The numerical solution of the incompressible Navier–Stokes equations on a low-cost, dedicated parallel computer

Maurizio Quadrio

Dipartimento di Ingegneria Aerospaziale Politecnico di Milano
Via La Masa 34, 20156 Milano, Italy
maurizio.quadrio@polimi.it

and

Paolo Luchini

Dipartimento di Ingegneria Meccanica Universitá di Salerno
via Ponte don Melillo, 84084 Fisciano (SA), Italy
luchini@unisa.it

A numerical method for the direct numerical simulation of the incompressible Navier–Stokes equations in rectangular and cylindrical geometries is presented. The method is designed for efficient shared-memory and distributed-memory parallel computing by using commodity hardware. A novel parallel strategy is implemented to minimize the amount of inter-node communication and by avoiding a global transpose of the data. The method is based on Fourier expansions in the homogeneous directions and fourth-order accurate, compact finite-difference schemes over a variable-spacing mesh in the wall-normal direction. Thanks to the small communication requirements, the computing machines can be connected each other with standard, low-cost network devices. The amount of physical memory deployed to each computing node can be minimal, since the global memory requirements are subdivided amongst the computing machines. The layout of a simple, dedicated and optimized computing system is described, and detailed instructions on how to assemble, install and configure such computing system are given. The basic structure of a numerical code implementing the method is briefly discussed.

Key Words: Navier–Stokes equations, direct numerical simulation, parallel computing, compact finite differences

1. INTRODUCTION

The direct numerical simulation (DNS) of the Navier–Stokes equations for incompressible fluids in geometrically simple, low-Reynolds number turbulent wall flows has become in the last years a valuable tool for basic turbulence research [16]. Among the most important such flows, one can mention turbulent plane channel flows and boundary layers,

turbulent pipe flows, and flows in ducts with annular cross-sections. The former naturally call for the use of a cartesian coordinate system, while the Navier–Stokes equations written in cylindrical coordinates are well suited for the numerical simulation of the latter.

The relevance of such flows is enormous, from the point of view of practical interest and basic turbulence research, and a number of studies exists, based on the DNS of the Navier–Stokes equations, concerning simple flows in cartesian coordinates. Flows which can be easily described in cylindrical coordinates are by no means less interesting; the cylindrical pipe flow, for example, is one of the of the cornerstones in the study of transition to turbulence and fully developed wall turbulent flows, since the pionieristic experimental work by O.Reynolds [27]. Annular duct flows play a role in important engineering applications like axial, coaxial and annular jets with and without swirl, and bear speculative interest, since the effects of the transverse curvature can significantly affect the mean flow and the low-order turbulence statistics, as numerically demonstrated first by Neves *et al.*[21]. The effects of streamwise curvature on the flow, described for example in [7] and [18], are even more important; flows with high streamwise curvature have been only recently addressedd through DNS [20].

Despite their practical relevance, turbulent flows in pipes and circular ducts have not been studied so deeply through DNS as their planar counterparts. This can be at least partially ascribed to the numerical difficulties associated with the cylindrical coordinate system. The first DNS of turbulent pipe flow by Eggels *et al.*[5] dates 7 years later than its planar counterpart [10], and in the following years a limited number of papers has followed. The turbulent flow in an annular duct has been only recently simulated for the first time with a DNS by Quadrio and Luchini [25], by using a preliminary version of the cylindrical numerical method described in this paper.

For the cartesian coordinate system, a very effective formulation of the equations of motion was presented almost 15 years ago by Kim, Moin & Moser in [10], their widely-referenced work on the DNS of turbulent plane channel flow. This formulation has since then been employed in many of the DNSs of turbulent wall flows in planar geometries. It consists in the replacement of the continuity and momentum equations written in primitive variables with two scalar equations, one (second-order) for the normal component of vorticity and one (fourth-order) for the normal component of velocity, much as the Orr–Sommerfeld and Squire decomposition of linear stability problems. In this way pressure disappears from the equations, and the two wall-parallel velocity components are easily computed through the solution of a 2×2 algebraic system (a cheap procedure from a computational point of view), when a Fourier expansion is adopted for the homogeneous directions. A high computational efficiency can thus be achieved. This particular formulation of the Navier–Stokes equation does not call for any particular choice for the discretization of the differential operators in the wall-normal direction. Many researchers have used spectral methods (mainly Chebyshev polynomials) in this direction too, even if in more recent years the use of finite difference schemes has seen growing popularity [16].

The extension of the efficient cartesian formulation to the cylindrical case is not obvious. Most of the existing numerical studies of turbulent flow in cylindrical coordinates write the governing equations in primitive variables, and use each a different numerical method: they range from second-order finite-difference schemes [22] to finite volumes [28] to complex spectral multi-domain techniques (as in [13] and [15]), but most often remain within the pressure-correction approach. The work [21] by Neves *et al.*is based on a spectral discretization, but calculation of pressure is still needed for the numerical solution

of the equations. Moser, Moin & Leonard in [19] presented a method based on a spectral expansion of the flow variables which inherently satisfies the boundary conditions and the continuity equations; this method has been subsequently used in [18] for the simulation of the turbulent flow over a wall with mild streamwise curvature. They indicate that the computational cost of the cylindrical solver is significantly higher than that for the cartesian case, for a given number of degrees of freedom.

The use of cylindrical coordinates is particularly hampered by the unwanted increase of the azimuthal resolution of the computational domain with decreasing radial coordinate. This is the main reason why DNS of the turbulent flow in an annular pipe has proven to be so difficult. The transversal resolution of DNS calculations is known to be crucial for the reliability of the computed turbulence statistics, especially in the near-wall region. If the azimuthal resolution is set according to the needs of the outer region of the computational domain, a waste of computational resources and potential stability problems are determined when the inner region is approached. If, on the other hand, the spatial discretization is adapted to the inner region, the turbulent scales are gradually less and less resolved in the azimuthal direction when the inner region is approached.

XXX CHECK In this paper we describe a numerical method designed for the DNS of turbulent wall flows both in cartesian and cylindrical coordinates. It can take advantage of parallel computing and works well on commodity hardware. The numerical method for the cartesian coordinate system solves the equations in the form described in [10] and recalled in §2. We will illustrate in §3 the main properties of the scheme concerning spatial and temporal discretization, as well as the use of compact, high-order finite differences for the wall-normal direction. The parallel strategy will be discussed in §4. A dedicated, low-cost computing machine, specialized to run efficiently a computer code based on this method, will be described in §5. We call this machine a Personal Supercomputer, since it gives the user perhaps less peak computing power when compared to a real supercomputer, but allows him to achieve a larger throughput, on a time scale typical of a research work.

The closely related method for the cylindrical coordinate system is then presented. First in §6 the governing equations for radial velocity and radial vorticity are worked out in a form suitable to keep the very same structure of the cartesian code. Numerical issues will be addressed in §7, by emphasizing the differences with the cartesian case. Moreover, in §7.5 a strategy for avoiding the unnecessary clustering of azimuthal resolution near the inner wall is introduces. While no particular difficulty is foreseen, we have not yet managed to consider the axis singularity, so that the cylindrical code can presently run only for geometries with an inner wall.

Further details are given in the Appendices. In Appendix A the main steps to design, install and configure a Personal Supercomputer are given. Further information can be requested from the Authors. In Appendix B the basic strucure of a computer code which implements the numerical method described herein is discussed, limited to the serial version of the cartesian code.

2. CARTESIAN COORDINATES: THE GOVERNING EQUATIONS

In this Section we recall the derivation of the equations of motion for the wall-normal velocity and wall-normal vorticity, as illustrated in [10]. In §6 the same formulation will be extended to cylindrical coordinates.

2.1. Problem definition

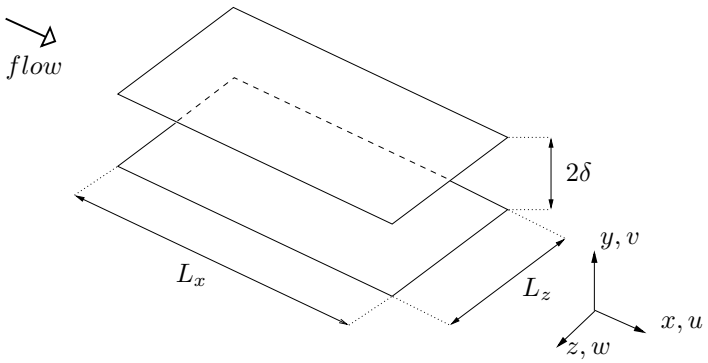

FIG. 1. Sketch of the computational domain for the cartesian coordinate system.

The cartesian coordinate system is illustrated in figure 1, where a sketch of an indefinite plane channel is shown: x, y and z denote the streamwise, wall-normal and spanwise coordinates, and u, v and w the respective components of the velocity vector. The flow is assumed to be periodic in the streamwise and spanwise directions. The lower wall is at position y_ℓ and the upper wall at position y_u. The reference length δ is taken to be one half of the channel height:

$$\delta = \frac{y_u - y_\ell}{2}.$$

Once an appropriate reference velocity U is chosen, a Reynolds number can be defined as:

$$\mathsf{Re} = \frac{U\delta}{\nu},$$

where ν is the kinematic viscosity of the fluid.

The non-dimensional Navier–Stokes equations for an incompressible fluid in cartesian coordinates can then be written as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0; \tag{1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{\partial p}{\partial x} + \frac{1}{\mathsf{Re}}\nabla^2 u; \tag{2a}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} = -\frac{\partial p}{\partial y} + \frac{1}{\mathsf{Re}}\nabla^2 v; \tag{2b}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} = -\frac{\partial p}{\partial z} + \frac{1}{\mathsf{Re}}\nabla^2 w. \tag{2c}$$

The differential problem is closed when an initial condition for all the fluid variables is specified, and suitable boundary conditions are chosen. At the wall the no-slip condition is physically meaningful. Periodic boundary conditions are usually employed in the x and z directions, where either the problem is homogeneous in wall-parallel planes, or a fringe-region technique [1] is adopted to address a non-homogeneous problem. See however [26] for a critical discussion of the subject. Once the periodicity assumption is made for

both the streamwise and spanwise directions, the equations of motion can be conveniently Fourier-transformed along the x and z coordinates.

2.2. Equation for the wall-normal vorticity component

The wall-normal component of the vorticity vector, which we shall indicate with η, is defined as

$$\eta = \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x},$$

and after transforming in Fourier space it is given by:

$$\hat{\eta} = i\beta\hat{u} - i\alpha\hat{w}, \tag{3}$$

where the hat indicates Fourier-transformed quantities, i is the imaginary unit, and the symbols α and β respectively denote the streamwise and spanwise wave numbers. A one-dimensional second-order evolutive equation for $\hat{\eta}$ which does not involve pressure can be easily written, following for example [10], by taking the y component of the curl of the momentum equation, obtaining:

$$\frac{\partial \hat{\eta}}{\partial t} = \frac{1}{\mathsf{Re}}\left(D_2(\hat{\eta}) - k^2\hat{\eta}\right) + i\beta\widehat{HU} - i\alpha\widehat{HW}. \tag{4}$$

In this equation, D_2 denotes the second derivative in the wall-normal direction, $k^2 = \alpha^2 + \beta^2$, and the nonlinear terms are grouped in the following definitions:

$$\widehat{HU} = i\alpha\widehat{uu} + D_1(\widehat{uv}) + i\beta\widehat{uw}; \tag{5a}$$

$$\widehat{HV} = i\alpha\widehat{uv} + D_1(\widehat{vv}) + i\beta\widehat{vw}; \tag{5b}$$

$$\widehat{HW} = i\alpha\widehat{uw} + D_1(\widehat{vw}) + i\beta\widehat{ww}. \tag{5c}$$

The numerical solution of equation (4) requires an initial condition for $\hat{\eta}$, which can be computed from the initial condition for the velocity field. The periodic boundary conditions in the homogeneous directions are automatically satisfied thanks to the Fourier expansions, whereas the no-slip condition for the velocity vector translates in $\hat{\eta} = 0$ to be imposed at the two walls at $y = y_\ell$ and $y = y_u$.

2.3. Equation for the wall-normal velocity component

An equation for the wall-normal velocity component $\hat{v}$, which does not involve pressure, is derived in [10] by summing (2a) derived two times w.r.t. x and y, and (2c) derived two times w.r.t. y and z, then subtracting (2b) derived w.r.t. x and x and substracting once again after derivation w.r.t. z and z. Further simplifications are obtained by invoking the continuity equation to cancel some terms, eventually obtaining the following fourth-order evolutive equation for $\hat{v}$:

$$\begin{aligned}\frac{\partial}{\partial t}\left(D_2(\hat{v}) - k^2\hat{v}\right) = \frac{1}{\mathsf{Re}}\left(D_4(\hat{v}) - 2k^2D_2(\hat{v}) + k^4\hat{v}\right) + \\ - k^2\widehat{HV} - D_1\left(i\alpha\widehat{HU} + i\beta\widehat{HW}\right).\end{aligned} \tag{6}$$

This scalar equation can be solved numerically once an initial condition for $\hat{v}$ is known. The periodic boundary conditions in the homogeneous directions are automatically satisfied

thanks to the Fourier expansions, whereas the no-slip condition for the velocity vector immediately translates in $\hat{v} = 0$ to be imposed at the two walls at $y = y_\ell$ and $y = y_u$. The continuity equation written at the two walls makes evident that the additional two boundary conditions required for the solution of the fourth-order equation (6) are $D_1(\hat{v}) = 0$ at $y = y_\ell$ and $y = y_u$.

2.4. Velocity components in the homogeneous directions

If the nonlinear terms are considered to be known, as is the case when such terms are treated explicitly in the time discretization, the two equations (4) and (6) become uncoupled and, after proper time discretization, can be solved for advancing the solution by one time step, provided the nonlinear terms (5a)-(5c) and their spatial derivatives can be calculated. To this aim, one needs to know how to compute $\hat{u}$ and $\hat{w}$ at a given time starting with the knowledge of $\hat{v}$ and $\hat{\eta}$. By using the definition (4) of $\hat{\eta}$ and the continuity equation (1) written in Fourier space, a 2×2 algebraic system can be written for the unknowns $\hat{u}$ and $\hat{w}$; its analytical solution reads:

$$\begin{cases} \hat{u} = \dfrac{1}{k^2} \left(i\alpha D_1(\hat{v}) - i\beta\hat{\eta} \right) \\ \\ \hat{w} = \dfrac{1}{k^2} \left(i\alpha\hat{\eta} + i\beta D_1(\hat{v}) \right) \end{cases} \tag{7}$$

The present method therefore enjoys its highest computational efficiency only when Fourier discretization is used in the homogeneous directions.

2.4.1. Mean flow in the homogeneous directions

The preceding system (7) is singular when $k^2 = 0$. This is a consequence of having obtained Eqns. (4) and (6) from the initial differential system through a procedure involving spatial derivatives.

Let us introduce a plane-average operator:

$$\widetilde{f} = \frac{1}{L_x} \frac{1}{L_z} \int_0^{L_x} \int_0^{L_z} f \, \mathrm{d}x \, \mathrm{d}z$$

The space-averaged streamwise velocity $\widetilde{u} = \widetilde{u}(y, t)$ is a function of wall-normal coordinate and time only, and in Fourier space it corresponds to the Fourier mode for $k = 0$. The same applites to the spanwise component $\widetilde{w}$. With the present choice of the reference system, with the x axis aligned with the mean flow, the temporal average of $\widetilde{u}$ is the streamwise mean velocity profile, whereas the temporal average of $\widetilde{w}$ will be zero throughout the channel (within the limits of the temporal discretization). This nevertheless allows $\widetilde{w}$ at a given time and at a given distance from the wall to be different from zero.

Two additional equations must be written to calculate $\widetilde{u}$ and $\widetilde{w}$; they can be worked out by applying the linear plane-average operator to the relevant components of the momentum equation:

$$\frac{\partial \widetilde{u}}{\partial t} = \frac{1}{\mathsf{Re}} D_2 \left(\widetilde{u} \right) - D_1 \left(\widetilde{uv} \right) + f_x$$

$$\frac{\partial \widetilde{w}}{\partial t} = \frac{1}{\mathsf{Re}} D_2 \left(\widetilde{w} \right) - D_1 \left(\widetilde{vw} \right) + f_z$$

In these expressions, f_x and f_z are the forcing terms needed to force the flow through the channel against the viscous resistence of the fluid. For the streamwise direction, f_x can be an imposed mean pressure gradient, and in the simulation the flow rate through the channel will oscillate in time around its mean value. f_x can also be a time-dependent spatially uniform pressure gradient, that has to be chosen in such a way that the flow rate remains constant in time at an imposed. The same distinction applies to the spanwise forcing term f_z: in this case however the imposed mean pressure gradient or the imposed mean flow rate is zero, while the other quantity will be zero only after time average.

3. CARTESIAN COORDINATES: THE NUMERICAL METHOD

In this Section the discretization of the continuous differential problem is illustrated. The spectral expansion in the homogeneous directions is a standard approach, whereas the finite-difference discretization of the wall-normal direction with explicit compact scheme will be reported with some detail, as well as the temporal discretization that permits the achievement of a substantial memory saving.

