We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 2

Multilevel Variable-Block Schur-Complement-Based Preconditioning for the Implicit Solution of the Reynolds-Averaged Navier-Stokes Equations Using Unstructured Grids

Bruno Carpentieri and Aldo Bonfiglioli

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72043

Abstract

Implicit methods based on the Newton's rootfinding algorithm are receiving an increasing attention for the solution of complex Computational Fluid Dynamics (CFD) applications due to their potential to converge in a very small number of iterations. This approach requires fast convergence acceleration techniques in order to compete with other conventional solvers, such as those based on artificial dissipation or upwind schemes, in terms of CPU time. In this chapter, we describe a multilevel variable-block Schur-complement-based preconditioning for the implicit solution of the Reynolds-averaged Navier-Stokes equations using unstructured grids on distributed-memory parallel computers. The proposed solver detects automatically exact or approximate dense structures in the linear system arising from the discretization, and exploits this information to enhance the robustness and improve the scalability of the block factorization. A complete study of the numerical and parallel performance of the solver is presented for the analysis of turbulent Navier-Stokes equations on a suite of three-dimensional test cases.

Keywords: computational fluid dynamics, Reynolds-averaged Navier-Stokes equations, Newton-Krylov methods, linear systems, sparse matrices, algebraic preconditioners, incomplete LU factorization, multilevel methods

1. Introduction

A considerable number of modern high-fidelity Computational Fluid Dynamics (CFD) solvers and codes still adopt either one-dimensional physical models based on the Riemann problem

using higher order shape functions, such as higher order Finite Volume (FV) and Discontinuous Galerkin Finite Element (FE) methods for the discrete data representation, or truly multidimensional physical models using linear shape functions, like Fluctuation Splitting (FS) schemes. Both of these approaches require fast convergence acceleration techniques in order to compete with conventional solvers based on artificial dissipation or upwind schemes in terms of CPU time. Implicit methods based on the Newton's rootfinding algorithm are receiving an increasing attention in this context for the solution of complex real-world CFD applications, for example in the analyses of turbulent flows past three-dimensional wings, due to their potential to converge in a very small number of iterations [1, 2]. In this chapter, we consider convergence acceleration strategies for the implicit solution of the Reynolds-averaged Navier-Stokes (RANS) equations based on the FS space discretization using a preconditioned Newton-Krylov algorithm for the integration. The use of a Newton solver requires the inversion of a large nonsymmetric system of equations at each step of the non-linear solution process. Choice of linear solver and preconditioner is crucial for efficiency especially when the mean flow and the turbulence transport equation are solved in fully coupled form. In this study, we use the restarted Generalized Minimal Residuals (GMRES) [3] algorithm for the inner linear solver, preconditioned by a block multilevel incomplete lower-upper (LU) factorization. We present the development lines of the multilevel preconditioning strategy that is efficient to reduce the number of iterations of Krylov subspace methods at moderate memory cost, and shows good parallel performance on three-dimensional turbulent flow simulations.

The chapter is structured as follows. The governing conservation equations for both compressible and incompressible flows are reviewed in Section 2. Section 3 briefly describes the fluctuation splitting space discretization, the time discretization and the Newton-Krylov method used to solve the space- and time-discretized set of governing partial differential equations (PDEs). In Section 4, we present the development of the multilevel preconditioning strategies for the inner linear solver. We illustrate the numerical and parallel performance of the preconditioner for the analysis of turbulent incompressible flows past a three-dimensional wing in Section 5. Some concluding remarks arising from the study are presented in Section 6.

2. Governing equations

In the case of inviscid and laminar flows, given a control volume C_i, fixed in space and bounded by the control surface ∂C_i with inward normal $\mathbf{n}$, the governing equations of fluid dynamics are obtained by considering the conservation of mass, momentum and energy. In the case of viscous turbulent flows, one approach to consider the effects of turbulence is to average the unsteady Navier-Stokes (NS) equations on the turbulence time scale. Such averaging procedure results in a new set of steady equations (the RANS equations) that differ from the steady NS equations for the presence of the Reynolds' stress tensor, representing the effects of turbulence on the averaged flow field. The appearance of this tensor yields a closure problem, which is often solved by adopting an algebraic or a differential turbulence model. In the present work, we use the Spalart-Allmaras [4] one-equation model for the turbulent viscosity. Thus the integral form of the conservation law of mass, momentum, energy and turbulence transport equations has the form

http://dx.doi.org/10.5772/intechopen.72043

$$\int_{C_i} \frac{\partial U_i}{\partial t} dV = \oint_{\partial C_i} \mathbf{n} \cdot F dS - \oint_{\partial C_i} \mathbf{n} \cdot G dS + \int_{C_i} S dV \tag{1}$$

where U is the vector of conserved variables. For compressible flows, we have $U = \left(\rho, \rho e^0, \rho \mathbf{u}, \tilde{\nu}\right)^T$, and for incompressible, constant density flows, $U = (p, \mathbf{u}, \tilde{\nu})^T$. The operators F and G represent the inviscid and viscous fluxes, respectively; for compressible flows, we have

$$F = \begin{pmatrix} \rho \mathbf{u} \\ \rho \mathbf{u} h^0 \\ \rho \mathbf{u}\mathbf{u} + pI \\ \tilde{\nu}\mathbf{u} \end{pmatrix}, \quad G = \frac{1}{Re_\infty} \begin{pmatrix} 0 \\ \mathbf{u} \cdot \underline{\underline{\tau}} + \mathbf{q} \\ \underline{\underline{\tau}} \\ \frac{1}{\mathrm{Pr}_T}[(\nu + \tilde{\nu})\nabla\tilde{\nu}] \end{pmatrix}, \tag{2}$$

and for incompressible, constant density flows,

$$F = \begin{pmatrix} a^2 \mathbf{u} \\ \mathbf{u}\mathbf{u} + pI \\ \tilde{\nu}\mathbf{u} \end{pmatrix}, \quad G = \frac{1}{Re_\infty} \begin{pmatrix} 0 \\ \underline{\underline{\tau}} \\ \frac{1}{\mathrm{Pr}_T}[(\nu + \tilde{\nu})\nabla\tilde{\nu}] \end{pmatrix}. \tag{3}$$

Finally, the source term S has a non-zero entry only in the row corresponding to the turbulence transport equation; its expression is not reported here for brevity, but can be found in [4]. Note that the standard NS equations are retrieved from (1) by removing the source term S and the differential equation associated with the turbulence variable, and setting the effective viscosity and thermal conductivity to their laminar values. The Euler equations are instead recovered by additionally removing the flux vector G.

3. Solution techniques

The model used in this study for the discrete data representation is based on the coupling of an hybrid class of methods for the space discretization, called Fluctuation Splitting (or residual distribution) schemes [5], and a fully coupled Newton algorithm. By "fully coupled" we mean that the mass, momentum and energy conservation equations on one hand, and the turbulent equation on the other, are solved simultaneously rather than in a decoupled or staggered fashion. We discuss in the following subsections, separately, the space and time discretization, the numerical integration of the set of equations resulting from the discretization, and the solution of the large linear system at each Newton's step.

3.1. Space discretisation

The Fluctuation Splitting approach has features common to both Finite Element (FE) and Finite Volume (FV) methods. Like in standard FE methods, the dependent variables are stored at the vertices of the computational mesh made up of triangles in the two-dimensional (2D) space, and tetrahedra in three-dimensional (2D), and are assumed to vary linearly and continuously

in space. Denoting Z_i as the nodal value of the dependent variable at the grid point i and N_i as the FE linear shape function, this dependence can be written as

$$Z(\mathbf{x}, t) = \sum_i Z_i(t) N_i(\mathbf{x}). \tag{4}$$

Note that, although the summation in Eq. (4) extends over all grid nodes, the computational molecule of each node is actually limited only to the set of its nearest neighbors due to the compact support of the linear shape functions. In the compressible case, Roe's parameter vector

$$Z = \left(\sqrt{\rho}, \sqrt{\rho} h^0, \sqrt{\rho}\mathbf{u}, \tilde{\nu}\right)^T \tag{5}$$

is chosen as the dependent variable to ensure discrete conservation [6]. In the incompressible case, discrete conservation is obtained by simply setting the dependent variable Z equal to the vector of conserved variables U. In our code, we group the dependent variables per gridpoint. The first m entries of the array Z are filled with the m flow variables of gridpoint 1, and these are followed by those of gridpoint 2, and so on. Blocking the flow variables in this way, also referred to as "field interlacing" in the literature, is acknowledged [7–9] to result in better performances than grouping variables per aerodynamic quantity.

