Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Developing Parallel Large Eddy Simulation Software with Libraries for Sparse Matrix Computations

Andrea Aprovitola, Pasqua D'Ambra, Filippo Maria Denaro,

Daniela di Serafino, Salvatore Filippone

RT-ICAR-NA-2012-01 **Febbraio 2012**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Developing Parallel Large Eddy Simulation Software with Libraries for Sparse Matrix Computations[*]

Andrea Aprovitola[1], Pasqua D'Ambra[2], Filippo Maria Denaro[3],

Daniela di Serafino[4], Salvatore Filippone[5]

Rapporto Tecnico N: **RT-ICAR-NA-2012-01** ***Data:*** **Febbraio 2012**

[*] Preprint sottomesso per la pubblicazione su rivista.
[1] Istituto di Ricerca sulla Combustione, IRC-CNR, P.le V. Tecchio, 80, 80125 Napoli.
[2] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, via P. Castellino, 111, 80131 Napoli.
[3] Dipartimento di Ingegnerai Aerospaziale e Meccanica, Seconda Università di Napoli, via Roma, 29, 81031 Aversa and ICAR-CNR, Napoli..
[4] Dipartimento di Matematica, Seconda Università di Napoli, Viale Lincoln, 5, 81100 Caserta and ICAR-CNR, Napoli.
[5] Dipartimento di Ingegneria Meccanica, Università di Roma "Tor Vergata", v.le del Policlinico, 1, 00133 Roma.

DEVELOPING PARALLEL LARGE EDDY SIMULATION SOFTWARE WITH LIBRARIES FOR SPARSE MATRIX COMPUTATIONS

ANDREA APROVITOLA[†], PASQUA D'AMBRA[‡], FILIPPO MARIA DENARO[§], DANIELA DI SERAFINO[¶], AND SALVATORE FILIPPONE[‖]

Abstract. We discuss the design and development of a parallel code for Large Eddy Simulation (LES) by exploiting libraries for sparse matrix computations. Specifically, we show how a numerical procedure for the LES of turbulent channel flows, based on an approximate projection method, can be formulated in terms of linear algebra operators involving sparse matrices and vectors, and can be implemented using general-purpose linear algebra libraries as building blocks. This approach allows to pursue goals such as modularity, flexibility, accuracy and robustness, as well as easy and fast exploitation of parallelism, with a relatively low coding effort. The parallel LES code developed in this work, named SParC-LES (Sparse Parallel Computations for LES), is based on two parallel libraries: PSBLAS, providing basic sparse matrix operators and Krylov solvers, and MLD2P4, providing a suite of algebraic multilevel Schwarz preconditioners. The results obtained by using SParC-LES for the simulation of an incompressible and homothermal flow in a plane channel at $Re_\tau = 590$ show the effectiveness of this approach.

Key words. Large eddy simulation, turbulent channel flows, sparse matrix computations, parallel software libraries.

AMS subject classifications. 76F65, 65F50, 65Y15, 65Y05.

1. Introduction.

The study of the motion of incompressible flows in wall-bounded domains finds its application in many scientific fields for which a decoupling from the acoustic field is suitable, such as several geophysical flows, micro-device flows, nano-fluidic flows and so on. For almost all of such flows, turbulence appears as the normal regime of the motion. Turbulent flows are multi-scale flows where main phenomena such as vortex stretching and pairing, energy production, cascade and dissipation take place at different length scales, therefore an adequate description of them imposes a resolution of all fluid spatial and temporal scales. However, the Direct Numerical Simulation (DNS), which solves the Navier-Stokes (N-S) equations without any arbitrary assumption, has (for homogeneous isotropic turbulence) a computational cost depending on the cube of the Reynolds number, thus its application results unfeasible for increasing Reynolds numbers in complex geometries [26, 28] .

A more feasible approach for the accurate numerical solution of turbulent flows is Large Eddy Simulation (LES), born more than 50 years ago in the field of geophysics. The basic idea of LES consists in computing directly the dynamics of the large, energetic flow scales, responsible for the inviscid energy transfer, while modelling the dynamics of the small scales where the energy dissipation takes place. This

[†]Institute for Combustion Research (IRC), National Research Council of Italy (CNR), piazzale V. Tecchio 80, I-80125 Napoli, Italy (andrea.aprovitola@irc.cnr.it).

[‡]Institute for High-Performance Computing and Networking (ICAR), National Research Council of Italy (CNR), via P. Castellino 111, I-80131 Napoli, Italy (pasqua.dambra@cnr.it).

[§]Department of Aerospace and Mechanical Engineering, Second University of Naples, via Roma 29, I-81031 Aversa, Italy, and ICAR-CNR, via P. Castellino 111, I-80131 Naples, Italy (filippomaria.denaro@unina2.it).

[¶]Department of Mathematics, Second University of Naples, viale A. Lincoln 5, I-81100 Caserta, Italy, and ICAR-CNR, via P. Castellino 111, I-80131 Naples, Italy (daniela.diserafino@unina2.it).

[‖]Department of Mechanical Engineering, University of Rome "Tor Vergata", viale del Policlinico 1, I-00133, Roma, Italy (salvatore.filippone@uniroma2.it).

scale separation is obtained by applying a filtering operator to the N-S equations, thereby decomposing the nonlinear term in a resolved tensor and in an unresolved one (see, e.g., [31]). Currently, LES is the state of the art in the numerical simulation of turbulence for small/medium-scale problems where it is required to achieve detailed information from the flow dynamics. Although the computational costs are reduced with respect to the DNS approach, LES remains a computationally expensive technique and its application to realistic flows is a usual context for high-performance computing.

Projection methods are among the methods of choice for the filtered N-S equations. Their application requires the solution of large and sparse linear systems, which, together with the flux computation, accounts for most of the execution time of the LES codes (this is generally true for CFD codes, as pointed out in [21]). Therefore, efficient solvers for sparse linear systems as well as efficient techniques for flux computation are key issues for an effective application of LES.

In this paper we show that a numerical procedure for the LES of incompressible flows, based on a projection method, can be formulated in terms of linear algebra operators involving sparse matrices and vectors. Starting from this formulation, we designed and developed a parallel code for the LES of incompressible and omothermal flows in a plane channel, relying upon open-source parallel libraries for sparse matrix computations. This code, named *SParC-LES (Sparse Parallel Computations for LES)*, is based on two parallel packages: PSBLAS (Parallel Sparse BLAS) [19], which implements basic sparse matrix and vector operators as well as sparse linear system solvers, and MLD2P4 (Multilevel Domain Decomposition Parallel Preconditioners Package based on PSBLAS) [9], which implements algebraic multilevel Schwarz preconditioners to be used in conjunction with the PSBLAS solvers. A main goal in designing SParC-LES was to obtain a modular and flexible code, where physical submodels or discretization schemes might be changed by only changing few modules in the code and without affecting the overall numerical procedure. Furthemore, by using general-purpose linear algebra libraries as building blocks, we wanted to make available to the code a suite of linear solvers and preconditioners, so that the most appropriate ones for the problem and the parallel machine under consideration could be chosen. A notable feature of SPARC-LES is the direct inheritance of the efficiency, portability and robustness of the packages PSBLAS/MLD2P4 on which it is based; leveraging the two base packages allows an easy and fast exploitation of parallelism, which is encapsulated in the computational routines and in the support routines for building and managing distributed data structures, available in these libraries. It is worth noting that this work also led to an analysis of the accuracy and reliability of the LES procedure implemented in SParC-LES, in a complete turbulent channel flow simulation. Actually, the whole procedure had been only tested, through a prototype implementation, on a flow with $Re_\tau = 180$ [1].

The paper is organized as follows. In section 2 we briefly present the approximate projection method which is at the basis of our LES approach. In section 3 we focus on the main computational kernels of this method, providing a description of them in terms of linear algebra operators, and in section 4 we show how SParC-LES has been built by translating this description into the application of suitable data structures and routines from PSBLAS and MLD2P4. In section 5 we discuss the results obtained with SParC-LES on a typical test case for wall-bounded flows. Specifically, we first analyze the results of a complete simulation, by comparing them with the data obtained with other well-known LES codes as well as with DNS data, and then we analyze the parallel performance of our code. Finally, we give some conclusions in section 6.

2. A deconvolution-based approximate projection method for the LES of turbulent channel flows. We are interested in the simulation of an incompressible and homothermal flow in a plane channel, with periodic boundary conditions assigned in the streamwise (x) and spanwise (z) directions, and no-slip boundary conditions on the walls. We consider the Finite Volume (FV)-based LES approach proposed in [1, 4], where a top-hat filter coupled with a differential deconvolution operator is applied to the N-S equations in non-dimensional weak conservation form, obtaining the following formulation of the continuity and momentum equations, respectively:

$$\int\limits_{\partial\Omega(\mathbf{x})} \widetilde{\mathbf{v}} \cdot \mathbf{n}\, dS = 0, \tag{2.1}$$

$$A_{\mathbf{x}}^{-1}\left(\frac{\partial \widetilde{\mathbf{v}}}{\partial t}\right) = \mathbf{f}_{conv} + \mathbf{f}_{diff} + \mathbf{f}_{press} + \mathbf{f}_{sgs}. \tag{2.2}$$

