} are the maximum numbers of pre and post-smoothing iterations; n_1 is the maximum number of level $l+1$ cycles per level l cycle. The algorithms with $n_1 = 1$ are V cycles, while the algorithms with $n_1 > 1$ are W cycles.
- F is a boolean function that, depending on the size of the matrix A and on the level l, decides if a correction equation is needed or not. Different criteria can be used, like the number of nodes of the current level or the number of levels already used.
- correction_lhs and correction_rhs are functions that evaluate the left and right hand sides of the correction equation, using respectively expressions (3.33) and (3.34).
- additive_correction is a function that implements expression (3.28) to do the block correction.

The function solve_level also allows both static or dynamic criteria to control the number of iterations:

- To use a static criteria, $\alpha = 0$ should be imposed. In these conditions, the condition $nr < \epsilon$ is never satisfied in level $l+1$, so the iterations are stopped when $i = n_1$.
- To use a dynamic criteria, n_1 is set to a high value (i.e., $n_1 = 10$) and the iterations are stopped when the norm of the residual is below α times the residual of the current level. Low α values impose a high precision in the solution of the correction equation. Typical α values lie in the range $(0.1 \cdots 2.0)$.

For algorithms based on CS formulation (section 3.1.3), it is important to use $\Phi = 0$ as the initial guess of the correction. On a two-levels algorithm, post smoothing can be suppressed as it is equivalent to the pre-smoothing of the next iteration.

Global algorithm. The global algorithm (Alg. 3.3) just calls the level solver for level 1 with the desired precision. Additional features, such as relaxation and relative residual have also been included in the implementation.

```
ACM(A,φ,b,ε) {
                (φ,normr)=level(A,φ,b,1,ε)
                 return(φ,normr)
}
```

Algorithm 3.3: ACM algorithm. Main function.

Additional features. There are additional features in the implementation that have not been presented in the algorithms for clarity:

- Constant A matrices. As discussed in section 1.2, there are important examples of applications where matrix A is needed to solve for thousands of vectors. In these conditions, a substantial amount of CPU time can be saved noting that the hierarchy of correction matrices only depends on A and not on the right hand sides. Additionally, other data needed by the smoothers such as ILU decompositions for MSIP or a $\frac{1}{a^p_{i,j}}$ field for Gauss-Seidel are also stored and reused.
- ACM allows to combine iterative and direct solvers. In the implementation, it is possible to use a direct LU decomposition for a certain level, for instance when the number of equations is below a desired limit. In the example of Fig. 3.1. the decomposition was done for level two. LU decomposition is of interest only in the context of constant A matrices. This strategy is an important point in the DDACM algorithm (section 7). If the decomposition is done in the first level, ACM collapses to a direct LU solver.

3.3 Benchmarking ACM

3.3.1 Effect of the parameters on the convergence process

The goal of this section is to clarify the role of the main parameters of the generic ACM algorithm previously described. To do so, the problem model (section 2.7.1) is solved with different variations of the algorithm. In all the tests done in this section, the number of unknowns in each direction is a power of two. This is done only for convenience: the implementation allows any number of unknowns.

Dynamic cycles

The effect of the parameters α and β, introduced in the previous algorithms is clarified in next paragraphs using different examples.

Effect of β. In dynamic cycles, iterate function calls the basic smoother until it is stalled. This is controlled by the parameter β. To illustrate its effect over the behavior of ACM, the problem model has been solved with a two-levels algorithm. To emphasize the effect of β, the correction equation is solved to machine accuracy with a LU solver and no post-smoothing is done. MSIP is used as smoother. For a problem with $N = 128^2$, the evolution of the residual has been represented in Fig. 3.4 for different β values.

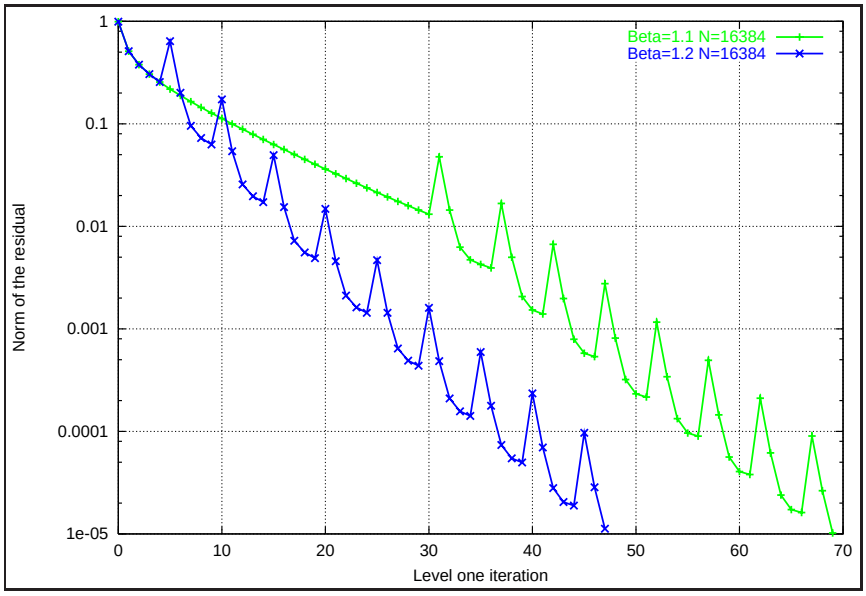

Figure 3.4: Evolution of the residual norm for a two levels ACM algorithm with different β values.

In the first group of iterations the residual evolution is the same for both algorithms. However, the case with $\beta = 1.1$ is more tolerant to the relatively low convergence rate of MSIP. If the cost of solving level 2 could be neglected, the algorithm with $\beta = 1.2$ would be clearly more efficient.

If the high frequency components are an important part of the error, the smoother can do a high number of iterations before stalling. The only high frequency components after the corrections are due to the extrapolation of the correction Φ to level 1, so only 5-6 iterations are done. This number depends on β and on the problem considered. The advantage of dynamic cycles is that they adjust themselves to a reasonable number of iterations per level, providing robustness (but not necessarily the optimal efficiency) to the algorithm. This is specially important for coupled solvers.

One of the parameters that influences the frequency distribution of the error and thus the number of smoothing iterations before stalling is the number of unknowns N. As an example, the previous experiment has been repeated for different problem sizes, with a fixed $\beta = 1.2$. The results can be seen in Fig. 3.5. For the smallest problem, the smoother does not even act as the problem is solved before stalling.

Effect of α. A two-level problem will be used to illustrate the effect of α. As the LU algorithm would solve the problem to machine accuracy (i.e., as if $\alpha = 0$), a full ACM algorithm (with multiple levels) has been used to allow a control of the residual of level 2. However, as in the previous paragraph, we concentrate on the analysis of the behavior of the first level: all we need to know about the rest is that level 2 has been solved to an accuracy $\epsilon^2 = \alpha\, nr$; the maximum number of iterations n_1 has been taken large enough to ensure that this condition is satisfied for all the α values considered. No post-smoothing is done $n_{2b} = 0$. To focus on the effect of α, a fixed $\beta = 1.2$ is used.

The result of the experiment, for $N = 16384$, is presented in Fig. 3.6. For an α value close to zero (10^{-2}), correction equation is solved accurately and the algorithm behaves almost like in the previous paragraph. However, to solve level 2 so accurately is expensive and it might be unnecessary.