The integral Eq. (1) is discretized over each control volume C_i using a FV-type approach. In two dimensions, the control volumes C_i are drawn around each gridpoint by joining the centroids of gravity of the surrounding cells with the midpoints of all the edges that connect that gridpoint with its nearest neighbors. An example of polygonal-shaped control volumes (so-called median dual cells) is shown by green lines in **Figure 1(a)**. With FS schemes, rather than calculating the inviscid fluxes by numerical quadrature along the boundary ∂C_i of the median dual cell, as would be done with conventional FV schemes, the net inviscid flux $\Phi^{e,\,inv}$ over each triangular/tetrahedral element

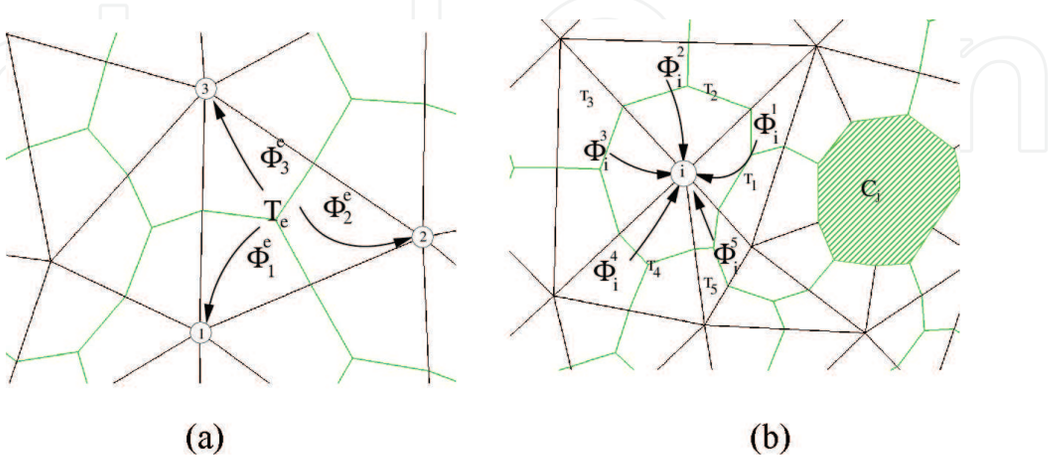


Figure 1. Residual distribution concept. (a) The flux balance of cell T is scattered among its vertices. (b) Gridpoint i gathers the fractions of cell residuals from the surrounding cells.

http://dx.doi.org/10.5772/intechopen.72043

$$\Phi^{e,\,inv} = \oint_{\partial T_e} \mathbf{n} \cdot F dS \tag{6}$$

is evaluated by means of a conservative linearization based on the parameter vector [6], and scattered to the element vertices using elemental distribution matrices B_i^e [5]. The inviscid contribution to the nodal residual $(R_\Phi)_i$ is then assembled by collecting fractions $\Phi_i^{e,\,inv}$ of the net inviscid fluxes $\Phi^{e,\,inv}$ associated with all the elements by which the node i is surrounded. This is schematically shown in **Figure 1(b)**. Concerning the viscous terms, the corresponding flux balance is evaluated by surface integration along the boundaries of the median dual cell: node i receives a contribution $\Phi_i^{e,\,vis}$ from cell e which accounts for the viscous flux through the portion of ∂C_i that belongs to that cell. This approach can be shown to be equivalent to a Galerkin FE discretization.

Summing up the inviscid and viscous contributions to the nodal residual of gridpoint i one obtains

$$(R_\Phi)_i = \sum_{e \ni i} \left(\Phi_i^{e,\,inv} + \Phi_i^{e,\,vis} \right) = \sum_{e \ni i} \left(B_i^e \Phi^{e,\,inv} + \Phi_i^{e,\,vis} \right). \tag{7}$$

In Eq. (7), the summation ranges over all the elements e that meet in meshpoint i, as shown in **Figure 1(b)**. The construction of the distribution matrices B_i^e involves the solution of $d+1$ small (of order m) dense linear systems for each triangular/tetrahedral element of the mesh thus making FS schemes somewhat more expensive than state-of-the-art FV schemes based upon either central differencing with artificial dissipation or upwind discretizations. The relatively high computational cost of FS discretizations has to be accounted for when deciding whether the Jacobian matrix should be stored in memory or a Jacobian-free method be used instead.

3.2. Time discretisation

One route to achieve second-order time accuracy with FS schemes is to use mass matrices that couple the time derivatives of neighboring grid points. This leads to implicit schemes, even if the spatial residual were treated explicitly. Although a more general framework for the derivation of the mass matrices can be devised [10], the approach adopted in our study consists of formulating the FS scheme as a Petrov-Galerkin FE method with elemental weighting function given by

$$\Omega_i^e = \left(N_i - \frac{1}{d+1} \right) I_{m \times m} - B_i^e. \tag{8}$$

The contribution of element e to the weighted residual equation for grid point i reads

$$\int_{T_e} \left(\Omega_i^e \frac{\partial U}{\partial t} \right) dV = \int_{T_e} \left(\Omega_i^e \frac{\partial U}{\partial Z} \frac{\partial Z}{\partial t} \right) dV, \tag{9}$$

where the chain rule is used to make the dependent variable Z appear. Since the conservative variables U are quadratic functions of the parameter vector Z, the transformation matrix

$\partial U/\partial Z$ is linear in Z and can thus be expanded using the linear shape functions N_j, just as in Eq. (4). A similar expansion applies to the time-derivative $\partial Z/\partial t$. Replacing both expansions in the RHS of Eq. (9), the discrete counterpart of the time derivative of Eq. (9) is given by the contribution of all elements sharing the node i:

$$\int_{C_i} \frac{\partial U_i}{\partial t}\, dV = \sum_e \int_{T_e} \left(\Omega_i^e \frac{\partial U}{\partial t} \right) dV = \sum_{e \ni i} \sum_{j \in e} M_{ij}^e \left(\frac{\partial Z}{\partial t} \right)_j . \tag{10}$$

In Eq. (10) the index j spans the vertices of the element e and the nodal values $(\partial Z/\partial t)_j$ are approximated by the three-level Finite Difference (FD) formula

$$\left(\frac{\partial Z}{\partial t} \right)_j = \frac{3Z_j^{n+1} - 4Z_j^n + Z_j^{n-1}}{2\Delta t}. \tag{11}$$

The matrix M_{ij}^e in Eq. (10) is the contribution of element e to the entry in the ith row and jth column of the global mass matrix $[M]$. Similarly to what is done in the assembly of the inviscid and viscous flux balance, Eq. (7), the discretization of the unsteady term, Eq. (10), is obtained by collecting elemental contributions from all the elements that surround the node i. Second-order space and time accuracy of the scheme described above has been demonstrated for inviscid flow problems by Campobasso et al. [11] using an exact solution of the Euler equations.

3.3. Numerical integration

Writing down the space- and time-discretized form of Eq. (1) for all gridpoints of the mesh, one obtains the following large, sparse system of non-linear algebraic equations

$$R_g(U(Z)) = R_\Phi(U(Z)) - [M] \frac{1}{\Delta t} \left(\frac{3}{2} Z^{n+1} - 2Z^n + \frac{1}{2} Z^{n-1} \right) = 0 \tag{12}$$

to be solved at time level $n+1$ to obtain the unknown solution vector U^{n+1}. The solution of Eq. (12) is obtained by means of an implicit approach based on the use of a fictitious time-derivative (Jameson's dual time-stepping [12]) that amounts to solve the following evolutionary problem

$$\frac{dU}{d\tau} [V_M] = R_g(U) \tag{13}$$

in pseudo-time τ until steady state is reached. Since accuracy in pseudo-time is obviously irrelevant, the mass matrix has been lumped into the diagonal matrix $[V_M]$ and a first-order accurate, two-time levels FD formula

$$\frac{dU}{d\tau} \approx \frac{U^{n+1,k+1} - U^{n+1,k}}{\Delta\tau} \tag{14}$$

is used to approximate the pseudo-time derivative in the LHS of Eq. (13). The outer iterations counter k has been introduced in Eq. (14) to label the pseudo-time levels.

Upon replacing Eq. (14) in Eq. (13), an implicit scheme is obtained if the residual R_g is evaluated at the unknown pseudo-time level $k+1$. Taylor expanding R_g about time level k, one obtains the following sparse system of linear equations

$$\left(\frac{1}{\Delta\tau_k}[V_M] - [J]\right)\Delta U = R_g\left(U^{n+1,k}\right) \qquad [J] = \frac{\partial R_g}{\partial U} \tag{15}$$

to be solved at each outer iteration until the required convergence of R_g is obtained. Steady RANS simulations are accommodated within the presented integration scheme by dropping the physical time-derivative term in Eq. (12). In the limit $\Delta\tau_k \to \infty$, Eq. (15) recovers Newton's rootfinding algorithm, which is known to yield quadratic convergence when the initial guess $U^{n+1,0} = U^n$ is sufficiently close to the sought solution U^{n+1}. This is likely to occur when dealing with unsteady flow problems because the solution of the flow field at a given physical time starts from the converged solution at the preceding time, and this latter constitutes a very convenient initial state. In fact, it is sufficiently close to the sought new solution to allow the use of the exact Newton's method (i.e. $\Delta\tau_k = \infty$ in Eq. (15)) since the first solution step.

The situation is different when dealing with steady flow problems. Newton's method is only locally convergent, meaning that it is guaranteed to converge to a solution when the initial approximation is already close enough to the sought solution. This is generally not the case when dealing with steady flows, and a "globalization strategy" needs to be used in order to avoid stall or divergence of the outer iterations. The choice commonly adopted by various authors [13, 14], and in this study as well, is a pseudo-transient continuation, which amounts to retain the pseudo-transient term in Eq. (15). At the early stages of the iterative process, the pseudo-time step length $\Delta\tau_k$ in Eq. (15) is kept small. The advantage is twofold: on one hand, it helps preventing stall or divergence of the outer iterations; on the other hand, it makes the linear system (15) easier to solve by means of an iterative solver since for moderate values of $\Delta\tau_k$ the term $[V_m]/\Delta\tau_k$ increases the diagonal dominance of the matrix. Once the solution has come close to the steady state, which can be monitored by looking at the norm of the nodal residual R_Φ, we let $\Delta\tau_k$ grow unboundedly so that Newton's method is eventually recovered during the last steps of the iterative process. The time step length $\Delta\tau_k$ is selected according to the Switched Evolution Relaxation (SER) strategy proposed by Mulder and van Leer [15], as follows:

$$\Delta\tau_k = \Delta\tau \min\left(C_{\max}, C_0 \frac{\left\|R_g\left(U^{n+1,0}\right)\right\|_2}{\left\|R_g\left(U^{n+1,k}\right)\right\|_2}\right), \tag{16}$$

where $\Delta\tau$ is the pseudo-time step based upon the stability criterion of the explicit time integration scheme, and C_0 and $C_{\max}$ are user-defined constants controlling the initial and maximum pseudo-time steps used in the actual calculations.

In the early stages of the iterative process, the turbulent transport equation and the mean flow equations are solved in tandem (or in a loosely coupled manner, following the nomenclature used by Zingg et al. [16]): the mean flow solution is advanced over a single pseudo-time step using an analytically computed, but approximate Jacobian while keeping turbulent viscosity

frozen, then the turbulent variable is advanced over one or more pseudo-time steps using a FD Jacobian with frozen mean flow variables. Due to the uncoupling between the mean flow and turbulent transport equations, this procedure will eventually converge to steady state, but never yields quadratic convergence. Close to steady state, when a true Newton strategy preceded by a "short" pseudo-transient continuation phase can be adopted, the mean flow and the turbulence transport equation are solved in fully coupled form, and the Jacobian is computed by FD. For the sake of completeness, we give further details of each of these two steps in the following two paragraphs.

