A Parallel Algorithm for the Direct Numerical Simulation of Turbulent Channel Flow

Maurizio Quadrio[1,*], Paolo Luchini[2], J.M. Floryan[3]

[1] *Dept. Aerospace Engineering, Politecnico di Milano, Italy*
[2] *Dept. Mechanical Engineering, University of Salerno, Italy*
[3] *Dept. Mechanical and Materials Engineering, University of Western Ontario, Ontario, Canada*

Email: *maurizio.quadrio@polimi.it, luchini@unisa.it, mfloryan@uwo.ca*

ABSTRACT

An algorithm for the Direct Numerical Simulation (DNS) of the incompressible Navier–Stokes equations for simple geometries in Cartesian coordinates is described. This algorithm presents excellent properties when used in large-scale or grid computing environments: an underlying fast scalar implementation, low memory requirements, a limited amount of communication and a high degree of parallelisation.

The parallel algorithm is designed for shared- as well as distributed-memory systems and, eventually, grid computing; data exchange between different computing machines is minimized, so as to take advantage of standard or non-optimal network devices and bandwidths. The key point for an efficient parallelization is the choice of finite differences for the discretizazion of the wall-normal direction, allowing the computational domain to be cut into wall-parallel slices: the most demanding computations can be carried out for different transverse planes without communication, and the global memory requirements can be subdivided amongst the computing machines.

1 INTRODUCTION

The Direct Numerical Simulation (DNS) of the incompressible Navier–Stokes equations is an essential research tool for basic studies on the physics of turbulent flow, for the development and testing of suitable sub-grid-scale models for Large Eddy Simulations, and for further refinement of turbulence models to be used in the solution of Reynolds-Averaged Navier-Stokes equations [1]. Since the computational size of the problems to be solved with DNS is very large and increases rapidly with the Reynolds number, it is mandatory for a DNS computer code to be highly efficient, both in terms of CPU time and RAM occupation, and to be suitable for an efficient parallelization.

An effective formulation of the Navier–Stokes equations was presented almost 15 years ago by Kim, Moin & Moser in [2], the pionieering work on the DNS of turbulent plane channel flow. It consists of the replacement of the continuity and momentum equations with two scalar equations, one (second-order) for the normal component of vorticity and one (fourth-order) for the normal component of velocity. This procedure is appealing, since pressure is removed from the equations, and the recovery of the other two velocity components can be done through the solution of a 2x2 *algebraic* system (a very cheap procedure from a computational point of view), when a Fourier expansion is adopted for the homogeneous directions. A very 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 Chebischev polynomials) in this direction, while in more recent years the use of finite difference schemes has seen growing popularity [1].

In this paper we present a numerical method designed for the DNS of turbulent wall flows in cartesian coordinates, with high computational efficiency, good parallel performances and very low memory requirements. The method solves the equations in the form introduced in [2]. We will illustrate the main properties concerning computational efficiency, the use of compact, high-order finite differences for the wall-normal direction, and the parallel strategy, with emphasis on memory performance.

*Visiting Professor, Dept. of Mechanical and Materials Engineering, University of Western Ontario

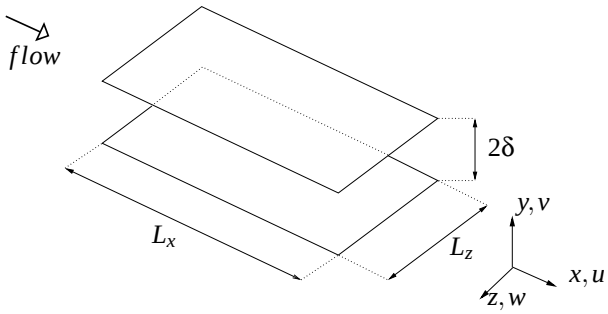

Figure 1: Sketch of the computational domain for the cartesian coordinate system.