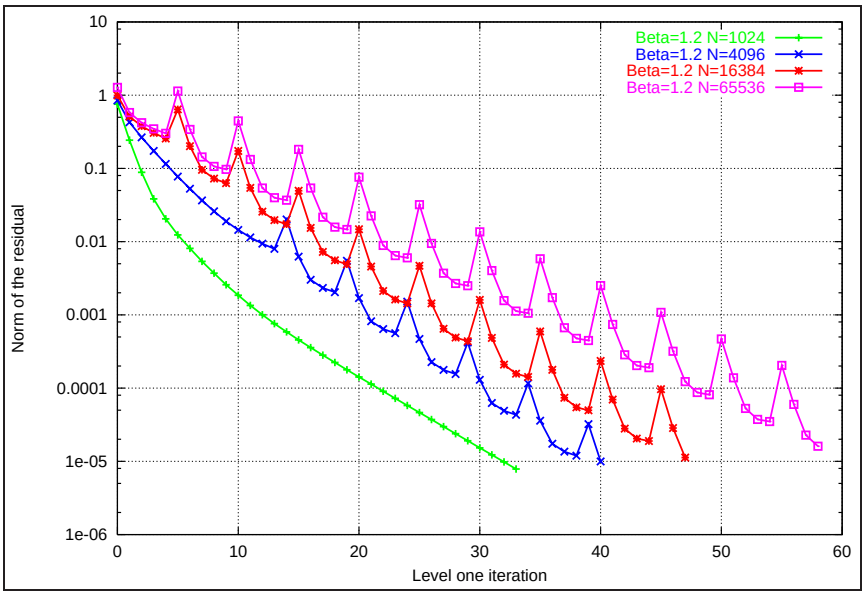

Figure 3.5: Evolution of the residual norm for a two levels ACM algorithm for different problem sizes and a fixed β value.

Increasing α, we can estimate the impact of the innacuracies in the solution of equation (3.32). If α is increased to 10^{-1} the behavior is similar, while for higher values the algorithm is degradated.

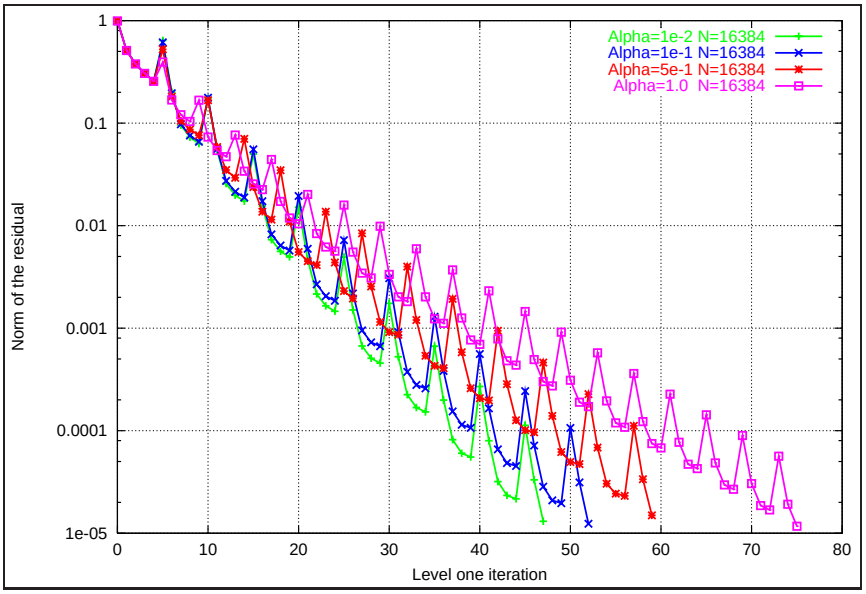

Figure 3.6: Evolution of the residual norm for a two level ACM algorithm with different α values.

Static cycles

Not necessarily the more sophisticated dynamic solutions are better: for a single scalar equation, better performance can usually be achieved with static control, if the parameters are tuned for each application. The main interest in dynamic cycles is the robustness, specially for coupled solvers. To use static cycles, $\alpha = \beta = 0$ must be imposed and the optimal values for n_1, n_{2a}, n_{2b} determinate experimentally.

As an example, the convergence histories for the problem model solved with different V and W static cycles have been presented in Figs. 3.7 and 3.8. For different reasons (section 5), V cycles are important for parallel MG, specially if pre-smoothing is suppressed. The behavior of such cycles can be seen in Fig. 3.8.

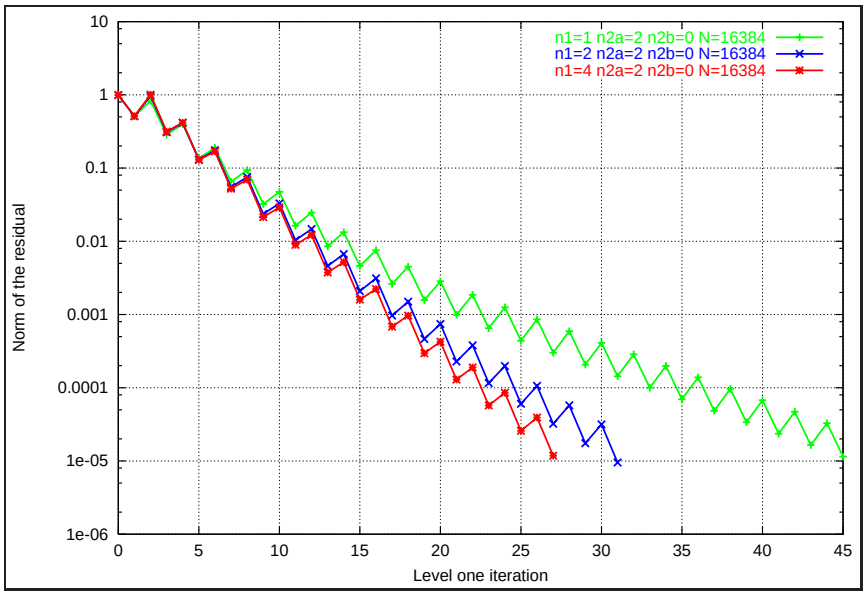

Figure 3.7: Convergence history of different V ($n_1 = 1$) and W ($n_1 > 1$) ACM cycles without post-smoothing ($n_{2b} = 0$).

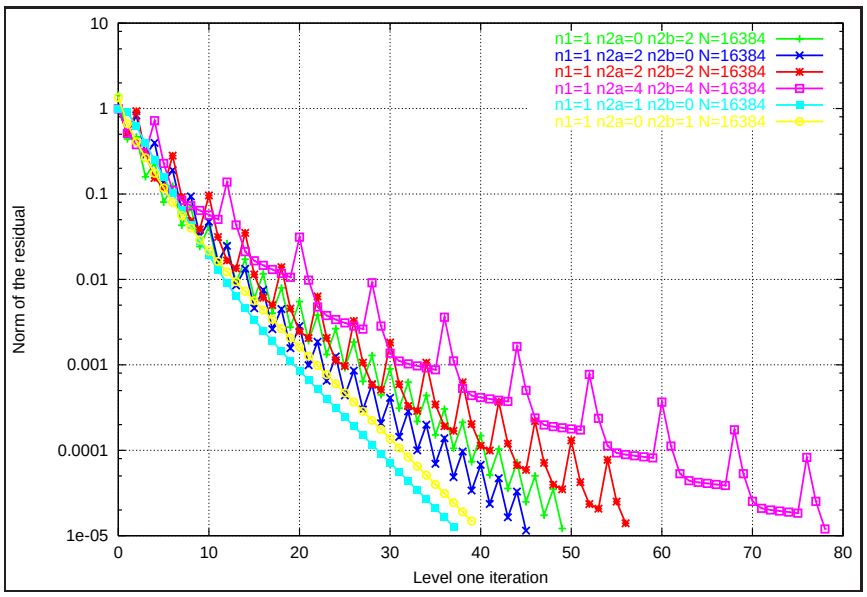

Figure 3.8: Convergence history of different V ACM cycles ($n_1 = 1$).

Effect of the frequency distribution of the error

The spatial frequency distribution of the error associated with the initial guess can be altered changing the parameters p_1 and p_2 of the problem model 2.7.1. As shown for an example in section 2.7.3, the smoother behavior is a function of the frequency distribution of the error.

The problem model with $N = 128 \times 128$ and different p_1 and p_2 has been solved with a $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$ static cycle. The convergence history is presented in Fig. 3.9.