3.3.1. Tandem solution strategy with (approximate) Picard linearization

Consider re-writing the steady nodal residual R_Φ, see [17] for full details, as

$$R_\Phi(U) = ([C] - [D])U, \tag{17}$$

where $[C]$ and $[D]$ are (sparse) matrices that account for the convective and diffusive contributions to the nodal residual vector R_Φ. Matrix $[D]$ is constant for isothermal, incompressible flows whereas it depends upon the flow variables through molecular viscosity in the case of compressible flows. Matrix $[C]$ depends upon U for both compressible and incompressible flows. Both matrices can be computed analytically as described in [17]. What we refer to as a Picard linearization consists in the following approximation

$$J \approx [C] - [D], \tag{18}$$

which amounts to neglect the dependence of matrices $[C]$ and $[D]$ upon U when differentiating the residual, written as in Eq. (17).

Once the mean flow solution has been advanced over a single pseudo-time step using the approximate Picard linearization, keeping the turbulent viscosity frozen, the turbulent variable is advanced over one or more (typically ten) pseudo-time steps using a FD Jacobian approximation (described in Section 3.3.2) with frozen mean flow variables. Blanco and Zingg [18] adopt a similar strategy, but keep iterating the turbulence transport equation until its residual has become lower than that of the mean flow equations. The loosely coupled solution strategy is a choice often made as it "allows for the easy interchange of new turbulence models" [19] and also reduces the storage [18], compared to a fully coupled approach. However, due to the uncoupling between the mean flow and the turbulent transport equations, the tandem solution strategy never yields quadratic convergence nor it is always able to drive the nodal residual to machine zero. The last statement cannot be generalized, since Blanco and Zingg [16, 20] report convergence to machine zero for their loosely coupled approach on two-dimensional unstructured grids. However, even if convergence to machine zero is difficult to achieve, the nodal residual is always sufficiently converged for any practical "engineering" purpose and close enough to "true" steady-state solution to be a good initial guess for Newton's method.

3.3.2. Fully coupled solution strategy with FD Newton linearization

Once the tandem solution strategy has provided a good approximation to the steady flow, or when dealing with unsteady flows, in which case the solution at a given time level is generally

a good approximation to the one sought at the next time level, it becomes very attractive to take advantage of the quadratic convergence of Newton's method. In order to do so, however, the mean flow and the turbulence transport equations must be solved fully coupled and the Jacobian matrix $[J]$ must be accurate. We take advantage of the compactness of the computational stencil required by FS schemes to compute a close approximation to the true Jacobian matrix even for second-order accurate discretizations. The analytical evaluation of the Jacobian matrix, though not impossible [5, 21], is rather cumbersome and thus this approach is not pursued here.

When the equations are fully coupled, the structure of the Jacobian matrix $[J]$ is naturally organized into small dense blocks of order m. This has implications both in terms of storage, since it is possible to reduce the length of the integer pointers that define the Compressed Sparse Row (CSR) data structure of the sparse matrix, and also in the design of the preconditioner for solving the large linear system at each Newton step, where division operations can be efficiently replaced by block matrix factorizations. We will address these issues in detail in the next section. Two neighboring gridpoints, i and j, in the mesh will contribute two block entries, J_{ij} and J_{ji}, to the global Jacobian matrix $[J]$. Each of these two block entries (say J_{ij}, for instance) will be computed by assembling elemental contributions coming from all the cells that share vertex i, as follows

$$J_{ij} = \sum_{e \ni i} J_{ij}^{e}. \tag{19}$$

Eq. (19) follows by applying the sum rule of differentiation and by observing that the nodal residual itself is a sum of contributions from the elements that share vertex i, see **Figure 1(b)**. Specifically, element J_{ij}^{e} accounts for the contribution of cell e to the residual change at gridpoint i, due to a change in the state vector of a neighboring gridpoint j that belongs to the same element e. The contribution of cell e to the element (p, q) of the block J_{ij} is computed from the following one-sided FD formula

$$\left(J_{i,j}^{e}\right)_{p,q} = \frac{\left(R_g^e\right)_i^p\left(U_i, \widehat{U}_j^q, \ldots\right) - \left(R_g^e\right)(U_i, U_j, \ldots)}{\varepsilon} \quad 1 \le p, q \le m; \;\; i, j \in e, \tag{20}$$

where $\left(R_g^e\right)_i^p$ is the p^{th} component of the contribution of cell e to the nodal residual of gridpoint i. In Eq. (20) we have emphasized that $\left(R_g^e\right)_i$ only depends upon the flow state of the $d+1$ vertices of cell e, which include both i and j. The first partial derivative, $\left(R_g^e\right)_i^p\left(U_i, \widehat{U}_j^q, \ldots\right)$ is computed by perturbing the q^{th} component of the conserved variables vector at gridpoint j as follows

$$\widehat{U}_j^q = \left(u_j^1, u_j^2, \ldots, u_j^q + \varepsilon\left(u_j^q\right), \ldots, u_j^m\right), \tag{21}$$

where ε is a "small" quantity. Due to the use of a one-sided FD formula, the FD approximation (20) of the Jacobian entry is affected by a truncation error which is proportional to the first

power of ε. Small values of ε keep the truncation error small, but too small values may lead to round-off errors. Following [21], ε is computed as

$$\varepsilon(x) = \sqrt{\varepsilon_{mc}}\max(|x|, 1) \quad \text{sgn}\,(x). \tag{22}$$

From a coding viewpoint, the same loop over all cells used to build the nodal residual R_g is also used to assemble matrix $[J]$. The operations to be performed within each cell are the following: *i*) perturb each of the m components of the conserved variables vector of the $d+1$ vertices of cell e; *ii*) evaluate the residual contribution to each of the vertices; *iii*) calculate the Jacobian entries according to Eq. (20). While looping over cell e, this contributes $(d+1)^2$ block entries to the global matrix $[J]$. Moreover, it follows that the cost of a Jacobian evaluation is equal to $m \times (d+1)$ residual evaluations, which can be quite a large number. For instance, for a 3D compressible RANS calculation using a one-equation turbulence model, $m \times (d+1) = 24$. In this study, it was decided to store the Jacobian matrix in memory rather than using a Jacobian-free (JFNK), as the Jacobian matrix is relatively sparse even for a second-order accurate discretization due to the compactness of the stencil. The JFNK approach avoids assembling and, more important, storing the Jacobian matrix. However, the matrix-vector product is replaced with the Jacobian matrix by FD formulae which requires extra costly FS residual evaluations. Note that the JFNK method still requires the construction of a preconditioner to be used by the iterative linear solver, often an Incomplete Lower Upper factorization, which is typically constructed using a lower order approximation of the residual vector. Matrix-free preconditioners might also be used [22], saving a huge storage at the expense of extra CPU cost. These latter are referred to as MFNK methods. Although JFNK or MFNK approaches should certainly be favored from the viewpoint of memory occupation, it cannot always be "assumed that the Jacobian-free matrix-vector products are inherently advantageous in terms of computing time" [13].

The compactness of the FS stencil, which never extends beyond the set of distance-1 neighbors even for a second-order-accurate space-time discretization, offers two advantages. On one hand, apart from the truncation and round-off errors involved in the FD derivatives, the numerical Jacobian matrix is a close approximation of the analytical Jacobian, even for a second-order-accurate discretization. This feature is crucial for retaining the quadratic convergence properties of Newton's algorithm. On the other hand, it is considerably sparser than that obtained using more traditional FV discretizations, which typically extend up to distance-2 [1] or even distance-3 neighbors [8]. In these latter cases, contributions from the outermost gridpoints in the stencil have to be neglected [1], or at least lumped [8], when constructing the Jacobian approximation upon which the ILU(ℓ) preconditioner is built. These approximations are a potential source of performance degradation as reported in [8]. The memory occupation required to store the Jacobian matrix still remains remarkable. Moreover, not only the Jacobian matrix, but also its preconditioner needs to be stored in the computer memory. It is therefore clear that a key ingredient that would help reducing memory occupation is an effective preconditioner having a relatively small number of non-zero entries, as close as possible to that of the Jacobian matrix. This demanding problem is addressed in the next section.

http://dx.doi.org/10.5772/intechopen.72043

4. Linear solve and preconditioning

The previous discussions have pointed that the solution of the large nonsymmetric sparse linear system (15) at each pseudo-time step is a major computational task of the whole flow simulation, especially when the mean flow and the turbulence transport equations are solved in fully coupled form, the Jacobian is computed *exactly* by means of FD, and the size of the time-step is rapidly increased to recover Newton's algorithm. For convenience, we write system (15) in compact form as

$$Ax = b, \tag{23}$$

where $A = [a_{ij}]$ is the large and sparse coefficient matrix of, say, size n, and b is the right-hand side vector. It is well established that, when A is highly nonsymmetric and/or indefinite, iterative methods need the assistance of preconditioning to transform system (23) into an equivalent one that is more amenable to an iterative solver. The transformed *preconditioned system* writes in the form $M^{-1}Ax = M^{-1}b$ when preconditioning is applied from the left, or $AM^{-1}y = b$ with $x = M^{-1}y$ when preconditioning is applied from the right. The matrix M is a nonsingular approximation to A called the *preconditioner matrix*. In the coming sections, we describe the development of an effective algebraic preconditioner for the RANS model.