2 THE GOVERNING EQUATIONS

The cartesian coordinate system is illustrated in Fig.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 $y = y_1$ and the upper wall at $y = y_2$. The reference length δ is taken to be one half of the channel height, i.e. $\delta = (y_2 - y_1)/2$.

The streamwise and spanwise lengths of the computational box are L_x and L_z respectively. Once an appropriate reference velocity U is chosen, a Reynolds number can be defined as $\mathrm{Re} = U\delta/\nu$, where ν is the kinematic viscosity of the fluid. Concerning the boundary conditions, at the wall the no-slip and no-penetration conditions are physically meaningful. The use of periodic boundary conditions is generally accepted for the homogeneous directions (see [3] for a critical discussion). 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.

2.1 Equation for the wall-normal vorticity component

The wall-normal component of the vorticity vector, which we shall indicate with η, after transforming in Fourier space is given by:

$$\hat{\eta} = i\beta\hat{u} - i\alpha\hat{w} \tag{1}$$

where the hat indicates Fourier-transformed quantities, i is the imaginary unit, and the symbols α and β 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 [2], by taking the y component of the curl of the momentum equation, obtaining:

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

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:

$$\widehat{HU} = -i\alpha\widehat{uu} - D_1(\widehat{uv}) - i\beta\widehat{uw} \tag{3a}$$

$$\widehat{HV} = -i\alpha\widehat{uv} - D_1(\widehat{vv}) - i\beta\widehat{vw} \tag{3b}$$

$$\widehat{HW} = -i\alpha\widehat{uw} - D_1(\widehat{vw}) - i\beta\widehat{ww} \tag{3c}$$

The numerical solution of Eq. (2) requires an initial condition for $\hat{\eta}$, which can be computed from the initial condition for the velocity field. The same initial condition can be used for the calculation of the nonlinear terms $\widehat{HU}$ and $\widehat{HW}$ and their derivatives. The periodic boundary conditions in the homogeneous directions are automatically satisfied thanks to the Fourier transform, whereas the no-slip condition translates in $\hat{\eta} = 0$ to be imposed at the two walls $y = y_1$ and $y = y_2$.

2.2 Equation for the wall-normal velocity component

An equation for the wall-normal velocity component $\hat{v}$, which does not involve pressure, can be written [2] summing the x component of the momentum equation differentiated with respect to x and y and the z component differentiated with respect to y and z, and subtracting the y component differentiated twice with respect to x and then twice with respect to z. Further simplifications arise if the continuity equation is invoked for cancelling some terms, eventually obtaining the following fourth-order evolution equation for $\hat{v}$:

$$\frac{\partial}{\partial t}\left(D_2(\hat{v}) - k^2\hat{v}\right) = \frac{1}{\mathrm{Re}}\left[D_4(\hat{v}) - 2k^2D_2(\hat{v}) + k^4\hat{v}\right] - k^2\widehat{HV} - D_1\left(i\alpha\widehat{HU} + i\beta\widehat{HW}\right) \tag{4}$$

This scalar equation can be solved numerically provided an initial condition for $\hat{v}$ is known. The no-penetration condition translates in $\hat{v} = 0$ to be imposed at $y = y_1$ and $y = y_2$. The continuity equation written at the two walls makes evident that the additional two boundary conditions required for the solution of Eq. (4) are $D_1(\hat{v}) = 0$ at $y = y_1$ and $y = y_2$.

2.3 Velocity components in the homogeneous directions

The two scalar equations (2) and (4) are uncoupled and, after proper time discretization, they can be solved for advancing the solution by one time step, provided the nonlinear terms (3a)-(3c) and their spatial derivatives can be calculated. For this being possible, one needs to know how to compute $\hat{u}$ and $\hat{w}$ at a given time knowing $\hat{v}$ and $\hat{\eta}$. By using the definition (2) of $\hat{\eta}$ and the continuity equation written in Fourier space, a 2x2 algebraic system can be written in the unknowns $\hat{u}$ and $\hat{w}$; its analytical solutions reads:

$$\begin{cases} \hat{u} = \dfrac{1}{k^2}\left(i\alpha D_1(\hat{v}) - i\beta\hat{\eta}\right) \\ \hat{w} = \dfrac{1}{k^2}\left(i\alpha\hat{\eta} + i\beta D_1(\hat{v})\right) \end{cases} \tag{5}$$

2.3.1 Plane-averaged homogeneous velocities

The preceding system is singular when $k^2 = 0$. This follows from the fact that Eqns. (2) and (4) have been derived from the initial differential system through a procedure involving spatial derivatives.