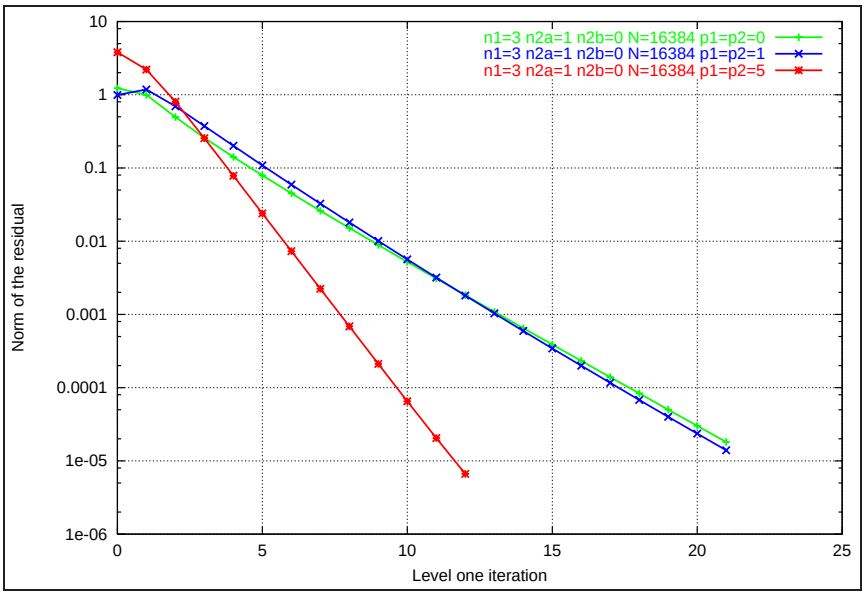

Figure 3.9: Convergence history of a static ACM cycle with different frequency distributions of the error.

For the case of a dynamic cycle, the number of iterations done in each level depends on the frequency distribution of the initial error. To illustrate this effect, the problem model (section 2.7.1) with $N = 1024 \times 1024$, has been solved using a dynamic cycle ($\alpha = 0.1$, $\beta = 1.3$). The number of iterations done in each level can be seen in Fig. 3.10. The distribution of the iterations is totally different, depending on the p_1 and p_2 parameters. In relation to the comment in the last paragraph of section 2.7.1, this is one of the reasons why the behavior of the iterative solution algorithms depend so much on the flow being solved.

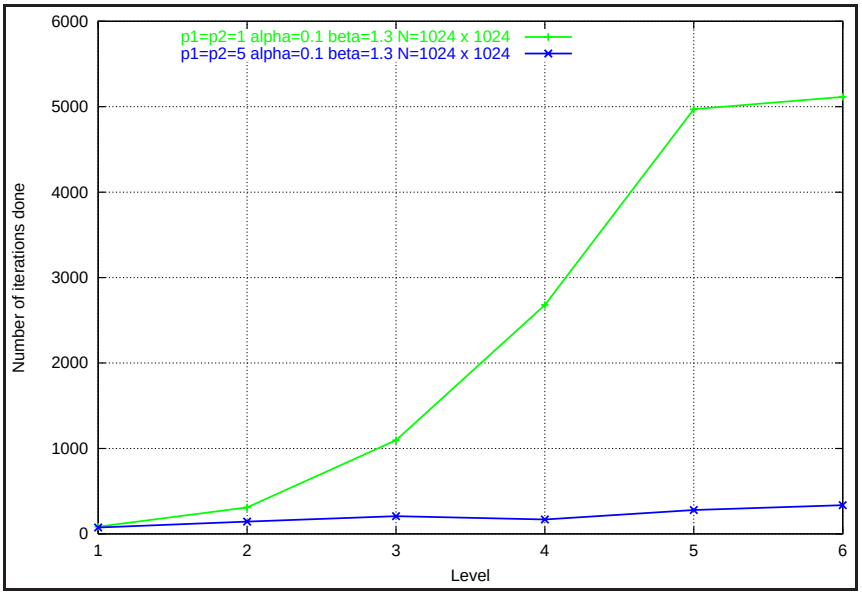

Figure 3.10: Iterations per level needed by a dynamic ACM algorithm, with $\alpha = 0.1$ and $\beta = 1.3$ for problems with different p_1 and p_2 parameters.

3.3.2 CPU time to solve the problem model

The relation between the norm of the residual and the iteration is not so important in practice as the relation between the residual and the CPU time, for problems with different numbers of unknowns. The CPU time to solve the problem model (section 2.7.1) with $\epsilon_r = 10^{-6}$ has been evaluated for different sets of parameters. The result has been represented versus the number of unknowns in Fig. 3.11. The parameters of the different variants are: (A) $\beta = 0$, $\alpha = 0.2$, $n_1 = 10$, $n_{2a} = 1$, $n_{2b} = 0$; (B) $\beta = \alpha = 0$, $n_1 = 0$, $n_{2a} = 1$, $n_{2b} = 0$; (C) $\beta = \alpha = 0$, $n_1 = 2$, $n_{2a} = 1$, $n_{2b} = 0$; (D) $\beta = \alpha = 0$, $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$.

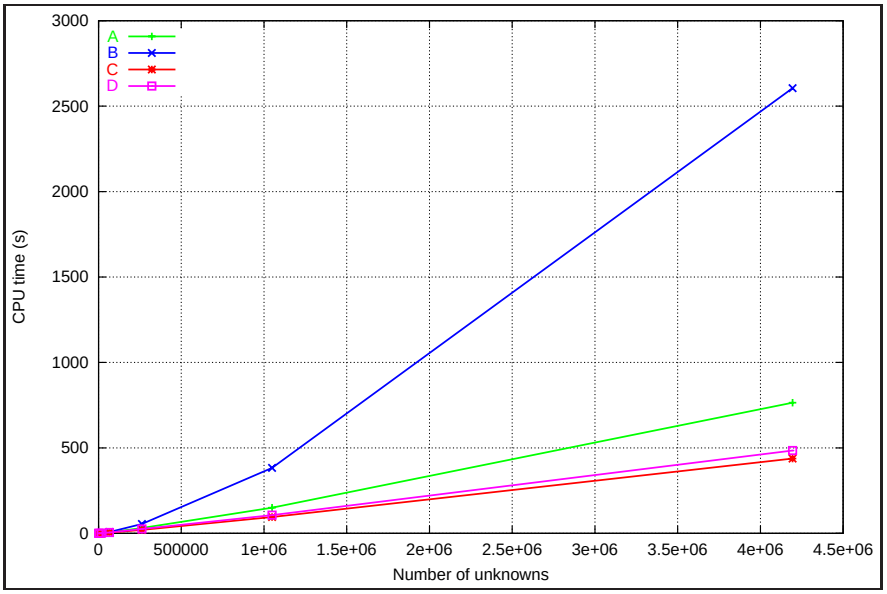

Figure 3.11: Time to solve the problem model with ACM versus N.

Breakdown of computing time

As can be seen in the example of Fig. 3.10, a large number of iterations has to be done in the smaller levels. However, for a sequential computer its cost is relatively low. A similar experiment is done here for the cycle $\beta = \alpha = 0$, $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$, also used in the previous section. Both the CPU time and the number of iterations on each level have been represented in Fig. 3.12. The problem size was $N = 1024 \times 1024$, and the parameters of the algorithm are as the (D) case of Fig. 3.11: $\beta = \alpha = 0$, $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$

This is not the case for a loosely coupled parallel computer, as the cost of the iterations on the smaller levels is dominated by latency (section 4.2.1). In practice this means that, for the smaller levels, the cost per iteration is very similar and independent of the number of unknowns. A cycle like this is very innefficient on a loosely coupled parallel computer. This consideration is crucial for the parallelization of ACM (section 7).