4.1. Multi-elimination ILU factorization preconditioner

Incomplete LU factorization methods (ILUs) are an effective, yet simple, class of preconditioning techniques for solving large linear systems. They write in the form $M = \overline{L}\,\overline{U}$, where $\overline{L}$ and $\overline{U}$ are approximations of the L and U factors of the standard triangular LU decomposition of A. The incomplete factorization may be computed directly from the Gaussian Elimination (GE) algorithm, by discarding some entries in the L and U factors according to various strategies, see [3]. A stable ILU factorization is proved to exist for arbitrary choices of the sparsity pattern of $\overline{L}$ and $\overline{U}$ only for particular classes of matrices, such as M-matrices [23] and H-matrices with positive diagonal entries [24]. However, many techniques can help improve the quality of the preconditioner on more general problems, such as reordering, scaling, diagonal shifting, pivoting and condition estimators [25–28]. As a result of this recent development, in the past decade successful experience have been reported using ILU preconditioners in areas that were of exclusive domain of direct solution methods like, in circuits simulation, power system networks, chemical engineering plants modeling, graphs and other problems not governed by PDEs, or in areas where direct methods have been traditionally preferred, such as structural analysis, semiconductor device modeling, computational fluid dynamics (see [29–33]).

Multi-elimination ILU factorization is a powerful class of ILU preconditioners, which combines the simplicity of ILU techniques with the robustness and high degree of parallelism of domain decomposition methods [34]. It is developed on the idea that, due to sparsity, many unknowns of a linear system are not coupled by an equation (i.e. they are *independent*) and thus they can be eliminated simultaneously at a given stage of GE. If the, say m, independent

unknowns are numbered first, and the other $n - m$ unknowns last, the coefficient matrix of the system is permuted in a 2×2 block structure of the form

$$PAP^T = \begin{pmatrix} D & F \\ E & C \end{pmatrix}, \tag{24}$$

where D is a diagonal matrix of dimension m and C is a square matrix of dimension $n - m$. In multi-elimination methods, a reduced system is recursively constructed from (24) by computing a block LU factorization of PAP^T of the form

$$\begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ G & I_{n-m} \end{pmatrix} \times \begin{pmatrix} U & W \\ 0 & A_1 \end{pmatrix}, \tag{25}$$

where L and U are the triangular factors of the LU factorization of D, $A_1 = C - ED^{-1}F$ is the Schur complement with respect to C, I_{n-m} is the identity matrix of dimension $n - m$, and we denote $G = EU^{-1}$ and $W = L^{-1}F$. The reduction process can be applied another time to the reduced system with A_1, and recursively to each consecutively reduced system until the Schur complement is small enough to be solved with a standard method such as a dense LAPACK solver [35]. Multi-elimination ILU factorization preconditioners may be obtained from the decomposition (25) by performing the reduction process inexactly, by dropping small entries in the Schur complement matrix and/or factorizing D approximately at each reduction step. These preconditioners exhibit better parallelism than conventional ILU algorithms, due to the recursive factorization. Additionally, for comparable memory usage, they may be significantly more robust especially for solving large problems as the reduced system is typically small and better conditioned compared to the full system.

The factorization (25) defines a general framework which may accommodate for many different methods. An important distinction between various methods is rooted in the choice of the algorithm used to discover sets of independent unknowns. Many of these algorithms are borrowed from graph theory, where such sets are referred to as *independent sets*. Denoting as $G = (V, E)$ the adjacency graph of A, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and E the set of edges, a *vertex independent set* S is defined as a subset of V such that

$$\forall v_i \in S, \forall v_j \in S : (v_i, v_j) \notin E. \tag{26}$$

The set S is maximal if there is no other independent set containing S strictly [36]. Independent sets in a graph may be computed by simple greedy algorithms which traverse the vertices in the natural order $1, 2, \ldots, n$, mark each visited vertex v and all of its nearest neighbors connected to v by an edge, and add v and each visited node that is not already marked to the independent set [37]. As an alternative to the greedy algorithm, the nested dissection ordering [38], mesh partitioning, or further information from the set of nested finite element grids of the underlying problem can be used [39–41].

The multilevel preconditioner considered in our study is the Algebraic Recursive Multilevel Solvers (ARMS) introduced in [25], which uses block independent sets computed by the simple

http://dx.doi.org/10.5772/intechopen.72043

greedy algorithm. Block independent sets are characterized by the property that unknowns of two different sets have no coupling, while unknowns within the same set may be coupled. In this case, the matrix D appearing in (24) is block diagonal, and may typically consist of large-sized diagonal blocks that are factorized by an ILU factorization with threshold (ILUT [42]) for memory efficiency. In the ARMS implementation described in [25], first the incomplete triangular factors $\overline{L}$, $\overline{U}$ of D are computed by one sweep of ILUT, and an approximation $\overline{W}$ to $\overline{L}^{-1}F$ is also computed. In a second loop, an approximation $\overline{G}$ to $E\overline{U}^{-1}$ and an approximate Schur complement matrix $\overline{A}_1$ are derived. This holds at each reduction level. At the last level, another sweep of ILUT is applied to the (last) reduced system. The blocks $\overline{W}$ and $\overline{G}$ are stored temporarily, and then discarded from the data structure after the Schur complement matrix is computed. Only the incomplete factors of D at each level, those of the last level Schur matrix, and the permutation arrays are needed for the solving phase. By this implementation, dropping can be performed separately in the matrices $\overline{L}$, $\overline{U}$, $\overline{W}$, $\overline{G}$, $\overline{A}_1$. This in turns allows to factor D accurately without incurring additional costs in $\overline{G}$ and $\overline{W}$, achieving high computational and memory efficiency. Implementation details and careful selection of the parameters are always critical aspects to consider in the design of sparse matrix algorithms. Next, we show how to combine the ARMS method with matrix compression techniques to exploit the block structure of A for better efficiency.

4.2. The variable-block ARMS factorization

The discretization of the Navier-Stokes equations for turbulent compressible flows assigns five distinct variables to each grid point (density, scaled energy, two components of the scaled velocity, and turbulence transport variable); these reduce to four for incompressible, constant density flows, and to three if additionally the flow is laminar. If the, say ℓ, distinct variables associated with the same node are numbered consecutively, the permuted matrix has a sparse block structure with non-zero blocks of size $\ell \times \ell$. The blocks are usually fully dense, as variables at the same node are mutually coupled. Exploiting any available block structure in the preconditioner design may bring several benefits [43], some of them are explained below:

1. *Memory.* A clear advantage is to store the matrix as a collection of blocks using the variable-block compressed sparse row (VBCSR) format, saving column indices and pointers for the block entries.
2. *Stability.* On indefinite problems, computing with blocks instead of single elements enables a better control of pivot breakdowns, near singularities, and other possible sources of numerical instabilities. Block ILU solvers may be used instead of pointwise ILU methods.
3. *Complexity.* Grouping variables in clusters, the Schur complement is smaller and hopefully the last reduced system is better conditioned and easier to solve.
4. *Efficiency.* A full block implementation, based on higher level optimized BLAS as computational kernels, may be designed leading to better flops to memory ratios on modern cache-based computer architectures.
5. *Cache effects.* Better cache reuse is possible for block algorithms.

It has been demonstrated that block iterative methods often exhibit faster convergence rate than their pointwise analogues for the solution of many classes of two- and three-dimensional partial differential equations (PDEs) [44–46]. For this reason, in the case of the simple Poisson's equation with Dirichlet boundary conditions on a rectangle $(0, \ell_1) \times (0, \ell_2)$ discretized uniformly by using $n_1 + 2$ points in the interval $(0, \ell_1)$ and $n_2 + 2$ points in $(0, \ell_2)$, it is often convenient to number the interior points by lines from the bottom up in the natural ordering, so that one obtains a $n_2 \times n_2$ block tridiagonal matrix with square blocks of size $n_1 \times n_1$; the diagonal blocks are tridiagonal matrices and the off-diagonal blocks are diagonal matrices. For large finite element discretizations, it is common to use substructuring, where each substructure of the physical mesh corresponds to one sparse block of the system. If the domain is highly irregular or the matrix does not correspond to a differential equation, finding the best block partitioning is much less obvious. In this case, graph reordering techniques are worth considering.

The PArameterized BLock Ordering (PABLO) method proposed by O'Neil and Szyld is one of the first block reordering algorithms for sparse matrices [47]. The algorithm selects groups of nodes in the adjacency graph of the coefficient matrix such that the corresponding diagonal blocks are either full or very dense. It has been shown that classical block stationary iterative methods such as block Gauss-Seidel and SOR methods combined with the PABLO ordering require fewer operations than their point analogues for the finite element discretization of a Dirichlet problem on a graded L-shaped region, as well as on the 9-point discretization of the Laplacian operator on a square grid. The complexity of the PABLO algorithm is proportional to the number of nodes and edges in both time and space.

Another useful approach to compute dense blocks in the sparsity pattern of a matrix A is the method proposed by Ashcraft in [48]. The algorithm searches for sets of rows or columns having the exact same pattern. From a graph viewpoint, it looks for vertices of the adjacency graph (V, E) of A having the same adjacency list. These are also called *indistinguishable nodes* or *cliques*. The algorithm assigns a *checksum* quantity to each vertex, using the function

$$chk(u) = \sum_{(u,w) \in E} w, \tag{27}$$

and then sorts the vertices by their checksums. This operation takes $|E| + |V| \log |V|$ time. If u and v are indistinguishable, then $chk(u) = chk(v)$. Therefore, the algorithm examines nodes having the same checksum to see if they are indistinguishable. The ideal checksum function would assign a different value for each different row pattern that occurs but it is not practical because it may quickly lead to huge numbers that may not even be machine-representable. Since the time cost required by Ashcraft's method is generally negligible relative to the time it takes to solve the system, simple checksum functions such as (27) are used in practice [48].