Let us introduce a plane-average operator:

$$\widetilde{f} = \frac{1}{L_x}\frac{1}{L_z}\int_0^{L_x}\int_0^{L_z} f\, dxdz$$

The space-averaged streamwise velocity $\widetilde{u} = \widetilde{u}(y,t)$ is a function of the wall-normal coordinate and the time only, and in Fourier space it corresponds to the Fourier mode for $k = 0$. The same applies to the spanwise component $\widetilde{w}$. With the present choice of the reference system, where the x axis is aligned with the mean velocity, 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 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 plane-average operator to the relevant components of the momentum equation:

$$\frac{\partial\widetilde{u}}{\partial t} = \frac{1}{\mathrm{Re}}\frac{\partial^2\widetilde{u}}{\partial y^2} - \frac{\partial(\widetilde{uv})}{\partial y} + f_x$$

$$\frac{\partial\widetilde{w}}{\partial t} = \frac{1}{\mathrm{Re}}\frac{\partial^2\widetilde{w}}{\partial y^2} - \frac{\partial(\widetilde{vw})}{\partial y} + f_z$$

In these expressions, f_x and f_z are the forcing terms needed to force the flow through the channel against the viscous resistance 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 be also a time-dependent spatially constant pressure gradient, chosen in such a way that the flow rate remains constant in time. 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 are zero for a standard channel flow.

3 THE NUMERICAL METHOD

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:

$$v(x,z,y,t) = \sum_{h=-N_x/2}^{+N_x/2}\sum_{\ell=-N_z/2}^{+N_z/2}\hat{v}_{h\ell}(y,t)\,\mathrm{e}^{i\alpha x}\mathrm{e}^{i\beta z} \tag{6}$$

where:

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

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 and L_z of the computational domain. The computational parameters given by the dimensions of the computational domain, L_x and L_z, and the truncation of the series, N_x and N_z, must be chosen so as to miminize computational errors. See again [3] for details regarding a proper choice of L_x.

The numerical evaluation of the non linear terms in Eqns. (2) and (4) would require computationally expensive convolutions in Fourier space. The same evaluation can be performed in a much cheaper way if first the Fourier components of the velocity are transformed back in physical space, then the velocity products are evaluated and eventually re-transformed into the Fourier space, where their spatial derivatives are computed. Fast Fourier Transform (FFT) algorithms are used to compute the nonlinear terms exactly in a computing-efficient manner. The exact removal of the aliasing error deriving from the nonlinear operations is performed by expanding the number of collocation points by a factor of (at least) 3/2 before going from the Fourier space into the physical space.

3.2 Time discretization

Time integration of the equations is performed by a partially-implicit method, as described for example in [2]. This is a standard approach in DNS: the explicitly integrated part of the equations benefit from a higher-accuracy scheme, while the stability-limiting implicit part is subjected to an implicit time advancement, thus relieving the constraint on time step size. Our preferred choice is to use an explicit third-order, low-storage Runge-Kutta scheme for the integration of the explicit part of the equations, and an implicit second-order Crank-Nicolson scheme for the implicit part. Here we present the equations with the Crank-Nicolson scheme for the viscous terms and with a generic two-levels scheme for the nonlinear terms ($k^2 = h^2\alpha_0^2 + \ell^2\beta_0^2$):