Computing times for two and three dimensional problems

For the same number of unknowns, the lowest frequency of the error decreases with the dimension of the problem. This has an important effect over the computing time. As an example, the cycle with $\beta = \alpha = 0$, $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$ has been used to compare the computing time to solve the two dimensional problem model and its three dimensional equivalent. The result can be seen in Fig. 3.13. The parameters of the algorithm used are as in (D) variant of Fig 3.11: $\beta = \alpha = 0$, $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$.

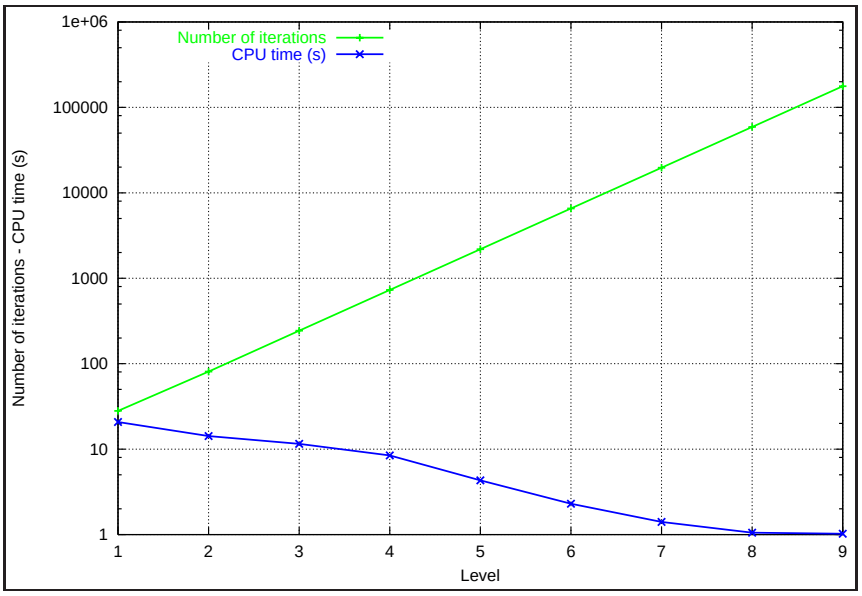

Figure 3.12: Number of iterations and cost of the smoothing in each level.

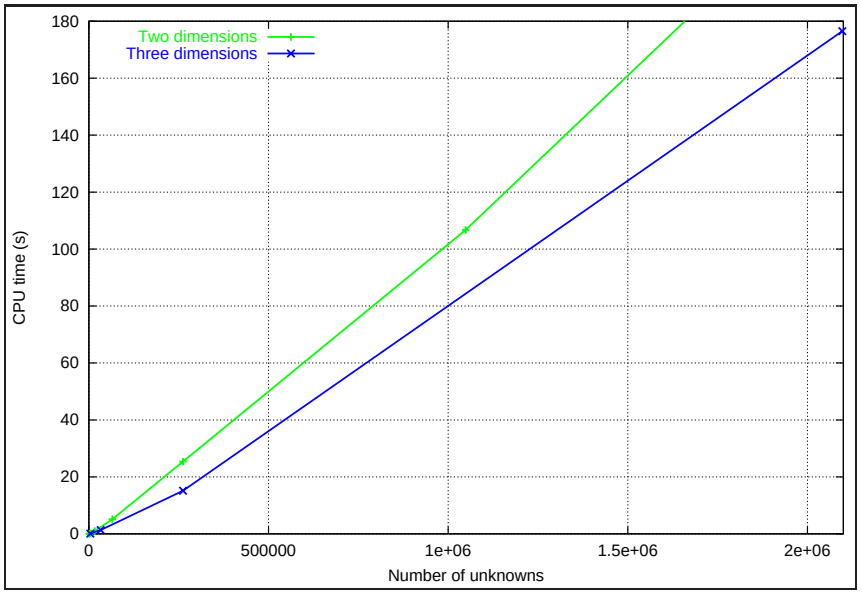

Figure 3.13: Time to solve the two-dimensional and three-dimensional systems with ACM versus N.

Comparison of ACM and LU

If the computing times to solve the problem model with ACM (with $\epsilon = 10^{-6}$) and LU are compared, the result is clearly favorable to LU, as it can be seen in Fig 3.14. The parameters of the ACM algorithm used are as in (D) variant of Fig 3.11: $\beta = \alpha = 0$, $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$. Only the CPU time of solution of the LU problem is considered, not its evaluation. The precision imposed to stop the ACM iterations is $\epsilon = 10^{-6}$, relative to the initial residual. However, as the time and memory to compute the LU decomposition are large, it is only of interest for the case of relatively small constant matrices. And, even in these conditions, depending on the precision needed, it may be worth using ACM. Fig. 3.15 is aimed to illustrate this aspect. For a problem with $N = 128 \times 128$, the precision obtained with ACM after each iteration in level 1, using $\beta = \alpha = 0$, $n_1 = 3$, $n_{2a} = 1$,

$n_{2b} = 0$ (as in the variant D of Fig. 3.11), is plotted versus the computing time.

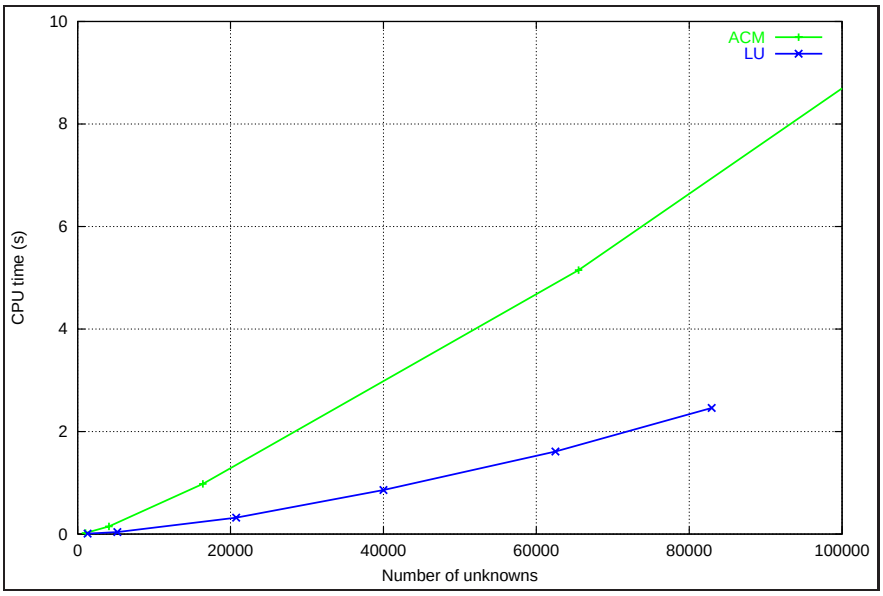

Figure 3.14: Time to solve the problem model versus N, with ACM and LU.

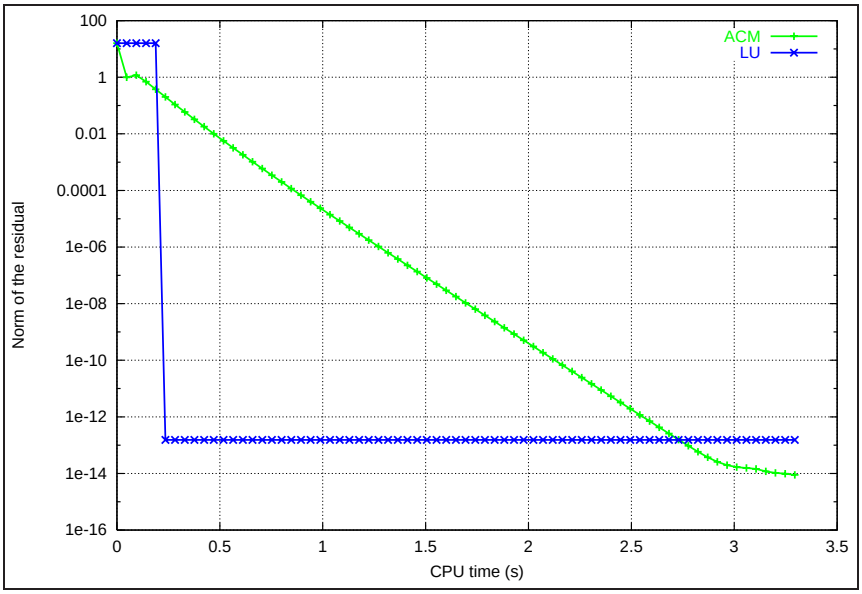

Figure 3.15: Norm of the residual versus computing time for ACM and LU.