In the above equations $\Omega(\mathbf{x})$ is a finite volume contained into the region of the flow, $\mathbf{n}$ is the outward-oriented unit vector normal to $\partial\Omega(\mathbf{x})$, $A_{\mathbf{x}}$ is the differential deconvolution operator, $\widetilde{\mathbf{v}} = A_{\mathbf{x}}(\overline{\mathbf{v}})$ is the deconvolved velocity field, where

$$\overline{\mathbf{v}}(\mathbf{x}, t) = \frac{1}{|\Omega(\mathbf{x})|} \int\limits_{\Omega(\mathbf{x})} \mathbf{v}(\mathbf{x}', t)\; d\mathbf{x}'$$

is the top-hat filtered velocity field. Furthermore, $\mathbf{f}_{conv}$, $\mathbf{f}_{diff}$ and $\mathbf{f}_{press}$ are the resolved convective, diffusive and pressure fluxes, respectively, and $\mathbf{f}_{sgs}$ represents the unresolved terms:

$$\begin{aligned}
\mathbf{f}_{conv} &= -\frac{1}{|\Omega(\mathbf{x})|} \int\limits_{\partial\Omega(\mathbf{x})} \widetilde{\mathbf{v}}\widetilde{\mathbf{v}} \cdot \mathbf{n}\, dS\ , \\
\mathbf{f}_{diff} &= \frac{2}{Re\ |\Omega(\mathbf{x})|} \int\limits_{\partial\Omega(\mathbf{x})} (\nabla^s \widetilde{\mathbf{v}}) \cdot \mathbf{n}\, dS, \\
\mathbf{f}_{press} &= -\frac{1}{|\Omega(\mathbf{x})|} \int\limits_{\partial\Omega(\mathbf{x})} p\mathbf{n}\, dS\ , \\
\mathbf{f}_{sgs} &= \frac{1}{|\Omega(\mathbf{x})|} \int\limits_{\partial\Omega(\mathbf{x})} \left[\frac{2}{Re}\left(\nabla^s \mathbf{v} - \nabla^s \widetilde{\mathbf{v}}\right) + \left(\widetilde{\mathbf{v}}\widetilde{\mathbf{v}} - \mathbf{v}\mathbf{v}\right)\right] \cdot \mathbf{n}\, dS,
\end{aligned}$$

where ∇^s is the zero-trace symmetric part of the gradient operator, Re the Reynolds number, and p the pressure term including the constant density. In our approach, an implicit SubGrid-Scale (SGS) modeling is used, i.e., $\mathbf{f}_{sgs} = 0$, thus the unresolved subgrid-scale terms do not appear explicitly in the equations. This choice is motivated by two main reasons: the use of the deconvolution operator for recovering the frequency content of the velocity field near the grid cutoff wavenumber, which, as shown in [31], is equivalent to the adoption of an explicit scale-similar SGS model on the filtered equation governing the top-hat velocity; and the adoption of an upwind discretization of the resolved fluxes, which has a local truncation error mimicking the diffusive behaviour of an eddy-viscosity SGS model (see, e.g., [4, 22]).

For the numerical solution of equations (2.1)-(2.2), we consider a time-splitting technique, based on the Approximate Projection Method (APM) described in [5]. APM allows to decouple the continuity and momentum equations by computing the unknown velocity field $\tilde{\mathbf{v}}$, at each time step, through the Helmholtz-Hodge decomposition:

$$\widetilde{\mathbf{v}}^{n+1} = \mathbf{v}^* - \Delta t \nabla \phi, \quad n = 1, 2, \dots \tag{2.3}$$

where $\mathbf{v}^*$ is an intermediate non-solenoidal velocity field, Δt is the time-step size, and ϕ is a suitable scalar field.

The intermediate velocity $\mathbf{v}^*$ is obtained by solving the deconvolved momentum equation (2.2), where the pressure term is neglected, with suitable Dirichlet boundary conditions at the walls [12]. The time integration of this equation is performed by applying the classical second order Adams-Bashforth/Crank-Nicolson (AB/CN) semi-implicit scheme. Specifically, the explicit AB method is used for the convective and diffusive terms in the x and z directions, while the implicit CN method is applied to the diffusion terms along the y direction; the latter choice ensures a wider stability range near the solid walls, where the grid used for the space discretization is finer, as explained below. Therefore, $\mathbf{v}^*$ is obtained by solving the following equation:

$$\begin{gathered}\left(A_{\mathbf{x}}^{-1} - \frac{\Delta t}{2Re} D_2\right)\mathbf{v}^* = \left(A_{\mathbf{x}}^{-1} + \frac{\Delta t}{2Re} D_2\right)\widetilde{\mathbf{v}}^n + \\ \frac{\Delta t}{2}\left(3\left(\frac{1}{Re}(D_1 + D_3)\widetilde{\mathbf{v}}^n + \mathbf{f}_{conv}^n\right) - \left(\frac{1}{Re}(D_1 + D_3)\widetilde{\mathbf{v}}^{n-1} + \mathbf{f}_{conv}^{n-1}\right)\right),\end{gathered} \tag{2.4}$$

with suitable Dirichlet boundary conditions in the y direction. The operators D_1, D_2 and D_3 are the components of

$$D(\,) = \frac{1}{|\Omega(\mathbf{x})|} \int\limits_{\partial\Omega(\mathbf{x})} \nabla^s(\,) \cdot \mathbf{n}\, dS$$

along the coordinate directions, and $\mathbf{f}_{conv}^n$ and $\mathbf{f}_{conv}^{n-1}$ are the convective fluxes at the time steps n and $n-1$, respectively.

The correction term in (2.3), needed to have a divergence-free velocity field $\widetilde{\mathbf{v}}$, is obtained by computing ϕ as the solution of the following Poisson-type elliptic equation:

$$(D_1 + D_2 + D_3)\phi \;=\; \frac{1}{\Delta t\, |\Omega\,(\mathbf{x})|} \int\limits_{\partial\Omega(\mathbf{x})} \mathbf{v}^* \cdot \mathbf{n}\, dS\ . \tag{2.5}$$

Non-homogeneous Neumann boundary conditions are prescribed in the wall-normal direction, such that the compatibility condition is fulfilled [5, 11]. In this way, the equation has a solution that is unique up to an additive constant. The above equation is known as pressure equation since $\nabla\phi$ is an $O(\Delta t)$ approximation of the pressure gradient. The time cycle of the APM procedure applied to the LES model of the turbulent channel flow is sketched in Figure 2.1.

The spatial domain is discretized by using a structured Cartesian grid, with uniform grid spacings in the streamwise and spanwise directions, where the flow is assumed to be homogeneous, and non-uniform grid spacing with refinement near the

```
! Nsteps = total number of time steps
for $n = 1, Nsteps$ do
    compute convective and diffusive fluxes and deconvolved velocity to build rhs of (2.4)
    compute $\mathbf{v}^*$ by solving deconvolved momentum equation (2.4)
    compute $\phi$ by solving pressure equation (2.5)
    update $\widetilde{\mathbf{v}}^{n+1}$ as in (2.3)
endfor
```

Fig. 2.1. *APM procedure.*

walls in the y direction, to adequately describe the boundary layer. A finite volume method, with the velocity components co-located at the centers of control volumes, is applied to equations (2.4)-(2.5). Such a choice is motivated by the simplicity in the implementation of the multidimensional upwind method used for the discretization of convective fluxes (see section 3.1). However, as a consequence of the grid co-location, the continuity equation in APM cannot be driven to the machine zero, but it vanishes according to the magnitude of the local truncation error associated with the time and space discretization [16]. Thus, the continuity error must be small enough to ensure that the kinetic energy does not increase in time [6].

The convective fluxes are discretized by a third-order multidimensional upwind scheme, while the diffusive ones by a classical second-order central scheme; fourth-order formulas are used for the discretization of the spatial derivatives involved in the inverse differential deconvolution in (2.4) (more details are given in the next section). The above space discretization leads to four sparse linear systems at each step of the simulation procedure. Their dimensions depend on the number of grid cells and hence increase with the Reynolds number, because of resolution needs. The solution of these systems, as well as the computation of the discrete convective and diffusive fluxes used to build their right-hand sides, are core tasks in the whole simulation procedure, therefore we focus on them in the next section.

3. Computational kernels in the APM procedure.

As in many CFD codes, the most computationally expensive tasks at each step of the APM procedure are the following:

- computation of the convective and diffusive fluxes and deconvolution of the velocity field,
- solution of the deconvolved momentum equation,
- solution of the pressure equation.

They account for almost all the computing time (details are given in section 5.2), and hence the effectiveness and the efficiency of the whole simulation is strongly dependent on their implementation. To clarify the rationale for our implementation choices we provide a description of these tasks in terms of linear algebra operators. Henceforth we assume that N_x, N_y and N_z are the numbers of grid nodes in the x, y and z directions, respectively; taking into account the boundary conditions, the total number of grid nodes where the velocity components and ϕ have to be actually computed is $N = N_x(N_y - 1)N_z$.

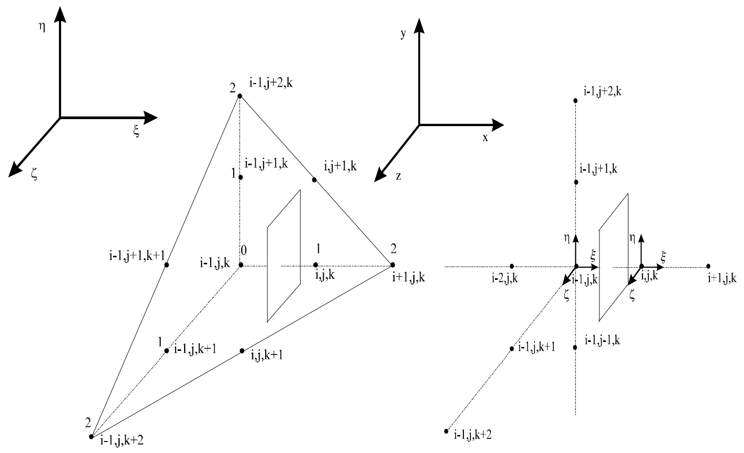

FIG. 3.1. *Multidimensional upwind criterion applied to the flux section corresponding to the west face of the (i, j, k)th control volume: Lagrangian simplex in case of negative velocity components (left) and local coordinate system (ξ, η, ζ), centered in i if $\tilde{v}_r$ is positive, or in $i-1$ if $\tilde{v}_r$ is negative (right).*

3.1. Computation of the fluxes and the deconvolved velocity. As mentioned in section 2, the convective fluxes at the time steps n and $n-1$, which are in the right-hand side of equation (2.4), are discretized by using the third-order upwind scheme proposed in [1]. This scheme applies a multidimensional upwind criterion, based on the interpolation of the velocity components over a Lagrangian simplex. We note that for strong multidimensional unsteady flows, such as those computed by LES, using a fully three-dimensional polynomial appears more suitable than using a factorized interpolation along the three coordinate directions. Furthermore, the interpolation over a Lagrangian simplex ensures the uniqueness of the polynomial over each of its faces, while such property is not guaranteed by classical factorized Lagrangian interpolations.