On the other hand, sparse unstructured matrices may sometimes exhibit *approximate dense blocks* consisting mostly of non-zero entries, except for a few zeros inside the blocks. By treating these few zeros as non-zero elements, with a little sacrifice of memory, a block ordering may be generated for an iterative solver. Approximate dense blocks in a matrix may be computed by numbering consecutively rows and columns having a similar non-zero structure. However, this would require a new checksum function that preserves the proximity of patterns, in the

sense that close patterns would result in close checksum values. Unfortunately, this property does not hold true for Ashcraft's algorithm in its original form. In [49], Saad proposed to compare angles of rows (or columns) to compute approximate dense structures in a matrix A. Let C be the pattern matrix of A, which by definition has the same pattern as A and the non-zero values are equal to 1. The method proposed by Saad computes the upper triangular part of CC^T. Entry (i, j) is the inner product (the cosine value) between row i and row j of C for $j > i$. A parameter τ is used to gauge the proximity of row patterns. If the cosine of the angle between rows i and j is smaller than τ, row j is added to the group of row i. For $\tau = 1$ the method will compute perfectly dense blocks, while for $\tau < 1$ it may compute larger blocks where some zero entries are padded in the pattern. To speed up the search, it may be convenient to run a first pass with the checksum algorithm to detect rows having an identical pattern, and group them together; then, in a second pass, each non-assigned row is scanned again to determine whether it can be added to an existing group. Two important performance measures to gauge the quality of the block ordering computed are the *average block density* (*av_bd*) value, defined as the amount of non-zeros in the matrix divided by the amount of elements in the non-zero blocks, and the *average block size* (*av_bs*) value, which is the ratio between the sum of dimensions of the square diagonal blocks divided by the number of diagonal blocks. The cost of Saad's method is closer to that of checksum-based methods for cases in which a good blocking already exists, and in most cases it remains inferior to the cost of the least expensive block LU factorization, i.e. block ILU(0).

Our recently developed variable-block variant of the ARMS method (VBARMS) incorporates an angle-based compression technique during the factorization to detect fine-grained dense structures in the linear system automatically, without any users knowledge of the underlying problem, and exploits them to improve the overall robustness and throughput of the basic multilevel algorithm [50]. It is simpler to describe VBARMS from a graph point of view. Suppose to permute A in block form as

$$\tilde{A} \approx P_B A P_B^T = \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} & \cdots & \tilde{A}_{1p} \\ \tilde{A}_{21} & \tilde{A}_{22} & \cdots & \tilde{A}_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{A}_{p1} & \tilde{A}_{p2} & \cdots & \tilde{A}_{pp} \end{bmatrix}, \tag{28}$$

where the diagonal blocks $\tilde{A}_{ii}$, $i = 1, \ldots, p$ are $n_i \times n_i$ and the off-diagonal blocks $\tilde{A}_{ij}$ are $n_i \times n_j$. We use upper case letters to denote matrix sub-blocks and lower case letters for individual matrix entries. We may represent the adjacency graph of $\tilde{A}$ by the quotient graph of $A + A^T$ [36]. Calling $\mathcal{B}$ the partition into blocks given by (28), we denote as $\mathcal{G}/\mathcal{B} = \{V_\mathcal{B}, E_\mathcal{B}\}$ the quotient graph obtained by coalescing the vertices assigned to the block $\tilde{A}_{ii}$ (for $i = 1, \ldots, p$) into a supervertex Y_i. In other words, the entry in position (i, j) of $\tilde{A}$ is a block of dimension $|Y_i| \times |Y_j|$, where $|X|$ is the cardinality of the set X. With this notation, the quotient graph $\mathcal{G}/\mathcal{B} = \{V_\mathcal{B}, E_\mathcal{B}\}$ is defined as

$$V_\mathcal{B} = \{Y_1, \ldots, Y_p\}, E_\mathcal{B} = \{(Y_i, Y_j) \mid \exists v \in Y_i, w \in Y_j \text{ s.t.} (v, w) \in E\}. \tag{29}$$

An edge connects two supervertices Y_i and Y_j if there exists an edge from a vertex in A_{ii} to a vertex in A_{jj} in the graph $\{V, E\}$ of $A + A^T$.

The complete pre-processing and factorization process of VBARMS consists of the following steps.

Step 1. Find the block ordering P_B of A such that, upon permutation, the matrix $P_B A P_B^T$ has fairly dense non-zero blocks. We use the angle-based graph compression algorithm proposed by Saad and described earlier to compute exact or approximate block structures in A.

Step 2. Scale the matrix at Step 1 in the form $S_1 P_B A P_B^T S_2$ using two diagonal matrices S_1 and S_2, so that the 1-norm of the largest entry in each row and column is smaller or equal than 1.

Step 3. Find the block independent sets ordering P_I of the quotient graph $\mathcal{G}/\mathcal{B} = \{V_{\mathcal{B}}, E_{\mathcal{B}}\}$. Apply the permutation to the matrix obtained at Step 2 as

$$P_I S_1 P_B A P_B^T S_2 P_I^T = \begin{pmatrix} D & F \\ E & C \end{pmatrix}. \tag{30}$$

We use a simple form of weighted greedy algorithm for computing the ordering P_I. The algorithm is the same as the one used in ARMS, and described in [25]. It consists of traversing the vertices $\mathcal{G}/\mathcal{B}$ in the natural order $1, 2, \ldots, n$, marking each visited vertex v and all of its nearest neighbors connected to v by an edge and adding v and each visited node that is not already marked to the independent set. We assign the weight $\|Y\|_F$ to each supervertex Y.

In the 2×2 partitioning (30), the upper left-most matrix D is block diagonal like in ARMS. However, due to the block permutation, the diagonal blocks of D are additionally block sparse matrices, as opposed to simply sparse matrices in ARMS and in other forms of multilevel incomplete LU factorizations, see [51, 52]. The matrices F, E, C are also block sparse because of the same reason.

Step 4. Factorize the matrix (30) in the form

$$\begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ EU^{-1} & I \end{pmatrix} \times \begin{pmatrix} U & L^{-1}F \\ 0 & A_1 \end{pmatrix}, \tag{31}$$

where I is the identity matrix of appropriate size, and form the reduced system with the Schur complement

$$A_1 = C - ED^{-1}F. \tag{32}$$

The Schur complement is also block sparse and has the same block partitioning of C.

Steps 2–4 can be repeated on the reduced system a few times until the Schur complement is small enough. After one additional level, we obtain

$$P_I^{(1)} S_1^{(1)} A_1 S_2^{(1)} (P_I^{(1)})^T = \left[\begin{array}{cc|c} D & F_1 & F_2 \\ E_1 & C_{11} & C_{12} \\ \hline E_2 & C_{21} & C_{22} \end{array}\right], \tag{33}$$

that can be factored as

http://dx.doi.org/10.5772/intechopen.72043

$$\left[\begin{array}{cc|c} L_D & 0 & 0 \\ L_{E_1} & I & 0 \\ \hline L_{E_2} & L_{C_{21}} & I \end{array}\right] \left[\begin{array}{cc|c} D & 0 & 0 \\ 0 & D_{C_{11}} & 0 \\ \hline 0 & 0 & A_2 \end{array}\right] \left[\begin{array}{cc|c} U_D & U_{F_1} & U_{F_2} \\ 0 & I & U_{C_{12}} \\ \hline 0 & 0 & I \end{array}\right]. \tag{34}$$

Denote as A_ℓ the reduced Schur complement matrix at level ℓ, for $\ell > 1$. After scaling and preordering A_ℓ, a system with the matrix

$$P_I^{(\ell)} D_1^{(\ell)} A_\ell D_2^{(\ell)} \left(P_I^{(\ell)}\right)^T = \begin{pmatrix} D_\ell & F_\ell \\ E_\ell & C_\ell \end{pmatrix} = \begin{pmatrix} L_\ell & 0 \\ E_\ell U_\ell^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_\ell & L_\ell^{-1} F_\ell \\ 0 & A_{\ell+1} \end{pmatrix} \tag{35}$$

needs to be solved, with

$$A_{\ell+1} = C_\ell - E_\ell D_\ell^{-1} F_\ell. \tag{36}$$

Calling

$$x_\ell = \begin{pmatrix} y_\ell \\ z_\ell \end{pmatrix}, \; b_\ell = \begin{pmatrix} f_\ell \\ g_\ell \end{pmatrix} \tag{37}$$

the unknown solution vector and the right-hand side vector of system (35), the solution process with the above multilevel VBARMS factorization consists of level-by-level forward elimination followed by an exact solution on the last reduced system and suitable inverse permutation. The solving phase is sketched in Algorithm 1.

In VBARMS, we perform the factorization approximately, for memory efficiency. We use block ILU factorization with threshold to invert inexactly both the upper leftmost matrix $D_\ell \approx \overline{L}_\ell \overline{U}_\ell$ at each level ℓ, and the last level Schur complement matrix $A_{\ell_{max}} \approx \overline{L}_S \overline{U}_S$. The block ILU method used in VBARMS is a straightforward block variant of the one-level pointwise ILUT algorithm. We drop small blocks $B \in R^{m_B \times n_B}$ in $\overline{L}_\ell$, $\overline{U}_\ell$, $\overline{L}_S$, $\overline{U}_S$ whenever $\frac{\|B\|_F}{m_B \cdot n_B} < t$, for a given user-defined threshold t. The block pivots in block ILU are inverted exactly by using GE with partial pivoting. In assembling the Schur complement matrix $A_{\ell+1}$ at level ℓ, we take advantage of the finest block structure of D_ℓ, F_ℓ, E_ℓ, C_ℓ, imposed by the block ordering P_B on the small (usually dense) blocks in the diagonal blocks of D_ℓ and the corresponding small off-diagonal blocks in E_ℓ and F_ℓ; we call optimized level-3 BLAS routines [53] for computing $A_{\ell+1}$ in Eq. (36). We do not drop entries in the Schur complement, except at the last level. The same threshold is applied in all these operations.

The VBARMS code is developed in the C language and is adapted from the existing ARMS code available in the ITSOL package [54]. The compressed sparse storage format of ARMS is modified to store block vectors and block matrices of variable size as a collection of contiguous non-zero dense blocks (we refer to this data storage format as VBCSR). First, we compute the factors $\overline{L}_\ell$, $\overline{U}_\ell$ and $\overline{L}_\ell^{-1} F_\ell$ by performing a variant of the IKJ version of the Gaussian Elimination algorithm, where index I runs from 2 to m_ℓ, index K from 1 to $(I-1)$ and index J from $(K+1)$ to n_ℓ. This loop applies implicitly $\overline{L}_\ell^{-1}$ to the block row $[D_\ell, F_\ell]$ to produce $\left[U_\ell, \overline{L}_\ell^{-1} F_\ell\right]$. In the second loop, Gaussian Elimination is performed on the block row $[E_\ell, C_\ell]$ using the multipliers

computed in the first loop to give $E_\ell \overline{U}_\ell^{-1}$ and an approximation of the Schur complement $A_{\ell+1}$. Then, after Step 1, we permute explicitly the matrix at the first level as well as the matrices involved in the factorization at each new reordering step. For extensive performance assessment results of the VBARMS method, we point the reader to [50].