.

After about 2.2×10^{-2} seconds, the LU system has been solved and the solution (with a residual close to the machine accuracy) is available. On the other hand, ACM needs much more time to reach machine accuracy precision, but approximate solutions are available sooner. If the precision obtained by ACM in the first iterations is enough for our application, it can be a better choice than LU, even for constant matrices. This is shown in a time-accurate simulation in section 3.3.3.

Fig. 3.15 is also useful to give an idea of the level of precision that can be reached with LU: the residual of the direct LU solution is less than one order of magnitude larger than the machine accuracy solution obtained with ACM.

3.3.3 ACM as a solver for the pressure-correction equation of a time-accurate CFD problem

As will be shown in section 3.4, segregated ACM approaches are more efficient than coupled ACM approaches for time accurate CFD problems. A two-dimensional simulation of a turbulent thermal driven cavity problem [141] will be used to show the behavior of ACM as a solver for a CFD problem.

Problem description

The governing equations (1.1-1.5) are directly used, without any modelization. In spite of the three-dimensional nature of the actual turbulent flow, two-dimensional simulations are of interest to provide information about the nature of the flow (as for instance in [142]) and also as benchmarks to enhance the numerical tools. As aforementioned in section 1.2, there is an important body of literature devoted to this and similar problems. In this section we are more concerned about the numerical aspects: our goal here is not to describe the physical system or to present a mesh independent simulation but just to show how ACM can be used to solve this problem.

The domain is a rectangle of dimensions L_x,L_y (horizontal and vertical, respectively). The left wall is kept isothermal at $T = T_h$, and the right wall at $T = T_c$. Horizontal walls are adiabatic. Velocity vector is null in all the boundaries. Initially, all the fluid is static and isothermal at a temperature $T = \frac{T_h+T_c}{2}$. The fluid is incompressible. Boussinesq hypothesis (section 1.2) is invoked. The characteristic dimensionless numbers are: $Pr = 0.71$ (air), $Ra = 1 \times 10^9$, $\frac{L_y}{L_x} = 4$. The dimensionless time step used for the integration is $\Delta\tau = 1.25 \times 10^{-7}$, where $\Delta\tau = \frac{\Delta t}{t_{ref}}$ and $t_{ref} = \frac{L^2 \rho C_p}{k}$.

These governing numbers are in the typical range of building heat transfer problem. As an example, one of the combinations that yields the previous dimensionless parameters is: $L_x = 1$ m, $L_y = 4$ m, $\Delta T = T_h - T_c \approx 10\,^0$C, $\Delta t = 5.64 \times 10^{-3}$ s.

To illustrate the structure of the flow, a instantaneous temperature field, corresponding to the pseudo-steady state is shown in Fig. 3.16. Instantaneous temperature (left) and streamfunction (right) maps, evaluated with a 235×470 control volumes mesh. For the map on the left, a *zebra* color map has been used: areas between isothermal are alternatively filled with black and with a color associated with the temperature (from dark blue for the coldest regions to red for the hottest regions). At the left, iso-streamfunction lines (tangent to the instantaneous velocity vector), are shown in black over a temperature map.

However, to have a picture of the flow dynamics, it is necessary to visualize an animation. An animation with the results of the present simulations can be downloaded from http://www.upc.es/lte. A part from being delightful, one can see for instance the periodic oscillations of the isotherms in the downstream parts of the boundary layers, apparently in concordance with the description given in [142].

Numerical method

SIMPLEC algorithm has been used for the pressure-momentum coupling. SMART [98] numerical scheme has been used for the discretization of the convective terms. Different meshes have been successfully used with ACM. To resolve the vertical boundary layers, a mesh strongly concentrated in the boundaries has been used: the ratio between the widths of the control volumes in the center of the cavity and adjacent to the vertical wall is ≈ 750.

To decide when are to be stopped the iterations at each time step, the dimensionless maximum difference between two consecutive iterations is controlled,

$$c_\phi^k = \frac{\max_{i,j}\left(|\phi_{i,j}^k - \phi_{i,j}^{k-1}|\right)}{\max_{i,j}\left(\phi_{i,j}^k\right)} \tag{3.35}$$

where $\phi^k_{i,j}$ is the value of ϕ field at the node (i,j) obtained after the k iteration of a certain time step. When the maximum increment is below a precision criteria for the time step, evaluated as the maximum of the c^k_ϕ,

$$c^k = \max\left(c^k_u, c^k_v, c^k_p, c^k_T\right) < \epsilon_t \tag{3.36}$$

the iterations are stopped.

A problem of this chaotic flow is to choose a correct ϵ_t value. For non-chaotic flows, an usual criteria to choose ϵ_t is to advance the simulation a number of time steps and then control the value of a parameter of interest, say the averaged Nu number on the right wall. This cannot be done for this problem: no matter how small is the ϵ_t chosen, if the simulation proceeds for long time enough, the instantaneous values of the flow will always be a function of ϵ_t. The numerical perturbations unavoidably introduced by finite precision arithmetics grow and cause this effect (on chaotic flows). This is analogous to the effect of perturbations in the physical flow (section 1.3.1).

As an example, the simulation has been carried out for 10^5 time steps with ϵ_t values of 10^{-3}, 10^{-4} and 10^{-5}, with a mesh of 121×241 control volumes has been used. If only a few time steps are considered, Fig. 3.17 (top), the three simulations seem instantaneously equal. After 10^4 time steps, the variant with 10^{-3} is clearly different but the difference between 10^{-4} and 10^{-5} is still relatively small (center). However, they also reveal totally different if the simulation continues (bottom). This instability explains the asymmetry of the flow [141] in spite of the symmetric boundary conditions, as aforementioned for the Poiseuille flow in section 1.3.1.

In this case, the criteria should be based on the time-average of the parameter of interest. The number of time steps to be averaged seems to be $\approx 10^5$.

As the time steps are small, scalar equations for u, v and T have a short area of influence (as discussed in section 2.4.2 for a one-dimensional diffusion equation) and one MSIP iteration is enough to solve then with sufficient precision. Actually, explicit or semi-explicit integration is frequently used for scalar equations in other algorithms (i.e. [141]). However, it is crucial to find an algorithm better than MSIP for pressure correction equation.

Benchmarking different algorithms for the pressure correction equation

An important fraction of the CPU cost for this problem is due to the mass conservation equation, expressed in terms of a pressure correction or Poisson equation. The cost of coefficient evaluation for scalar equation grows linearly with N and, even using high order schemes, it should be relatively low. The solution of the three scalar equations (momentum and energy) is also relatively cheap.

The CPU time needed to solve this problem on a single JFF node (see appendix B) has been measured using different algorithms, for a mesh with 235×470 control volumes, $\epsilon_t = 10^{-4}$. As the initial stage of the flow is not representative, a map from the pseudo-steady state is used as initial conditions for the benchmarks.

In order to optimize the computing time due to the solution of the pressure correction equation, the first consideration is the precision to impose. The number of iterations needed per time step can be reduced, solving the pressure correction equation with higher accuracy. Multigrid iterations to solve it are stopped when its residual is smaller than ϵ_p times the initial residual.

Different ϵ_p values have been used to stop the solver iterations on pressure correction equation. The averaged number of iterations $\overline{n}$, for a sample of 1000 time steps is shown in Table 3.3.