3.1. Spatial discretization in the homogeneous directions

The equations written in Fourier space readily call for an expansion of the unknown functions in terms of truncated Fourier series in the homogeneous directions. For example the wall-normal component v of the velocity vector is represented as:

$$v(x,z,y,t) = \sum_{h=-nx/2}^{+nx/2} \sum_{\ell=-nz/2}^{+nz/2} \hat{v}_{h\ell}(y,t) \mathrm{e}^{i\alpha x} \mathrm{e}^{i\beta z} \tag{8}$$

where:

$$\alpha = \frac{2\pi h}{L_x} = \alpha_0 h; \qquad \beta = \frac{2\pi \ell}{L_z} = \beta_0 \ell.$$

Here h and ℓ are integer indexes corresponding to the streamwise and spanwise direction respectively, and α_0 and β_0 are the fundamental wavenumbers in these directions, defined in terms of the streamwise and spanwise lengths $L_x = 2\pi/\alpha_0$ and $L_z = 2\pi/\beta_0$ of the computational domain. The computational parameters given by the streamwise and spanwise lenght of the computational domain, L_x and L_z, and the truncation of the series, nx and nz, must be chosen so as to miminize computational errors. See again [26] for details regarding the proper choice of a value of L_x.

The numerical evaluation of the non linear terms in Eqns. (4) and (6) would require computationally expensive convolutions in Fourier space. The same evaluation can be performed efficiently by first transofrming the three Fourier components of velocity back in physical space, multiplyng them in all six possible pair combinations and eventually retransforming the results into the Fourier space. Fast Fourier Transform algorithms are used to move from Fourier- to physical space and viceversa. The aliasing error is removed by expanding the number of modes by a factor of at least 3/2 before the inverse Fourier transforms, to avoid the introduction of spurious energy from the high-frequency into the low-frequency modes during the calculation.

3.2. Time discretization

Time integration of the equations is performed by a partially-implicit method, implemented in such a way as to reduce the memory requirements of the code to a minimum,

by exploiting the finite-difference discretization of the wall-normal direction. The use of a partially-implicit scheme is a common approach in DNS [10]: the explicit part of the equations can benefit from a higher-accuracy scheme, while the stability-limiting viscous part is subjected to an implicit time advancement, thus relieving the stability constraint on the time-step size Δt. Our preferred choice, following [17, 9], is to use an explicit third-order, low-storage Runge-Kutta method for the integration of the explicit part of the equations, and an implicit second-order Crank-Nicolson scheme is used for the implicit part. This scheme has been anyway embedded in a modular coding implementation that allows us to change the time-advancement scheme very easily without otherwise affecting the structure of the code. In fact, we have a few other time-advancement schemes built into the code for testing purposes. Here we present the time-discretized version of Eqns. (4) and (6) for a generic wavenumber pair and a generic two-levels scheme for the explicitly-integrated part coupled with the implicit Crank-Nicolson scheme:

$$\begin{aligned}\frac{\lambda}{\Delta t}\hat{\eta}_{h\ell}^{n+1} - \frac{1}{\mathsf{Re}}\left[D_2(\hat{\eta}_{h\ell}^{n+1}) - k^2\hat{\eta}_{h\ell}^{n+1}\right] = \\ \frac{\lambda}{\Delta t}\hat{\eta}_{h\ell}^{n} + \frac{1}{\mathsf{Re}}\left[D_2(\hat{\eta}_{h\ell}^{n}) - k^2\hat{\eta}_{h\ell}^{n}\right] + \\ \theta\left(i\beta_0\ell\widehat{HU}_{h\ell} - i\alpha_0 h\widehat{HW}_{h\ell}\right)^n + \xi\left(i\beta_0\ell\widehat{HU}_{h\ell} - i\alpha_0 h\widehat{HW}_{h\ell}\right)^{n-1}\end{aligned} \tag{9}$$

$$\begin{aligned}\frac{\lambda}{\Delta t}\left(D_2(\hat{v}_{h\ell}^{n+1}) - k^2\hat{v}_{h\ell}^{n+1}\right) - \frac{1}{\mathsf{Re}}\left[D_4(\hat{v}_{h\ell}^{n+1}) - 2k^2D_2(\hat{v}_{h\ell}^{n+1}) + k^4\hat{v}_{h\ell}^{n+1}\right] = \\ \frac{\lambda}{\Delta t}\left(D_2(\hat{v}_{h\ell}^{n}) - k^2\hat{v}_{h\ell}^{n}\right) + \frac{1}{\mathsf{Re}}\left[D_4(\hat{v}_{h\ell}^{n}) - 2k^2D_2(\hat{v}_{h\ell}^{n}) + k^4\hat{v}_{h\ell}^{n}\right] + \\ \theta\left(-k^2\widehat{HV}_{h\ell} - D_1\left(i\alpha_0 h\widehat{HU}_{h\ell} + i\beta_0\ell\widehat{HW}_{h\ell}\right)\right)^n + \\ \xi\left(-k^2\widehat{HV}_{h\ell} - D_1\left(i\alpha_0 h\widehat{HU}_{h\ell} i\beta_0\ell\widehat{HW}_{h\ell}\right)\right)^{n-1}\end{aligned} \tag{10}$$

The three coefficients λ, θ and ξ define a particular time-advancement scheme. For the simplest case of a 2nd-order Adams-Bashfort, for example, we have $\lambda = 2$, $\theta = 3$ and $\xi = -1$.

The procedure to solve these discrete equations is made by two distinct steps. In the first step, the RHSs corresponding to the explicitly-integrated parts part have to be assembled. In the representation (8), at a given time the Fourier coefficients of the variables are represented at different y positions; hence the velocity products can be computed through inverse/direct FFT in wall-parallel planes. Their spatial derivatives are then computed: spectral accuracy can be achieved for wall-parallel derivatives, whereas the finite-differences compact schemes described in §3.3 are used in the wall-normal direction. These spatial derivatives are eventually combined with values of the RHS at previous time levels. The whole y range from one wall to the other must be considered.

The second step involves, for each α, β pair, the solution of a set of two ODEs, derived from the implicitly integrated viscous terms, for which the RHS is now known. A finite-differences discretization of the wall-normal differential operators produces two real banded matrices, in particular pentadiagonal matrices when a 5-point stencil is used. The solution of the resulting two linear systems gives $\hat{\eta}_{h\ell}^{n+1}$ and $\hat{v}_{h\ell}^{n+1}$, and then the planar velocity components $\hat{u}_{h\ell}^{n+1}$ and $\hat{w}_{h\ell}^{n+1}$ can be computed by solving system (7) for each wavenumber pair. For each α, β pair, the solution of the two ODEs requires the simultaneous knowledge

of their RHS in all y positions. The whole α, β space must be considered. In the $\alpha - \beta - y$ space the first step of this procedure proceeds per wall-parallel planes, while the second one proceeds per wall-normal lines.

3.2.1. A memory-saving implementation

A memory-efficient implementation of the time integration procedure is possible, by leveraging the finite-difference discretization of the wall normal derivative. As an example, let us consider the following differential equation for the one-dimensional vector $\mathbf{f} = \mathbf{f}(y)$:

$$\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}t} = \mathcal{N}(\mathbf{f}) + A \cdot \mathbf{f}, \tag{11}$$

where $\mathcal{N}$ denotes non-linear operations on $\mathbf{f}$, and A is the coefficient matrix which describes the linear part. After time discretization of this generic equation, that has identical structure to both the $\hat{\eta}$ and $\hat{v}$ equations, the unknown at time level $n+1$ stems from the solution of the linear system:

$$(A + \lambda I) \cdot \mathbf{f} = \mathbf{g} \tag{12}$$

where $\mathbf{g}$ is given by a linear combination (with suitable coefficients which depend on the particular time integration scheme and, in the case of Runge-Kutta methods, on the particular sub-step too) of $\mathbf{f}$, $\mathcal{N}(\mathbf{f})$ and $A \cdot \mathbf{f}$ evaluated at time level n and at a number of previous time levels. The number of previous time levels depends on the chosen explicit scheme. For the present, low-storage Runge-Kutta scheme, only the additional level $n-1$ is required.

The quantities $\mathbf{f}$, $\mathcal{N}(\mathbf{f})$ and $A \cdot \mathbf{f}$ can be stored in distinct arrays, thus resulting in a memory requirement of 7 variables per point for a two-levels time integration scheme. An obvious, generally adopted optimization is the incremental build into the same array of the linear combination of $\mathbf{f}$, $\mathcal{N}(\mathbf{f})$ and $A \cdot \mathbf{f}$, as soon as the single addendum becomes available. The RHS can then be efficiently stored in the array $\mathbf{f}$ directly, thus easily reducing the memory requirements down to 3 variables per point.

The additional optimization we are able to enforce here relies on the finite-difference discretization of the wall-normal derivatives. Referring to our simple example, the incremental build of the linear combination is performed contemporary to the computation of $\mathcal{N}(\mathbf{f})$ and $A \cdot \mathbf{f}$, the result being stored into the same array which already contained $\mathbf{f}$. The finite-difference discretization ensures that, when dealing with a given y level, only a little slice of values of $\mathbf{f}$, centered at the same y level, is needed to compute $\mathcal{N}(\mathbf{f})$. Hence just a small additional memory space, of the same size of the finite-difference stencil, must be provided, and the global storage space reduces to two variables per point for the example equation (11).

The structure of the time integration procedure implemented in our DNS code is symbolically shown in the bottom chart of figure 2, and compared with the standard approach, illustrated in the top chart. Within the latter approach, in a main loop over the wall-parallel planes (integer index j) the velocity products are computed pseudo-spectrally with planar FFT, their spatial derivatives are taken and the result is eventually stored in the three-dimensional array $\mathbf{nl}$. After the loop has completed, the linear combination of $\mathbf{f}$, $\mathbf{nl}$ and $A \cdot \mathbf{f}$ is assembled in a temporary two-dimensional array rhs, then combined into the three-dimensional array $\mathbf{f}$ with the contribution from the previous time step, and eventually stored in the three-dimensional array $\mathbf{rhsold}$ for later use. The RHS, which uses the storage space of the unknown itself, permits now to solve the linear system which yields

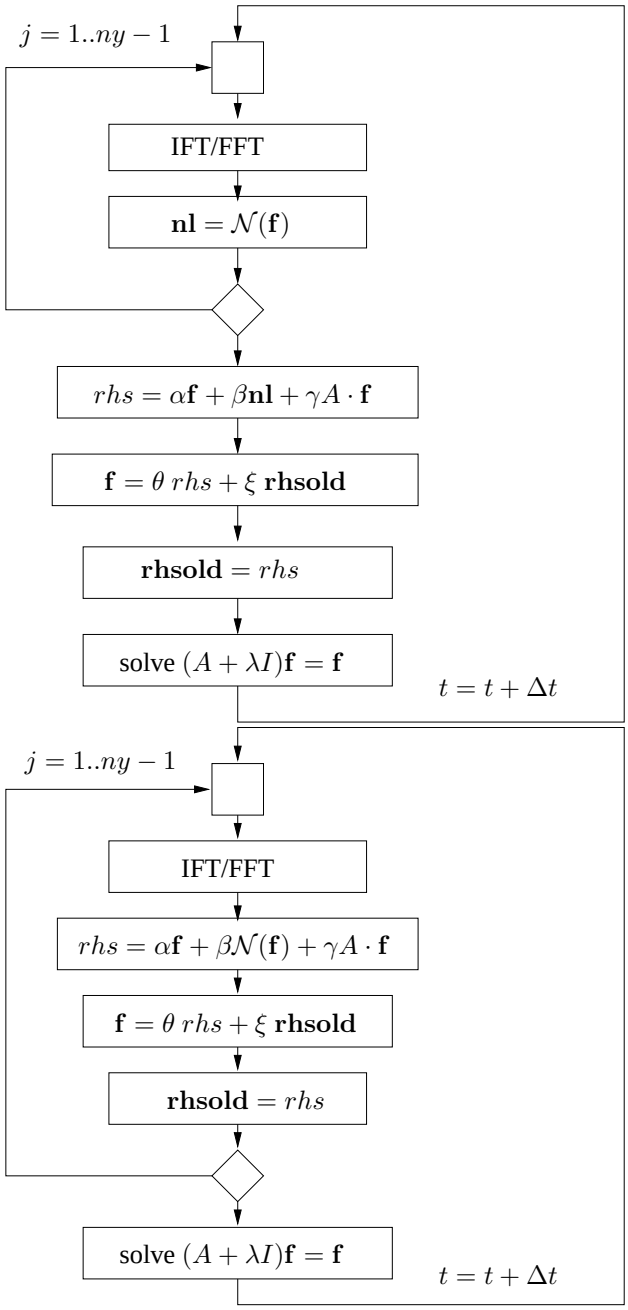

FIG. 2. Comparison between the standard implementation of a two-level time-advancement scheme (top), and the present, memory-efficient implementation (bottom). Variables printed in bold require three-dimensional storage space, while italics marks temporary variables which can use two-dimensional arrays. Greek letters denote coefficients defining a particular time scheme. The present implementation reduces the required memory space for a single equation from 3 to 2 three-dimensional variables.

the unknown at the future time step, and the procedure is over, requiring storage space for 3 three-dimensional arrays.

The flow chart on the bottom of figure 2 illustrates the present approach. In the main loop over wall-parallel planes, not only the non-linear terms are computed, but the RHS of the linear system is assembled plane-by-plane and stored directly in the three-dimensional array $\mathbf{f}$, provided the value of the unknown in a small number of planes (5 when a 5-point finite-difference stencil is employed) is conserved. As a whole, this procedure requires only 2 three-dimensional arrays for each scalar equation.

3.3. High-accuracy compact, explicit finite-difference schemes

The discretization of the wall-normal derivatives D_1, D_2 and D_4, required for the numerical solution of the present problem, is performed through finite difference (FD) compact schemes [11] with fourth-order accuracy over a computational molecule composed by five arbitrarily spaced (with smooth stretching) grid points. We indicate here with $d_1^j(i), i = -2, \ldots, 2$ the five coefficients discretizing the exact operator D_1 over five adjacent grid points centered at y_j:

$$D_1(f(y))|_{y=y_j} = \sum_{i=-2}^{2} d_1^j(i) f(y_{j+i}).$$

The basic idea of compact schemes can be most easily understood by thinking of a standard FD formula in Fourier space as a polynomial interpolation of a trascendent function, with the degree of the polynomial corresponding to the formal order of accuracy of the FD formula. Compact schemes improve the interpolation by replacing the polynomial with a ratio of two polynomials, i.e. with a rational function. This obviously increases the number of available coefficients, and moreover gives control over the behavior at infinity (in frequency space) of the interpolant, whereas a polynomial necessarily diverges. This allows a compact FD formula to approximate a differential operator in a wider frequency range, thus achieving resolution properties similar to those of spectral schemes [11].

Compact schemes are also known as implicit finite-differences schemes, because they typically require the inversion of a linear system for the actual calculation of a derivative [11, 14]. Here we are able to use compact, fourth-order accurate schemes at the cost of explicit schemes, owing to the absence of the third-derivative operator from the equations of motion. Thanks to this property, it is possible to find rational function approximations for the required three FD operators, where the denominator of the function is always the same, as highlighted first in the original Gauss-Jackson-Noumerov compact formulation exploited in his seminal work by Thomas [30], concerning the numerical solution of the Orr-Sommerfeld equations.

To illustrate Thomas' method, let us consider an 4th-order one-dimensional ordinary differential equation, linear for simplicity, in the form:

$$D_4\left(a_4 f\right) + D_2\left(a_2 f\right) + D_1\left(a_1 f\right) + a_0 f = g, \tag{13}$$

where the coefficients $a_i = a_i(y)$ are arbitrary functions of the independent variable y, and $g = g(y)$ is a known RHS. Let us moreover suppose that a differential operator, for example D_4, is approximated in frequency space as the ratio of two polynomials, say $\mathcal{D}_4$ and $\mathcal{D}_0$. Polynomials like $\mathcal{D}_4$ and $\mathcal{D}_0$ have their counterpart in physical space, and d_4 and d_0 are the corresponding FD operators. The key point is to impose that *all* the differential operators

appearing in the example equation (13) admit a representation such as the preceding one, in which the polynomial $\mathcal{D}_0$ at the denominator remains *the same*.

Eq. (13) can thus be recast in the new, discretized form:

$$d_4\left(a_4 f\right) + d_2\left(a_2 f\right) + \ldots + d_1\left(a_1 f\right) + d_0\left(a_0 f\right) = d_0\left(g\right),$$

and this allows us to use explicit FD formulas, provided the operator d_0 is applied to the right-hand-side of our equations. The overhead related to the use of implicit finite difference schemes disppears, while the advantage of using high-accuracy compact schemes is retained.

3.3.1. Calculation of the finite-difference coefficients

The actual computation of the coefficients d_0, d_1, d_2 and d_4 to obtain a formal accuracy of order 4 descends from the requirement that the error of the discrete operator $d_4 d_0^{-1}$ decreases with the step size according to a power law with the desired exponent -4. In practice, following a standard procedure in the theory of Padé approximants [24], this can be enforced by choosing a set t_m of polynomials of y of increasing degree:

$$t_m(y) = 1, y, y^2, ..., y^m, \tag{14}$$

by analytically calculating their derivatives $D_4(t_m)$, and by imposing that the discrete equation:

$$d_4\left(t_m\right) - d_0\left(D_4(t_m)\right) = 0 \tag{15}$$

is verified for the nine polynomials from $m = 0$ up to $m = 8$.

Our computational stencil contains 5 grid points, so that the unknown coefficients d_0 and d_4 are 10. There is however a normalization condition, and we can write the equations in a form where for example:

$$\sum_{i=-2}^{2} d_0(i) = 1. \tag{16}$$

The other 9 conditions are given by Eqn. (15) evaluated for $m = 0, 1, \ldots 8$. We thus can set up, for each distance from the wall, a 10×10 linear system which can be easily solved for the unknown coefficients. The coefficients of the derivatives of lesser degree are derived from analogous relations, leading to two 5×5 linear systems once the d_0's are known. An additional further simplification is possible. Since the polynomials (14) have vanishing D_4 for $m < 4$, thanks to the normalization condition (16) the 10×10 system can be split into two 5×5 subsystems, separately yielding the coefficients d_0 and d_4.