The filtered velocity component $\widetilde{v}_r$ along the rth coordinate axis ($r = 1, 2, 3$), at a point of the flow domain, is approximated through a quadratic polynomial interpolation over 10 grid points suitably chosen on a local three-dimensional Lagrangian simplex, which is built taking into account the velocity direction (see Figure 3.1, left). This approximation can be expressed as

$$\widetilde{v}_r(\xi, \eta, \zeta) \cong (\mathbf{p}(\xi, \eta, \zeta))^T \mathbf{c}^{\tilde{v}_r} \tag{3.1}$$

where the triple (ξ, η, ζ) represents the point in a local coordinate system depending on the velocity direction (see Figure 3.1, right), and

$$\mathbf{p}(\xi, \eta, \zeta) = \left[1,\ \xi,\ \eta,\ \zeta,\ \xi\eta,\ \xi\zeta,\ \eta\zeta,\ \xi^2,\ \eta^2,\ \zeta^2\right]^T, \quad \mathbf{c}^{\tilde{v}_r} = \mathbf{A}^{-1}\,\mathbf{q}^{\tilde{v}_r};$$

here $\mathbf{q}^{\tilde{v}_r} \in \mathbb{R}^{10}$ is the vector containing the values of $\widetilde{v}_r$ in the interpolation nodes and $\mathbf{A} \in \mathbb{R}^{10\times 10}$ is a nonsingular matrix depending on these nodes.

The convective flux across the surface of a control volume is obtained by adding the convective fluxes across the faces of the volume. Let us consider, for example, the control volume identified by its center (x_i, y_j, z_k). By using (3.1), the rth component of the flux across the "west" face $x \equiv x_i^- = x_i - \Delta x/2$, where Δx is the FV spacing in the x direction, is approximated as

$$\frac{1}{|\Omega_{i,j,k}|} \int_{z_k^-}^{z_k^+} dz \int_{y_j^-}^{y_j^+} (\tilde{v}_1 \tilde{v}_r)|_{x_i^-} \, dy \cong \frac{1}{\Delta x \Delta y_i \Delta z} \left(\mathbf{q}_W^{\tilde{v}_1}\right)^T \mathbf{M}_W \, \mathbf{q}_W^{\tilde{v}_i}, \tag{3.2}$$

where $y_i^\pm = y_i \pm \Delta y_i/2$, $z_i^\pm = z_i \pm \Delta z/2$, Δy_i and Δz are the FV spacings in the y and z directions, respectively, W refers to the west face, and

$$\mathbf{M}_W = \int_{z_k^-}^{z_k^+} dz \int_{y_j^-}^{y_j^+} \mathbf{m}_W \, \mathbf{m}_W^T \, dy, \quad \mathbf{m}_W^T = \mathbf{p}_W^T \, \mathbf{A}_W^{-1}.$$

The same technique can be applied to the remaining faces of the control volume, obtaining the following approximation of the rth component of the convective flux:

$$\begin{aligned} f_{conv}^r \cong \frac{1}{\Delta x \Delta y_i \Delta z} \Big[&\left(\mathbf{q}_E^{\tilde{v}_1}\right)^T \mathbf{M}_E \, \mathbf{q}_E^{\tilde{v}_r} - \left(\mathbf{q}_W^{\tilde{v}_1}\right)^T \mathbf{M}_W \, \mathbf{q}_W^{\tilde{v}_r} \\ &+ \left(\mathbf{q}_N^{\tilde{v}_2}\right)^T \mathbf{M}_N \, \mathbf{q}_N^{\tilde{v}_r} - \left(\mathbf{q}_S^{\tilde{v}_2}\right)^T \mathbf{M}_S \, \mathbf{q}_S^{\tilde{v}_r} \\ &+ \left(\mathbf{q}_U^{\tilde{v}_3}\right)^T \mathbf{M}_U \, \mathbf{q}_U^{\tilde{v}_r} + \left(\mathbf{q}_L^{\tilde{v}_3}\right)^T \mathbf{M}_L \, \mathbf{q}_L^{\tilde{v}_r} \Big], \end{aligned} \tag{3.3}$$

where W, E, N, S, U and L identify the six faces of the control volume. We note that only the matrices $\mathbf{M}_W$, $\mathbf{M}_S$, $\mathbf{M}_L$ are actually needed, since the FV scheme is flux-conservative and hence $(\mathbf{M}_E)_{i,j,k} = (\mathbf{M}_W)_{i+1,j,k}$, $(\mathbf{M}_N)_{i,j,k} = (\mathbf{M}_S)_{i,j+1,k}$, and $(\mathbf{M}_U)_{i,jk,} = (\mathbf{M}_L)_{i,j,k+1}$. Furthermore, these matrices are independent of the time step.

The contribution of the operator D_r to the component of the diffusive flux along the rth coordinate direction is discretized by using a classical second-order centered finite-volume scheme [1, 5], leading to three-point stencils in each coordinate direction. Since the operators D_r appear in the left- and right-hand side of the deconvolved momentum equation (2.4) and in the left-hand side of the pressure equation (2.5), it is convenient, for reuse, to build the matrices $\mathbf{D}_r \in \mathbb{R}^{N \times N}$, $r = 1, 2, 3$, that represent the discrete operators on the overall computational domain. By assuming a natural ordering of the grid cells, i.e., first along x, then along y, and finally along z, these matrices can be written as

$$\mathbf{D}_1 = \mathbf{I}_{N_z} \otimes \mathbf{I}_{N_y-1} \otimes \mathbf{D}_x, \quad \mathbf{D}_2 = \mathbf{I}_{N_z} \otimes \mathbf{D}_y \otimes \mathbf{I}_{N_x}, \quad \mathbf{D}_3 = \mathbf{D}_z \otimes \mathbf{I}_{N_y-1} \otimes \mathbf{I}_{N_x},$$

where $\mathbf{D}_x \in \mathbb{R}^{N_x \times N_x}$, $\mathbf{D}_y \in \mathbb{R}^{(N_y-1)\times(N_y-1)}$, $\mathbf{D}_z \in \mathbb{R}^{N_z \times N_z}$ represent the discrete counterparts of D_1, D_2, D_3 along the x, y, z directions, $\mathbf{I}_{N_x}$, $\mathbf{I}_{N_y-1}$, $\mathbf{I}_{N_z}$ are the identity matrices of dimensions N_x, $N_y - 1$, N_z, and $\otimes$ is the Kronecker product. It is immediate to see that the matrices $\mathbf{D}_1$ and $\mathbf{D}_3$ are symmetric, while $\mathbf{D}_2$ is nonsymmetric, but has a symmetric sparsity pattern; furthermore, each matrix has only three nonzero diagonals, i.e., the main diagonal plus two diagonals whose distance from the main one depends on the coordinate axis associated with the matrix itself.

The computation of the right-hand side of equation (2.4) requires also the application of the inverse deconvolution operator $A_{\mathbf{x}}^{-1}$ to the velocity vector $\widetilde{\mathbf{v}}^n$. To get an accurate representation of the spectral content of the numerical solution, $A_{\mathbf{x}}^{-1}$ is discretized by using a fourth-order centered scheme [23]:

$$\begin{aligned}A_{\mathbf{x}}^{-1}\widetilde{\mathbf{v}}^n \cong \; & \alpha_j \widetilde{\mathbf{v}}^n_{i,j,k} + \beta\left(\widetilde{\mathbf{v}}^n_{i-1,j,k} + \widetilde{\mathbf{v}}^n_{i+1,j,k} + \widetilde{\mathbf{v}}^n_{i,j,k-1} + \widetilde{\mathbf{v}}^n_{i,j,k+1}\right) \\ & - \gamma\left(\widetilde{\mathbf{v}}^n_{i-2,j,k} + \widetilde{\mathbf{v}}^n_{i+2,j,k} + \widetilde{\mathbf{v}}^n_{i,j,k-2} + \widetilde{\mathbf{v}}^n_{i,j,k+2}\right) + \sum_{\substack{s=-2 \\ s\neq 0}}^{2} \alpha_{j+s}\widetilde{\mathbf{v}}^n_{i,j+s,k},\end{aligned}$$

where the coefficients α_l are all dependent on Δy_j. Therefore, the inverse of the discrete deconvolution operator can be represented as a matrix $\bar{\mathbf{A}} \in \mathbb{R}^{N\times N}$ that has only 13 nonzero non-adjacent diagonals and a symmetric sparsity pattern, but is unsymmetric in the values, due to the non-uniform grid spacing in the y direction. It can be verified that the matrix $\bar{\mathbf{A}}$ is diagonally dominant and its sparsity pattern contains the sparsity patterns of the diffusion matrices $\mathbf{D}_r$.

Finally, we note that the matrices $\mathbf{D}_r$ and $\bar{\mathbf{A}}$ do not depend on the time step and can be computed just once, before the APM procedure starts.