Algorithm 1 `VBARMS_Solve`$(A_{\ell+1}, b_\ell)$. The solving phase with the VBARMS method.

Require: $\ell \in \mathbb{N}^*$, $\ell_{max} \in \mathbb{N}^*$, $b_\ell = \left(f_\ell, g_\ell\right)^T$

1: Solve $L_\ell y = f_\ell$

2: Compute $g'_\ell = g_\ell - E_\ell U_\ell^{-1} y$

3: **if** $\ell = \ell_{max}$ **then**

4: Solve $A_{\ell+1} z_\ell = g'_\ell$

5: else

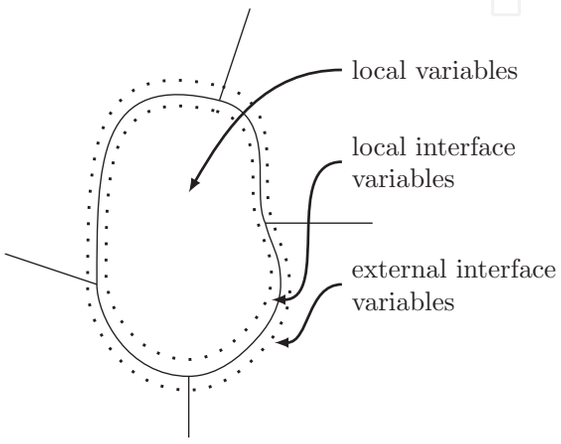
6: Call `VBARMS_Solve`$(A_{\ell+1}, g'_\ell)$

7: **end if**

8: Solve $U_\ell y_\ell = \left[y - L_\ell^{-1} F_\ell z_\ell\right]$

5. Numerical experiments

In this section, we illustrate the performance of the VBARMS method for solving a suite of block structured linear systems arising from an implicit Newton-Krylov formulation of the RANS equations in the turbulent incompressible flow analysis past a three-dimensional wing. On multicore machines, the quotient graph $\mathcal{G}/\mathcal{B}$ is split into distinct subdomains, and each of them is assigned to a different core. Following the parallel framework described in [55], we separate the nodes assigned to the ith subdomain into *interior nodes*, that are those coupled by the equations only with the local variables, and *interface nodes*, those that may be coupled with the local variables stored on processor i as well as with remote variables stored on other processors (see Figure below).



The vector of the local unknowns x_i and the local right-hand side b_i are split accordingly in two separate components: the subvector corresponding to the internal nodes followed by the subvector of the local interface variables

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix}, \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{38}$$

The rows of A indexed by the nodes of the ith subdomain are assigned to the ith processor. These are naturally separated into a local matrix A_i acting on the local variables $x_i = (u_i, y_i)^T$, and an interface matrix U_i acting on the remotely stored subvectors of the external interface variables $y_{i,\text{ext}}$. Hence, we can write the local equations on processor i as

$$A_i x_i + U_{i,\text{ext}} y_{i,\text{ext}} = b_i \tag{39}$$

or, in expanded form, as

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}, \tag{40}$$

where N_i is the set of subdomains that are neighbors to subdomain i and the submatrix $E_{ij}y_j$ accounts for the contribution to the local equation from the jth neighboring subdomain. Note that matrices B_i, C_i, E_i, and F_i still preserve the fine block structure imposed by the block ordering P_B. From a code viewpoint, the quotient graph is initially distributed amongst the available processors; then, the built-in parallel hypergraph partitioner available in the Zoltan package [56] is applied on the distributed data structure to compute an optimal partitioning of the quotient graph that can minimize the amount of communications.

At this stage, the VBARMS method described in Section 4.2 can be used as a local solver for different types of global preconditioners. In the simplest parallel implementation, the so-called block-Jacobi preconditioner, the sequential VBARMS method can be applied to invert approximately each local matrix A_i. The standard Jacobi iteration for solving $Ax = b$ is defined as

$$x_{n+1} = x_n + D^{-1}(b - Ax_n) = D^{-1}(Nx_n + b), \tag{41}$$

where D is the diagonal of A, $N = D - A$ and x_0 is some initial approximation. In cases we have a graph partitioned matrix, the matrix D is block diagonal and the diagonal blocks of D are the local matrices A_i. The interest to consider the block Jacobi preconditioner is its inherent parallelism, since the solves with the matrices A_i are performed independently on all the processors and no communication is required.

If the diagonal blocks of the matrix D are enlarged in the block-Jacobi method so that they overlap slightly, the resulting preconditioner is called Schwarz preconditioner. Consider again a graph partitioned matrix with N nonoverlapping sets W_i^0, $i = 1, \ldots, N$ and $W_0 = \cup_{i=1}^{N} W_0^i$. We define a δ-overlap partition

$$W^{\delta} = \bigcup_{i=1}^{N} W_i^{\delta} \tag{42}$$

where $W_i^{\delta} = adj\left(W_i^{\delta-1}\right)$ and $\delta > 0$ is the level of overlap with the neighboring domains. For each subdomain, we define a restriction operator R_i^{δ}, which is an $n \times n$ matrix with the (j,j)th element equal to 1 if $j \in W_i^{\delta}$, and zero elsewhere. We then denote

$$A_i = R_i^{\delta} A R_i^{\delta}. \tag{43}$$

The global preconditioning matrix M_{RAS} is defined as

$$M_{RAS}^{-1} = \sum_{i=1}^{s} R_i^T A_i^{-1} R_i \tag{44}$$

and named as the Restricted Additive Schwarz (RAS) preconditioner [3, 57]. Note that the preconditioning step still offers a good scope par parallelism, as the different components of the error update are formed independently. However, due to overlapping some communication is required in the final update, as the components are added up from each subdomain. In our experiments, the overlap used for RAS was the level 1 neighbors of the local nodes in the quotient graph.

A third global preconditioner that we consider in this study is based on the Schur complement approach. In Eq. (40), we can eliminate the vector of interior unknowns u_i from the first equations to compute the local Schur complement system

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i', \tag{45}$$

where S_i denotes the local Schur complement matrix

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{46}$$

The local Schur complement equations considered altogether write as the global Schur complement system

$$\begin{pmatrix} S_1 & E_{12} & \dots & E_{1p} \\ E_{21} & S_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \dots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}, \tag{47}$$

where the off-diagonal matrices E_{ij} are available from the parallel distribution of the linear system. One preconditioning step with the Schur complement preconditioner consists in solving

http://dx.doi.org/10.5772/intechopen.72043

approximately the global system (47), and then recovering the u_i variables from the local equations as

$$u_i = B_i^{-1}\left[f_i - F_i y_i\right] \tag{48}$$

at the cost of one local solve. We solve the global system (47) by running a few steps of the GMRES method preconditioned by a block diagonal matrix, where the diagonal blocks are the local Schur complements S_i. The factorization

$$S_i = L_{S_i} U_{S_i} \tag{49}$$

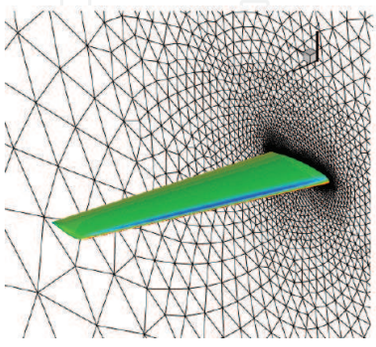
is obtained as by-product of the LU factorization of the local matrix A_i,

$$A_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix} \tag{50}$$

which is by the way required to compute the u_i variables in Eq. (48).

5.1. Results

The parallel experiments were run on the large-memory nodes (32 cores/node and 1 TB of memory) of the TACC Stampede system located at the University of Texas at Austin. TACC Stampede is a 10 PFLOPS (PF) Dell Linux Cluster based on 6400+ Dell PowerEdge server nodes, each outfitted with 2 Intel Xeon E5 (Sandy Bridge) processors and an Intel Xeon Phi Coprocessor (MIC Architecture). We linked the default vendor BLAS library, which is MKL. Although MKL is multi-threaded by default, in our runs we used it in a single-thread mode since our MPI-based parallelisation employed one MPI process per core (communicating via the shared memory for the same-node cores). We used the Flexible GMRES (FGMRES) method [58] as Krylov subspace method, a tolerance of $1.0e-6$ in the stopping criterion and a



Ref. Area, S = 290322 mm^2 = 450 in^2
Ref. Chord, c = 197.556 mm = 7.778 in
Ref. Span, b = 1524 mm = 60 in

RANS1 :	$n = 4918165$	$nnz = 318370485$
RANS2 :	$n = 4918165$	$nnz = 318370485$
RANS3 :	$n = 9032110$	$nnz = 670075950$
RANS4 :	$n = 12085410$	$nnz = 893964000$
RANS5 :	$n = 22384845$	$nnz = 1659721325$

Figure 2. Geometry and mesh characteristics of the DPW3 Wing-1 problem proposed in the 3rd AIAA drag prediction workshop. Note that problems RANS1 and RANS2 correspond to the same mesh, and are generated at two different Newton steps.

maximum number of iteration equal to 1000. Memory costs were calculated as the ratio between the sum of the number of non-zeros in the local preconditioners and the sum of the number of non-zeros in the local matrices A_i.