$$\begin{aligned}
&\frac{\lambda}{\Delta t}\hat{\eta}_{h\ell}^{n+1} - \frac{1}{\mathsf{Re}}\left[D_2(\hat{\eta}_{h\ell}^{n+1}) - k^2\hat{\eta}_{h\ell}^{n+1}\right] = \\
&\qquad \frac{\lambda}{\Delta t}\hat{\eta}_{h\ell}^{n} + \frac{1}{\mathsf{Re}}\left[D_2(\hat{\eta}_{h\ell}^{n}) - k^2\hat{\eta}_{h\ell}^{n}\right] + \\
&\qquad \theta\left(i\beta_0\ell\widehat{HU}_{h\ell} - i\alpha_0 h\widehat{HW}_{h\ell}\right)^{n} + \\
&\qquad\qquad \xi\left(i\beta_0\ell\widehat{HU}_{h\ell} - i\alpha_0 h\widehat{HW}_{h\ell}\right)^{n-1} \qquad (7)
\end{aligned}$$

$$\begin{aligned}
&\frac{\lambda}{\Delta t}\left(D_2(\hat{v}_{h\ell}^{n+1}) - k^2\hat{v}_{h\ell}^{n+1}\right) - \\
&\quad \frac{1}{\mathsf{Re}}\left[D_4(\hat{v}_{h\ell}^{n+1}) - 2k^2D_2(\hat{v}_{h\ell}^{n+1}) + k^4\hat{v}_{h\ell}^{n+1}\right] = \\
&\qquad \frac{\lambda}{\Delta t}\left(D_2(\hat{v}_{h\ell}^{n}) - k^2\hat{v}_{h\ell}^{n}\right) + \\
&\quad \frac{1}{\mathsf{Re}}\left[D_4(\hat{v}_{h\ell}^{n}) - 2k^2D_2(\hat{v}_{h\ell}^{n}) + k^4\hat{v}_{h\ell}^{n}\right] + \\
&\theta\left(-k^2\widehat{HV}_{h\ell} - D_1\left(i\alpha_0 h\widehat{HU}_{h\ell} + i\beta_0\ell\widehat{HW}_{h\ell}\right)\right)^{n} + \\
&\xi\left(-k^2\widehat{HV}_{h\ell} - D_1\left(i\alpha_0 h\widehat{HU}_{h\ell} i\beta_0\ell\widehat{HW}_{h\ell}\right)\right)^{n-1} \qquad (8)
\end{aligned}$$

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 $\xi = -1$.

After time discretization, the continuous problem is reduced to the numerical solution of Eqns. (7) and (8), which is carried out in two logical steps. The first consists in building the r.h.s. of (7) and (8), computing psuedospectrally the non-linear terms and their spatial derivatives from the known velocity field. The second one involves the solution of a set of two linear systems, one for each value of k. A finite-differences discretization of the wall-normal derivative operators produces real banded matrices. The solution of the 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 with (5).

3.3 High-accuracy compact finite difference schemes

The discretization of the wall-normal derivatives D_1, D_2 and D_4 is performed through finite difference (FD) compact Padé schemes with fourth-order accuracy over a computational molecule composed by five arbitrarily spaced grid points. We indicate here with $d_1(i), i = -2,\ldots,2$ the five coefficients discretizing the exact operator D_1 over five adjacent grid points:

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

where y_j is the y position on the computational mesh where the derivative has to be evaluated. A similar notation is adopted for d_2 and d_4.

A compact FD formula is able to approximate a differential operator in a wider frequency range compared to a centered scheme, thus achieving resolution properties similar to those of spectral schemes [4]. The use of compact schemes, known also as implicit finite-differences schemes, typically require the inversion of a linear system for the actual calculation of a derivative [5]. Here we are able to use compact, fourth-order accurate schemes at the cost of standard centered schemes, taking advantage of the fact that the equations of motion do not contain the third-derivative operator D_3. 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. By multiplying the equations times this denominator, we hence obtain explicit formulas again, provided a suitable FD operator d_0 is applied to the right-hand-side of our equations. This simplification is contained in the original Gauss-Jackson-Noumerov compact formulation exploited in his seminal work by Thomas [6], concerning the numerical solution of the Orr-Sommerfeld equation.