3.2. Setup and solution of the momentum and pressure equations. The discretization of the left-hand side of equation (2.4) is obtained as a byproduct of the discretization of the diffusive flux and deconvolved momentum operators. The discrete deconvolved momentum equations consists of three linear systems, henceforth referred to as velocity systems:

$$\left(\bar{\mathbf{A}} - \frac{\Delta t}{2Re}\mathbf{D}_2\right)(\mathbf{v}^*)_r = (\mathbf{w})_r, \tag{3.4}$$

where $(\mathbf{v}^*)_r$, $r = 1, 2, 3$, is the component along the rth coordinate axis of the discrete intermediate velocity $\mathbf{v}^*$ and $(\mathbf{w})_r$ is the discretization of the right-hand side of the corresponding component of (2.4), obtained as explained in the previous section. Since the sparsity pattern of $\bar{\mathbf{A}}$ includes the sparsity pattern of $\mathbf{D}_2$, the matrix $\bar{\mathbf{A}} - \frac{\Delta t}{2Re}\mathbf{D}_2$ has at most 13 nonzero entries per row, distributed over as many non-adjacent diagonals (see Figure 3.2, left). Furthermore, it is diagonally dominant and well conditioned. Therefore, a natural choice for the solution of (3.4) is a nonsymmetric Krylov method such as GMRES [30], possibly coupled with a simple preconditioner such as Jacobi or block-Jacobi. As for the matrices involved in the flux computation, the matrix in 3.4 can be built before the beginning of the APM procedure.

The discretization of the left-hand side of the pressure equation results from the discretization of the three diffusion operators $\mathbf{D}_r$, outlined in section 3.1. A small modification must be applied to the discretization of $\mathbf{D}_2$, to take into account the Neumann boundary conditions in the wall-normal direction; the corresponding matrix is denoted by $\bar{\mathbf{D}}_2$. The right-hand side is discretized by approximating the integral through a centered scheme, involving the velocity components on the faces of the finite volumes, which are obtained through linear interpolation over the volume centers [1]. At the walls, the exact normal component of the velocity is used, thus the boundary value problem is equivalent to a problem with homogeneous Neumann boundary conditions and a modified source term. In this way the compatibility condition is ensured and the spatial accuracy of the discretized pressure equation is of second order. The resulting linear system has the form

$$\left(\mathbf{D}_1 + \bar{\mathbf{D}}_2 + \mathbf{D}_3\right)\boldsymbol{\varphi} = \mathbf{g}, \tag{3.5}$$

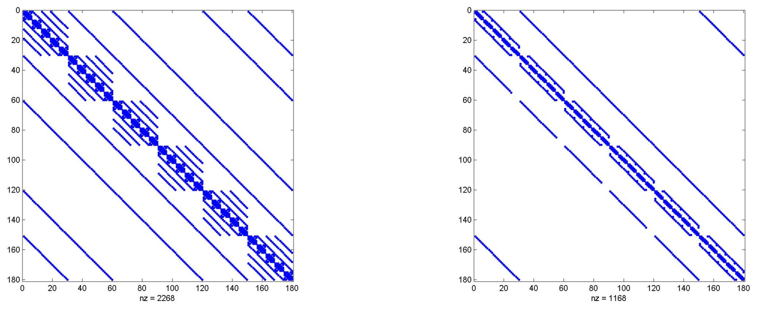

FIG. 3.2. *Sparsity patterns of the matrices of the velocity (left) and pressure (right) systems.*

where $\boldsymbol{\varphi}$ denotes the discrete approximation of ϕ and the right-hand side $\mathbf{g}$ also includes the boundary conditions. In the following, this system is referred to as pressure system. The matrix $\mathbf{D} = \mathbf{D}_1 + \bar{\mathbf{D}}_2 + \mathbf{D}_3$ has at most seven entries per row, distributed over seven non-adjacent diagonals plus four diagonals arising from the periodicity in the x and z directions; it is nonsymmetric, because of the non-uniform spacing along the y direction, but has a symmetric sparsity pattern (see Figure 3.2, right). Again, it can be computed before the beginning of the APM procedure.

Note that $\mathbf{D}$ is singular; furthermore, it can be verified that $\mathbf{D}$ has the following property:

$$\mathcal{R}(\mathbf{D}) \cap \mathcal{N}(\mathbf{D}) = \{0\}, \tag{3.6}$$

where $\mathcal{R}(\mathbf{D})$ and $\mathcal{N}(\mathbf{D})$ are the range space and the null space of $\mathbf{D}$. Thus, the GMRES method is able to compute a solution of the pressure system before a breakdown occurs [36]. This allows to use a standard GMRES implementation. The application of an effective preconditioner is required to achieve a sufficiently accurate solution with a small number of iterations. Multilevel Schwarz preconditioners are known to be optimal in the solution of linear systems arising from the discretization of elliptic partial differential equations, in the sense that the preconditioned solvers can achieve convergence with a number of iterations independent of the problem size [32, 34]. These preconditioners may result very efficient in a parallel computing setting, provided that a suitable balance between scalability of implementation and optimality is achieved [2].

4. SParC-LES: a parallel code for LES. SParC-LES is a novel parallel code which implements the LES approach described in the previous sections, by exploiting the formulation of the numerical methods in terms of computational kernels. A driving principle in designing the code has been the use of portable, robust and efficient numerical building blocks, whenever possible. Since the main kernels of the APM procedure involve sparse matrix computations and the solution of sparse linear systems, the code development has been based on the PSBLAS library and on the relevant parallel preconditioners package MLD2P4. PSBLAS and MLD2P4 are written in Fortran 95, using an object-based approach and a modular design that favour flexibility and extensibility, implemented through the language features for data abstraction and functional overloading. Special attention is devoted to memory management and performance issues, to obtain runtime efficiency; furthermore, portability is ensured through the use of MPI [33] as message-passing environment. A very short description of the PSBLAS and MLD2P4 functionalities is given next. For details the reader is referred to [8, 9, 10, 17, 18, 19].

PSBLAS provides the basic operators needed to implement iterative solvers for the solution of sparse linear systems on distributed-memory parallel computers. It includes parallel versions of most of the Sparse BLAS computational kernels proposed in [15], as well as implementations of many popular Krylov subspace solvers for systems of linear equations. It also provides functionalities for sparse matrix management, e.g., for the setup and storage of distributed sparse matrices and for the implementation of data communication patterns typically involved in sparse matrix computations. An interesting feature of PSBLAS is the decoupling of the internal storage format from the application structure: an application code based on PBLAS can be easily adapted to different machine architectures by plugging in the appropriate internal formats and computational kernels. Moreover, the storage formats can be changed at runtime to be adapted to the different needs of the various application phases. In this way we are moving towards a higher-level design of scientific application codes, as advocated by multiple authors (see, e.g., [7, 29]).

MLD2P4 implements a suite of parallel multilevel Schwarz preconditioners that can be used with the Krylov solvers available in PSBLAS. It is built on the top of PSBLAS, thus exploiting the above-mentioned PSBLAS functionalities. The MLD2P4 preconditioners work in an algebraic way, i.e., exploiting only information on the matrix and not on the geometry of the problem from which it originates; to build coarse-level corrections they use the smoothed aggregation technique [35]. MLD2P4 provides the Jacobi and block-Jacobi preconditioners, the basic additive Schwarz preconditioners, and multilevel preconditioners combining the previous one-level preconditioners with coarse-level corrections, in an additive or multiplicative framework. Note that by making available different solvers and preconditioners to an application code, we can easy experiment with different methods within the code, in order to select the most approprite ones. In SParC-LES we tested the GMRES method with several preconditioners for the solution of the velocity and pressure systems, to select the most efficient ones (see section 4.3).

Parallelism is introduced in SParC-LES by a domain decomposition approach at the discrete level, i.e, by partitioning the computational grid into subgrids and assigning a subgrid to each available processing unit. In order to obtain the best computation-to-communication ratio, we implement a 3D block decomposition of the computational grid that produces the well-known *surface-to-volume effect*, i.e., minimizes the surface area of each partition (data to be communicated) for a given volume (data to be locally computed) [20]. Suitable MPI functionalities are used to define a virtual 3D Cartesian topology of running processes that matches the decomposition of the computational grid. We implicitly assume that the number of processes is equal to the number of available processing units, although this is not required to run our code. The grid decomposition results into a a general row-block distribution of the matrices involved in the computation, managed through suitable PSBLAS functionalities, as explained in section 4.1.

Next, we provide some details on the use of PSBLAS and MLD2P4 in the development of SParC-LES.

4.1. Parallel deconvolution and diffusion operators. The first phase of the parallel computation is the definition of the basic discrete operators involved in the APM procedure, i.e., the inverse deconvolution matrix $\bar{\mathbf{A}}$ and the three diffusion matrices $\mathbf{D}_r$ described in Section 3.1. In SParC-LES each matrix is represented through a corresponding PSBLAS distributed data structure. Each data structure has a set of associated methods, and thus, using the computer science language, it is considered an "object". The necessary data structure is actually split into a *sparse matrix* data structure, containing the part of the operator assigned to a given process in the row-block distribution, and a *communication descriptor* data structure, containing the information needed for handling all necessary data exchanges.

As discussed in [9, 19], each application of a matrix operator, i.e., each computation of a matrix-vector product, requires a specific data exchange among all processes, according to a scheme that is implicitly defined by the sparsity pattern of the matrix itself and by the assignment of its rows to the available processes. Thus, the construction of the communication descriptor requires the knowledge of the sparsity pattern of the matrix, in turn determined by the grid and the stencil chosen to discretize the PDE under consideration, and of the partitioning of the computational domain among the various processes. Therefore, the communication descriptors of the discrete deconvolution operator $\bar{\mathbf{A}}$ and the diffusion operators $\mathbf{D}_r$ are determined by the Cartesian discretization grid, using a natural numbering of the grid cells, and by its 3D partitioning.