Due to the turbulence anisotropy, the use of a mesh with variable size in the wall-normal direction is advantageous. The procedure outlined above must then be performed numerically at each y_j station, but only at the very beginning of the computations. The computer-based solution of the systems requires a negligible computing time.

We end up with FD operators which are altogether fourth-order accurate; the sole operator D_4 is discretized at sixth-order accuracy. As suggested in [11] and [14], the use of all the degrees of freedom for achieving the highest formal accuracy might not always be the optimal choice. We have therefore attempted to discretize D_4 at fourth-order accuracy only, and to spend the remaining degree of freedom to improve the spectral characteristics of *all* the FD operators at the same time. Our search has shown however that no significant

advantage can be achieved: the maximum of the errors can be reduced only very slightly, and - more important - this reduction does not carry over to the entire frequency range.

The boundaries obviously require non-standard schemes to be designed to properly compute derivatives at the wall. For the boundary points we use non-centered schemes, whose coefficients are computed following the same approach as the interior points, thus preserving by construction the formal accuracy of the method. Nevertheless the numerical error contributed by the boundary presumably carries a higher weight than interior points, albeit mitigated by the non-uniform discretization. A systematic study of this error contribution and of alternative more refined treatments of the boundary are ongoing work.

4. THE PARALLEL STRATEGY

In this Section the parallel strategy, hinge of our numerical method, is described. It is designed with the aim to minimizing the amount of communication, so that commodity network hardware can be used. The same strategy can be used in the cylindrical case.

4.1. Distributed-memory computers

If the calculations are to be executed in parallel by p computing machines (nodes), data necessarily reside on these nodes in a distributed manner, and communication between nodes will take place. Our main design goal is to keep the required amount of communication to a minimum.

When a fully spectral discretization is employed, a transposition of the whole dataset across the computing nodes is needed every time the numerical solution is advanced by one time (sub)step when non-linear terms are evaluated. This is illustrated for example in the paper by Pelz [23], where parallel FFT algorithms are discussed in reference to the pseudo-spectral solution of the Navier–Stokes equations. Pelz shows that there are basically two possibilities, i.e. using a distributed FFT algorithm or actually transposing the data, and that they essentially require the same amount of communication. The two methods are found in [23] to perform, when suitably optimized, in a comparable manner, with the distributed strategy running in slightly shorter times when a small number of processors is used, and the transpose-based method yielding an asymptotically faster behavior for large p. The large amount of communication implies that very fast networking hardware is needed to achieve good parallel performance, and this restrict DNS to be carried out on very expensive computers only.

Of course, when a FD discretization in the y direction is chosen instead of a spectral one, it is conceivable to distribute the unknowns in wall-parallel slices and to carry out the two-dimensional inverse/direct FFTs locally to each machine. Moreover, thanks to the locality of the FD operators, the communication required to compute wall-normal spatial derivatives of velocity products is fairly small, since data transfer is needed only at the interface between contiguous slices. The reason why this strategy has not been used so far is simple: a transposition of the dataset seems just to have been delayed to the second half of the time step advancement procedure. Indeed, the linear systems which stem from the discretization of the viscous terms require the inversion of banded matrices, whose principal dimension span the entire width of the channel, while data are stored in wall-parallel slices.

A transpose of the whole flow field can be avoided however when data are distributed in slices parallel to the walls, with FD schemes being used for wall-normal derivatives. The arrangement of the data across the machines is schematically shown in figure 3: each machine holds all the streamwise and spanwise wavenumbers for ny/p contiguous y positions. As

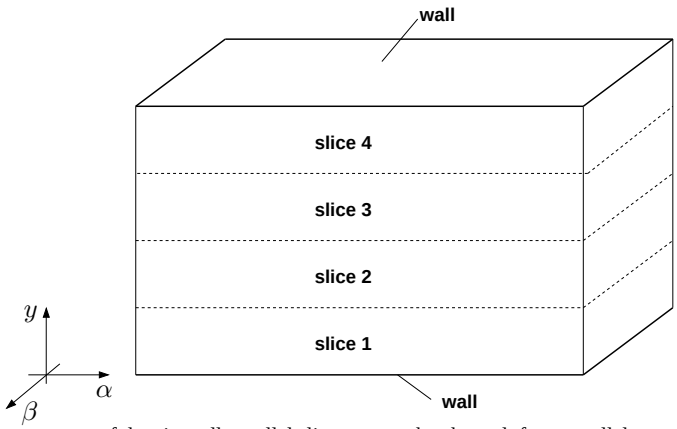

FIG. 3. Arrangement of data in wall-parallel slices across the channel, for a parallel execution with $p = 4$ computing nodes.

said, the planar FFTs do not require communication at all. Wall-normal derivatives needed for the evaluation of the RHSs do require a small amount of communication at the interface between contiguous slices. However, this communication can be avoided at all if, when using a 5-point stencil, two boundary planes on each internal slice side are duplicated on the neighboring slice. This duplication is obviously a waste of computing time, and translates into an increase of the actual size of the computational problem. However, since the duplicated planes are $4(p-1)$, as long as $p \ll ny$ this overhead is negligible. When p becomes comparable to ny, an alternative, slightly different procedure becomes convenient. This alternative strategy is still in development at the present time.

The critical part of the procedure lies in the second half of the time-step advancement, i.e. the solution of the set of two linear systems, one for each h, ℓ pair, and the recovery of the planar velocity components: the necessary data just happen to be spread over all the p machines. It is relatively easy to avoid a global transpose, by solving each system in a *serial* way across the machines: adopting a LU decomposition of the pentadiagonal, distributed matrices, and a subsequent sweep of back-substitutions, only a few coefficients at the interface between two neighboring nodes must be transmitted. The global amount of communication remains very low and, at the same time, local between nearest neighbors only. The problem here is obtaining a reasonably high parallel efficiency: if a single system had to be solved, the computing machines would waste most of their time waiting for the others to complete their task. In other words, with the optimistic assumption of infinite communication speed, the total wall-clock time would be simply equal to the single-processor computing time.

The key observation to obtain high parallel performance is that the number of linear systems to be solved at each time (sub)step is very large, i.e. $(nx+1)(nz+1)$, which is at least 10^4 and sometimes much larger in typical DNS calculations [3]. This allows the solution of the linear systems to be efficiently pipelined as follows. When the LU decomposition of the matrix of the system for a given h, ℓ pair is performed (with a standard Thomas algorithm adapted to pentadiagonal matrices), there is a first loop from the top row of the matrix down to the bottom row (elimination of the unknowns), and then a second loop in the opposite direction (back-substitution). The machine owning the first slice performs the elimination in the local part of the matrix, and then passes on the boundary coefficients to the neighboring machine, which starts its elimination. Instead of

waiting for the elimination in the h, ℓ system matrices to be completed across the machines, the first machine can now immediately start working on the elimination in the matrix of the following system, say $h, \ell + 1$, and so on. After the elimination in the first p systems is started, all the computing machines work at full speed. A synchronization is needed only at the end of the elimination phase, and then the whole procedure can be repeated for the back-substitution phase.

Clearly this pipelined-linear-system (PLS) strategy involves an inter-node communication made by frequent sends and receives of small data packets (typically two lines of a pentadiagonal matrix, or two elements of the RHS array). While the global amount of transmitted data is very small, this poses a serious challenge to out-of-the-box communication libraries, like MPI, which are known to incur in a significant overhead for very small data packets. In fact, we have found unacceptably poor performance when using MPI-type libraries. On the other hand we have succeeded in developing an effective implementation of inter-node communication using only the standard i/o functions provided by the C library. Details of this alternative implementation are illustrated in §5 and Appendix A.

4.2. Estimate of communication requirements

The amount of data which has to be exchanged by each machine for the advancement of the solution by one time step made by 3 Runge–Kutta substeps by the PLS method can be quantified as follows. The number of bytes D_p transmitted and received by each computing node for $p > 2$ and in one complete time step is:

$$D_p = 3 \times 8 \times nx \times nz \times 88 = 2112\, nx \times nz$$

where 3 is the number of temporal substeps, 8 accounts for 8-bytes variables, and 88 is the total number of scalar variables that are exchanged at the slice interfaces for each wavenumber pair (during solution of the linear systems and of the algebraic system to compute $\hat{u}$ and $\hat{w}$). To quantify, in a simulation with $nx = ny = nz = 128$ D_p amounts to ≈ 276 MBit of network traffic. It is interesting to note that D_p is linear in the total number of Fourier modes $nx \times nz$, but is independent upon ny. Moreover, the amount of traffic does not change when p increases.

To appreaciate this estimate, we can also carry out the same estimate when a standard parallel FFT, i.e. the transpose method, is used. In this case the amount of data (in bytes) D_t exchanged by each machine for the complete advancement by one time step method is as follows:

$$D_t = 3 \times 8 \times (p-1)\frac{nx}{p} \times \frac{3}{2}\frac{nz}{p} \times ny \times 18 = 648\, \frac{p-1}{p^2}\, nx \times nz \times ny$$

Again, the factors 3 and 8 account for the number of temporal substeps and the 8-bytes variables respectively. In the whole process of computing non-linear terms 9 scalars have to be sent and received (3 velocity components before IFT and 6 velocity products after FFT); for each wall-parallel plane, each machine must exchange with each of the others $p-1$ nodes an amount of $nx \times nz/p^2$ grid cells, and the factor $3/2$ corresponds to dealiasing in one horizontal direction (the $3/2$ expansion, and the subsequent removal of higher-wavenumber modes, in the other horizontal direction can be performed after transmission).

The ratio between the communication required by the transpose-based method and the PLS method can thus be written as:

$$\frac{D_t}{D_p} = 0.307\,\frac{p-1}{p^2}\,ny$$

which corresponds to the intuitive idea that the transpose method exchanges all the variables it stores locally, whereas the PLS method only exchanges a (small) number of wall-parallel planes, independent on ny and p. Moreover the ratio D_t/D_p, being proportional to ny for a given p, is expected to increase with the Reynolds number of the simulation, since so does the number of points needed to discretize the wall-normal direction. More important, when the transpose-based method is employed, the global amount of communication that has to be managed by the switch increases with the number of computing machines and is all-to-all rather than between nearest neighbors only, so that its performance is expected to degrade when a large p is used. This is perhaps the main advantage of the PLS parallel strategy.

4.3. Shared-memory machines

The single computing node may be single-CPU or multi-CPU. In the latter case, it is possible to exploit an additional and complementary parallel strategy, which does not rely on message-passing communication anymore, and takes advantage of the fact that local CPUs have direct access to the same, local memory space. We stress that this is different from using a message-passing strategy on a shared-memory machine, where the shared memory simply becomes a faster transmission medium. Using multiple CPUs on the same memory space may yield an additional gain in computing time, at the only cost of having the computing nodes equipped with more than one (typically two) CPUs. For example the FFT of a whole plane from physical to Fourier-space and vice-versa can be easily parallelized this way, as well as the computing-intensive part of building up the RHS terms. With SMP machines, high parallel efficiencies can be obtained quite easily by "forking" new processes which read from and write to the same memory space; the operating system itself then handles the assignment of tasks to different CPUs, and only task synchronization is a concern at the programming level.

5. THE PERSONAL SUPERCOMPUTER

While a computer program based on the numerical method described heretoforth can be easily run on a general-purpose cluster of machines, connected through a network in a star topology with a switch, for maximum efficiency a dedicated computing system can be specifically designed and built on top of the parallel algorithm described above.

At the CPU level, the mass-marketed CPUs which are commonly found today in desktop systems are the perfect choice: their performance is comparable to the computing power of the single computing element of any supercomputer [4], at a fraction of the price. The single computing node can hence be a standard desktop computer; SMP mainboards with two CPUs are very cheap and easily available.

The present PLS parallel strategy allows an important simplification in the connection topology of the machines. Since the transposition of the whole dataset is avoided, communications are always of the *point-to-point* type; moreover, each computing machine needs to exchange data with and only with two neighboring machines only. This can be exploited with a simple ring-like connection topology among the computing machines, sketched in

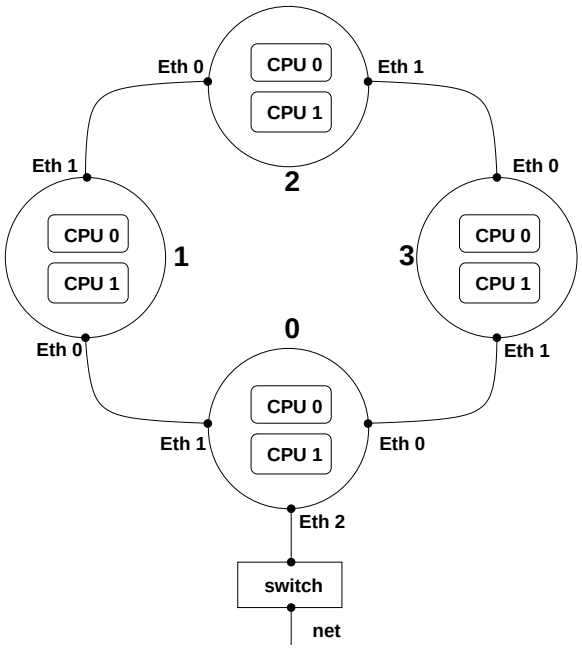

FIG. 4. Conceptual scheme of the connection topology for a computing system made by 4 nodes; one machine may be connected to the local net through a switch, if the system has to be operated remotely.

figure 4, which replicates the logical exchange of information and the data structure previously illustrated in figure 3: each machine is connected through two network cards only to the previous machine and to the next. The necessity of a switch (with the implied additional latencies in the network path) is thus eliminated, in favor of simplicity, performance and cost-effectiveness.

Concerning the transmission protocol, the simplest choice is the standard, error-corrected TCP/IP protocol. We have estimated that on typical problem sizes the overall benefits from using a dedicated protocol (for example the GAMMA protocol described in [2]) would be negligible: since the ratio between communication time and computing time is very low, the improvements by using such a protocol are almost negligible, and to be weighted against the increase in complexity and decrease in portability.

The simplest and fastest strategy we have devised for the communication type is to rely directly on the standard networking services of the Unix operating system, i.e. sockets (after all, message-passing libraries are socket-based). At the programming level, this operation is very simple, since a socket is seen as a plain file to write into and to read from. Using sockets allows us to take advantage easily and efficiently of the advanced buffering techniques incorporated in the management of the input/output streams by the operating system: after opening the socket once and for all, it is sufficient to write (read) data to (from) the socket whenever they are available (needed), and the operating system itself manages flushing the socket when its associated buffer is full. We have found however that for best performances the buffer size had to be empirically adjusted: for Fast Ethernet hardware, the optimum has been found at the value of 800 bytes, significantly smaller than the usual value (the Linux operating system defaults at 8192).

In the year 2001 we have built at Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano the first prototype of such a dedicated system, composed of 8 SMP Personal Computers. Each node is equipped with 2 Pentium III 733MHz CPU and 512MB of

133MHz SDRAM. The nodes are connected to each other by two cheap 100MBits Fast Ethernet cards. This machine is still heavily used today. In the meanwhile, we have installed in 2003 a second-generation machine, made by 10 SMP nodes. Each node carries 2 Intel Xeon 2.66 GHz CPU, and 512MB of 266 MHz SDRAM; the interconnects are two onboard Gigabit Ethernet cards. In late 2004 the third, largest machine entered production stage, at Dipartimento di Ingegneria Meccanica dell'Università di Salerno. This machine is made by 64 SMP nodes connected each other in a ring with two onboard Gigabit ethernet. Eight of the nodes carry a third PCI Gigabit Ethernet card, through which they are interconnected with a switch, so that the number of hops between any two nodes is limited below 5. Each node is made by two AMD Opteron 1.6 GHz CPU, with 1GB of SDRAM installed.

We call such machines Personal Supercomputers. The performance of our numerical method used on these system is indeed comparable to that of a supercomputer. In addition, such machines enjoy the advantages of a simple desktop Personal Computer: low cost and easy upgrades, unlimited availability even to a single user, low weight, noise and heat production, small requirements of floor space, etc. Further details and instructions to build and configure such a machine can be found in Appendix B.

5.1. Summary of performance measurements

A thourough evaluation of the performance of the present numerical method (referred to as the PLS method in the following), as well as the performance of our Personal Supercomputers when used with the present method, is contained in the paper [12]. Here we report only the main results from that paper.

The amount of required RAM is dictated by the number and the size of the three-dimensional arrays, and it is typically reported [10, 29, 8] to be no less than 7 $nx \times ny \times nz$ floating-point variables. Cases where RAM requirements are significantly higher are not uncommon: for example in [6] a channel flow simulation of $128 \times 65 \times 128$ reportedly required 1.2GB of RAM, suggesting a memory occupation approximately 18 times larger.

In our code all the traditional optimizations are employed, and an additional saving specific to the present method comes from the implementation of the time advancement procedure, discussed in §3.2.1, which takes advantage of the finite-difference discretization of the wall-normal derivatives. Thus our code requires a memory space of $5\ nx \times ny \times nz$ floating-point variables, plus workspace and two-dimensional arrays. For example a simulation with $nx = ny = nz = 128$ takes only 94 MBytes of RAM (using 64-bit floating-point variables).

In a parallel run the memory requirement can be subdivided among the computing machines. With $p = 2$ the same 128^3 case runs with 53 MBytes of RAM (note that the amount of RAM is slightly larger than one half of the $p = 1$ case, due to the aforementioned duplication of boundary planes). The system as a whole therefore allows the simulation of turbulence problems of very large computational size even with a relatively small amount of RAM deployed in each node. A problem with computational size of 400^3 would easily fit into our 8 nodes equipped with 512MB RAM each.