In our experiments, we analyzed the turbulent incompressible flow past a three-dimensional wing illustrated in **Figure 2** using the *EulFS* code developed by the second author [59]. The geometry, called DPW3 Wing-1, was proposed in the 3rd AIAA Drag Prediction Workshop [35]. Flow conditions are 0.5° angle of attack and Reynolds number based on the reference chord equal to $5 \cdot 10^6$. The freestream turbulent viscosity is set to 10% of its laminar value. In

Matrix	Method	Graph time (s)	Factorization time (s)	Solving time (s)	Total time (s)	Its	Mem
	BJ + VBARMS	17.3	8.58	41.54	50.13	34	2.98
RANS1	RAS + VBARMS	17.4	10.08	42.28	52.37	19	3.06
	SCHUR + VBARMS	17.6	11.94	55.99	67.93	35	2.57
	BJ + VBARMS	17.0	16.72	70.14	86.86	47	4.35
RANS2	RAS + VBARMS	16.8	21.65	80.24	101.89	39	4.49
	SCHUR + VBARMS	17.5	168.85	173.54	342.39	24	6.47
	BJ + VBARMS	27.2	99.41	187.95	287.36	154	4.40
RANS3	RAS + VBARMS	25.2	119.32	90.47	209.79	71	4.48
	SCHUR + VBARMS	22.0	52.65	721.67	774.31	140	4.39

Table 1. Experiments on the DPW3 Wing-1 problem. The RANS1, RANS2 and RANS3 test cases are solved on 32 processors. We ran one MPI process per core, so in these experiments we used shared memory on a single node.

Matrix	Method	Graph time (s)	Factorization time (s)	Solving time (s)	Total time (s)	Its	Mem
	BJ + VBARMS	51.5	12.05	105.89	117.94	223	3.91
RANS4	RAS + VBARMS	43.9	14.05	91.53	105.58	143	4.12
	SCHUR + VBARMS	39.3	15.14	289.89	305.03	179	3.76
RANS5	RAS + VBARMS	1203.94(1)	16.80	274.62	291.42	235	4.05

Table 2. Experiments on the DPW3 Wing-1 problem. The RANS4 and RANS5 test cases are solved on 128 processors. Note (1): due to a persistent problem with the Zoltan library on this run, we report on the result of our experiment with the metis (sequential) graph partitioner [60].

Solver	Number of processors	Graph time (s)	Total time (s)	Its	Mem
	8	38.9	388.37	27	5.70
	16	28.0	219.48	35	5.22
RAS + VBARMS	32	17.0	101.49	39	4.49
	64	16.0	54.19	47	3.91
	128	18.2	28.59	55	3.39

Table 3. Strong scalability study on the RANS2 problem using parallel graph partitioning.

Matrix	Method	Factorization time (s)	Solving time (s)	Total time (s)	Its	Mem
	pARMS	—	—	—	—	6.63
RANS3	BJ + VBARMS	99.41	187.95	287.36	154	4.40
	BJ + VBILUT	20.45	8997.82	9018.27	979	13.81
	pARMS	—	—	—	—	5.38
RANS4	BJ + VBARMS	12.05	105.89	117.94	223	3.91
	BJ + VBILUT	1.16	295.20	296.35	472	5.26

Table 4. Experiments on the DPW3 Wing-1 problem. The RANS3 test case is solved on 32 processors and the RANS4 problem on 128 processors. The dash symbol − in the table means that in the GMRES iteration the residual norm is very large and the program is aborted.

Tables 1 and **2** we show experiments with the parallel VBARMS solver on the five meshes of the DPW3 Wing-1 problem. On the largest mesh we report on only one experiment, in **Table 2**, as this is a resource demanding problem. In **Table 3** we report on a strong scalability study on the problem denoted as RANS2 by increasing the number of processors. Finally, in **Table 4** we show comparative results with parallel VBARMS against other popular solvers; the method denoted as pARMS is the solver described in [55] using default parameters while the method VBILUT is a variable-block incomplete lower-upper factorization with threshold from the ITSOL package [54]. The results of our experiments show that the proposed preconditioner is effective to reduce the number of iterations especially in combination with the Restricted Additive Schwarz method, and exhibits good parallel scalability. A truly parallel implementation of the VBARMS method that may offer better numerical scalability will be considered as the next step of this research.

6. Conclusions

The applicability of Newton's method in steady flow simulations is often limited by the difficulty to compute a good initial solution, namely, one lying in a reasonably small neighborhood of the sought solution. This problem can now be overcome by introducing some approximations in the first stages of the solution procedure. In the case of unsteady flow problems, on the other hand, the use of Newton's method in conjunction with a dual-time stepping procedure is even more effective since the flow field computed at the preceding physical time level is likely to be sufficiently close to the sought solution at the next time level to allow the use of Newton's algorithm right from the beginning of the sub-iterations in pseudo-time. On the downside of Newton-Krylov methods is the need for efficiently preconditioned iterative algorithms to solve the sparse linear system arising at each inner iteration (Newton step). The stiffness of the linear systems to be solved increases when the Jacobian is computed "exactly" and the turbulence transport equations are solved fully coupled with the mean flow equations.

In this chapter, we have presented a block multilevel incomplete factorization preconditioner for solving sparse systems of linear equations arising from the implicit RANS formulation. The method detects automatically any existing block structure in the matrix, without any user's prior

knowledge of the underlying problem, and exploits it to maximize computational efficiency. The results of this chapter show that, by taking advantage of this block structure, the solver can be more robust and efficient. Other recent studies on block ILU preconditioners have drawn similar conclusions on the importance of exposing dense blocks during the construction of the incomplete LU factorization for better performance, in the design of incomplete multifrontal LU-factorization preconditioners [61] and adaptive blocking approaches for blocked incomplete Cholesky factorization [62]. We believe that the proposed VBARMS method can be useful for solving linear systems also in other areas, such as in Electromagnetics applications [63–65].

Acknowledgements

The authors acknowledge the Texas Advanced Computing Center (TACC) at the University of Texas at Austin for providing HPC resources that have contributed to the research results reported in this chapter. URL: http://www.tacc.utexas.edu. The authors are grateful to Prof. Masha Sosonkina at the Department of Modeling Simulation & Visualization Engineering, Old Dominion University for help and insightful comments.

Nomenclature

Roman symbols

a	artificial sound speed
d	dimension of the space, $d = 2, 3$
e^0	specific total energy
h^0	specific total enthalpy
ℓ	level of fill in incomplete lower upper factorizations
m	number of degrees of freedom within a gridpoint
n	order of a matrix
nnz	number of non-zero entries in a sparse matrix
$\mathbf{n}$	unit inward normal to the control surface
p	static pressure
$\mathbf{q}$	flux vector due to heat conduction
t	time
$\mathbf{u}$	velocity vector
$\mathbf{x}$	vector of the d Cartesian coordinates

http://dx.doi.org/10.5772/intechopen.72043

C_i	median dual cell (control volume)
∂C_i	boundary of the median dual cell (control surface)
E	edges of a graph
$\mathbf{G}(A)$	graph of matrix A
M	global mass matrix
M_L	left preconditioning matrix
Ma	mach number
I	identity matrix
N_i	shape function
P	permutation matrix
Pr_T	turbulent Prandtl number
R_Φ	spatial residual vector
Re	Reynolds' number
T	triangle or tetrahedron
$\mathbf{U}$	conserved variables vector
V	vertices of a graph
V_M	lumped mass matrix
z	parameter vector

Greek symbols

α	angle of attack
Δt	physical time step
$\Delta\tau$	pseudo-time step
ΔU	$= U^{n+1,k+1} - U^{n+1,k}$
ε_{mc}	machine zero
ρ	density
ν	kinematic viscosity
$\tilde{\nu}$	working variable in the turbulence transport equation
τ	pseudo-time variable
$\underline{\underline{\tau}}$	Newtonian stress tensor

Φ	flux balance
Ω_i^e	Petrov Galerkin weighting function
Subscript	
i	nodal index or row index of a matrix
j	nodal index or column index of a matrix
e	cell index
∞	Free-stream condition
Superscript	
inv	inviscid
k	inner iterations counter
n	physical time step counter
T	transpose
vis	viscous

Author details

Bruno Carpentieri[1]* and Aldo Bonfiglioli[2]

*Address all correspondence to: bcarpentieri@gmail.com

1 Free University of Bolzano-Bozen, Faculty of Computer Science, Italy

2 Scuola di Ingegneria, University of Basilicata, Potenza, Italy

References

[1] Wong P, Zingg DW. Three-dimensional aerodynamic computations on unstructured grids using a Newton-Krylov approach. Computers & Fluids. 2008;**37**(2):107-120

[2] Bonfiglioli A, Campobasso S, Carpentieri B. Parallel unstructured three-dimensional turbulent flow analyses using efficiently preconditioned Newton-Krylov solvers. In: Proceedings of the 19th AIAA Computational Fluid Dynamics Conference 2009, San Antonio, Texas; 2009

[3] Saad Y. Iterative Methods for Sparse Linear Systems. 2nd edition. Philadelphia: SIAM; 2003

[4] Spalart PR, Allmaras SR. A one-equation turbulence model for aerodynamic flows. LaRecherche-Aerospatiale. 1994;**1**:5-21

[5] van der Weide E, Deconinck H, Issman E, Degrez G. A parallel, implicit, multi-dimensional upwind, residual distribution method for the Navier-Stokes equations on unstructured grids. Computational Mechanics. 1999;**23**:199-208

[6] Deconinck H, Roe PL, Struijs R. A multi-dimensional generalization of Roe's flux difference splitter for the Euler equations. Journal of Computers and Fluids. 1993;**22**(2/3):215-222

[7] Gropp WD, Kaushik DK, Keyes DE, Smith BF. High-performance parallel implicit cfd. Parallel Computing. 2001;**27**(4):337-362

[8] Lucas P, Bijl H, van Zuijlen AH. Efficient unsteady high Reynolds number flow computations on unstructured grids. Computers & Fluids. 2010;**39**(2):271-282

[9] Lucas P, van Zuijlen AH, Bijl H. Fast unsteady flow computations with a Jacobian-free Newton-Krylov algorithm. Journal of Computational Physics. 2010;**229**(24):9201-9215

[10] De Palma P, Pascazio, Rossiello G, Napolitano M. A second-order-accurate monotone implicit fluctuation splitting scheme for unsteady problems. Journal of Computational Physics. 2005;**208**(1):1-33