In Figure 4.1 we report a pseudo-code showing the main steps for building the sparse matrix data structures `D1`, `D2` and `D3` holding $\mathbf{D}_1$, $\mathbf{D}_2$ and $\mathbf{D}_3$, respectively, and the associated communication descriptor `desc_d`. Note that we define a single descriptor including the combined sparsity patterns of the three operators because they will be used via their sums/differences. A similar pseudo-code describes the construction of the sparse matrix `DEC` and the communication descriptor `desc_dec` for the operator $\bar{\mathbf{A}}$. Once the mapping between the grid decomposition and the processes has been performed and the list of indices assigned to each process, `vl`, has been identified, the communication descriptor and the sparse matrix data structures are allocated by using `psb_cdall` and `psb_spall`, respectively. Then, the entries of the diffusion operators are inserted into the sparse matrix structure row by row, through `psb_spins`. Finally, the descriptor and the sparse matrices are assembled through `psb_cdasb` and `psb_spasb`, to make them ready for use by the other PSBLAS and MLD2P4 routines.

Then, the basic discrete operators are used for computing the matrices of the velocity and pressure systems. The sparse matrix data structure corresponding to the coefficient matrix of system (3.4) can be obtained as the difference between `DEC` and `D2`, the latter suitably scaled. Analogously, the sparse matrix holding the coefficient matrix in (3.5) can be computed by adding up `D1`, `D2` and `D3`, after a suitable small change to `D2` to take into account the Neumann boundary conditions in the wall-normal direction. We also note that the computation of the right-hand side of (3.4) requires the sum of `DEC` and `D2`, with the same scaling as above, and the sum of `D1` and `D3` (see (2.4)). In all these cases we use the sparse matrix sum routine `psb_sp_add`, which computes

$$\mathbf{Z} = \alpha\mathbf{V} + \beta\mathbf{W},$$

where $\mathbf{Z}$, $\mathbf{V}$, and $\mathbf{W}$ are distributed sparse matrices and α and β are scalars. The corresponding pseudo-code is reported in Figure 4.2, where `D2BAR` holds the modified diffusion operator, `MATD` and `MATV` hold the coefficient matrices of the pressure and velocity systems, and `DD` and `AD2` the matrices in the right-hand sides of the discrete momentum equations.

```
< choose a map between (X,Y,Z) and (i=1,...,N) >
< choose a distribution of the grid to the processes >
< generate the list of indices, vl, for current process >
call psb_cdall(ictxt,desc_d,info,vl=vl(1:lenvl))
call psb_spall(D1,desc_d,info)
call psb_spall(D2,desc_d,info)
call psb_spall(D3,desc_d,info)
do i = 1, N
  if ( <index i belongs to me> ) then
    nz = < number of nonzeros in i-th row >
    irow(:)  = (/ < i repeated nz times >/)
    icol(:)  = (/ <list of nonzero column indices in the row >/)
    val1(:)  = (/ <coefficients of D1 in i-th row> /)
    val2(:)  = (/ <coefficients of D2 in i-th row> /)
    val3(:)  = (/ <coefficients of D3 in i-th row> /)
    call psb_spins(nz,irow,icol,val1,D1,desc_d,info)
    call psb_spins(nz,irow,icol,val2,D2,desc_d,info)
    call psb_spins(nz,irow,icol,val3,D3,desc_d,info)
  endif
enddo
call psb_cdasb(desc_d,info)
call psb_spasb(D1,desc_d,info)
call psb_spasb(D2,desc_d,info)
call psb_spasb(D3,desc_d,info)
```

FIG. 4.1. *Setup of diffusion operators.*

```
< modify D2 to get D2BAR >
alpha = 1.0
beta1 = 1.0
beta2 = deltat/(2.0*Re)
call psb_sp_add(alpha,D1,beta1,D3,DD,info)
call psb_sp_add(alpha,DD,beta1,D2BAR,MATD,info)
call psb_sp_add(alpha,DEC,-beta2,D2,MATV,info)
call psb_sp_add(alpha,DEC,beta2,D2,AD2,info)
```

FIG. 4.2. *Computation of the matrices of the velocity and pressure systems.*

4.2. Parallel computation of the convective and diffusive fluxes. At each time step of the APM procedure, the convective and diffusive fluxes and the deconvolved velocity field must be computed to build the right-hand sides of the velocity systems (3.4).

For each component of the intermediate velocity field, the computation of the corresponding discrete convective fluxes at two subsequent time steps is required (see (2.4)). In this case, the 10×10 matrices involved in the flux computations, as specified in (3.3), are not explicitely constructed because of their very small size, and the corresponding operations are directly implemented in Fortran 95. This phase requires that each process exchanges the velocity values in two layers of grid cells with the processes holding the adjacent subgrids; the data exchange is implemented through the basic send and receive PSBLAS routines, i.e., `psb_snd` and `psb_rcv`.

Once the convective fluxes have been computed, the right-hand sides of the three

```
do ia = < loop on all locally owned indices >
  < map index ia onto triple (i,j,k) >
  bval = < convective flux in (i,j,k) at time step n >
  call psb_geins(1,(/ia/),(/bval/),b,desc_dec,info)
end do
call psb_geasb(b,desc_dec,info)
alpha1 = 1.0
alpha2 = 1/Re
beta1 = deltat/2.0
beta2 = 3.0
beta3 = 1.0
call psb_spmm(alpha1,AD2,v,-beta1,b0,desc_dec,info)
call psb_geaxpby(-alpha1,v0,beta2,v,desc_dec,info)
call psb_spmm(alpha2,DD,v,beta3,b,desc_d,info)
call psb_geaxpby(alpha1,b0,beta1,b,desc_d,info)
```

FIG. 4.3. *Computation of the right-hand side of one of the three velocity systems.*

velocity systems, i.e., the discretizations of the components of the right-hand side of (2.4), are obtained by applying the PSBLAS computational kernel implementing the following sparse matrix by vector operation:

$$\mathbf{y} = \alpha \mathbf{V}\mathbf{x} + \beta \mathbf{y},$$

where $\mathbf{V}$ is a distributed sparse matrix, $\mathbf{x}$ and $\mathbf{y}$ are vectors distributed according to the matrix $\mathbf{V}$, and α and β are scalars. This operation is implemented in the `psb_spmm` routine, as a special case of the sparse matrix by dense matrix product. Specifically, in order to obtain the discretization of the first term in the right-hand side of (2.4), we compute the sparse matrix by vector products between the sparse matrix stored in `AD2`, mentioned at the end of the previous section, and each of the intermediate velocity vectors $(\mathbf{v}^*)_r$. The remaining terms in the right-hand sides of the velocity systems, stemming from the discretization of the x and z components of the diffusive fluxes, are obtained through the sparse matrix by vector product involving the matrix in `DD` and the vectors $3(\widetilde{\mathbf{v}}^n)_r - (\widetilde{\mathbf{v}}^{n-1})_r$. Then, the right-hand sides are obtained by adding up the results of the previous computations, by using the `psb_geaxpby` routine, which computes

$$\mathbf{Y} = \alpha \mathbf{X} + \beta \mathbf{Y},$$

where $\mathbf{X}$ and $\mathbf{Y}$ are distributed dense matrices (including vectors as a special case) and α and β are scalars.

The pseudo-code for the computation of the right-hand side of any of the three velocity systems is given in Figure 4.3 (some redundancy has been introduced in the pseudo-code to ease its readability). In this case, before `psb_spmm` and `psb_geaxpby` are called, `v` and `b` contain the velocity component along the selected coordinate axis and the corresponding flux at time step n, while `v0` and `b0` contain the same vectors at time step $n-1$. Note that only `b` is built, through `psb_geins` (insertion of entries) and `psb_geasb` (vector assembly), at the beginning of the current time step, since the other vectors result from the computations at the previous time step. Finally, after the calls to `psb_spmm` and `psb_geaxpby`, the right-hand side of the velocity system is in `b`.

```
call mld_precinit(PV,'JACOBI',info)
call mld_precbld(MATV,desc_dec,PV,info)
.....................................
call psb_krylov('RGMRES',MATV,PV,b,x,tol,desc_dec,info, &
    & itmax,iter,err,itrace,irst,istop)
```

FIG. 4.4. *Solution of a velocity system: setup of the Jacobi preconditioner and application of the preconditioned RGMRES.*

The right-hand side of the pressure system, arising from the discretization of the right-hand side of equation (2.5), is obtained by simple linear combinations of velocity components defined on classical seven-point stencils. As for the discrete convective fluxes, this requires the exchange of the velocity components in two layers of grid points among nearest-neighbour processes, that are managed through the `psb_snd` and `psb_rcv` data communication routines.

4.3. Parallel solution of the velocity and pressure systems. The velocity and pressure systems are solved using the GMRES implementation provided by PSBLAS; several preconditioners are available from MLD2P4 to accelerate the convergence.

As observed in section 3.2, the matrix of the velocity systems is diagonally dominant and well conditioned, thus no preconditioner or very simple preconditioners, such as the Jacobi and block-Jacobi ones, are expected to work well. Figure 4.4 shows the setup and the construction of the Jacobi preconditioner for the matrix under consideration, through the MLD2P4 routines `mld_precinit` and `mld_precbld`; no preconditioner or the block-Jacobi preconditioner can be selected by specifying `'NOPREC'` or `'BJAC'` instead of `'JACOBI'` in the call to `mld_precinit`. Note that, by default, the block-Jacobi preconditioner is applied by using the ILU(0) factorization of the blocks; different solvers can be chosen for the blocks, by specifying them through a suitable routine, named `mld_precset`. The preconditioned GMRES is applied via the routine `psb_krylov`, which provides an interface for all the Krylov methods implemented in PSBLAS (see [17] for details). Actually, the restarted GMRES (RGMRES) method is applied, with restart parameter `irst`; in our simulations we choose `irst`=30, which makes GMRES equal to RGMRES because of the small number of iterations performed (see section 5.2). Of course, since the matrix of the velocity systems does not change during the APM procedure, the preconditioner is setup only once, before the time cycle starts, and is reused at each system solution.