A value of $\epsilon_p = 10^{-3}$ seems a reasonable choice for this problem. Beyond that point, there is no gain in increasing the precision, as the iterations are not due to mass imbalance but to the non linearity of the problem and maybe to the use of a deferred correction approach for the second order spatial discretization (section 2.3).

Next consideration is about the best option to solve pressure correction equation with $\epsilon_p = 10^{-3}$. The averaged CPU cost per time step, for a sample of 1000 time steps, using different alternatives to do so, is presented in Table 3.4. For this relatively small problem, a possible option is to use LU. Doing so, the code needs about 655 Mbytes of RAM (close to the limit of the "fat" JFF node),

ϵ_p	$\overline{n}$	**Algorithm**
10^{-1}	4.150	ACM
10^{-2}	3.220	ACM
10^{-3}	3.165	ACM
10^{-4}	3.163	ACM
10^{-5}	3.163	ACM
≈ 0	3.162	LU

Table 3.3: Averaged number of iterations needed per time step for different precisions in the solution of the pressure correction equation (relative to the initial residual of the equation).

Algorithm	**Parameters**	$\overline{CPU}$ **per time step (s)**
LU		20.4
ACM	$\alpha = \beta = 0$, $n_1 = 3$, $n_{2a} = 1$, $n_{2b} = 0$	37.5
ACM	$\alpha = \beta = 0$, $n_1 = 4$, $n_{2a} = 1$, $n_{2b} = 0$	22.4
ACM	$\beta = 0$, $\alpha = 0.1$, $n_{2a} = 1$, $n_{2b} = 0$	17.7
ACM	$\beta = 0$, $\alpha = 1.0$, $n_{2a} = 1$, $n_{2b} = 0$	16.2

Table 3.4: Averaged CPU time (s) to solve each time step, using different algorithms.

while only 61 Mbytes of RAM are needed for ACM. On the other hand, it can solve the pressure correction equation to machine precision faster than ACM, but ACM is faster to reduce the residual the first three orders of magnitude[8]. Experimenting with the ACM parameters, we can easily find combinations that have a better CPU time than LU. The CPU times obtained with different algorithms are in Table 3.4.

However, the CPU time to solve this problem is still too high. A sample of 10^5 time steps takes about 19 CPU days. This is tolerable, but we must remember that the turbulent flow is three dimensional. Even optimizing the CPU cost of coefficient evaluation, using periodic boundary conditions to avoid solving two additional boundary layers, and considering that the cost per node of three dimensional problems is lower, parallel computers are necessary to work out the full three dimensional simulation.

3.4 Coupled Multigrid Solvers

Using multigrid only on individual linear equations does not resolve the low-frequency errors in the pressure-velocity coupling [120]. To solve this problem, for the case of steady state applications where such components tend to be important due to the large time steps used, the ACM algorithm can be applied to the discrete coupled momentum - mass conservation equations. The resulting algorithm can be called Coupled Additive Correction Multigrid (CACM). An overview of the method is given, and the limitations of the current implementations discussed.

The method, as implemented in DPC, is due to Sathyamurthy and Patankar [133, 134, 135], but it can be considered as a variant of the algorithm proposed in [137]. It was probably inspired in the coupled ACM devised by Hutchison, Galpin and Raithby [143] and in the geometric coupled MG algorithms introduced by Vanka. Initially, Vanka proposed a method for the direct (coupled) solution of the velocity components and pressure. However, this approach was limited by the RAM memory available. Then, Vanka [144] suggested to split the domain and use a direct solver for

[8]The difference seems less in Fig. 3.15, but the behaviour of MG depends on the equation solved, as discussed in section 2.7.1.

each of the zones. The limit of this idea is the coupled solution of the equations for each control volume, using explicit values of the neighbouring control volumes. This iterative algorithm is called Symmetrically Coupled Gauss Seidel (SCGS) [99]. Vanka (and later Wesseling [86] and others) used this method in the context of a coupled, classic MG solver.

Sathyamurthy applied the same idea to the ACM concept [137]. The correction equations for a staggered mesh are obtained using the previously described ACM procedure. The contribution of pressure gradient terms to momentum equations, that is included in the source term for the SIMPLE algorithm and its variants (Table 2.1), is treated separately in CACM. The correction equations can be found for instance in [133]. Due to the use of a staggered mesh, the treatment of the boundary conditions presents difficulties that to our knowledge are not totally clarified in any published paper, so a little bit of experimentation is needed to implement the algorithm. SCGS is used as smoother. Dynamic criteria are used to control the number of iterations at each level. It was described as a method for incompressible flows but it can be extended to compressible flows.

The method is attractive as it has the main advantages of ACM (algebraic approach and robustness) but it allows a multigrid solution of the pressure-velocity coupling that is better than a SIMPLEC+ACM approach. It is specially well suited to solve pseudo-transient formulations. In general, it provides both a better convergence region than SIMPLEC+ACM and a better convergence ratio, specially for large meshes.

This algorithm, with slight modifications to allow the presence of internal discontinuities such as solids, has been the core of DPC code for years, providing an efficient solution of many problems, such as natural convection flow in solar collector channels [145], or thermal energy storage tanks [77].

The same general idea can be implemented using more complex smoothers, such as CELS [88] or CSIP. CELS (standing for Coupled Equation Line Solver) is to SCGS what line-by-line Gauss-Seidel is to Gauss-Seidel. The coupled momentum and mass conservation equations are solved for each line or column of the domain, using explicit values for the nodes outside of the line being considered. There is a version [146] that extends the coupling to the energy equation. CSIP [100] (standing for Coupled Strongly Implicit) is to SCGS [120] what SIP [107] is to Gauss-Seidel. An ILU decomposition of the coupled momentum-mass conservation equations is done. The different efficiencies of these algorithms can be important if used as iterative solvers, without multigrid correction. However, in general terms, they behave similarly as smoothers in a MG context. Other smoothing techniques for staggered meshes are reviewed in [147].

The main inconvenients of CACM, implemented as described in [133] or using any of the aforementioned smoothers are: *(i)* It relies on the specific algebraic structure obtained from the discretization of the momentum-mass conservation equation on staggered meshes; *(ii)* It is limited to the pressure-velocity coupling. If other couplings are included, they are restricted to the interaction of all the unknowns in the same control volume or immediate neighbours, unless the algorithm is totally changed.

To overcome these inconvenience and allow the easy extension to more complex equations, such as those arising from Newton linearization, reactive flows with a large number of scalar equations to describe the concentration of the different species, or $k-\epsilon$ models, a Block-wise Additive Correction Multigrid (BACM) method has recently been developed. It does not rely on the nature of the equations arising from staggered meshes. It can be easily implemented to solve the coupling between an arbitrary number of fields. Using a systematic approach, different block-wise smoothers have been developed. Initial results are promising [97].

3.4.1 CACM benchmark

For a thermal driven cavity problem, with a uniform mesh, $Ra = 1 \times 10^6$, $Pr = 0.7$, the number of iterations to achieve convergence is represented versus the dimensionless time step $\Delta\tau = \frac{\Delta t}{t_{ref}}$. The reference time used is $t_{ref} = \frac{L^2 \rho c_p}{k}$. Iterations are started from $u = v = 0$, $T = T_l$. A non-uniform mesh is used. Only one iteration is allowed per time step. To control the convergence, a criteria similar to equation (3.36) is used, but here the superindex k refers to the discrete time step instead

to the iteration in the time step. The ϵ value used is 10^{-6}.

For the case of SIMPLEC [59] algorithms, ACM algorithm has been used for all the scalar fields (one iteration on u, v and T fields and 8 iterations for pressure correction equation). For the case of CACM algorithms, two iterations where done to solve the momentum-mass conservation set. The default DPC parameters, based on dynamic control of the number of iterations, have been used for both solvers.