The actual computation of the coefficients d_0,d_1,d_2 and d_4 for a certain order p of accuracy must, from a conceptual point of view, descend from the condition 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. In practice, following a standard procedure in the theory of Padé approximants, this can be enforced by choosing a set t_m of polynomials of y of

increasing degree

$$t_m(y) = 1, y, y^2, ..., y^m \tag{9}$$

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

$$d_4(t_m) - d_0(D_4(t_m)) = 0 \tag{10}$$

is verified for polynomials from $m = 0$ up to $m = 4 + p$.

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

$$\sum_{i=-2}^{2} d_0(i) = 1 \tag{11}$$

The other 9 conditions are given by Eqn. (10) written for $m = 0, 1, \ldots 8$. We thus can set up, for each distance from the wall, a 10x10 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 5x5 linear systems since the d_0 are known.

An additional further simplification is possible. Since the polynomials (9) have vanishing D_4 for $m < 4$, thanks to the normalization condition (11) the 10x10 system can be split into two 5x5 subsystems, separately yielding the coefficients d_0 and d_4.

The use of a variable mesh size in the wall-normal direction in such a way as to still keep a fourth-order accuracy requires this procedure to be performed numerically at each y 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 fourth-order accurate (except the operator d_4, which is sixth-order), at essentially the same computational cost required for using a second-order-accurate method.

4 THE PARALLEL STRATEGY

There are many strategies for building a computer code suitable for parallel execution. We have favoured the availability of a large amount of CPU time at low or no costs, focusing towards the use of commodity computing and networking hardware with a limited number of processors. As a future step we are presently considering a different parallel implementation, still in development at the present time, which is well suited for use over a large number of processors.

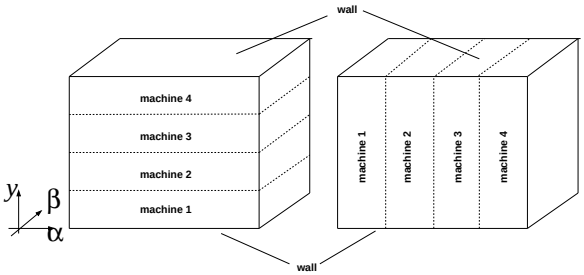

Figure 2: Arrangement of data across the channel.

4.1 Distributed-memory machines

If the calculations are to be executed in parallel by N_p computing machines (nodes), communication between nodes will necessarily have to take place. Our design goal has been to keep the required amount of communication to a minimum. At first, however, it appears that, every time the numerical solution is advanced by one time (sub)step, a transposition of the whole dataset across the computing nodes cannot be avoided. If data are stored by grouping contiguous streamwise (or spanwise) wavenumbers on the same machine, the pseudo-spectral evaluation of the nonlinear terms requires all the wavenumbers to be simultaneously present on each machine, and this calls for a large amount of communication. If, on the other hand, data are stored in wall-parallel slices, the FFTs can be done locally to each machine. When in addition the spectral discretization in the y direction is avoided in favour of a FD discretization (recall that the FD operators are local), the necessary communication is greatly reduced, since it is needed only at the interface between contiguous slices. However, even when this strategy is adopted, a full transposition seems nonetheless to be necessary, since the linear systems which stem from Eqns. (7) and (8) require the simultaneous inversion of banded matrices, whose principal dimension span the entire width of the channel, while data are stored in wall-parallel slices.

This appearance is deceiving however, and a transpose of the whole flow field can be avoided with the following procedure. Data are distributed in slices parallel to the walls, as shown schematically in Fig.2: if N_y is the number of collocation points, each machine stores all the streamwise and spanwise wavenumbers for N_y/N_p contiguous y positions. In this way the planar FFTs do not require communication, since the coefficients are local to each machine. Wall-normal derivatives required in the r.h.s. of Eqns. (7) and (8) do require some communication at the interface between contiguous slices; however this communication can be

avoided if, when using a 5-point stencil, two boundary planes on each side of each slice are duplicated.