As explained in section 3.2, an appropriate choice for the solution of the pressure system is GMRES with a multilevel Schwarz preconditioner. MLD2P4 provides several multilevel preconditioners, obtained by choosing the multilevel framework (i.e., additive or multiplicative), the number of levels, the smoother at each level, the coarsest-level solver, etc., as explained in [10]. To provide an example, in Figure 4.5 we show the setup of a 4-level multiplicative (V-cycle) preconditioner, using 1 block-Jacobi sweep as pre- and post-smoother, and 4 block-Jacobi sweeps as coarsest-level (approximate) solver; the ILU(0) factorization is applied to the blocks within the block-Jacobi method. A threshold equal to 0.01 is set in the aggregation algorithm, to perform the matrix coarsening at each level. Actually, most of these choices are the default ones, but they are explicitly done via `psb_precset` for illustration purposes. As in the case of the velocity systems, the preconditioner for the pressure equation is only built once before the start of the APM procedure.

```
call mld_precinit(PD,'ML',info,nlev=4)
call mld_precset(PD,mld_ml_type_,'MULT',info)
call mld_precset(PD,mld_smoother_pos_,'TWOSIDE',info)
call mld_precset(PD,mld_smoother_type_,'BJAC',info)
call mld_precset(PD,mld_smoother_sweeps_,1,info)
call mld_precset(PD,mld_sub_solve_,'ILU',info)
call mld_precset(PD,mld_sub_fillin_,0,info)
call mld_precset(PD,mld_coarse_mat_,'DISTR',info)
call mld_precset(PD,mld_coarse_solve_,'BJAC',info)
call mld_precset(PD,mld_coarse_sweeps_,4,info)
call mld_precset(PD,mld_coarse_subsolve_,'ILU',info)
call mld_precset(PD,mld_coarse_fillin_,0,info)
call mld_precset(PD,mld_aggr_thresh_,0.01,info)
call mld_precbld(MATD,desc_d,PD,info)
```

FIG. 4.5. *Setup of a 4-level V-cycle preconditioner for the pressure system.*

We note that the choice of the parameters defining the multilevel preconditioner usually affects the performance of the preconditioned solver; therefore, we tested different combinations of parameters in order to identify the best one for the test problem considered in this paper on the available parallel machines. This issue was also discussed in [2, 3].

5. Numerical experiments. In this section we analyze the results obtained with SParC-LES in the simulation of a turbulent flow in a plane channel at $Re_\tau = 590$, for which a DNS database widely accepted in the literature is available [27]. The channel height is $H = 2\delta$, the streamwise length $L_x = 2\pi\delta$ and the spanwise length $L_z = \pi\delta$. The turbulent flow, assumed to be periodic in the x and z directions, is sustained in the streamwise direction by a driven force provided by a suitable constant pressure gradient. The non-dimensional lengths are defined by using the channel half-height δ, while the non-dimensional velocity is defined by means of the shear velocity $u_\tau = \sqrt{\frac{\Delta p_0 \delta}{\rho_0 L_x}}$, where Δp_0 is the forcing pressure gradient and ρ_0 is the homogeneous density. In this way, a forcing non-dimensional pressure gradient equal to 1 is obtained. An initial parabolic velocity profile with a superimposed Gaussian perturbation field is assigned along the streamwise direction. More details on this test case can be found, e.g., in [27, 31].

The numerical experiments were carried out with the double aim of assessing the numerical results obtained with SParC-LES in a complete LES simulation and of analysing the parallel performance of the code on modern architectures.

5.1. Analysis of a complete simulation. A complete LES simulation was performed by using a computational grid with $N_x = N_z = 48$ nodes uniformly distributed in the homogeneity plane (x, z); a trigonometric stretching law was applied in the wall-normal direction y, with $N_y = 100$ nodes, corresponding to a resolved boundary layer. The time step was set as $\Delta t = 10^{-5}$ and the run was executed until an energy equilibrium was achieved, after which 15 samples of the velocity field were dumped for computing time-averaged statistics. The complete simulation corresponds to 100 time units, for a total number of 10^7 time steps. The GMRES method and the preconditioners were set as in the experiments reported in section 5.2. The simulation was carried out on 64 processors of a HP XC 6000 Linux cluster operated by the

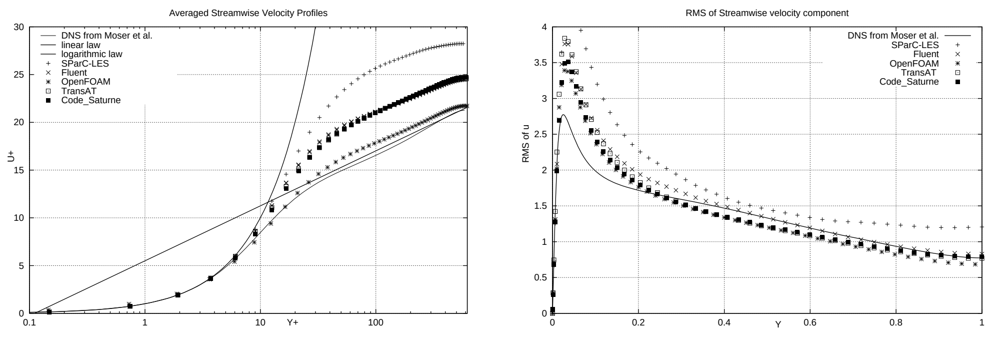

FIG. 5.1. *SParC-LES vs available LES data: averaged streamwise velocity profiles (left) and RMS of the fluctuations of the streamwise streamwise velocity components (right).*

Naples branch of ICAR-CNR, using PSBLAS 2.4.0, MLD2P4 1.2.1., HP MPI 2.01, and the GNU 4.6.1 Fortran compiler.

In Figure 5.1 (left) we compare the SParC-LES time-averaged streamwise velocity profile with the corresponding DNS results from [27] and with other LES velocity profiles provided by the LESinItaly Initiative [25]. All the quantities are in wall units, i.e., $y^+ = u_\tau y/\nu$ and $u^+ = u/u_\tau$, where ν is the kinematic viscosity, and, for the LES results, u denotes the averaged streamwise component of the velocity. As usual, we also plot the linear low and the logarithmic low, which are the reference models for u^+ in the viscous sublayer ($y^+ < 5$) and in the log-layer region ($y^+ > 30$) [31]. We note that, among the data available for our test case in the LESinItaly database, we selected only those obtained with widely used codes based on FV formulations (Fluent, OpenFOAM, TransAT, Code_Saturne), where second-order-accurate schemes both in space and time and an eddy-viscosity closure for the subgrid terms were used. These data come from simulations on a finer grid than the one used in our experiments, i.e., a Cartesian grid where $N_x = N_z = 64$ and $N_y = 100$, with a trigonometric stretching law along y; nevertheless, they are useful to assess the "quality" of the solution computed by SParC-LES.

We see that the SParC-LES solution exhibits a good agreement with both the DNS and LES results in the viscous sublayer, up to the beginning of the buffer layer, i.e., of the region $5 < y^+ < 30$; a departure from the DNS and other LES data is observed in the log-layer region. Several considerations can be made to explain this behaviour. First, the horizontal grid used for the SParC-LES simulation is coarser than for the other LES codes. This corresponds to different filtered width and filtered velocity. As noted in [24], in the LES literature there is evidence that the turbulent structures originating along the spanwise direction (streaks), and responsible of turbulent energy production, are particularly sensitive to the spanwise grid resolution. Therefore, the coarser grid used by SParC-LES drives toward a worse description of such flow structures. Owing also to the stretching law in the y direction, the grid appears quite coarsened near the channel centerline and thus the contribution of the implicit subgrid model becomes relevant. Such a contribution is due the implicit dissipative nature of the local truncation error associated with the polynomial upwind scheme, whose magnitude depends on the cell size. As a consequence, a smaller shear velocity u_τ is computed by SParC-LES, and the non-dimensional variable u^+ is larger, driving to a shift in the log-layer region. However, it is worth noting that a similar,

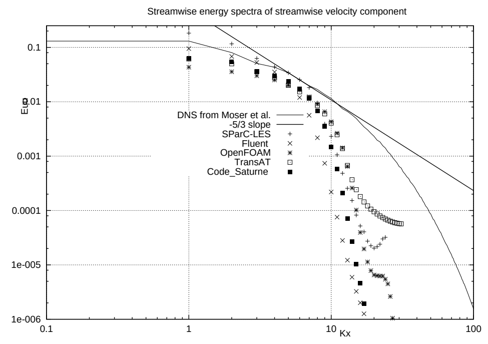

FIG. 5.2. *SParC-LES vs available LES and DNS data: streamwise energy spectra at $y+ = 590$.*

although less pronounced, behaviour is observed in the other LES solutions.

The previous conclusions are confirmed by the root mean square (RMS) of the fluctuations of the streamwise component of the velocity along the channel height (see Figure 5.1, right). For SParC-LES the peak of turbulence intensity of the solution is displaced farther from the wall than for the other LES codes, thus indicating that the stronger turbulent fluctuations obtained with our simulation result in a thickened boundary layer.

We also compare SParC-LES with the other codes in terms of the energy spectrum of the averaged streamwise velocity component at $y^+ = 590$ (see Figure 5.2). The energy content of the streamwise velocity shows an interesting feature. At low wavenumbers, which correspond to the largest energy containing turbulent structures, the spectral components obtained with SParC-LES fit the DNS ones better than the spectral components from the other LES codes. This good recovering of the energy content can be ascribed to the deconvolution procedure implemented in SParC-LES. As pointed out in [14], the deconvolution operator has the effect of making the filtered velocity field closer to a spectral-like resolution. This good behaviour is relevant when thinking about practical applications in the simulation of turbulence where large structures are responsible of important effects and must accurately resolved. For instance, in non-premixed combustion computations, the flame behaviour is governed by the large-scale turbulent motions, which set the rate at which fuel and oxidant mix. Hence, an accurate description of such flow scale is fundamental to describe the flame stabilization mechanism.