[11] Campobasso MS, Bonfiglioli A, Baba-Ahmadi M. Development of efficient and accurate CFD technologies for wind turbine unsteady aerodynamics. In CMFF'09 Conference on Modelling fluid flow; 2009

[12] Jameson A. Time-dependent calculations using multigrid with applications to unsteady flows past airfoils and wings. AIAA Paper. 1991;**91-1596**

[13] Chisholm TT, Zingg DW. A Jacobian-free Newton-Krylov algorithm for compressible turbulent fluid flows. Journal of Computational Physics. 2009;**228**(9):3490-3507

[14] Geuzaine P. Newton-Krylov strategy for compressible turbulent flows on unstructured meshes. AIAA Journal. October 2001;**39**(3):528-531

[15] Mulder W, van Leer B. Experiments with an implicit upwind method for the Euler equations. Journal of Computational Physics. 1985;**59**:232-246

[16] Blanco M, Zingg DW. A Newton-Krylov Algorithm with a loosely-coupled turbulence model for aerodynamic flows. 44th AIAA Aerospace Sciences Meeting and Exhibit, 9–12 Jan. Reno: Nevada Paper 2006–691; 2006

[17] Bonfiglioli A. Fluctuation splitting schemes for the compressible and incompressible Euler and Navier-Stokes equations. IJCFD. 2000;**14**:21-39

[18] Blanco M, Zingg DW. Newton-Krylov algorithm with a loosely coupled turbulence model for aerodynamic flows. AIAA Journal. 2007;**45**(5):980

[19] Anderson WK, Bonhaus DL. An implicit upwind algorithm for computing turbulent flows on unstructured grids. Journal of Computers and Fluids. 1994;**23**(1):1-21

[20] Nichols JC, Zingg DW. A Three-Dimensional Multi-Block Newton-Krylov Flow Solver for the Euler Equations. 2005. AIAA Paper 2005–5230

[21] Issman E. Implicit Solution Strategies for Compressible Flow Equations on Unstructured Meshes. [PhD thesis]. Bruxelles: Université Libre de Bruxelles; 1997

[22] Qin N, Ludlow DK, Shaw ST. A matrix-free preconditioned Newton/GMRES method for unsteady Navier-Stokes solutions. International Journal for Numerical Methods in Fluids. 2000;**33**(2):223-248

[23] Meijerink JA, van der Vorst HA. An iterative solution method for linear systems of which the coeffcient matrix is a symmetric M-matrix. Mathematics of Computation. 1977;**31**: 148-162

[24] Varga RS, Saff EB, Mehrmann V. Incomplete factorizations of matrices and connections with H-matrices. SIAM Journal on Numerical Analysis. 1980;**17**:787-793

[25] Saad Y, Suchomel B. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. Numerical Linear Algebra with Applications. 2002;**9**(5):359-378

[26] Bollhöfer M, Saad Y. Multilevel preconditioners constructed from inverse–based ILUs. SIAM Journal on Scientific Computing. 2006;**27**(5):1627-1650

[27] Duff IS, Koster J. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. SIAM Journal on Matrix Analysis and Applications. 1999;**20**(4): 889-901

[28] Magolu monga Made M, Beauwens R, Warzee G. Preconditioning of discrete Helmholtz operators perturbed by a diagonal complex matrix. Communications in Numerical Methods in Engineering. 2000;**11**:801-817

[29] Benzi M. Preconditioning techniques for large linear systems: A survey. Journal of Computational Physics. 2002;**182**:418-477

[30] Bollhöfer M. A robust and efficient ILU that incorporates the growth of the inverse triangular factors. SIAM Journal on Scientific Computing. January 2003;**25**(1):86-103

[31] Manguoglu M. A domain-decomposing parallel sparse linear system solver. Journal of Computational and Applied Mathematics. 2011;**236**(3):319-325

[32] Saad Y. Multilevel ILU with reorderings for diagonal dominance. SIAM Journal on Scientific Computing. 2005;**27**:1032-1057

[33] Saad Y, Soulaimani A, Touihri R. Variations on algebraic recursive multilevel solvers (ARMS) for the solution of CFD problems. Applied Numerical Mathematics. 2004;**51**:305-327

[34] Saad Y. ILUM: A multi-elimination ILU preconditioner for general sparse matrices. SIAM Journal on Scientific Computing. 1996;**17**(4):830-847

[35] Drag Prediction Workshop. URL: http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/Workshop3/workshop3.html

[36] George A, Liu JW. Computer Solution of Large Sparse Positive Definite Systems. Englewood Cliffs, New Jersey: Prentice-Hall; 1981

[37] Saad Y. ILUM: A multi-elimination ILU preconditioner for general sparse matrices. SIAM Journal on Scientific Computing. 1996;**17**(4):830-847

[38] George J. Nested dissection of a regular finite element mesh. SIAM Journal on Numerical Analysis. 1973;**10**:345-363

[39] Axelsson O, Vassilevski PS. Algebraic multilevel preconditioning methods, I. Numerische Mathematik. 1989;**56**:157-177

[40] Axelsson O, Vassilevski PS. Algebraic multilevel preconditioning methods. II. SIAM Journal on Numerical Analysis. 1990;**27**:1569-1590

[41] Botta EFF, van der Ploeg A, Wubs FW. Nested grids ILU-decomposition (NGILU). Journal of Computational and Applied Mathematics. 1996;**66**:515-526

[42] Saad Y. ILUT: A dual threshold incomplete LU factorization. Numerical Linear Algebra with Applications. 1994;**1**:387-402

[43] Chapman A, Saad Y, Wigton L. High-order ILU preconditioners for CFD problems. International Journal for Numerical Methods in Fluids. 2000;**33**(6):767-788

[44] Concus P, Golub GH, Meurant G. Block preconditioning for the conjugate gradient method. SIAM Journal on Scientific and Statistical Computing. 1985;**6**:220-252

[45] Meurant GA. Computer Solution of Large Linear Systems, Volume 28 of Studies in Mathematics and Its Applications. Amsterdam: North-Holland;1999

[46] Magolu monga Made M, Polman B. Experimental comparison of three-dimensional point and line modified incomplete factorizations. Numerical Algorithms, **23**(1):51-70, 2000

[47] O'Neil J, Szyld DB. A block ordering method for sparse matrices. SIAM Journal on Scientific and Statistical Computing. 1990;**11**(5):811-823

[48] Ashcraft C. Compressed graphs and the minimum degree algorithm. SIAM Journal on Scientific Computing. 1995;**16**(6):1404-1411

[49] Saad Y. Finding exact and approximate block structures for ilu preconditioning. SIAM Journal on Scientific Computing. 2002;**24**(4):1107-1123

[50] Carpentieri B, Liao J, Sosonkina M. VBARMS: A variable block algebraic recursive multilevel solver for sparse linear systems. Journal of Computational and Applied Mathematics. 2014;**259**(A):164-173

[51] Saad Y, Zhang J. BILUM: Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems. SIAM Journal on Scientific Computing. 1999;**20**:2103-2121

[52] Saad Y, Zhang J. BILUTM: A domain-based multilevel block ILUT preconditioner for general sparse matrices. SIAM Journal on Matrix Analysis and Applications. 1999;**21**(1):279-299

[53] Dongarra JJ, Du Croz J, Duff IS, Hammarling S. A set of level 3 basic linear algebra subprograms. ACM Transactions on Mathematical Software. 1990;**16**:1-17

[54] Na L, Suchomel B, Osei-Kuffuor D, Saad Y. ITSOL: Iterative Solvers Package. https://www-users.cs.umn.edu/~saad/software/ITSOL/

[55] Li Z, Saad Y, Sosonkina M. pARMS: A parallel version of the algebraic recursive multilevel solver. Numerical Linear Algebra with Applications. 2003;**10**:485-509

[56] Boman E, Devine K, Fisk LA, Heaphy R, Hendrickson B, Leung V, Vaughan V, Catalyurek U, Bozdag D, Mitchell W. Zoltan home page. http://www.cs.sandia.gov/Zoltan, 1999

[57] Quarteroni A, Valli A. Domain Decomposition Methods for Partial Differential Equations. Oxford: Clarendon Press Oxford; 1999

[58] Saad Y. A flexible inner-outer preconditioned GMRES algorithm. SIAM Journal on Scientific and Statistical Computing. 1993;**14**:461-469

[59] Bonfiglioli A, Carpentieri B, Sosonkina M. EulFS: a parallel CFD code for the simulation of Euler and Navier-Stokes problems on unstructured grids. In: Kågström B, Elmroth E, Dongarra J, Waśniewski J, editors. Applied Parallel Computing. State of the Art in Scientific Computing., volume 4699 of Lecture Notes in Computer Science. Springer-Verlag; 2007. pages 676-685

[60] Karypis G, Kumar V. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0. http://glaros.dtc.umn.edu/gkhome/views/metis. University of Minnesota, Department of Computer Science/Army HPC Research Center Minneapolis, MN 55455

[61] Gupta A, George T. Adaptive techniques for improving the performance of incomplete factorization preconditioning. SIAM Journal on Scientific Computing. 2010;**32**(1):84-110

[62] Vannieuwenhoven N, Meerbergen K. IMF: An incomplete multifrontal LU-factorization for element-structured sparse linear systems. SIAM Journal on Scientific Computing. 2013;**35**(1):A270-A293

[63] Carpentieri B, Liao J, Sosonkina M. Parallel Processing and Applied Mathematics, volume 8385 of Lecture Notes in Computer Science, chapter Variable block multilevel iterative solution of general sparse linear systems. In: Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J, editors. 11th International Conference, PPAM 2015. Krakow: Springer-Verlag. 2014. pp. 520-530

[64] Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra JJ, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D. LAPACK Users' Guide. third ed. Philadelphia, PA: Society for Industrial and Applied Mathematics; 1999

[65] Li X. An overview of SuperLU: Algorithms, implementation, and user interface. ACM Transactions on Mathematical Software. September 2005;**31**(3):302-325