Of course, in the second part of the time-step advancement (the solution of the set of 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 N_p machines. It is possible to avoid a global transpose of the data, by solving each pentadiagonal system in a serial way across the machines. The key observation to obtain reasonable parallel performance is that the number of the linear systems to be solved at each time step is very large, i.e. $(N_x+1)(N_z+1)$, which is at least 10^4 in a typical calculation. When the LU decomposition of the matrix of the system is performed (with a standard Thomas algorithm), the flow of information is from the top row of the matrix down to the bottom row (elimination of the unknowns), and then vice-versa (backsubstitution). A single machine can then perform the elimination in the local part of the matrix for a given h,ℓ pair, and then start working on the same elimination for the matrix of the following system, say $h,\ell+1$, while waiting for the backsubstituion of the results for the h,ℓ system.

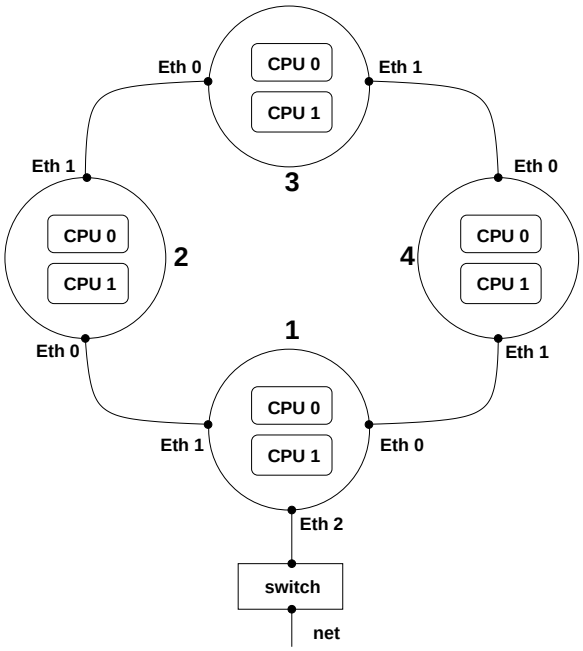

Figure 3: Conceptual scheme of the connection topology between computing machines

4.2 Shared-memory machines

The single computing node can 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 shared-memory facilities of a Unix system. We note that this is different from using a message-passing strategy with shared-memory, that simply becomes a faster tramsission medium. Typically 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 r.h.s. term of Eqs. (7) and (8). With SMP machines, high parallel efficiencies can be obtained quite easily by "forking" new processes as needed: the operating system itself then handles the assignment of tasks to different CPUs, and only task synchronization is a concern at the programming level.

4.3 A low-cost computing system

While the numerical method described heretoforth can be used on a general-purpose cluster of machines employing traditional message-passing libraries for communication, a dedicated computing system can be specifically designed and built on top of the computer code, for maximum efficiency.

Concerning the communication type, it is possible to rely directly on standard networking services of the Unix operating system, i.e. sockets, instead of message-passing libraries. Using sockets allow to take advantage easily and efficiently of the advanced buffering techniques incorporated in the management of the input/output streams by the operating system (provided stdio file buffer size is empirically adjusted to its best size). Concerning transmission protocol, the simplest choice is the standard, error-corrected TCP/IP protocol: we have estimated that on typical problem sizes the benefits from using a dedicated protocol would be negligible.