On the other hand, the results of SParC-LES, as well as of the other codes, show a strong decay of the high-wavenumber energy content, typical of the use of filters that are smooth in the wavenumber space. For SParC-LES this is due to the artificial dissipation implicitly introduced by the upwind scheme, which can be considered comparable with a central discretization scheme plus an explicit eddy-viscosity contribution. It is useful to remark that several CFD codes can employ an artificial dissipation to stabilize the numerical computation of the convective fluxes. In our comparison, FLUENT uses a second-order central bounded scheme for the convective fluxes, while OpenFOAM uses a second-order unbounded scheme. Without bounding, FLUENT does not achieve convergence, whereas the SParC-LES computation is stable, although exhibiting a sort of oscillation on the high-frequency components, similarly to the OpenFOAM solution.

Finally, we note that the approach used to develop SParC-LES allows to easily change the discretization scheme used for the convective fluxes. For instance, high-order central methods coupled with an explicit SGS model, such as a dynamic scale-similar model, could be implemented in the future, with the aim of better correlating the energy content in the inertial range and reducing the dissipation.

5.2. Analysis of parallel performance. A performance analysis of SParC-LES on the selected test problem was carried out by running the code on the Hopper supercomputer, a Cray XE6 operated by the National Energy Research Scientific Computing Center in Berkeley, CA. Hopper has 6384 compute nodes made up of 2 twelve-core AMD processors for a total of 153216 cores, and uses the Cray "Gemini" interconnect for inter-node communication; it has a peak performance of 1.28 Petaflops/sec. and is ranked the eighth in the November 2011 TOP 500 list (see `http://www.top500.org/`). In our experiments we used PSBLAS 2.4.0 and MLD2P4 1.2.1, installed on the top of the xt-mpich2/xt-shmem 5.4.0 MPI implementations. The software was compiled with the GNU 4.6.1 Fortran compiler.

We considered a Cartesian grid with $N_x = N_z = 64$ and $N_y = 100$; the resulting dimension of the velocity and pressure matrices is $N = 405504$, with 5263360 nonzero entries for the velocity matrix and 2826430 for the pressure one. Taking into account the grid size, we used from 1 to 64 cores to analyze the strong scalability of our code. We run the code for 1000 time steps only, because we had previously observed that the behaviour of the linear solvers in such a number of steps is representative of the behaviour in a complete simulation. On the velocity systems we tested GMRES with no preconditioner and with the Jacobi and block-Jacobi preconditioners, while on the pressure systems we tested several multilevel Schwarz preconditioners. The iterations were stopped when the 2-norm of the relative residual was lower than 10^{-7}. For each velocity or pressure system, the solution obtained at the previous time step was used as starting guess, except at the first time step where the zero vector was considered. On the velocity systems the smallest execution times were obtained by using GMRES with no preconditioner. On the pressure systems the most efficient preconditioners generally were the 2-level and 3-level V-cycle preconditioners, with 1 block-Jacobi sweep as pre- and post-smoother, 4 block-Jacobi sweeps as coarsest level approximate solver, and ILU(0) to factorize the blocks. Specifically, the 3-level preconditioner led to the smallest times when using 1 to 16 cores, while the 2-level one was the best on 32 and 64 cores. With both preconditioners, a threshold equal to 10^{-2} was chosen by numerical experiments in the aggregation algorithm. For the sake of space, here we show only the results concerning the 2-level preconditioner. In this case, the size of the coarse matrix ranges from 48108, on a single core, to 52752, on 64 cores; this variability is due to the uncoupled aggregation implemented in MLD2P4 [9].

To better understand the overall parallel performance of SParC-LES, resulting from the performances of its different tasks, we first analyze the execution profile of the code on 1 core (Figure 5.3). The computation of the convective fluxes and the computation of the deconvolved velocity and diffusive fluxes account for about 52% of the total execution time; they have almost the same weight in the overall run, i.e., 26.5% and 25.4%, respectively. The solution of the velocity and pressure systems requires about 45% of the total time, with the pressure systems having a weight of 17.4% and the velocity ones of 28%. The remaining part of the code, which also includes the velocity updates (2.3) and the computation of the right-hand sides of the pressure systems, accounts for less than 3% of the execution.

Now we analyze the parallel performance of SParC-LES in the solution of the

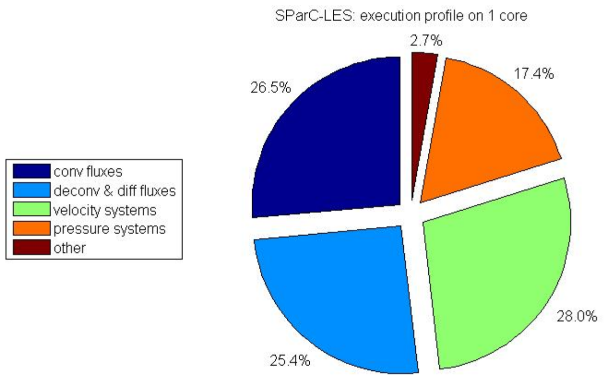

FIG. 5.3. *Profile of a sequential SParC-LES run.*

velocity and pressure systems; the corresponding execution times and speedups are shown in Figure 5.4. For both types of systems we see a similar time decrease, which leads to a speedup of 20 on 64 cores for both types of systems, corresponding to an efficiency of about 31%. We report that the mean number of GMRES iterations for solving a single velocity system is about 9; for a single pressure system it ranges from 4 to 6, thus confirming the effectiveness of the multilevel preconditioner. Taking into account the size and the sparsity pattern of the matrices, we can conclude that the performance of SParC-LES in the solution of the linear systems is satisfactory.

In Figure 5.5 we depict the execution time and the speedup of the code for building the fluxes and the deconvolved velocity. In this case, a stronger time reduction than in the case of the linear solvers is obtained in going from 1 to 64 cores. The reason is that this part of SParC-LES mainly performs operations having a smaller communication-to-computation ratio than GMRES and the multilevel preconditioners. On 64 cores, the speedup is about 32 for the computation of the convective fluxes, and it is about 46 for the computation of the diffusive fluxes and the deconvolved velocity; the overall speedup of this phase is about 37.

Finally, in Figure 5.6, we show the overall execution time and speedup of the SParC-LES code. To provide a single picture of the performance of the whole code and its phases, we also display the time and the speedup for the linear solvers and the flux computations, already depicted in the previous figures, as well as for the remaining part of SParC-LES. The latter has a modest scalability, with a speedup of about 13 on 64 cores, because it includes some communication-bounded tasks. We see that SParC-LES is able to achieve a satisfactory overall performance, with a speedup of about 26 on 64 cores, corresponding to a decrease of the execution time from about 78 minutes, on 1 core, to 3 minutes, on 64 cores (note that a complete simulation, with 10^7 time steps, would take more than 20 days on 64 cores). Therefore, we can conclude that using PSBLAS and MLD2P4 in the development of SParC-LES allowed an effective explotation of parallelism with a relatively low coding effort.

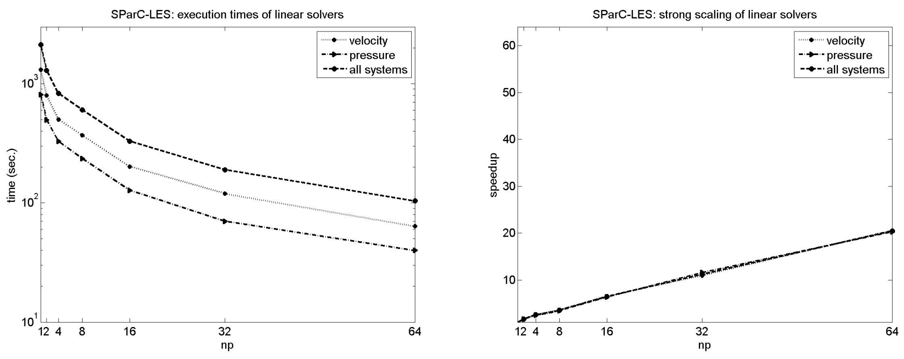

FIG. 5.4. *Parallel performance of the linear solvers in SParC-LES.*

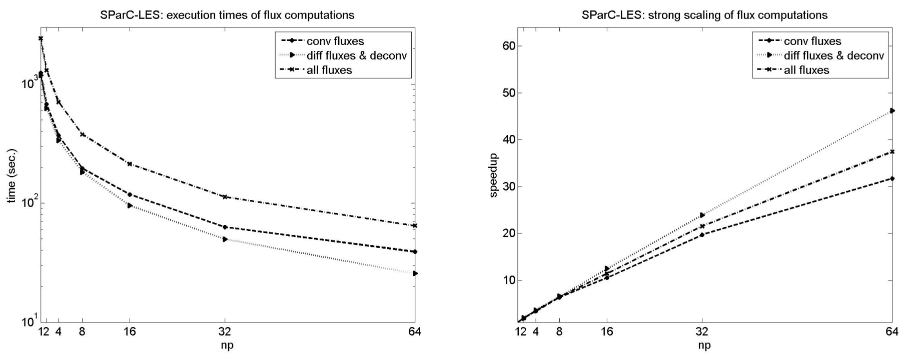

FIG. 5.5. *Parallel performance of the computation of the fluxes and the deconvolved velocity in SParC-LES.*