As far as CPU efficiency is concerned, without special optimization the 128^3 test case mentioned above requires 42.8 CPU seconds for the computation of a full three-sub-steps Runge-Kutta temporal step on a single Pentium III 733MHz processor. Internal timings show that the direct/inverse two-dimensional FFT routines take the largest part of the CPU time, namely 56%. The calculation of the RHS of the two governing equations (where wall-normal derivatives are evaluated) takes 25% of the total CPU time, the solution of

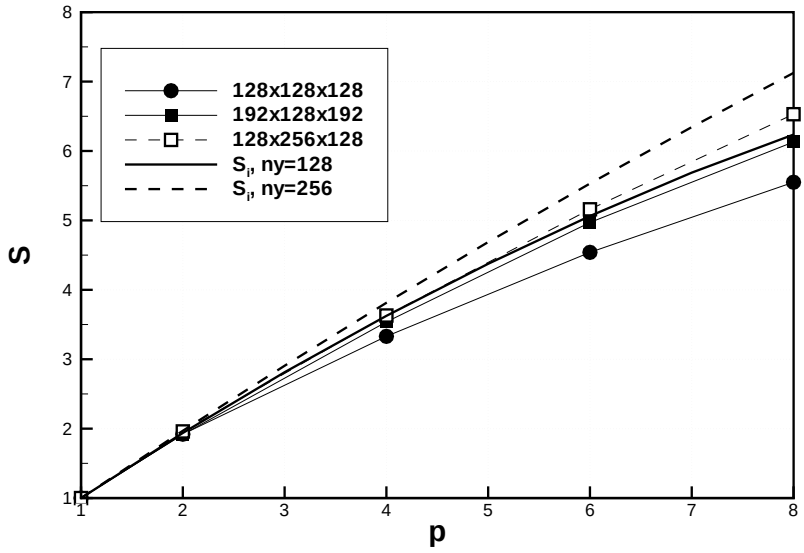

FIG. 5. Measured speedup on the Pentium III-based machine as a function of the number p of computing nodes. Thick lines are the ideal speedup S_i from Eq. (17) for $ny = 128$ (continuous line) and $ny = 256$ (dashed line).

the linear systems arising from the implicit part around 12%, and the calculation of the planar velocity components 3%. The time-stepping scheme takes 3% and computing a few runtime statistics requires an additional 1% of the CPU time.

The parallel (distributed-memory) performance of the code is illustrated in figure 5, where speedup ratios are reported as a function of the number of computing nodes. We define the speedup factor as the ratio of the actual wall-clock computing time t_p obtained with p nodes and the wall-clock time t_1 required by the same computation on a single node:

$$S(p) = \frac{t_p}{t_1}.$$

The maximum or ideal speedup factor S_i that we can expect with our PLS algorithm, corresponding to the assumption of infinite communication speed, is less than linear, and can be estimated with the formula:

$$S_i(p) = p\left(1 - \frac{4(p-1)}{ny}\right), \tag{17}$$

where the factor 4 accounts for the two wall-parallel planes duplicated at each side of interior slices. Eq. (17) reduces to a linear speedup when $ny \to \infty$ for a finite value of p. A quantitative evaluation of the function (17) for typical values of $ny = \mathcal{O}(100)$ shows that the maximum achievable speedup is nearly linear as long as the number of nodes remains moderate, i.e. $p < 10$.

The maximum possible speedup S_i is shown with thick lines. S_i approaches the linear speedup for large ny, being reasonably high as long as p remains small compared to ny: with $p = 8$ it is 6.25 for $ny = 128$ and 7.125 for $ny = 256$. Notwithstanding the commodity networking hardware and the overhead implied by the error-corrected TCP protocol, the actual performance compared to S_i is extremely good, and improves with the

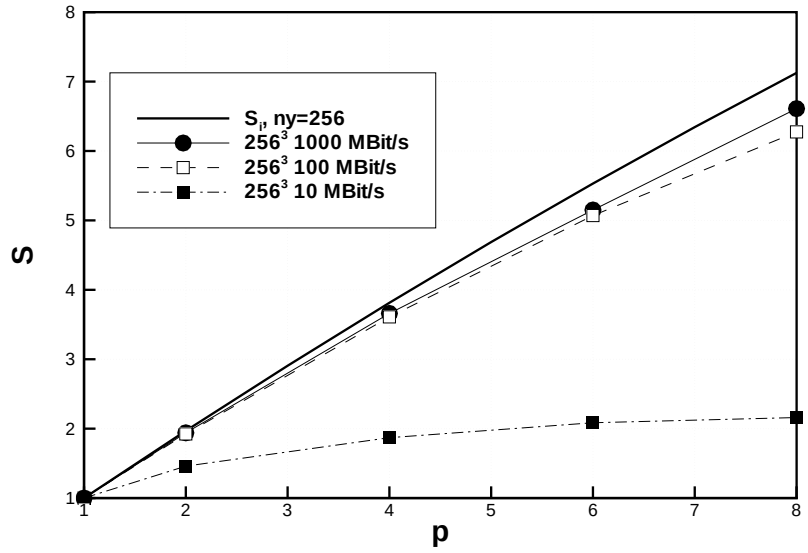

FIG. 6. Measured speedup on the Opteron-based machine as a function of the number p of computing nodes. Thick line is the ideal speedup from Eq. (17) for $ny = 256$. Speedup measured when using Gigabit Ethernet cards (circles), and the same cards run at the slower speed of 100MBit/s (empty squares) and 10 MBit/s (filled squares).

size of the computational problem. The case $192 \times 128 \times 192$ is hardly penalized by the time spent for communication, which is only 2% of the total computing time when $p = 8$. The communication time becomes 7% of the total computing time for the larger case of $nx = 128$, $ny = 256$ and $nz = 128$, and is 12% for the worst (i.e. smallest) case of 128^3, which requires 7.7 seconds for one time step on our machine, with a speedup of 5.55.

Figure 6 illustrates the speedup achieved with the faster Opteron machines connected via Gigabit Ethernet cards in the ring-topology layout, compared with S_i. The test case has a size of 256^3. The CPUs of this system are significantly faster than the Pentium III, and the network cards, while having 10 times larger bandwidth, have latency characteristics typical of Fast Ethernet cards. It is remarkable how well the measured speedup still approaches the ideal speedup, even at the largest tested value of p. Furthermore, we report also the measured speedup when the Opteron machines are used with the Gigabit cards set up to work at the lower speeds of 100 MBit/s and 10MBit/s. It is interesting to observe how slightly performance is degraded in the case at 100MBit/s, whose curve is nearly indistinguishable form that at 1GBit/s. Even with the slowest 10MBit/s bandwidth connecting such fast processors, and with a problem of large computational size, it is noteworthy how the present method is capable to achieve a reasonable speedup for low p and not to ever degrade below $S = 1$. This relative insensitivity to the available bandwidth can be ascribed to the limited amount of communication required by the present method.

Lastly, the PLS method is compared on the Opteron machine with the transpose-based method of performing parallel FFT. By taking advantage of the connection topology of the Opteron machines, which are connected both in the ring topology with the onboard network cards and in the star topology with the switch, the same code can be used where only the parallel strategy is modified. Figure 7 reports comparative measurements between the PLS and the transpose-based method. The PLS method is run with the machines connected

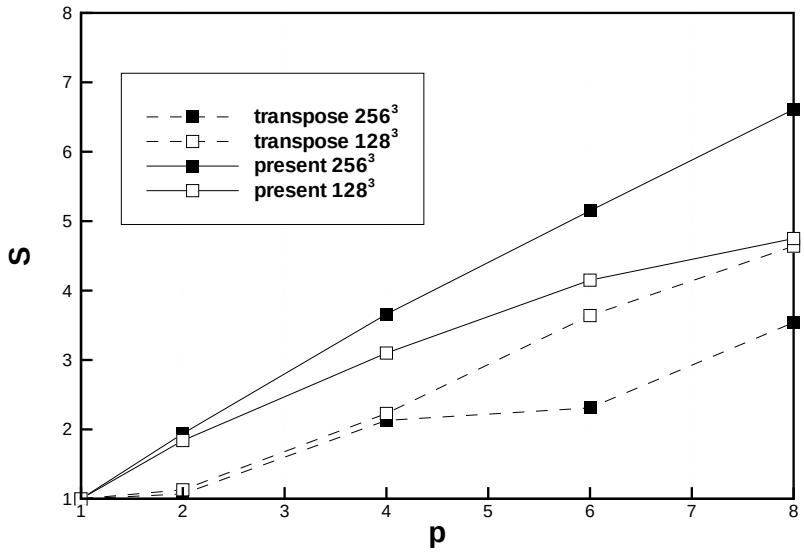

FIG. 7. Measured speedup on the Opteron-based machine as a function of the number p of computing nodes. Continuous line is the PLS method, and dashed line is the transpose-based method.

in a ring, while the transpose-based method is tested with machines linked through the switch. Measurements show that $S > 1$ can now be achieved with the transpose-based method. However, the transpose method performs best for the smallest problem size, while the PLS shows the opposite behavior. For the 256^3 case, which is a reasonable size for such machines, the speedup from the transpose-based method is around one half of what can be obtained with PLS.

6. CYLINDRICAL COORDINATES: THE GOVERNING EQUATIONS

In this Section we present the extension of the numerical method previously described the cylindrical coordinate system. First the procedure to write a two-equations formulation of the differential problem for the radial velocity and radial vorticity is described. This has been already published in [25]. The material in §7.3 illustrates how fourth-order accruacy can still be achieved, and it has never been published elsewhere.

6.1. Problem definition

The cylindrical coordinate system is illustrated in figure 8, where a sketch of an annular duct is shown: x, r and θ denote the axial, wall-normal (radial) and azimuthal coordinates, and u, v and w the respective components of the velocity vector. The flow is assumed to be periodic in the axial and azimuthal directions. The inner cylinder has radius $\mathcal{R}_i$ and the outer cylinder has radius $\mathcal{R}_o$. The reference length δ is taken to be one half of the gap width:

$$\delta = \frac{\mathcal{R}_o - \mathcal{R}_i}{2}$$

Once an appropriate reference velocity U is chosen, a Reynolds number can be defined as:

$$\mathsf{Re} = \frac{U_b \delta}{\nu}$$

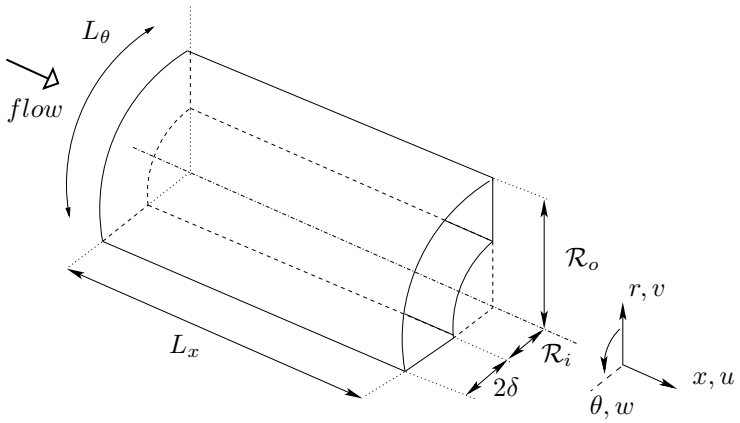

FIG. 8. Sketch of the computational domain for the cylindrical coordinate system

where ν is the kinematic viscosity of the fluid.

The non-dimensional Navier–Stokes equations for an incompressible fluid in cylindrical coordinates can then be written as:

$$\frac{\partial u}{\partial x} + \frac{1}{r}\frac{\partial (rv)}{\partial r} + \frac{1}{r}\frac{\partial w}{\partial \theta} = 0; \tag{18}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial r} + \frac{w}{r}\frac{\partial u}{\partial \theta} = -\frac{\partial p}{\partial x} + \frac{1}{\mathsf{Re}}\nabla^2 u; \tag{19a}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial r} + \frac{w}{r}\frac{\partial v}{\partial \theta} - \frac{w^2}{r} = -\frac{\partial p}{\partial r} + \frac{1}{\mathsf{Re}}\left(\nabla^2 v - \frac{v}{r^2} - \frac{2}{r^2}\frac{\partial w}{\partial \theta}\right); \tag{19b}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial r} + \frac{w}{r}\frac{\partial w}{\partial \theta} + \frac{vw}{r} = -\frac{1}{r}\frac{\partial p}{\partial \theta} + \frac{1}{\mathsf{Re}}\left(\nabla^2 w - \frac{w}{r^2} + \frac{2}{r^2}\frac{\partial v}{\partial \theta}\right), \tag{19c}$$

where the Laplacian operator in cylindrical coordinates takes the form:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}. \tag{20}$$

The differential problem is closed when an initial condition for all the fluid variables is specified, and suitable buondary conditions are chosen. At the walls the no-slip condition is physically meaningful, whereas periodic boundary conditions are used for the azimuthal direction, as well as for the axial direction, under the same assumptions discussed in §2.1 for the cartesian case.

Once the periodicity assumption is made both in the axial and azimuthal directions, the equations of motion can be conveniently Fourier-transformed along the x and θ coordinates. The symbols α and m denote the axial and azimuthal wave numbers, respectively. By defining $k^2 = (m/r)^2 + \alpha^2$, and by introducing the Chandrasekar notation:

$$D_1(f) = \frac{\partial f}{\partial r}; \qquad D_*(f) = \frac{\partial f}{\partial r} + \frac{f}{r}, \tag{21}$$

the Fourier-transformed Laplacian operator (20) can be written in the more compact form:

$$\nabla^2 = D_* D_1 - k^2$$

The transformed equations, where the hat indicates the Fourier components of the transformed variable, are:

$$i\alpha\hat{u} + D_*(\hat{v}) + \frac{im}{r}\hat{w} = 0; \tag{22}$$

$$\frac{\partial \hat{u}}{\partial t} = -i\alpha\hat{p} + \frac{1}{\mathsf{Re}}\left(D_* D_1(\hat{u}) - k^2\hat{u}\right) + \widehat{HU}; \tag{23a}$$

$$\frac{\partial \hat{v}}{\partial t} = -D_1(\hat{p}) + \frac{1}{\mathsf{Re}}\left(D_1 D_*(\hat{v}) - k^2\hat{v} - \frac{2im}{r^2}\hat{w}\right) + \widehat{HV}; \tag{23b}$$

$$\frac{\partial \hat{w}}{\partial t} = -\frac{im}{r}\hat{p} + \frac{1}{\mathsf{Re}}\left(D_1 D_*(\hat{w}) - k^2\hat{w} + \frac{2im}{r^2}\hat{v}\right) + \widehat{HW}. \tag{23c}$$

In these expressions, the nonlinear convective terms have been grouped under the following definitions:

$$\widehat{HU} = -i\alpha\widehat{uu} - D_*(\widehat{uv}) - \frac{im}{r}\widehat{uw}; \tag{24a}$$

$$\widehat{HV} = -i\alpha\widehat{uv} - D_*(\widehat{vv}) - \frac{im}{r}\widehat{vw} + \frac{1}{r}\widehat{ww}; \tag{24b}$$

$$\widehat{HW} = -i\alpha\widehat{uw} - D_1(\widehat{uw}) - \frac{im}{r}\hat{w}^2 - \frac{2}{r}\widehat{vw}. \tag{24c}$$

It can be noticed that the main difference between (22), (23a-c) and the analogous equations in cartesian coordinates is the dependence of k^2 upon r. As a consequence thereof, k^2 does not commute with the operators for radial derivatives. In addition, the components of the momentum equations are coupled through the viscous and convective terms; therefore it can be anticipated that in the time advancement procedure a fully implicit treatment of the viscous terms, as usually done in the cartesian case, will not be possible.

6.2. Equation for the radial vorticity component

The wall-normal (radial) component of the vorticity vector, which we shall indicate with η, is defined as

$$\eta = \frac{1}{r}\frac{\partial u}{\partial \theta} - \frac{\partial w}{\partial x},$$

and after transforming in Fourier space it is given by:

$$\hat{\eta} = \frac{im}{r}\hat{u} - i\alpha\hat{w} \tag{25}$$

Following a procedure which resembles that of the cartesian case, an equation for $\hat{\eta}$, which does not involve pressure, can be written by taking the radial component of the curl

of the momentum equation. By multiplying equation (23a) times im/r and subtracting equation (23c) times $i\alpha$, one gets:

$$\frac{im}{r}\frac{\partial \hat{u}}{\partial t} - i\alpha\frac{\partial \hat{w}}{\partial t} = \frac{1}{\mathsf{Re}}\left[\frac{im}{r}D_*D_1(\hat{u}) - i\alpha D_1D_*(\hat{w}) - k^2\left(\frac{im}{r}\hat{u} - i\alpha\hat{w}\right) + 2\frac{m\alpha}{r^2}\hat{v}\right] + \frac{im}{r}\widehat{HU} - i\alpha\widehat{HW} \quad (26)$$

By writing down the expression for $\nabla^2\hat{\eta}$:

$$\nabla^2\hat{\eta} = -k^2\left(\frac{im}{r}\hat{u} - i\alpha\hat{w}\right) + D_*D_1\left(\frac{im}{r}\hat{u} - i\alpha\hat{w}\right),$$

and remembering the definitions (21) of the operators D_1 and D_* and the fact that:

$$D_1D_* = D_*D_1 - \frac{1}{r^2}$$

one can substitute in the preceding equation, and write the following second-order equation for $\hat{\eta}$:

$$\frac{\partial \hat{\eta}}{\partial t} = \frac{1}{\mathsf{Re}}\left(D_1D_*(\hat{\eta}) - k^2\hat{\eta} + 2\frac{im}{r^2}D_1(\hat{u}) + 2\frac{m\alpha}{r^2}\hat{v}\right) + \frac{im}{r}\widehat{HU} - i\alpha\widehat{HW} \quad (27)$$

The numerical solution of Eqn. (27) requires an initial condition for $\hat{\eta}$, which can be computed form the initial condition for the velocity field. The periodic boundary conditions in the homogeneous directions are automatically satisfied thanks to the Fourier transform, whereas the no-slip condition for the velocity vector translates in $\hat{\eta} = 0$ to be imposed at the two walls at $r = \mathcal{R}_i$ and $r = \mathcal{R}_o$.