At a hardware level, the present parallel strategy allows one further simplification. Since the transposition of the whole dataset is avoided, communications are always point-to-point, and each computing machine needs to exchange data with two neighboring machines only. This can be exploited with a very simple ring-like connection topology between the computing machines, schematically illustrated in Fig.3, which replicates the logical exchange of information illustrated in Fig.2: each machine can be connected through two dedicated Ethernet cards only to the previous machine and to the next. The necessity of a hub or switch is thus eliminated, in favour of simplic-

ity, cost-effectiveness and performance. We have built such a dedicated system, composed by 8 SMP Personal Computers. Each node is equipped with 2 Pentium III 733MHz CPU and 512MB of 133MHz SDRAM. The nodes are connected each other with two cheap 100MBits Fast Ethernet cards. A similar system can be built with relative ease by anyone.

5 PERFORMANCE

For performance comparison we use data from [7], a recent work conducted by one of the leading research groups working on DNS, and focused on the optimization of a similar code for parallel computing. We must note however that we are not able to compare two different DNS codes running on identical machines.

5.1 Memory requirements

One important requirement for a DNS code is to save RAM. The size of the required RAM is dictated by the number of the 3-dimensional arrays, and is typically reported in [2] and [7] to be no less than $7N_xN_yN_z/2$ words, where the factor 7 accounts for the 3 velocity components, the 2 r.h.s. of the Eqs. (7) and (8), and the same two r.h.s. stored at a previous time level for a two-levels time integration scheme (the factor 2 accounts for the saving allowed by the simmetry in the Fourier expansion of a real function).

In our code the required memory space is reduced to $5N_xN_yN_z/2$: this saving comes from using the velocity arrays for storing the (current) r.h.s. terms, exploiting the fact that velocities and r.h.s. are needed in different parts of the time integration procedure. A test case with $N_x = 129$, $N_y = 129$ and $N_z = 129$ hence requires only 95 MBytes of RAM (double precision).

The most important feature of our method is that this memory requirement can be subdivied amongst computing machines. For example the same test case run on four machines requires only 34 MBytes of RAM on each machine. This peculiar property of our code descends from the particular parallel implementation, which does not require transposing the whole dataset.

5.2 CPU requirements

Concerning the CPU efficiency, the test case mentioned above requires 44 CPU seconds for the computation of a full three-substeps temporal step on a single PIII processor. One can deduce from [7] that the same case can be run on a single processor of the

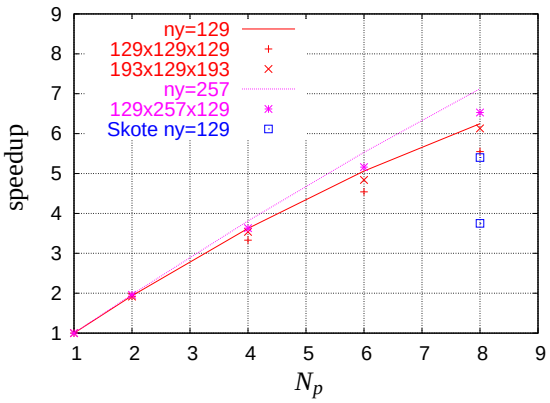

Figure 4: Speedup as a function of the number N_p of nodes for the Intel PIII dedicated system.

256 nodes Cray T3E of the National Supercomputer Center of Linkoeping (Sweden) in 30 seconds, and on a single processor of the 152 nodes IBM SP2 machine, available at the Center for Parallel Computers of KTH University, with 7.5 seconds of CPU time. This single-node comparison shows that the present computer code runs on a commodity PC with performance comparable with those of similar codes running on expensive supercomputers. With more modern hardware the execution times are further reduced: the same test case runs in 20.3 seconds on a desktop AMD Athlon XP 1800+ at 1500 MHz.

5.3 Parallel efficiency

The parallel performances of the code are illustrated in Fig.4, where speedup ratios are reported as a function of the number N_p of computing nodes. The speedup is the ratio between the wall-clock time needed for a single-node calculation and the actual wall-clock time with N_p machines.