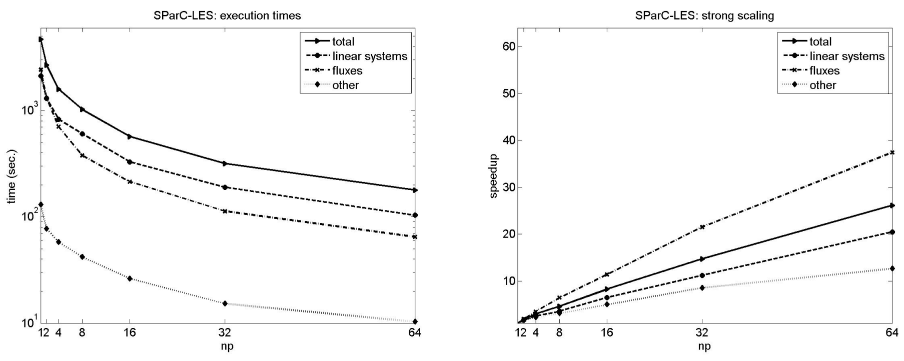

FIG. 5.6. *Parallel performance of SParC-LES and its tasks.*

6. Conclusions. We presented the design and the development of SParC-LES, a parallel code for the LES of turbulent channel flows, which uses as building blocks general-purpose numerical libraries implementing sparse matrix operators, Krylov solvers and algebraic multilevel preconditioners. The key issue in building SParC-LES was the formulation of the main tasks of the numerical simulation procedure in terms of sparse linear algebra kernels, to exploit the computation and data management functionalities provided by PSBLAS and MLD2P4. This approach resulted in a modular and flexible code, where accuracy, robustness and parallel efficiency are committed to the library framework.

SParC-LES was run to simulate a turbulent flow in a plane channel at $Re_\tau = 590$, not only to evaluate its parallel efficiency, but also to assess the overall numerical procedure implemented in it, which, before this work, was only applied to a turbulent channel flow with $Re_\tau = 180$. The parallel performance of the code is more than satisfactory and shows that our approach is able to exploit the computational power offered by modern parallel computers at a relatively moderate coding effort. The analysis of the simulation results show the strengths and drawbacks of the numerical procedure, thus allowing to identify possible modifications to improve the quality of the solution, that can be easily applied thanks to the design methodology used for SParC-LES.

Acknowledgments. We express our deepest thanks to Damian Rouson of the Sandia National Labs, for his help in setting up the computational experiments on Hopper. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] A. Aprovitola, *Sull'applicabilità degli schemi upwind multidimensionali ai volumi finiti per la simulazione numerica delle grandi scale di flussi turbolenti confinati*, Ph.D. dissertation (in Italian), Second University of Naples, 2006.

[2] A. Aprovitola, P. D'Ambra, F. M. Denaro, D. di Serafino, and S. Filippone, *Scalable algebraic multilevel preconditioners with application to CFD*, in "Parallel Computational Fluid Dynamics 2008", D. Tromeur-Dervout, G. Brenner, D. Emerson, J. Erhel, eds., Lecture Notes in Computational Science and Engineering, vol. 74, Springer, 2010, pp. 15–27.

[3] A. Aprovitola, P. D'Ambra, D. di Serafino, and S. Filippone, *On the use of aggregation-based parallel multilevel preconditioners in the LES of wall-bounded turbulent flows*, in "Large-Scale Scientific Computing", I. Lirkov, S. Margenov, and J. Wasniewski, eds., Lecture Notes in Computer Science, vol. 5910, Springer, 2010, pp. 67–75.

[4] A. Aprovitola and F. M. Denaro, *On the application of congruent upwind discretizations for large eddy simulations*, J. Comput. Phys., 194, 1 (2004), pp. 329–343.

[5] A. Aprovitola and F. M. Denaro, *A Non-diffusive, divergence-free, finite volume-based double projection method on non-staggered grids*, Internat. J. Numer. Methods Fluids, 53 (2007), pp. 1127–1172.

[6] A. Aprovitola and F.M. Denaro, *On the control of the mass errors in finite volume-based approximate projection methods for large eddy simulations*, in "Quality and Reliability of Large-Eddy Simulations", J. Mayers, B. J. Geurts, and P. Sagaut, eds., ERCOFTAC Series, vol. 12, Springer, 2008, pp. 213–224.

[7] D. Barbieri, V. Cardellini, S. Filippone, and D. Rouson, *Design patterns for scientific computations on sparse matrices* in "Proceedings of Euro-Par 2011, Parallel Processing Workshops", M. Alexander et al., eds., Lecture Notes in Computer Science, Springer, to appear.

[8] P. D'Ambra, D. di Serafino, and S. Filippone, *On the development of PSBLAS-based parallel two-level Schwarz preconditioners*, Appl. Numer. Math., 57 (2007), pp. 1181–1196.

[9] P. D'Ambra, D. di Serafino, and S. Filippone, *MLD2P4: a package of parallel algebraic multilevel domain decomposition preconditioners in Fortran 95*, ACM Trans. Math. Software, 37 (2010), art. 30, 23 pages.

[10] P. D'Ambra, D. di Serafino, and S. Filippone, *MLD2P4 user's and reference guide*, March 2011, available from `http://www.mld2p4.it/docs.php`.

[11] F. M. Denaro, *On the application of the Helmholtz-Hodge decomposition in projection methods for the numerical solution of the incompressible Navier-Stokes equations with general boundary conditions*, Int. J. Numer. Meth. Fluids, 43 (2003), pp. 43–69.

[12] F. M. Denaro, *Time-accurate intermediate boundary conditions for large eddy simulation based on projection methods*, Int. J. Numer. Meth. Fluids, 48 (2005), pp. 869–9085.

[13] F. M. Denaro, *What does finite volume-based implicit filtering really resolve in large-eddy simulations?*, J. Comp. Physics, 230 (2011), pp. 3849–3883.

[14] G. De Stefano, F. M. Denaro, and G. Riccardi, *High order filtering formulation for control volumes flow simulation*, Int. J. Numer. Meth. Fluids, 37 (2001), pp. 797–835.

[15] I. S. Duff, M. Marrone, G. Radicati, and C. Vittoli, C., *Level 3 basic linear algebra subprograms for sparse matrices: a user-level interface*, ACM Trans. Math. Softw., 23 (1997), pp. 379–401.

[16] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*, Springer, 1996.

[17] S. Filippone and A. Buttari, *PSBLAS 2.4.0 user's guide*, June 2010, available from `http://www.ce.uniroma2.it/psblas/`.

[18] S. Filippone and A. Buttari, *Object-oriented techniques for sparse matrix computations in Fortran 2003*, ACM Trans. Math. Software, to appear.

[19] S. Filippone and M. Colajanni, *PSBLAS: a library for parallel linear algebra computation on sparse matrices* ACM Trans. Math. Software, 26 (2000), pp. 527–550.

[20] I. Foster, *Designing and building parallel programs*, Addison-Wesley, 1995.

[21] W. Gropp, D. Kaushik, D. Keyes, and B. Smith, *High-performance parallel implicit CFD*, Parallel Computing, 27 (2001), pp. 337–362.

[22] S. Hickel, T. Kempe, and N. A. Adams, *Implicit large-eddy simulation applied to turbulent channel flow with periodic constrictions*, Theoretical and Computational Fluid Dynamics, 22 (2008), pp. 227–242.

[23] P. Iannelli, F. M. Denaro, and G. De Stefano, *A deconvolution-based fourth order finite volume method for incompressibile flows on non-uniform grids*, Int. J. Numer. Meth. Fluids, 43 (2003), pp. 431–462.

[24] J. Kim, P. Moin, and R. Moser, *Turbulent statistics in fully developed channel flow at low Reynolds number*, J. Fluid Mechanics, 177 (1987), pp. 133–166.

[25] LESinItaly Group, *A comparative test for assessing the performances of large-eddy simulation codes*, in "Atti del XX Congresso dell'Associazione Italiana di Meccanica Teorica e Applicata", F. Ubertini, E. Viola, S. de Miranda, and G. Castellazzi, eds., 2011, ISBN 978-88-906340-1-7 (electronic), available from `http://www.lamc.ing.unibo.it/aimeta2011/`.

[26] P. Moin and J. Kim, *Tackling turbulence with supercomputers*, Scientific American, 276 (1997), pp. 62–68.

[27] R. D. Moser, J. Kim, and N. N. Mansour, *Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$*, Physics of Fluids, 11 (1999), art. 943, 3 pages.

[28] U. Piomelli, A. Scotti, and E. Balaras, *Large-Eddy simulations of turbulent flows, from desktop to supercomputer*, in "Vector and Parallel Processing - VECPAR 2000", J. M. L. M. Palma, J. Dongarra, and V. Hern/'andez, eds., Lecture Notes in Computer Science, vol. 1981, Springer, 2001, pp. 551–577.

[29] D. W. I. Rouson, J. Xia, and X. Xu, *Scientific software design: The object-oriented way*, Cambridge University Press, 2011.

[30] Y. Saad, *Iterative methods for sparse linear systems, 2nd edition,* SIAM, 2003.

[31] P. Sagaut, *Large eddy simulation for incompressible flows: an introduction*, Springer, 2010.

[32] B. F. Smith, P. E. Bjørstad, and W. D. Gropp, *Domain decomposition. Parallel multilevel methods for elliptic partial differential equations*, Cambridge University Press, 1996.

[33] M. Snir, S. Otto, S. Huss-Lederman, D. W. Walker, and J. J. Dongarra, *MPI: the complete reference. Vol. 1 – The MPI core*, second edition, Scientific and Engineering Computation Sereis, The MIT Press, 1998.

[34] A. Toselli and O. Widlund, *Domain decomposition methods. Algorithms and theory*, Springer, 2005.

[35] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[36] P. N. Brown and H. F. Walker, *GMRES on (nearly) singular systems*, SIAM J. Matrix. Anal. Appl., 18 (1997), pp. 37–51.