The thermal driven cavity problem was solved using a pseudo-transient method with different dimensionless time steps with $Ra = 1 \times 10^6$, $Pr = 0.7$. The results of the comparison can be seen in Fig. 3.18. For the coupled solver, the number of iterations is roughly independent of the mesh size. This important property is achieved by the SIMPLEC+ACM solver only for the smaller time steps, where the integration done is almost time accurate. However, if the time step is increased, the segregated approach fails to solve the discrete equations of each time step. Global convergence can still be achieved, but at the cost of doing more outer iterations.

For the larger mesh considered, the total CPU time for both methods has been compared in Fig. 3.19. The results are not as relevant as the number of iterations because the time presented includes the evaluation of the coefficients, that is a significant fraction of the total. However, the general trend of the data is relevant: the coupled approach is more effective for larger time steps, where there are long range interactions between pressure and velocity that are better solved by CACM. At the opposite side, time accurate integrations are more effectively solved by the segregated approach, as used for the example in section 3.3.3.

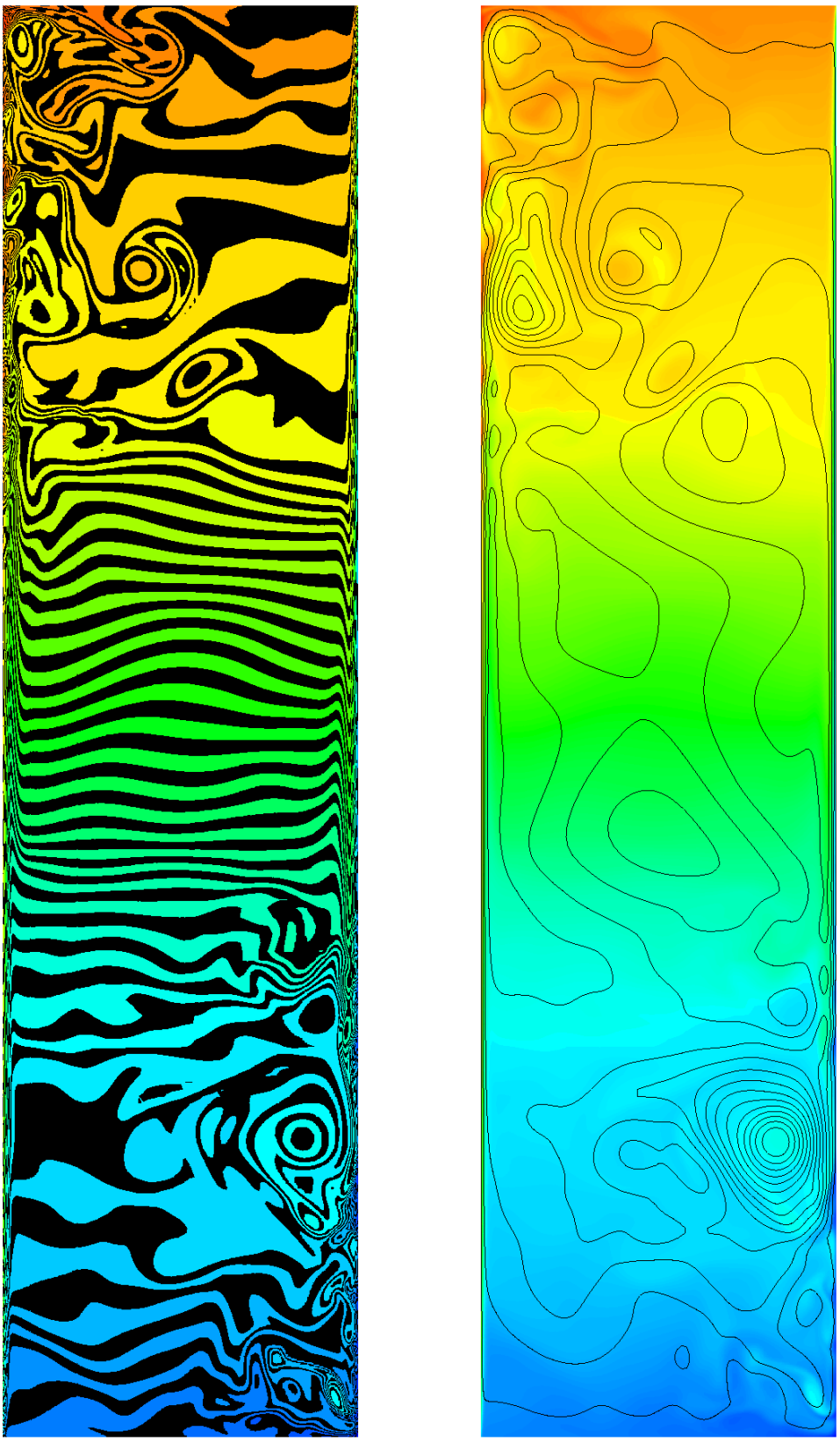

Figure 3.16: Instantaneous velocity and temperature fields of a turbulent natural convection simulation.

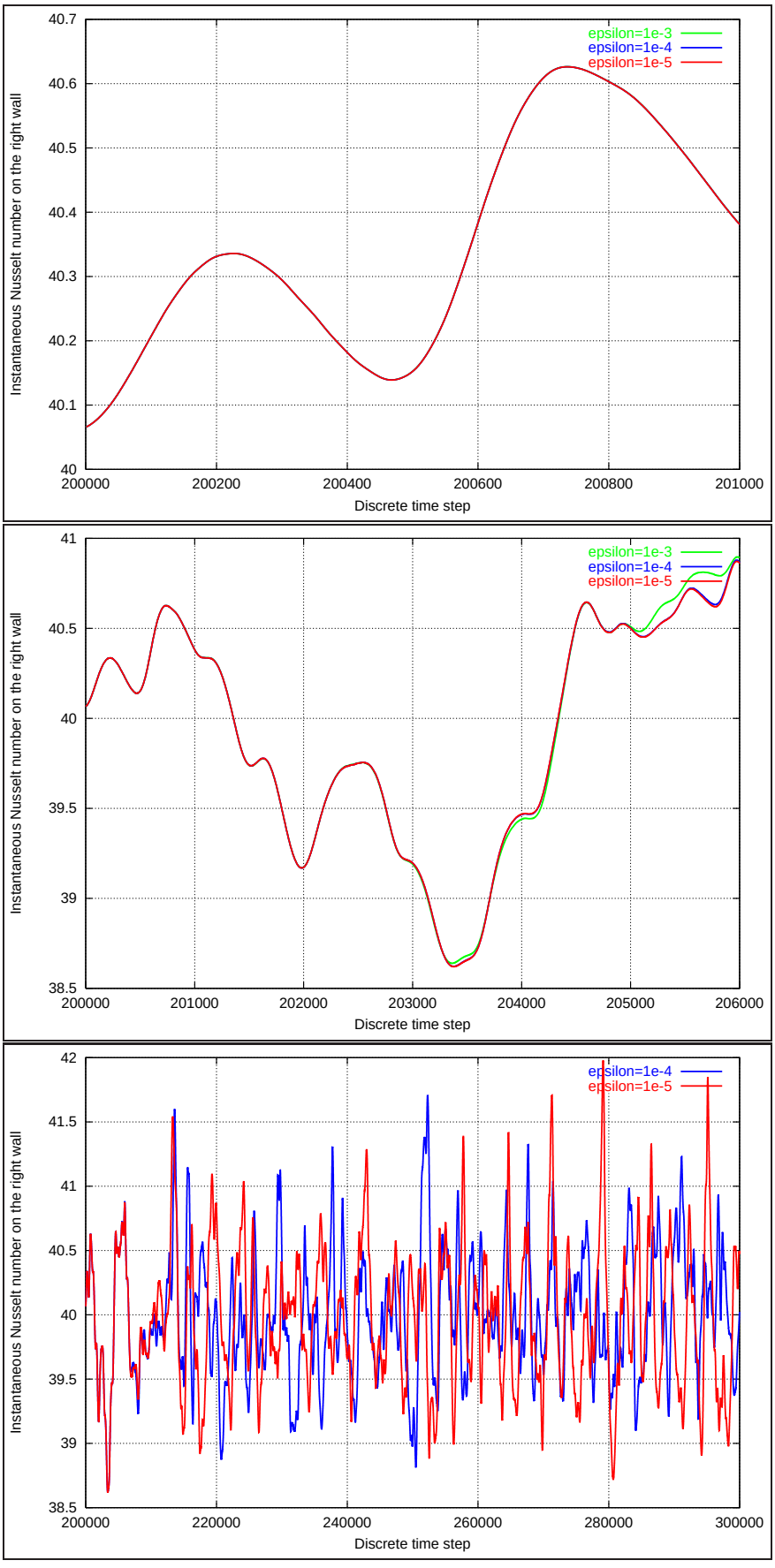

Figure 3.17: Instantaneous right wall Nusselt numbers for integrations carried out with different ϵ values.