This equation has an overall structure which is analogous to that of the corresponding cartesian equation (4), except that it is not independent upon $\hat{v}$. Moreover, a curvature term proportional to the first radial derivative of $\hat{u}$ appears.

6.3. Equation for the radial velocity component

The derivation of an equation for the radial component $\hat{v}$ of the velocity vector, again without pressure terms, is less straightforward, and requires the use of the continuity equation in order to obtain an expression for $\hat{p}$ as a function of the velocity components.

The first step consists in taking the time derivative of the Fourier-transformed continuity equation (22):

$$\frac{\partial D_*(\hat{v})}{\partial t} = -i\alpha\frac{\partial \hat{u}}{\partial t} - \frac{im}{r}\frac{\partial \hat{w}}{\partial t}.$$

The time derivatives of $\hat{u}$ and $\hat{w}$ can be replaced by the corresponding expression from equations (23a) and (23c), thus giving:

$$\frac{\partial D_*(\hat{v})}{\partial t} = -k^2\hat{p} - i\alpha\left[\frac{1}{\mathsf{Re}}\left(D_*D_1(\hat{u}) - k^2\hat{u}\right) + \widehat{HU}\right] - \frac{im}{r}\left[\frac{1}{\mathsf{Re}}\left(D_1D_*(\hat{w}) - k^2\hat{w} + 2\frac{im}{r^2}\hat{v}\right) + \widehat{HW}\right].$$

The continuity equation can be invoked again to simplify some terms, together with the relations obtained by applying the operators D_1/r and D_2 to it, namely:

$$-D_2 D_*(\hat{v}) = i\alpha D_2(\hat{u}) + \frac{im}{r} D_2(\hat{w}) + 2\frac{im}{r^3}\hat{w} - 2\frac{im}{r^2} D_1(\hat{w});$$

$$-\frac{1}{r} D_1 D_*(\hat{v}) = \frac{i\alpha}{r} D_1(\hat{u}) + \frac{im}{r^2} D_1(\hat{w}) - \frac{im}{r^3}\hat{w}.$$

By also applying the identity:

$$D_* D_1 D_*(\hat{v}) = D_2 D_*(\hat{v}) + \frac{1}{r} D_1 D_*(\hat{v}),$$

after some algebra, the following expression for $\hat{p}$ is obtained:

$$\hat{p} = -\frac{1}{\mathsf{Re}}\frac{1}{k^2}\left[k^2 D_*(\hat{v}) - D_* D_1 D_*(\hat{v}) - 2\frac{m^2}{r^3}\hat{v} + 2\frac{im}{r^2} D_1(\hat{w}) - 2\frac{im}{r^3}\hat{w}\right] + \\ - \frac{1}{k^2}\left[\frac{\partial D_*(\hat{v})}{\partial t} + i\alpha\widehat{HU} + \frac{im}{r}\widehat{HW}\right].$$

This expression for $\hat{p}$ can now be differentiated with respect to the radial coordinate, and then substituted into equation (23b) to get rid of $\hat{p}$ altogether. Eventually the fourth-order equation for $\hat{v}$ emerges in the final form:

$$\frac{\partial}{\partial t}\left[\hat{v} - D_1\left(\frac{1}{k^2} D_*(\hat{v})\right)\right] = \frac{1}{\mathsf{Re}} D_1\left\{\frac{1}{k^2}\left[k^2 D_*(\hat{v}) - D_* D_1 D_*(\hat{v}) - 2\frac{m^2}{r^3}\hat{v} + \right.\right. \\ \left.\left. 2\frac{im}{r^2} D_1(\hat{w}) - 2\frac{im}{r^3}\hat{w}\right]\right\} + \frac{1}{\mathsf{Re}}\left(-k^2\hat{v} + D_1 D_*(\hat{v}) - 2\frac{im}{r^2}\hat{w}\right) + \\ D_1\left[\frac{1}{k^2}\left(i\alpha\,\widehat{HU} + \frac{im}{r}\widehat{HW}\right)\right] + \widehat{HV}. \quad (28)$$

This scalar equation can be solved numerically provied an initial condition for $\hat{v}$ is known. The periodic boundary conditions in the homogeneous directions are automatically satisfied thanks to the Fourier transform, whereas the no-slip condition for the velocity vector immediately translates in $\hat{v} = 0$ to be imposed at the two walls. The continuity equation written at the two walls makes evident that the additional two boundary conditions required for the solution of (28) are $D_1(\hat{v}) = 0$ at $r = \mathcal{R}_i$ and $r = \mathcal{R}_o$. Equation (28) shares with its cartesian counterpart (6) the general structure, in particular the fact that it is independent of $\hat{\eta}$. Curvature terms proportional to $\hat{w}$ and to its first radial derivative are present.

6.4. Velocity components in the homogeneous directions

The two equations (27) and (28) are not uncoupled anymore, since (27) contains $\hat{v}$. With explicitely-integrated non-linear terms, they can however be solved separately at each time step, provided one solves first (28) for $\hat{v}$ and then (27) for $\hat{\eta}$.

For computing the nonlinear terms and their spatial derivatives, one needs to know the velocity components $\hat{u}$ and $\hat{w}$ in the homogeneous directions at a given time by knowing $\hat{v}$ and $\hat{\eta}$. By using the definition defintion (25) of $\hat{\eta}$ and the continuity equation (22) written

in Fourier space, a 2×2 algebraic system can be written for the unknowns $\hat{u}$ and $\hat{w}$; its analytical solution reads:

$$\begin{cases} \hat{u} = \dfrac{1}{k^2}\left(i\alpha D_*(\hat{v}) - \dfrac{im}{r}\hat{\eta}\right) \\ \hat{w} = \dfrac{1}{k^2}\left(i\alpha\hat{\eta} + \dfrac{im}{r}D_*(\hat{v})\right) \end{cases} \tag{29}$$

Like in the cartesian case, this system lends itself to an analytical solution only when the variables are expanded in Fourier series.

6.4.1. Mean (shell-averaged) flow in the homogeneous directions

The preceding system (29) is singular when $k^2 = 0$. This is a consequence of having obtained Eqns. (27) and (28) through a procedure involving spatial derivatives.

Let us introduce an averaging operator over the homogeneous directions:

$$\widetilde{f} = \frac{1}{L_x}\frac{1}{L_\theta}\int_0^{L_x}\int_0^{L_\theta} f \,\mathrm{d}x r \mathrm{d}\theta$$

The space-averaged streamwise velocity $\widetilde{u} = \widetilde{u}(r,t)$ is a function of radial coordinate and time only, and in Fourier space it corresponds to the Fourier mode for $k = 0$. The same applites to the azimuthal component $\widetilde{w}$. With the present choice of the reference system, where the x axis is aligned with the mean flow, the temporal average of $\widetilde{u}$ is the streamwise mean velocity profile, whereas the temporal average of $\widetilde{w}$ will be zero (within the limits of the temporal discretization). This nevertheless allows $\widetilde{w}$ at a given time and at a given distance from the wall to be different from zero.

Two additional equations must then be written for calculating $\widetilde{u}$ and $\widetilde{w}$; they can be worked out by applying the linear, shell-average operator to the relevant components of the momentum equation:

$$\frac{\partial\widetilde{u}}{\partial t} = \frac{1}{\mathsf{Re}}D_*D_1\left(\widetilde{u}\right) - D_*\left(\widetilde{uv}\right) + f_x$$

$$\frac{\partial\widetilde{w}}{\partial t} = \frac{1}{\mathsf{Re}}D_1D_*\left(\widetilde{w}\right) - D_1\left(\widetilde{uw}\right) - \frac{2}{r}\widetilde{vw} + f_\theta$$

In these expressions, f_x and f_θ are the forcing terms needed to force the flow through the channel against the viscous resistence of the fluid. For the streamwise direction, f_x can be a given mean pressure gradient, and in the simulation the flow rate through the channel will oscillate in time around its mean value. f_x can also be a time-dependent spatially uniform the pressure gradient, to be chosen in such a way that the flow rate remains constant in time. The same distinction applies to the forcing term f_θ in the azimuthal direction.

7. CYLINDRICAL COORDINATES: THE NUMERICAL METHOD

The numerical techniques and the PLS parallel strategy employed in the cartesian case and described in §3 and §4 must be transferred to the present formulation in cylindrical coordinates without significant penalty, so that the Personal Supercomputer described in §5 can be used efficiently in the cylindrical case too. In what follows, emphasis will then be given to the differences with the cartesian case.

7.1. Spatial discretization in the homogeneous directions

In full analogy with the cartesian case, the unknown functions are expanded in truncated Fourier series in the homogeneous directions. For example the radial component v of the velocity vector is represented as:

$$v(x,\theta,r,t) = \sum_{h=-nx/2}^{+nx/2} \sum_{\ell=-n\theta/2}^{+n\theta/2} \hat{v}_{h\ell}(r,t) \mathrm{e}^{i\alpha x} \mathrm{e}^{im\theta} \tag{30}$$

where:

$$\alpha = \frac{2\pi h}{L_x} = \alpha_0 h; \qquad m = \frac{2\pi \ell}{L_\theta} = m_0 \ell$$

Here h and ℓ are integer indexes corresponding to the axial and azimuthal direction respectively, and α_0 and m_0 are the fundamental wavenumbers in these directions, defined in terms of the axial length L_x of the computational domain and its azimuthal extension L_θ, expressed in radians.

The numerical evaluation of the nonlinear terms in (27) and (28) is done following the same pseudo-spectral approach involving FFTs and the use of proper dealiasing.

7.2. Time discretization

Once the equations for $\hat{\eta}$ and $\hat{v}$ are discretized in time, as said before they are not independent anymore; they can however still be solved in a sequentialy way. In fact the evolution equation (27) for $\hat{\eta}_{h\ell}^{n+1}$ contains $\hat{v}_{h\ell}^{n+1}$, but luckily Eqn. (28) for $\hat{v}_{h\ell}^{n+1}$ does not contain $\hat{\eta}_{h\ell}^{n+1}$. The only difference with the cartesian case is that the order of solution of the two equations here matters, and Eqn. (28) must be solved then before Eqn. (27).

The two equations can be advanced in time using the same partially implicit time schemes described for the cartesian case. Now the explicit part contains the nonlinear terms plus some additional viscous curvature terms. No stability limitations have been encountered in our numerical experiments, since curvature terms contain low-order derivatives and do not reduce the time step size allowed by the time integration method. The same memory-efficient implementation of the cartesian case can be used, provided the compact finite-difference schemes can still be written in explicit form, as will be shown below.

7.3. High-accuracy compact finite difference schemes

The extension of the cartesian method described in §3.3 to obtain fourth-order accuracy over a five unevenly spaced points stencil, is not immediate: there are three main points which make the extension difficult. First, third-derivative terms are present in Eq.(28), thus preventing the possibility of finding explicit compact schemes. Second, both Eqns. (27) and (28) do contain r-dependent coefficients which are not in the innermost position. Last, Eqn. (28) for $\hat{v}$ is a fourth-order equation, but the highest differential operator is not D_4, but DD_*DD_*.

7.3.1. The third derivative

The third derivatives in Eq. (28) can be removed by using the continuity equation (22), which allows the first radial derivative of $\hat{v}$ be substituted with terms not containing radial derivatives:

$$D_*(\hat{v}) = -i\alpha\hat{u} - \frac{im}{r}\hat{w}.$$

As a consequence, some new terms will enter the part of the equation that will be integrated with the explicit time scheme (see below Eqns. (31) and (32) for their final form). Again, no problems of numerical stability have been encountered with this formulation of the explicit part.

7.3.2. The r-dependent coefficients

All the r-dependent coefficients in the middle of radial derivatives must be moved at the innermost position of the radial operators, as required by the example equation (13). This is done by applying repeated integrations by parts, i.e. repeatedly performing the following substitutions, where a indicates the generic r-dependent coefficient:

$$aD_1(f) = D_1(af) - D_1(a)f; \qquad aD_*(f) = D_*(af) - D_1(a)f.$$

In Eqn. (28), the first term which needs to be rewritten with an integration by part is:

$$\frac{\partial}{\partial t}\left[-D_1\left(\frac{1}{k^2}D_*(\hat{v})\right)\right] = \frac{\partial}{\partial t}\left[-D_1D_*\left(\frac{1}{k^2}\hat{v}\right) + D_1\left(\hat{v}D_1(\frac{1}{k^2})\right)\right].$$

In the righ-hand-side of Eqn. (28), perhaps the most complicated term is:

$$D_1\left[\frac{1}{k^2}\left(-D_*D_1D_*\hat{v}\right)\right],$$

where the continuity equation must be invoked to cancel the third derivative, and repeated integrations by parts allow the r-dependent coefficients to remain only in the innermost positions. After some algebra, the result is:

$$\begin{aligned}-D_1\left[\frac{1}{k^2}\left(D_*D_1D_*\hat{v}\right)\right] = -D_1D_*D_1D_*\left(\frac{1}{k^2}\hat{v}\right) + D_1\left[\frac{1}{r}D_2\left(\frac{1}{k^2}\hat{v}\right)\right] + \\ -2D_1D_*\left[\frac{1}{r}D_1\left(\frac{1}{k^2}\right)\hat{v}\right] + D_1\left[D_3\left(\frac{1}{k^2}\right)\hat{v}\right] - D_1\left[\frac{1}{r^2}D_1\left(\frac{1}{k^2}\right)\hat{v}\right] + \\ -3D_1D_*\left[D_1\left(\frac{1}{k^2}\right)\left(i\alpha\hat{u} + \frac{im}{r}\hat{w}\right)\right],\end{aligned}$$

where, as a result of the use of the continuity equation, the last term cannot enter the impicit part of the equations, and must be treated explicitely, similarly to what is done for the curvature terms.

The last term of eq. (28) which needs further manipulation is:

$$D_1\left[\frac{1}{k^2}\left(2\frac{im}{r^2}D_1(\hat{w})\right)\right] = 2im\left\{D_2\left(\frac{\hat{w}}{k^2r^2}\right) - D_1\left[\frac{1}{r^2}D_1\left(\frac{1}{k^2}\right)\hat{w}\right] + D_1\left(\frac{2}{k^2r^3}\hat{w}\right)\right\}.$$

The same sequnce of integration by parts must be carried out for Eqn. (27) for the radial vorticity, arriving at the following substitution:

$$2\frac{im}{r}D_1(\hat{u}) = 2im\left[D_1\left(\frac{u}{r^2}\right) + 2\frac{u}{r^3}\right].$$

The nonlinear terms (24a-c) contain radial derivatives too, and some terms therein must be integrated by parts in order to have all the coefficient in the innermost position.

This procedure leads to the final, rather long form of the equations for $\hat{v}$ and $\hat{\eta}$, which lends itself to a discretization in the radial direction with explicit compact finite differrence schemes of fourth-order accuracy over a five point stencil. It is written here for completness, without time discretization for notational simplicity:

$$\frac{\partial}{\partial t}\left[\hat{v} - D_1 D_*\left(\frac{1}{k^2}\hat{v}\right) + D_1\left(\hat{v}D_1(\frac{1}{k^2})\right)\right] =$$
$$\frac{1}{\mathsf{Re}}\left\{2D_1D_*(\hat{v}) - D_1D_*D_1D_*\left(\frac{1}{k^2}\hat{v}\right) + D_1\left[\frac{1}{r}D_2\left(\frac{1}{k^2}\right)\hat{v}\right] - 2D_1D_*\left[\frac{1}{r}D_1\left(\frac{1}{k^2}\right)\hat{v}\right] + \right.$$
$$+D_1\left[D_3\left(\frac{1}{k^2}\right)\hat{v}\right] - D_1\left[\frac{1}{r^2}D_1\left(\frac{1}{k^2}\right)\hat{v}\right] - 3D_1D_*\left[D_1\left(\frac{1}{k^2}\right)\left(i\alpha\hat{u} + \frac{im}{r}\hat{w}\right)\right] +$$
$$-2m^2D_1\left(\frac{1}{k^2r^2}\hat{v}\right) + imD_1\left[\frac{1}{k^2r^2}\hat{w}\right] + 2im\left[D_2\left(\frac{1}{k^2r^2}\hat{w}\right) - D_1\left[\frac{1}{r^2}D_1\left(\frac{1}{k^2}\right)\hat{w}\right]\right] +$$
$$\left.-k^2\hat{v} - 2\frac{im}{r^2}\hat{w}\right\} - i\alpha\left[D_1D_*\left(\frac{1}{k^2}\widehat{uv}\right) - D_1\left(D_1\left(\frac{1}{k^2}\widehat{uv}\right)\right)\right] - im\left[D_2\left(\frac{1}{rk^2}\widehat{vw}\right) +$$
$$+3D_1\left(\frac{1}{r^2k^2}\widehat{vw}\right) - D_1\left(\frac{1}{r}D_1\left(\frac{1}{k^2}\right)\widehat{vw}\right)\right] + D_1\left[\frac{1}{k^2}\left(\alpha^2\hat{u}^2 + 2\frac{\alpha m}{r}\widehat{uw} + \frac{m^2}{r^2}\hat{w}^2\right)\right] +$$
$$- i\alpha\widehat{uv} - D_1(\hat{v}^2) - \frac{im}{r}\widehat{vw} - \frac{1}{r}\hat{v}^2 + \frac{1}{r}\hat{w}^2; \quad (31)$$

$$\frac{\partial\hat{\eta}}{\partial t} = \frac{1}{\mathsf{Re}}\left\{D_1D_*(\hat{\eta}) - k^2\hat{\eta} + 2im\left[D_1\left(\frac{1}{r^2}\hat{u}\right) + 2\frac{\hat{u}}{r^3} - \frac{i\alpha}{r^2}\hat{v}\right]\right\} - im\left[\frac{i\alpha}{r}\hat{u}^2 + \right.$$
$$\left. D_1\left(\frac{\widehat{uv}}{r}\right) + \frac{2}{r^2}\widehat{uv} + \frac{im}{r^2}\widehat{uw}\right] + i\alpha\left[i\alpha\widehat{uw} + D_1\left(\widehat{vw}\right) + \frac{im}{r}\hat{w}^2 + \frac{2}{r}\widehat{vw}\right]; \quad (32)$$

Though of complicated appeareance, these equations can be solved by employing again most f the numerical tools developed for the cartesian case. As a consequence, only a few lines of the cylindrical source code are different from its cartesian counterpart.