The speedup is not expected to be linear, due to the previously discussed duplication of four data planes at the interface between different slices. The maximum possible speedup, shown with lines in Fig. 4, is estimated by:

$$speedup = N_p \left(1 - \frac{4(N_p - 1)}{N_y}\right)$$

and increases with the problem size; it is reasonably high as long as the number of computing nodes is small compared to the number N_y of points in the y direction. Compared to the maximum speedup, the measured performances are extremely good, and become even better when the size of the computational prob-

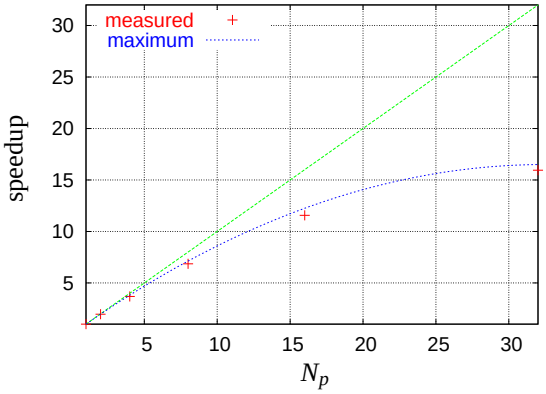

Figure 5: Speedup vs N_p for the Itanium II SHARCNET cluster, with $N_y = 257$.

lem increases. The communication time is not a bottleneck, despite the fact that communication has been enforced with commodity hardware and with the overhead of the TCP protocol. With $N_p = 8$, the communication time is as low as 7% of the total computing time for the most demanding case of $N_x = 129$, $N_y = 257$ and $N_z = 129$, and is 14% for the worst case of $N_x = 129$, $N_y = 129$ and $N_z = 129$, which requires 6.5 seconds for iteration with a speedup of 5.5. The same test case runs in approximately 2 seconds when 8 processors are used on the *SP*2 (speedup 3.75), while it requires 7.5 seconds with 8 processors of the T3E.

Of course the parallel strategy described here targets only systems where the number N_p of computing nodes is not very high compared to the number N_y of points in the y direction. Increasing N_p leads to gradually worse performance: this is illustrated for example in Fig.5, where we measure performances on a cluster of 64 SMP machines equipped with 2 Itanium II processors, available at SHARCNET. We observe in Fig.5 that the actual communication times are barely discernible. This is due to the faster communication hardware available on the Itanium machines, namely Gigabit Ethernet cards.

6 Conclusions

We have described the design strategy of a computer code for the Direct Numerical Simulation of the incompressible Navier–Stokes equations. The code manages the various discretization choices to achieve the highest computational efficiency.

Similar codes described in the literature require 40% more RAM for a given problem size. Most important, our parallel strategy allows the memory requirements to be subdivided between computing machines, so making feasible the largest DNS computations.

One parallel implementation has been described in detail, which shows very good performances as long as the number of machines is small compared to the number of discretization points in the wall-normal direction. The layout of a computing system built on top of this code has been described, which makes available to the scientist a computing power comparable to that of a supercomputer, at a fraction of the price and in a dedicated environment. The modification of this parallel strategy to obtain high efficiencies also in the case of high number of computing machines is currently underway. This will pave the way towards the use of this code in heavily distributed computing environments, a.k.a. grid computing.

Acknowledgments

The first two authors have been supported by the SHARCNET through the Senior Visiting Fellowship Program.

References

[1] P. Moin and K. Mahesh. Direct numerical simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics*, 30:539–578, 1998.

[2] J. Kim, P. Moin, and R. Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *Journal of Fluid Mechanics*, 177:133–166, 1987.

[3] M. Quadrio and P. Luchini. Integral time-space scales in turbulent wall flows. *Submitted to Physics of Fluids*, 2003.

[4] S.K. Lele. Compact Finite Difference Schemes with Spectral-like Resolution. *Journal of Computational Physics*, 103:16–42, 1992.

[5] K. Mahesh. A Family of High Order Finite Difference Schemes with Good Spectral Resolution. *Journal of Computational Physics*, 145(1):332–358, 1998.

[6] L.H. Thomas. The stability of plane Poiseuille flow. *Physical Review*, 91(4):780–783, 1953.

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