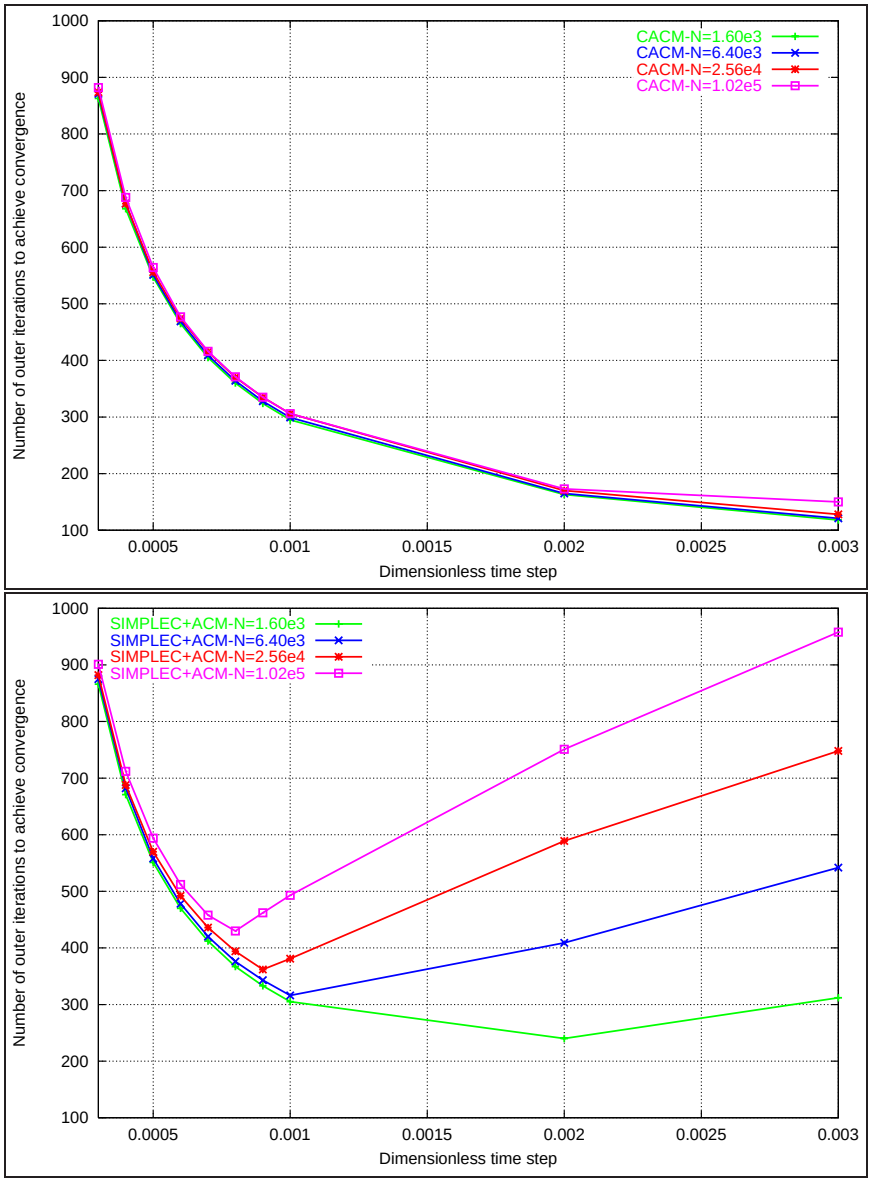

Figure 3.18: Number of iterations to achieve convergence for a steady-state flow with ACM and CACM.

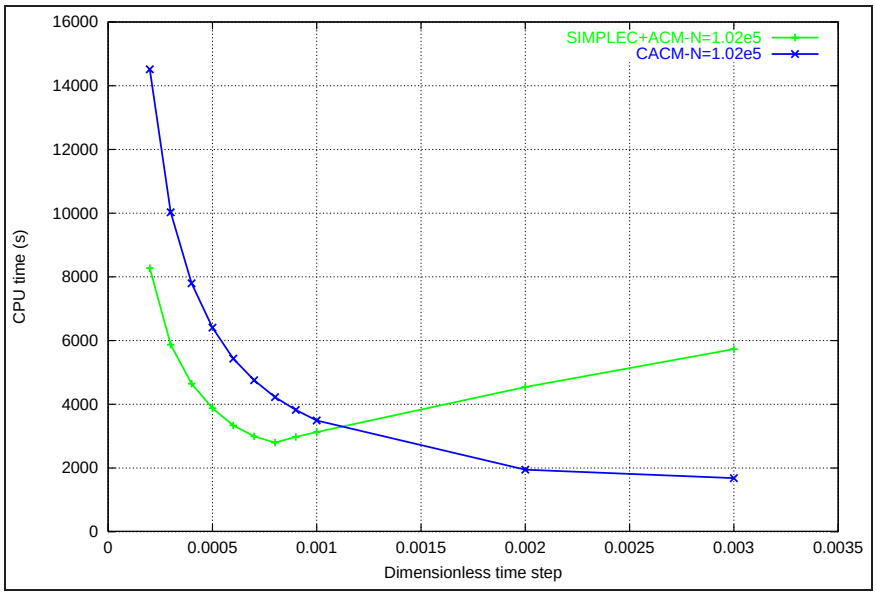

Figure 3.19: CPU time to achieve convergence for a steady-state flow with ACM and CACM.

3.5 Nomenclature

b	right hand side
c_p	specific heat
d	domain dimension
k	thermal conductivity
I_o^d	integrid transfer operator o origin level d destination level
L	generic differential operator
L_x	horizontal domain length
L_y	vertical domain length
$\mathbf{x}$	position vector
f	right hand side
$\overline{n}$	averaged number of iterations per time step
nr	residual norm
n	number of iterations
n_1	number of correction cycles (Alg. 3.2)
n_{2a}, n_{2b}	number of ACM pre and post-smoothing iterations (Alg. 3.2)
N	number of nodes
M	number of levels
r	residual
Pr	Prandtl number
Ra	Rayleigh number
R	block residual
Δ_t	time step
t_{ref}	reference time
T_h	cold temperature
T_h	hot temperature
u	generic unknown (for geometric MG)
$\tilde{u}$	approximate value of the generic unknown

Greek symbols

α	dynamic ACM criteria (Alg. 3.2)
β	dynamic ACM criteria (Alg. 3.1)
ϵ	precission
ϵ_t	precission of a time step
Ω	domain
ν	number of iterations
ϕ	generic unknown (for ACM)
Φ	correction
$\mathbf{\Phi}$	vector of all the unknowns
$\Delta\tau$	dimensionless time step
ρ	density

Subindices

1,2,3	Cartesian components
(i, j)	position in the discretization map
(I, J)	position in the correction map
ϕ	generic unknown
u	horizontal velocity
v	horizontal velocity
p	pressure
T	temperature

Superindices

$1 \cdots M$	levels from fine to coarse
e	east neighbour
w	west neighbour
n	north neighbour
s	south neighbour
p	discretization node
k	iteration