It is important to note that this procedure introduce additional coefficients, which are function both of the radial coordinate directly and/or via the wavenumbers: one of the simplest among them is $D_1(1/k(r)^2)$. With some overhead in CPU time these coefficients can be computed on the fly during the execution of the program; alternatively, they can be precomputed once at the beginning at the expense of some memory space, if the available storage allows.

7.4. Calculation of the finite-difference coefficients

Six finite-differences groups of coefficients must be computed. Besides d_0^j, d_1^j and d_2^j we introduce the following three sets of coefficients:

$$D_*\left(f(y)\right)\big|_{y=y_j} = \sum_{i=-2}^{2} d_*^j(i) f(y_{j+i})$$

$$D_1D_*\left(f(y)\right)\big|_{y=y_j} = \sum_{i=-2}^{2} d_{1*}^j(i) f(y_{j+i})$$

$$D_1 D_* D_1 D_* \left(f(y)\right)|_{y=y_j} = \sum_{i=-2}^{2} d^j_{1*1*}(i) f(y_{j+i})$$

The actual calculation of the FD operators at fourth order accuracy on a five point stencil centered at r_j can still be perfomed in the way described in §3.3, since the only form in which the fourth derivative enters the equations is through the operator $D_1 D_* D_1 D_*$. Thus, given a set of polynomials $t_m(r)$ of increasing degree in the independent variable r, the corresponding derivative $DD_*DD_*(t)$ can be computed analitically and evaluated at $r = r_j$. A 10×10 linear system yielding the coefficients of the FD operators d^j_{1*1*} and d^j_0 follows from the condition that:

$$d^j_{1*1*}\left(t_m\right) - d_0\left(D_1 D_* D_1 D_*\left(t_m\right)\right) = 0.$$

The normalization condition (16) still gives a relation among the five coefficients d_0, so that nine additional conditions are needed, and polynomials from $m = 0$ up to $m = 8$ have to be considered. The 10×10 system cannot be decoupled into two smaller systems, and must now be solved at once. The remaining coefficients are then computed in analogy to the cartesian case.

7.5. The spatial resolution in the azimuthal direction

The cylindrical coordinate system presents the general problem that the azimuthal extension L_θ of the computational domain decreases with the radial coordinate; if the necessary spatial resolution (for example the number of Fourier modes, or the collocation points in a finite-difference calculation) is set up based on the most demanding region of the flow field, i.e. the outer wall, then the spatial resolution becomes unnecessarily high when the inner wall is approached. This not only implies a waste of computational resources, but might also induce numerical stability problems.

To overcome this difficulty, we have made the truncation of the azimuthal Fourier series a function of the radial position. Whereas in a collocation approach changing the resolution with the radial coordinate would involve multiple interpolations and numerical diffusion, in a spectral representation dropping a few Fourier modes at the high end of the spectrum is a smooth operation, which does not induce any spatially localized error.

Instead of the expansion (30), we use the following representation for the variable, e.g. v:

$$v(x,\theta,r,t) = \sum_{h=-nx/2}^{+nx/2} \sum_{\ell=-N_\theta(r)/2}^{+N_\theta(r)/2} \hat{v}_{h\ell}(r,t) \mathrm{e}^{i\alpha x} \mathrm{e}^{im\theta}$$

where, thanks to the intrinsic smoothness of the Fourier series, the number of modes in the azimuthal direction can be an arbitrary function $N_\theta(r)$ of the radial coordinate. The simplest and most natural choice for the function $N_\theta(r)$ is a linear function from a maximum $N_{\theta,max}$ at $r = \mathcal{R}_o$ down to a minimum $N_{\theta,min}$ at $r = \mathcal{R}_i$, with $N_{\theta,max}$ and $N_{\theta,min}$ being proportional to the outer and inner radii themselves so as to keep the same spatial resolution throughout the domain.

This is equivalent to assuming that the Fourier modes $\hat{v}_{h\ell}$ with $|\ell| \leq N_{\theta,min}$ are defined through the whole annular gap, i.e. for $\mathcal{R}_i \leq r \leq \mathcal{R}_o$, while any mode $\hat{v}_{h\ell}$ with $N_{\theta,min} < |\ell| \leq N_{\theta,max}$ only exists for $\overline{r}(\ell) < r < \mathcal{R}_o$, where $\overline{r}(\ell)$ is a suitable radial position, function of the index ℓ, intermediate between the two walls. These modes are assumed to become zero at the lower end of this interval, just as all modes beyond $N_{\theta,max}$ implicitly are

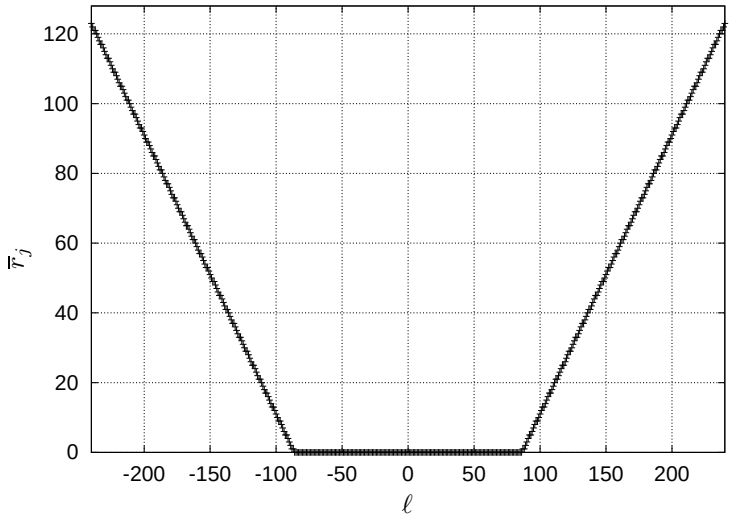

FIG. 9. XXX WRONG FIGURE!! Distribution Radial position $\overline{r}_j(\ell)$ below which the azimuthal wavenumber $m = 2\pi/L_\theta$ is assumed to be zero, as a function of the integer index ℓ. $\mathcal{R}_i = 2$, $\mathcal{R}_o = 4$, $N_{\theta,min} = 160$ and $N_{\theta,max} = 320$. This example corresponds to a radial discretization with $N_r = 128$, and a non uniform mesh with a hyperbolic tangent law.

everywhere, and the necessary boundary conditions for their governing radial differential equation are thus provided.

From the point of view of computer programming, a comb array of Fourier coefficients whose number varies with ℓ (and possibly h too, even if this feature is not presently used) has been implemented through a suitable memory management, where a two-dimensional array of pointers is used to reference variable-sized one-dimensional arrays, each of which stores all and only the nonzero coefficients in a radial line, from $r = \mathcal{R}_o$ down to $\overline{r}(\ell)$. This procedure reduces the computational cost of DNS in cylindrical geometry with significant curvature, thanks to the reduction in the number of active Fourier modes, and at the same time to avoid the numerical stability problems which could otherwise derive from an overfine resolution of the innermost region. Fig. 9 provides an example of the change of the lower dimension of the arrays as a function of the integer index ℓ corresponding to the azimuthal wavenumber. It can be seen that the lower wavenumbers, namely $|\ell| < 80$, are defined throughout the whole channel, i.e. for $j \geq 0$. Conversely the highest spanwise wavenumber $\ell = \pm 160$ is defined only very near to the outer wall at $r = \mathcal{R}_o$.

7.6. Performance

The cylindrical version of the computer code shares with its cartesian counterpart the basic structure, as well as the high computational efficiency when executed in serial or parallel mode. The differences in source code are actually very limited, allowing us to re-use most of the numerical routines. The performance evaluation made in §5.1 for the cartesian code thus applies here too, in particular concerning the properties of the PLS parallel method. For a problem of the same computational size, the CPU overhead of the cylindrical version compared to the cartesian case is approximately 40%. Pre-computing the r-dependent coefficients increases memory requirements by about 13%.

8. CONCLUSIONS

In this paper we have given a detailed description of a numerical method suitable for the parallel direct numerical simulation of incompressible wall turbulence, and capable of achieving high efficiency by using commodity hardware. The method can be used when the governing equations are written either in cartesian or in cylindrical coordinates.

The key point in its design is the choice of compact finite differences of fourth-order accuracy for the discretization of the wall-normal direction. The use of finite differences schemes, while retaining a large part of the accuracy enjoyed by spectral schemes, is crucial to the development of the parallel strategy, which exploits the locality of the FD operators to largely reduce the amount of inter-node communication. Finite differences are also key to the implementation of a memory-efficient time integration procedure, which permits a minimal storage space of 5 variables per point, compared to the commonly reported minimum of 7 variables per point. This significant saving is available in the present case too, the use of compact schemes notwithstanding, since they can be written in explicit form, leveraging the missing third derivative in the governing equations.

The formulation of the cylindrical Navier–Stokes equations in terms of radial velocity and radial vorticity has allowed us to solve them numerically with high computational efficiency, employing numerical techniques already developed for the cartesian geometry and writing a computer code which shares the basic structure with the cartesian version. The problem of the radial resolution, which varies with r in cylindrical coordinates, has been circumvented by adopting a representation of the flow variables with finite Fourier series whose number of modes depends on the radial coordinate itself. This procedure is promising for future works employing the cylindrical coordinate system.

The parallel method described in this paper, based on the pipelined solution of the linear systems (PLS) arising from the discretization of the viscous terms, achieves its best performance on systems where the number of computing nodes is significantly smaller than the number of points in the wall-normal direction. This limitation is not essential however, and it can be removed at the expense of an additional, small amount of communication in the calculation of the non-linear terms of the Navier-Stokes equations. The global transpose of the data, which constrains DNS codes to run on machines with very large networking bandwidth, is avoided anyway.

The computing effort, as well as the required memory space, can be efficiently subdivided among a number of low-cost computing nodes. Moreover, the distribution of data in wall-parallel slices allows us to exploit a particular, efficient and at the same time cost-effective connection topology, where the computing machines are connected to each other in a ring.

A dedicated system can be easily built, using commodity hardware and hence at low cost, to run a computer code based on the PLS method. Such a system grants high availability and throughput, as well as ease in expanding/upgrading. It is our opinion that the concept of Personal Supercomputer can be successful: such a specialized system, yet built with mass-market components, can be fully dedicated to a single research group or even to a single researcher, rather than being shared among multiple users through a queueing system. The smaller investment, together with additional advantages like reduced power consumption and heat production, minimal floor space occupation, etc, allows the user to have dedicated access to the machine for unlimited time, thus achieving the highest throughput.

The sole significant difference performance-wise between such a system and a real supercomputer lies in the networking hardware, which offers significantly larger bandwidth and better latency characteristics in the latter case. However the negative effects of this

difference are not felt when the present parallel algorithm is employed, since the need for a large amount of communication is removed *a priori*, thanks to the algorithm itself.

ACKNOWLEDGMENT

Financial support from ASI and MURST for the years 1999 and 2000 is acknowledged. The Authors whish to thank ing. Patrick Morandi, ing. Fabio Brenna for their help in part of the work concerning cylindrical coordinates.

REFERENCES

1. F. P. Bertolotti, T. Herbert, and P. R. Spalart. Linear and nonlinear stability of the Blasius boundary layer. *J. Fluid Mech.*, 242:441–474, 2002.
2. G. Ciaccio and G. Chiola. Porting MPICH ADI on GAMMA with Flow Control. In *Midwest Workshop on Parallel Processing, Kent, Ohio*, 1999.
3. J.C. del Álamo and J. Jiménez. Spectra of the very large anisotropic scales in turbulent channels. *Phys. Fluids*, 15(6):L41–L44, 2003.
4. J. J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report). (CS-89-85), 2004.
5. J.G.M. Eggels, F. Unger, M.H. Weiss, J. Westerweel, R.J. Adrian, R. Fiedrich, and F.T.M. Nieuwstadt. Fully developed turbulent pipe flow: a comparison between direct numerical simulation and experiment. *Journal of Fluid Mechanics*, 268:175–209, 1994.
6. A. Günther, D.V. Papavassilou, M.D. Warholic, and T.J. Hanratty. Turbulent flow in a channel at a low Reynolds number. *Exp. Fluids*, 25:503–511, 1998.
7. P.H. Hoffmann, K.C. Muck, and P. Bradshaw. The effect of concave surface curvature on turbulent boundary layers. *Journal of Fluid mechanics*, 161:371–403, 1985.
8. J. Jiménez. Computing high-Reynolds-number turbulence: will simulations ever replace experiments? *J. Turbulence*, 4:22, 2003.
9. J. Kim. Control of turbulent boundary layers. *Phys. Fluids*, 15(5):1093–1105, 2003.
10. J. Kim, P. Moin, and R. Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *J. Fluid Mech.*, 177:133–166, 1987.
11. S.K. Lele. Compact Finite Difference Schemes with Spectral-like Resolution. *J. Comp. Phys.*, 103:16–42, 1992.
12. P. Luchini and M. Quadrio. A low-cost parallel implementation of direct numerical simulation of wall turbulence. *Submitted to J. Comp. Phys.*, 2004.
13. B. Ma, Z. Zhang, F.T.M. Nieuwstadt, and C.W.H. van Doorne. On the spatial evolution of a wall-imposed periodic disturbance in pipe Poiseuille flow at Re=3000. Part 1. Subcritical disturbance. *J. Fluid Mech.*, 398:181–224, 1999.
14. K. Mahesh. A Family of High Order Finite Difference Schemes with Good Spectral Resolution. *J. Comp. Phys.*, 145(1):332–358, 1998.
15. M. Manna and A. Vacca. An efficient method for the solution of the incompressible Navier-Stokes equations in cylindrical geometries. *J. Comp. Phys.*, 151:563–584, 1999.
16. P. Moin and K. Mahesh. Direct numerical simulation: A tool in turbulence research. *Ann. Rev. Fluid Mech.*, 30:539–578, 1998.
17. R. Moser, J. Kim, and N.N. Mansour. Direct numerical simulation of turbulent channel flow up to $Re_\theta = 590$. *Phys. Fluids*, 11(4):943–945, 1999.
18. R. D. Moser and P. Moin. The effects of curvature in wall-bounded turbulent flows. *J. Fluid Mech.*, 175:479–510, 1987.
19. R.D. Moser, P. Moin, and A. Leonard. A spectral numerical method for the Navier–Stokes equations with application to Taylor-Couette flow. *Journal of Computational Physics*, 52:524–544, 1983.
20. M. Nagata and N. Kasagi. Spatio-temporal evolution of coherent vortices in wall turbulence with streamwise curvature. *J. Turbulence*, 2004.
21. J. C. Neves, P. Moin, and R. D. Moser. Effects of convex transverse curvature on wall-bounded turbulence. Part 1. The velocity and vorticity. *Journal of Fluid Mechanics*, 272:349–381, 1994.
22. P. Orlandi and M. Fatica. Direct simulations of turbulent flow in a pipe rotating about its axis. *J. Fluid Mech.*, 343:43–72, 1997.

23. R. B. Pelz. The Parallel Fourier Pseudospectral Method. 92:296–312, 1991.

24. A. Pozzi. *Application of Padé's Approximation Theory in Fluid Dynamics*. Advances in Mathematics for Applied Sciences. World Scientific, 1994.

25. M. Quadrio and P. Luchini. Direct numerical simulation of the turbulent flow in a pipe with annular cross-section. *Eur. J. Mech. B / Fluids*, 21:413–427, 2002.

26. M. Quadrio and P. Luchini. Integral time-space scales in turbulent wall flows. *Phys. Fluids*, 15(8):2219–2227, 2003.

27. O. Reynolds. An experimental investigation on the circumstances which determine whether the motion of water shall be direct or sinuous, and the law of resistance in parallel channels. *Proc. R. Soc. London A*, 35:84, 1883.

28. I. Shapiro, L. Shtilman, and A. Tumin. On stability of flow in an annular channel. *Phys. Fluids*, 11(10):2984–2992, 1999.

29. M. Skote. *Studies of turbulent boundary layer flow through direct numerical simulation*. PhD thesis, Royal Institute of Technology Department of Mechanics, 2001.

30. L.H. Thomas. The stability of plane Poiseuille flow. *Phys. Rev.*, 91(4):780–783, 1953.

APPENDIX A

Design, installation and configuration of a Personal Supercomputer

The focus of the numerical method presented in this paper has been towards its use on a low-cost computing machine. Here we describe how to install and configure such a machine, schematically illustrated in figure 4, and made by a certain number of desktop Personal Computers.

The main ingredient to the machine is the *computing node*. This is a general-purpose PC, where the Unix operating system is installed. We have always used the Debian/Gnu Linux distribution, but any Unix flavour will do the job. To decide the most cost-effective configuration of a node, two main options have to be evaluated. The first one is the size of the case: 1U cases require much more expensive hardware components, but allow a significant saving in floor space compared to the standard desktop cases (tower or minitower). The second one is to decide whether one or more CPU are installed. With the present market situation, and given the SMP parallel speedup achievable, the use of a second CPU is advantageous in terms of ratio cost/benefit. This may change in the future, but presently the cost increase due to the second CPU (and the more expensive SMP motherboard) is smaller than the SMP speedup achievable (which is in the range 1.7–1.8, independent upon the number of nodes). This leaves us with the added advantage that the number of nodes for a given budget is smaller, thus allowing a larger fraction of the peak computing power to be reached (see for example fig. 5).

The choice of the *CPU type* is, by and large, a matter of taste, since commodity processors are very near each other when evaluated with the ratio between floating-point performance and unit price. The choice of the *CPU speed* can be conveniently made by looking for the kink in the price-performance curve at the time of buying: the most recent, faster processors are far more expensive, since peak power is a premium in general. In the present context, however, we can buy a certain amount of computing power by deciding the speed of the computing nodes and the number of nodes. More precise indications cannot be given, given the etreme volatily of the market situation.

Particular attention must be paid to the mainboard specifications, (like bus speed, bandwidth, etc), since memory access is the real limiting factor of the simulation.

The amount of *disk* space to be installed on each machine is not relevant, one could also go with diskless nodes, or with nodes where the system is run from a live CD, and certainly there is no need of SCSI or high-performance disks. In general, cheap and relatively small

EIDE disks installed on every machine has been for us a perfect choice, as long as a single disk is able to store the whole dataset pertaining to the same project. The machines need one disk mounted in such a way as to be shared among the other nodes with a networked file system, like NFS or the like. Performance of the network-mounted disk too is typically not critical, since during the computations the disk is accessed only to read the initial file, to write flow fields periodically and to (over)write the restart file. Our strategy has been most often to store datasets on the network-mounted disk, and to move them to the local (empty) disk of another node as soon as the free space reaches 30% or so of the capacity.

For special cases where a large database has to be built, or when the number of nodes becomes very large, poor NFS performance may become a bottleneck. To increase performance, as well as to avoid cache problems, we mount the NFS volumes with the following options in the `/etc/fstab` file: `soft,rsize=8192,wsize=8192,noac`. In extreme cases, another viable option is to use a distributed i/o, i.e. each node writes to its local disk. This is very easy and gives maximum performance, but it requires a distributed post-processing of the databases, or the availability of a very large storage space for centralized post-processing.

The amount of *memory* to install on each node must not be very large, since the global memory requirements of the simulations are subdivided among the nodes. In all of our machines we have installed less than 1GB of memory. The optimal amount can be estimated from the size of the largest simulation affordable, which in turn depends mainly on the CPU speed. As a rule of thumb, today (late 2004) the memory in MBytes can be comparable or smaller than the CPU speed in GHz. In case of doubt, it is advisable to install a smaller amount of memory, and subsequently add one stick of RAM to each node when needed. It may be rewarding to fine-tune the memory configuration (acces time, latency, etc) in the BIOS, since such large simulations are limited by memory bandwidth and in our experience the computing time has been found to depend linearly upon memory performance.

Each node must have two *network cards*. They are used to link the nodes each other in a ring-like topology, without switch, as illustrated in Fig.4 for the case of 4 nodes. Getting rid of the switch is something that should not be underestimated, since the switch can result in a serious degradation in performance. Any Linux-supported network card will do the job, but a preliminary check for full support even under heavy load in SMP mode will potentially save troubles. Today Gigabit-Ethernet cards with a speed of 1000Mbit/s are commonplace, and they can be efficently used at their full bandwidth even when the LAN (for example the departmental network) is cabled at lower speed, since the machines communicate directly.

One of the nodes may have a slightly different configuration. This machine is typically used with remote access from the network, so that a third NIC is needed on this special machine. It may also be useful that it is assigned a public IP number with a registered hostname. The other nodes can be set up to live on a private network, with IP numbers for example in the 192.168.0.xx range.

The main node (see Fig.4) is configured as a standard, standalone machine, with one of its 3 interfaces, say eth2, managing the connection to the local public network. It has a registered hostname, say `publichost`, corresponding to its public IP number assigned from the relevant authority. Any machine, including `publichost`, is assigned a private IP number, and a non-registered hostname which is made available to the others by copying on any node the same, complete `/etc/hosts` file. It may be handy to assign the private IPs serially, and to use a consistent host name: for example one machine may have IP

192.168.0.02 and hostname `host02`. The same IP can be assigned to each of the two network interfaces.

```
 # Example /etc/hosts file
192.168.0.0 host00
192.168.0.1 host01
192.168.0.2 host02
192.168.0.3 host03
```

On each machine a routing table has to be set up. The minimal requirement is that each machine is connected *point-to-point* with the two neighbours. It is useful moreover that each machine can reach any other machine, though with a series of hops through the intermediate hosts. This is used when the program is started, and when results are written to disk. The routing table for machine `host02` of Fig.4 is set up at boot by a script that (for the Debian/Gnu Linux operating system) looks as follows:

```
 # /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
auto eth0
iface eth0 inet static
address 192.168.0.102
netmask 255.255.255.255
pointopoint 192.168.0.101
up route add -host 192.168.0.100 gw 192.168.0.101
iface eth1 inet static
address 192.168.0.102
netmask 255.255.255.255
pointopoint 192.168.0.103
```

Of course the special machine with three interfaces will have `eth0` and `eth1` configured with the private IP 192.168.0.0, and the third interface configured with the registered IP, with a default gateway on the local network.

Several variants of this simple example are possible, the larger complexity being balanced by a possibly large number of computing nodes to be configured. It is also possible to automate the process of creating the setup scripts by writing a short shell script.

With perhaps the exception of `publichost`, the other nodes are identical each other, if exception is made for the two files `/etc/hostname` and `/etc/network/interfaces`. This means that also automated processes of installing the system on the nodes can be used when the number of nodes is large. We use the software suite `systemimager`. The packages that have to be installed on the nodes are minimal. Bejond those needed for the machine to boot and mount the NFS volume, we essentially add only the capability of issuing and receiving SSH (or RSH) commands. For a small numer of nodes, each machine can first have a standard installation, then the two files can be modified by hand and the machine properly connected and rebooted.

Once the machine is up and running, it may be useful to have available a simple script that takes a generic command as an argument, and executes it machine-wide, i.e. on each node. The script can be run from `publichost` or, depending on the installation policy, from an external machine, and should look as follows:

```
 #!/bin/bash
COMMAND="$@"
```

```
for N in 0 1 2 3
do
echo "host0$N::"
ssh host0$N $COMMAND
done
```

Jobs can be run either from the externally-accessible node `publichost`, or from any other host, possibly being not too far from the machine and with a decent network connection to it. Of course the internal nodes too may work as submitting machines. We prefer however not to access these nodes directly, since the implicit load-balancing of our SMP parallel strategy, which is completely left to the kernel, assumes that machines are idle.

We run our code through another simple script, so that the only requirement is that the nodes can be accessed through `ssh`. Our example script takes two arguments, `$1` is the working directory and `$2` is the name of the executable to be launched. The commands contained between "" are what is needed to run the DNS code. The script in its simplest form is as follows:

```
 ssh host03 "cd ~/$1; nohup ./$2 1 4 >& /dev/null &"
ssh host02 "cd ~/$1; nohup ./$2 2 4 host03 >& /dev/null &"
ssh host01 "cd ~/$1; nohup ./$2 3 4 host02 >& /dev/null &"
ssh host00 "cd ~/$1; nohup ./$2 4 4 host01 >& /dev/null &"
```

To have this script run a simple command, it is useful that the nodes can execute a command received via ssh from the submitting machines without requewst for password authentication. This requires the public signature of the submitting machine, contained in the file `$HOME/.ssh/id_dsa.pub`, to be copied into the file `$HOME/.ssh/authorized_keys` of each node. This operation is very simple, inasmuch the home directory of the user(s) is network-mounted. Thus, once a user is created in any node and its home directory resides on a network-mounted volume, the public key has to be copied only once.

APPENDIX B

Structure of the computer code (cartesian version)

In the following the source code listing of the cartesian program is reported, to the aim of giving a general idea of its structure. Only the serial version is discussed: the listing of the parallel version, as well as the source for the cylindrical program, can be obtained by contacting the Authors. The differences between the serial and the parallel codes are however minimal, and at the program-source level they concern only the strategy for reading/writing results on disk. The inter-node communication is hidden behind the routine which solves the linear systems, then there is no need to expose such details to the user.

The program is written in CPL, a programming language (with related compiler) written by Paolo Luchini. In a simple procedure, which is made transparent to the user by the invokation of a `make`-like command, the CPL source is subjected first to a preprocessing pass to generate an ANSI-C source, which is then compiled by any ANSI-compliant C compiler. The meaning of CPL statements, keywords and programming structures can be easily understood, since they are modeled after the most common programming languages. The name of the variables in the source closely follow the symbolic names used in this paper: for exaple `oldrhs.eta` stores the value of the right-hand-side of eq. (4) for the wall-normal vorticity component at the previous time level.

```
USE rtchecks
USE fft
USE rbmat

FILE dati=OPEN("dns.in")
ARRAY(0..21) OF CHAR input_file
INTEGER CONSTANT nx,ny,nz
REAL CONSTANT alfa0,beta0,htcoeff,ymax=2,ymin=0,t_max,dt_field,dt_save
REAL ni, meanpx=0, meanpz=0, meanflowx=0, meanflowz=0, deltat, time
READ BY NAME FROM dati input_file
READ BY NAME FROM dati nx,ny,nz,alfa0,beta0,htcoeff,ni
DO WHILE READ BY NAME FROM dati meanpx OR meanflowx OR meanpz OR meanflowz
READ BY NAME FROM dati deltat, t_max, dt_field, dt_save
WRITE BY NAME nx,ny,nz,2*PI/alfa0,2*PI/beta0,ni; ni=1/ni
WRITE BY NAME deltat,t_max,dt_field,dt_save
REAL y(-1..ny+1) !; DO y(i)=ymax*i/ny FOR ALL i
DO y(i)=ymin+tanh(htcoeff*(2*i/ny-1))/tanh(htcoeff) + 1 FOR ALL i

STRUCTURE[ARRAY(-2..2) OF REAL d0,d1,d2,d4] derivatives(1..ny-1)
ARRAY(-2..2) OF REAL d040,d140,d14m1, d04n,d14n,d24n,d14np1
MODULE setup_derivatives
REAL M(0..4,0..4),t(0..4)
LOOP FOR iy=1 TO ny-1 WITH derivatives(iy)
  DO M(i,j)=(y(iy-2+j)-y(iy))**(4-i) FOR ALL i,j; LUdecomp M
  t=0; t(0)=24
  d4(-2+(*))=M\t
  DO M(i,j)=(5-i)*(6-i)*(7-i)*(8-i)*(y(iy-2+j)-y(iy))**(4-i) FOR ALL i,j; LUdecomp M
  DO t(i)=SUM {d4(j)*(y(iy+j)-y(iy))**(8-i)} FOR ALL j FOR ALL i
  d0(-2+(*))=M\t
  DO M(i,j)=(y(iy-2+j)-y(iy))**(4-i) FOR ALL i,j; LUdecomp M
  t=0; DO t(i)=SUM d0(j)*(4-i)*(3-i)*(y(iy+j)-y(iy))**(2-i) FOR ALL j FOR i=0 TO 2
  d2(-2+(*))=M\t
  t=0; DO t(i)=SUM d0(j)*(4-i)*(y(iy+j)-y(iy))**(3-i) FOR ALL j FOR i=0 TO 3
  d1(-2+(*))=M\t
REPEAT
DO M(i,j)=(y(-1+j)-y(0))**(4-i) FOR ALL i,j; LUdecomp M
t=0; t(3)=1; d140(-2+(*))=M\t
DO M(i,j)=(y(-1+j)-y(-1))**(4-i) FOR ALL i,j; LUdecomp M
t=0; t(3)=1; d14m1(-2+(*))=M\t
d04n=0; d04n(1)=1; d040=0; d040(-1)=1
DO M(i,j)=(y(ny-3+j)-y(ny))**(4-i) FOR ALL i,j; LUdecomp M
t=0; t(3)=1; d14n(-2+(*))=M\t
t=0; t(2)=2; d24n(-2+(*))=M\t
DO M(i,j)=(y(ny-3+j)-y(ny+1))**(4-i) FOR ALL i,j; LUdecomp M
t=0; t(3)=1; d14np1(-2+(*))=M\t
END setup_derivatives

INLINE REAL FUNCTION D0(REAL f(*)) = d0(-2)*f(-2)+d0(-1)*f(-1)+d0(0)*f(0)+d0(1)*f(1)+d0(2)*f(2)
INLINE REAL FUNCTION D1(REAL f(*)) = d1(-2)*f(-2)+d1(-1)*f(-1)+d1(0)*f(0)+d1(1)*f(1)+d1(2)*f(2)
INLINE REAL FUNCTION D2(REAL f(*)) = d2(-2)*f(-2)+d2(-1)*f(-1)+d2(0)*f(0)+d2(1)*f(1)+d2(2)*f(2)
INLINE REAL FUNCTION D4(REAL f(*)) = d4(-2)*f(-2)+d4(-1)*f(-1)+d4(0)*f(0)+d4(1)*f(1)+d4(2)*f(2)
INLINE COMPLEX FUNCTION D0(COMPLEX f(*))=D0(f.REAL)+I*D0(f.IMAG)
INLINE COMPLEX FUNCTION D1(COMPLEX f(*))=D1(f.REAL)+I*D1(f.IMAG)
INLINE COMPLEX FUNCTION D2(COMPLEX f(*))=D2(f.REAL)+I*D2(f.IMAG)
INLINE COMPLEX FUNCTION D4(COMPLEX f(*))=D4(f.REAL)+I*D4(f.IMAG)

REAL FUNCTION yintegr(REAL f(*))
  RESULT=0
  LOOP FOR iy=1 TO ny-1 BY 2
   yp1=y(iy+1)-y(iy); ym1=y(iy-1)-y(iy)
   a1=-1/3*ym1+1/6*yp1+1/6*yp1*yp1/ym1
   a3=+1/3*yp1-1/6*ym1-1/6*ym1*ym1/yp1
   a2=yp1-ym1-a1-a3
   RESULT=+a1*f(iy-1) + a2*f(iy) + a3*f(iy+1)
  REPEAT
END yintegr

VELOCITY=STRUCTURE(COMPLEX u,v,w)
MOMFLUX=STRUCTURE(COMPLEX uu,uv,vv,vw,ww,uw)
INTEGER nxd=3*nx DIV 2 - 1; DO INC nxd UNTIL FFTfit(nxd)
INTEGER nzd=3*nz - 1; DO INC nzd UNTIL FFTfit(nzd)
ARRAY(0..nxd-1,0..nzd-1) OF VELOCITY Vd
ARRAY(0..nxd-1,0..nzd-1) OF MOMFLUX VVd

SUBROUTINE convolutions(ARRAY(*,*) OF VELOCITY V; POINTER TO ARRAY(*,*) OF MOMFLUX VV)
  Vd=0
  LOOP FOR ix=0 TO nx
    DO Vd(ix,iz)=V(ix,iz) FOR iz=0 TO nz
    DO Vd(ix,nzd+iz)=V(ix,iz) FOR iz=-nz TO -1
    WITH Vd(ix,*): IFT(u); IFT(v); IFT(w)
  REPEAT LOOP
  DO WITH Vd(*,iz): RFT(u); RFT(v); RFT(w); FOR ALL iz
  DO WITH Vd(ix,iz), VVd(ix,iz)
    uu.REAL=u.REAL*u.REAL; uu.IMAG=u.IMAG*u.IMAG
    uv.REAL=u.REAL*v.REAL; uv.IMAG=u.IMAG*v.IMAG
    vv.REAL=v.REAL*v.REAL; vv.IMAG=v.IMAG*v.IMAG
    vw.REAL=v.REAL*w.REAL; vw.IMAG=v.IMAG*w.IMAG
    ww.REAL=w.REAL*w.REAL; ww.IMAG=w.IMAG*w.IMAG
    uw.REAL=u.REAL*w.REAL; uw.IMAG=u.IMAG*w.IMAG
  FOR ALL ix,iz
  DO WITH VVd(*,iz): HFT(uu); HFT(uv); HFT(vv); HFT(vw); HFT(ww); HFT(uw) FOR ALL iz
  LOOP FOR ix=0 TO nx
    WITH VVd(ix,*): FFT(uu); FFT(uv); FFT(vv); FFT(vw); FFT(ww); FFT(uw)
    DO VV(ix,iz)=VVd(ix,iz) FOR iz=0 TO nz
    DO VV(ix,iz)=VVd(ix,nzd+iz) FOR iz=-nz TO -1
  REPEAT LOOP
END convolutions

maxtimelevels=1
rhstype=ARRAY(0..nx,-nz..nz) OF STRUCTURE(COMPLEX eta,D2v)
ARRAY(0..nx,-nz..nz,-1..ny+1) OF VELOCITY V=0
ARRAY(1..maxtimelevels,1..ny-1) OF rhstype oldrhs=0
ARRAY(-2..0) OF POINTER TO rhstype newrhs
DO newrhs(i) = NEW rhstype FOR ALL i
MOMFLUX VV(0..nx,-nz..nz,-2..2)

!READ BINARY FROM input_file V, oldrhs
DO WITH V(0,0,iy): u.REAL=1-[1-y(iy)]^2 FOR ALL iy

INLINE FUNCTION OS(INTEGER iy,i)=ni*[d4(i)-2*k2*d2(i)+k2*k2*d0(i)]
INLINE FUNCTION SQ(INTEGER iy,i)=ni*[d2(i)-k2*d0(i)]
SUBROUTINE buildrhs[SUBROUTINE(COMPLEX rhs^,old^(*),unknown,implicit_part,explicit_part) timescheme]
DO convolutions(V(*,*,iy),VV(*,*,iy)) FOR iy=-1 TO 2
LOOP FOR iy=1 TO ny-1
  DO VV(ix,iz,i)=VV(ix,iz,i+1) FOR ALL ix,iz AND i=-2 TO 1
  convolutions(V(*,*,iy+2),VV(*,*,2))
  WITH derivatives(iy) LOOP FOR ALL ix AND ALL iz
    ialfa=I*alfa0*ix; ibeta=I*beta0*iz
    k2=(alfa0*ix)**2+(beta0*iz)**2
    WITH VV(ix,iz,*), V(ix,iz,iy+(*)):
    rhsu=-ialfa*D0(uu)-D1(uv)-ibeta*D0(uw)
    rhsv=-ialfa*D0(uv)-D1(vv)-ibeta*D0(vw)
    rhsw=-ialfa*D0(uw)-D1(vw)-ibeta*D0(ww)
    D2vimpl = SUM OS(iy,i)*v(i) FOR i=-2 TO 2
    timescheme{newrhs(0,ix,iz).D2v,oldrhs(*,iy,ix,iz).D2v,D2(v)-k2*D0(v),
               D2vimpl,
               ialfa*[ialfa*D1(uu)+D2(uv)+ibeta*D1(uw)]+
               ibeta*[ialfa*D1(uw)+D2(vw)+ibeta*D1(ww)]-k2*rhsv}
    IF ix=0 AND iz=0 THEN
      ! u media conservata in eta.REAL e w media in eta.IMAG
      timescheme{newrhs(0,0,0).eta,oldrhs(*,iy,0,0).eta,D0(u.REAL)+D0(w.REAL)*I,
                 ni*D2(u.REAL)+ni*D2(w.REAL)*I,
                 rhsu.REAL+meanpx+[rhsw.REAL+meanpz]*I}
    ELSE
      etaimpl=SUM SQ(iy,i)*[ibeta*u(i)-ialfa*w(i)] FOR i=-2 TO 2
      timescheme{newrhs(0,ix,iz).eta,oldrhs(*,iy,ix,iz).eta,ibeta*D0(u)-ialfa*D0(w),
                 etaimpl,
                 ibeta*rhsu-ialfa*rhsw}
```

```
    END IF
    V(ix,iz,iy-2).u=newrhs(-2,ix,iz).eta; V(ix,iz,iy-2).v=newrhs(-2,ix,iz).D2v
  REPEAT LOOP
  temp=newrhs(-2); newrhs(-2)=newrhs(-1); newrhs(-1)=newrhs(0) ;newrhs(0)=temp
REPEAT LOOP
DO V(ix,iz,ny+i).u=newrhs(i,ix,iz).eta; V(ix,iz,ny+i).v=newrhs(i,ix,iz).D2v
  FOR ALL ix,iz AND i=-2 TO -1
END buildrhs

ARRAY(1..ny-1,-2..2) OF REAL D0mat, etamat, D2vmat
D0mat=derivatives.d0; LUdecomp D0mat
SUBROUTINE deriv(ARRAY(*) OF REAL f0,f1^)
  f1(0)=SUM d140(i)*f0(1+i) FOR i=-2 TO 2
  f1(-1)=SUM d14m1(i)*f0(1+i) FOR i=-2 TO 2
  f1(ny)=SUM d14n(i)*f0(ny-1+i) FOR i=-2 TO 2
  f1(ny+1)=SUM d14np1(i)*f0(ny-1+i) FOR i=-2 TO 2
  DO WITH derivatives(i) f1(i)=D1(f0(i+(*))) FOR i=1 TO ny-1
  WITH derivatives(1): f1(1)=~-(d0(-1)*f1(0)+d0(-2)*f1(-1))
  WITH derivatives(2): f1(2)=~-d0(-2)*f1(0)
  WITH derivatives(ny-1): f1(ny-1)=~-(d0(1)*f1(ny)+d0(2)*f1(ny+1))
  WITH derivatives(ny-2): f1(ny-2)=~-d0(2)*f1(ny)
  f1(1..ny-1)=D0mat\f1(1..ny-1)
END deriv

ARRAY(-2..2) OF REAL v0bc,v0m1bc,vnbc,vnp1bc,eta0bc,eta0m1bc,etanbc,etanp1bc
v0bc=d040; v0m1bc=d140; eta0bc=d040
vnbc=d04n; vnp1bc=d14n; etanbc=d04n
etanp1bc=derivatives(ny-1).d4
eta0m1bc=derivatives(1).d4
DO v0bc(i)=~-v0bc(-2)*v0m1bc(i)/v0m1bc(-2) FOR i= -1 TO 2
DO vnbc(i)=~-vnbc(2)*vnp1bc(i)/vnp1bc(2) FOR i= -2 TO 1
DO eta0bc(i)=~-eta0bc(-2)*eta0m1bc(i)/eta0m1bc(-2) FOR i= -1 TO 2
DO etanbc(i)=~-etanbc(2)*etanp1bc(i)/etanp1bc(2) FOR i= -2 TO 1
SUBROUTINE applybc_0(ARRAY(*) OF REAL eq^(*); ARRAY(*) OF REAL bc0,bc0m1;
                     COMPLEX rhs^(*), rhs0, rhs0m1)
  DO eq(1,i)=~-eq(1,-2)*bc0m1(i)/bc0m1(-2) FOR i=-1 TO 2
  DO eq(1,i)=~-eq(1,-1)*bc0(i)/bc0(-1) FOR i=0 TO 2
  DO eq(2,i-1)=~-eq(2,-2)*bc0(i)/bc0(-1) FOR i=0 TO 2
  rhs(1)=~-eq(1,-2)*rhs0m1/bc0m1(-2)
  rhs(1)=~-eq(1,-1)*rhs0/bc0(-1)
  rhs(2)=~-eq(2,-2)*rhs0/bc0(-1)
END applybc_0
SUBROUTINE applybc_n(ARRAY(*) OF REAL eq^(*); ARRAY(*) OF REAL bcn,bcnp1;
                     COMPLEX rhs^(*), rhsn, rhsnp1)
  DO eq(ny-1,i)=~-eq(ny-1,2)*bcnp1(i)/bcnp1(2) FOR i=-2 TO 1
  DO eq(ny-1,i)=~-eq(ny-1,1)*bcn(i)/bcn(1) FOR i=-2 TO 0
  DO eq(ny-2,i+1)=~-eq(ny-2,2)*bcn(i)/bcn(1) FOR i=-2 TO 0
  rhs(ny-1)=~-eq(ny-1,2)*rhsnp1/bcnp1(2)
  rhs(ny-1)=~-eq(ny-1,1)*rhsn/bcn(1)
  rhs(ny-2)=~-eq(ny-2,2)*rhsn/bcn(1)
END applybc_n

SUBROUTINE linsolve(REAL lambda)
COMPLEX A0, B0, An, Bn
LOOP FOR ALL ix,iz
  A0=0;An=A0; B0=0; Bn=0
  A0=~-v0bc(-2)*B0/v0m1bc(-2); An=~-vnbc(2)*Bn/vnp1bc(2)

  ialfa=I*alfa0*ix; ibeta=I*beta0*iz
  k2=(alfa0*ix)**2+(beta0*iz)**2
  LOOP FOR ALL iy,i WITH derivatives(iy)
    D2vmat(iy,i)=lambda*[d2(i)-k2*d0(i)]-OS(iy,i)
    etamat(iy,i)=lambda*d0(i)-SQ(iy,i)
  REPEAT
  ! condizioni al contorno
  applybc_0(D2vmat,v0bc,v0m1bc,V(ix,iz,*).v,A0,B0)
  applybc_n(D2vmat,vnbc,vnp1bc,V(ix,iz,*).v,An,Bn)
  applybc_0(etamat,eta0bc,eta0m1bc,V(ix,iz,*).u,0,0)
  applybc_n(etamat,etanbc,etanp1bc,V(ix,iz,*).u,0,0)
  LUdecomp D2vmat; LUdecomp etamat
  WITH V(ix,iz,*):
  v.REAL=D2vmat\v.REAL; v.IMAG=D2vmat\v.IMAG
```

```
  v(0)=(A0-SUM v(1+i)*v0bc(i) FOR i=0 TO 2)/v0bc(-1)
  v(-1)=(B0-SUM v(1+i)*v0m1bc(i) FOR i=-1 TO 2)/v0m1bc(-2)
  v(ny)=(An-SUM v(ny-1+i)*vnbc(i) FOR i=-2 TO 0)/vnbc(1)
  v(ny+1)=(Bn-SUM v(ny-1+i)*vnp1bc(i) FOR i=-2 TO 1)/vnp1bc(2)
  u.REAL=etamat\u.REAL; u.IMAG=etamat\u.IMAG
  u(0) =-(SUM u(1+i)*eta0bc(i) FOR i=0 TO 2)/eta0bc(-1)
  u(-1)=-(SUM u(1+i)*eta0m1bc(i) FOR i=-1 TO 2)/eta0m1bc(-2)
  u(ny)=-(SUM u(ny-1+i)*etanbc(i) FOR i=-2 TO 0)/etanbc(1)
  u(ny+1)=-(SUM u(ny-1+i)*etanp1bc(i) FOR i=-2 TO 1)/etanp1bc(2)
  IF ix=0 AND iz=0 THEN
    IF ABS(meanflowx)>1E-10 THEN
      REAL ucor(-1..ny+1); DO ucor(iy)=1 FOR ALL iy
      ucor=etamat\ucor
      ucor(0) =-(SUM ucor(1+i)*eta0bc(i) FOR i=0 TO 2)/eta0bc(-1)
      ucor(-1)=-(SUM ucor(1+i)*eta0m1bc(i) FOR i=-1 TO 2)/eta0m1bc(-2)
      ucor(ny)=-(SUM ucor(ny-1+i)*etanbc(i) FOR i=-2 TO 0)/etanbc(1)
      ucor(ny+1)=-(SUM ucor(ny-1+i)*etanp1bc(i) FOR i=-2 TO 1)/etanp1bc(2)
      V(0,0,*).u.REAL=~+(meanflowx-yintegr(V(0,0,*).u.REAL))/yintegr(ucor)*ucor
      V(0,0,*).w.REAL=~+(meanflowz-yintegr(V(0,0,*).w.REAL))/yintegr(ucor)*ucor
    END IF
  ELSE
    deriv(v.REAL,w.REAL)
    deriv(v.IMAG,w.IMAG)
    DO temp=(ialfa*w(iy)-ibeta*u(iy))/k2
      w(iy)=(ibeta*w(iy)+ialfa*u(iy))/k2
      u(iy)=temp
    FOR iy=-1 TO ny+1
  END IF
REPEAT
END linsolve

SUBROUTINE simple(COMPLEX rhs^,old^(*),unkn,impl,expl)
  rhs=unkn/deltat+expl
END simple
REAL CONSTANT simple_coeff=1
SUBROUTINE CN_AB(COMPLEX rhs^,old^(*),unkn,impl,expl)
  rhs=2/deltat*unkn+impl+3*expl-old(1)
  old(1)=expl
END CN_AB
CONSTANT INTEGER CN_AB_coeff=2

INTEGER cont=0
LOOP timeloop WHILE time < t_max-deltat/2
!  buildrhs(simple);  linsolve(simple_coeff/deltat)
  buildrhs(CN_AB);   linsolve(CN_AB_coeff/deltat)
  time=~+deltat

  WRITE              time,SUM d140(i)*V(0,0,1+i).u.REAL FOR i=-2 TO 2,
                          -SUM d14n(i)*V(0,0,ny-1+i).u.REAL FOR i=-2 TO 2
  IF FLOOR(time / dt_field) > FLOOR((time-deltat) / dt_field) THEN
    cont=~+1; ARRAY(0..20) OF CHAR field_name
    field_name = WRITE("field"cont".dat"); FILE field_file = CREATE(field_name)
    LOOP FOR iy=LO TO HI
      LOOP FOR ix=LO TO HI AND iz=LO TO HI WITH V(ix,iy,iz)
        ialfa=I*alfa0*ix; ibeta=I*beta0*iz
        WRITE BINARY TO field_file v, ibeta*u-ialfa*w
      REPEAT LOOP
      WRITE BINARY TO field_file V(0,0,iy).u.REAL, V(0,0,iy).w
    REPEAT LOOP
  END IF

  IF FLOOR(time / dt_save) > FLOOR((time-deltat) / dt_save) THEN
    WRITE BINARY TO "dati.out" V, oldrhs
  END IF
REPEAT timeloop
```

The main parts of the code are as follows. There is first an introductory section, where input data are read from the file `dns.in` to describe the simulation: parameters to define the spatial discretization, and parameters specific to the simulation strategy (time step size `deltat`, total integration time `t_max`, etc). This section is for lines 5–17. Note that the variables `nx` and `nz` used in the program correspond to half the number of Fourier modes, since the expansions go from `-nx` to `nx` and from `-nz` to `nz`. Note moreover (see for example the `ARRAY` dimensions in line 101) that the simmetry property of a complex function which is the Fourier transform of a real function is exploited. Thus one half of the Fourier coefficients (namely, the negative streamwise wavenumbers) does not need to be explicitely computed and stored.

In the `MODULE setup_derivatives` the finite-difference coefficients are computed for the interior points, whereas in lines 36-46 specific coefficients at the two walls are computed, based on non-centered stencils. For example the coefficients `d140` evaluate the first derivative at IV order accuracy at the inner wall for `iy=0`, whereas the `d14n` do the same job for the opposite wall at `iy=ny`. The wall-normal discretization is defined in lines 16 and 17, the easiest possibility of uniform mesh is commented out, while a hyperbolic-tangent law is applied in line 17.

These coefficients are then used compactly in the remaining parts of the code thanks to the functions defined in lines 48–55 and inlined for runtime efficiency. The `COMPLEX FUNCTION`s are defined via `REAL FUNCTION`s to help the compiler optimize the code better, since in the C language the type `COMPLEX` does not exist.

The variables reside in the Fourier space, so that the unknowns are the Fourier coefficients of expansions similar to (8). A type `VELOCITY` made by a `STRUCTURE` of 3 complex numers is introduced in line 68, and an `ARRAY(0..nx,-nz..nz,-1..ny+1) OF VELOCITY` is allocated in line 101.

The `SUBROUTINE convolutions` performs the task of transforming the velocity components in physical space, computing their products and transforming the results, of type `MOMFLUX`, back in Fourier space. The aliasing error is removed by the 3/2 rule.

Perhaps the most important routine is the `SUBROUTINE buildrhs`, which assembles the right-hand-side of Eqns. (9) and (10). One important observation is that the particular explicit scheme for temporal integration to be used for the convective terms is not predetermined, and can be easily changed. In fact the variable `timescheme` appears in the calling list of `buildrhs` (line 112). The actual invocation of `buildrhs` is shown at line 254 in side the main temporal loop `timeloop`. In this example the used time scheme is Crank-Nicolson and Adams-Bashfort, as defined in lines 245–249 by `SUBROUTINE CN_AB`. The structure of `buildrhs` is basically a main loop over the y positions, to compute velocity products with the pseudo-spectral method within `convolutions`, then build the spatial derivatives of `MOMFLUX`es, and lastly assemble these quantities into the r.h.s. of the two equations. Note the definition of the `INLINE FUNCTION`s `OS` and `SQ`, standing for Orr-Sommerfeld and SQuire, to define only once the parts of the equations (for $\hat{v}$ and $\hat{\eta}$ respectively) that will be used again in the following implicit part. `IF ix=0 AND iz=0 THEN`, i.e. when $k^2 = 0$, the mean velocity profile in both the homogeneous directions is computed. Note that the streamwise mean profile is stored in `V(0,0,*).u.REAL` and the spanwise profile into `V(0,0,*).u.IMAG`: at this stage of the `timeloop`, in fact, `V.u` stores the values of the r.h.s. for the η equation.

So far, only points in the interior of the channel, i.e. for the index `1 <= iy <= ny-1`, have been involved. In the second part of the time step, in the `SUBROUTINE linsolve`,

the boundary points and the boundary conditions come into play. The linear systems arising from the implicit time discretization of the viscous terms are solved, in a sequence for each wavenumber pair. So, for each pair of `ix` and `iz`, the two system matrices `D2vmat` and `etamat` are computed, also by taking advantage of the previously defined `INLINE FUNCTION`s `OS` and `SQ`. Then the boundary lines of the matrices (and of the r.h.s.) are modified to account for the boundary conditions. This task is accomplished by the routines `applybc_0` and `applybc_n`, which allow for the most general form of the boundary conditions. Then the system is solved, in line 209 for $\hat{v}$ and in line 214 for $\hat{\eta}$. In the case the simulations are carried out at a constant flow rate, the actual flow rate and the velocity profile for the required correction are computed in lines 221-226, and then the correction is applied in lines 227 and 228 for both homogeneous components to achieve the desired flowrate `meanflowx` and/or `meanflowz`. Finally, the `timeloop` is completed once the Fourier components of the homogeneous velocity components are computed. This is the only point in the whole code where actual derivatives have to be computed. This task is left to the `FUNCTION deriv` (defined in lines 149–160 and then used in line 231–232), which again acts on the `REAL` and `IMAG`inary parts of $\hat{v}$ separately, to the aim of improving the optimizing efficiency of the compiler. It is only at this stage of the `timeloop` that the three fields `u`, `v` and `w` of the `COMPLEX ARRAY V` actually contain the Fourier coefficients of the velocity components.