Proceedings of FEDSM'02
2002 ASME Fluids Engineering Division Forum
July 14-18, 2002, Montreal, Quebec, Canada

FEDSM2002-31048

ADJOINT DNS OF TURBULENT CHANNEL FLOW

Paolo Luchini[*]
Department of Mechanical Engineering
Università di Salerno
84084 Fisciano
Italy
Email: luchini@unisa.it

Maurizio Quadrio
Department of Aerospace Engineering
Politecnico di Milano
via La Masa 34 - Milano, 20134
Italy
Email: maurizio.quadrio@polimi.it

ABSTRACT

A discrete adjoint is developed on top of a parallel-computing code for the direct numerical simulation of wall turbulence. The architecture of the direct code is designed to obtain good scalar performance and high parallel efficiency using a cluster of dual-CPU Personal Computers. The adjoint code, written in a way that mimicks the basic structure of its direct counterpart, can also benefit of the same computational and parallel efficiency. After describing how we tested the adjoint part of the code against its direct counterpart, we present results concerning the sensitivity of the wall drag to blowing/suction applied at the wall for the case of a turbulent channel flow at $Re_\tau = 180$.

INTRODUCTION

Adjoint computations are an increasingly important tool in fluid mechanics, with applications that include aerodynamic shape optimization, state identification in weather predictions, the quantification of transient growth and its effects on bypass transition, the control of transition and of turbulence. Turbulence control at large has always attracted the interest of researchers for the technological benefits that can potentially derive from either a drag reduction or a mixing increase in different applications; most varied devices have been proposed for this purpose, that range from passive modifications of the wall structure or of the fluid properties to moving walls to active feedback control, either empirically determined or based on optimal and robust control theory. This latter approach has been very recently popularized by the successful efforts of (Bewley *et al.*, 2001), (Bewley, 2001), (Högberg and Bewley, 2000), and the field may rightfully be said to be blooming.

An essential component of an optimal control strategy based on direct numerical simulations of turbulence is an adjoint code. This is, basically, a means to compute the derivatives of an arbitrary objective function with respect to the control parameters. Just as in the optimization of a simple analytical formula, the optimum is identified by the vanishing of derivatives, and before applying any of several possible optimization strategies an efficient evaluation of these derivatives must be available. "efficient" is the keyword here: a direct numerical simulation of turbulence is a very computation-intensive task, and in order to be useful an adjoint code must first of all be as efficient as the direct code. This is not at first sight an obvious task, as the derivatives of the objective function with respect to all control parameters are simultaneously needed, and the control parameters are often the discretized values of a velocity or other field variable distributed all over a surface or even a volume. Yet computing the adjoint in a time comparable to the direct simulation is theoretically possible. Here we describe an adjoint DNS code that achieves this level of performance in a parallel-computing environment.

DIRECT NUMERICAL METHOD

The adjoint DNS code presented in this paper is built on top of a recently developed parallel solver of the Navier–Stokes equations for an incompressible fluid, a description of which can be found in (Quadrio and Luchini, 2001).

[*]Address all correspondence to this author.

Our Navier–Stokes solver is based on a mixed discretization: Fourier modes are used for the homogeneous, wall-parallel directions, and high-order finite differences in the wall-normal direction. Finite differences are produced from a fourth-order accurate compact scheme acting on a five-point computational molecule in a variable-spacing mesh. We have preferred finite differences to spectral schemes for their flexibility in dealing with inhomogeneous boundary conditions and their better properties for the parallelization of the code.

The Navier–Stokes equations are formulated, following a procedure described for example in (Kim *et al.*, 1987), in terms of a scalar equation for the normal component of velocity and a scalar equation for the normal component of vorticity that imitate the Squire decomposition of stability problems, thus achieving the largest computational efficiency when a Fourier discretization is adopted for the homogeneous directions. The non-linear terms of the equations are computed with a pseudo-spectral approach, transforming the flow variables back from Fourier space to physical space before computing the products, and taking advantage of computationally efficient Fast-Fourier-Transforms algorithms. Dealiasing in the homogeneous directions is performed by expanding the number of Fourier modes by a factor of $3/2$ before going from Fourier space to physical space, to avoid the introduction of spurious energy from the high-frequency into the low-frequency modes during the calculations.

The advancement in time of the solution is performed through a commonly used partially implicit approach: nonlinear terms are advanced with an explicit scheme (a low-storage, three-substeps, third order Runge-Kutta scheme), and linear terms with an implicit method (a second order Crank-Nicholson scheme) in order to overcome the stability limitations deriving from the viscous terms. Even if this scheme is rather classical, it has been 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.

Thanks to the use of finite differences for the discretization of the wall-normal derivatives, the parallel algorithm requires no inter-node communication as far as the computation of the explicit part is concerned. Fourier transforms are in fact executed in planes parallel to the walls, and all the necessary data always reside on the same machine. The evaluation of wall-normal derivatives through finite differences can be done without communication as well, at the expense of duplicating in each machine the four boundary planes shared with the two neighboring machines. Only the numerical solution of the linear systems arising from the discretization of the implicit part requires communication, but the impact on the overall computing time decreases when the problem size increases, and even with cheap connection hardware calculations never happen to be communication-bound.

The computer code is built to be run in parallel, on shared-memory SMP architectures and/or a cluster of distributed-memory computers. We focused the design of the code towards the use of low-cost, commodity hardware, since with commodity (and hence dedicated) machines an unshared amount of CPU time becomes available with limited financial effort. The code is currently running on a dedicated cluster of 8 SMP Personal Computers, purposely built in late 2000 at the Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano at the cost of approximately 10000 Euro. Each node of the cluster is equipped with 2 Pentium III 733-MHz CPUs and 256-MB 133-MHz SDRAM; the nodes are connected in a ring with two cheap 100-MBits Fast Ethernet cards each.

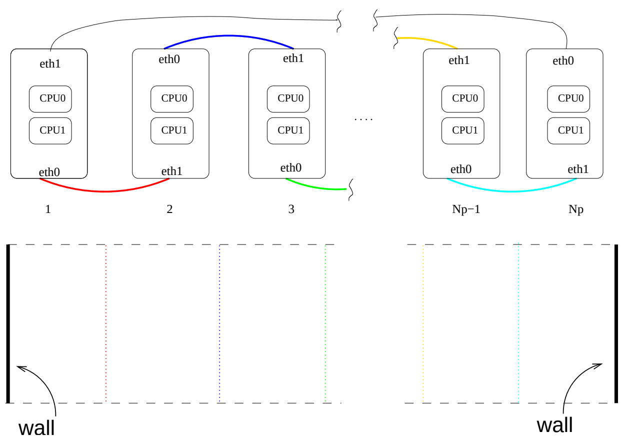

Figure 1. CONNECTION TOPOLOGY OF THE PC CLUSTER AT POLITECNICO DI MILANO

The parallel algorithm is perfectly suited to a very simple ring topology, schematically illustrated in figure 1, where each machine is directly connected to the previous machine and to the next only, and contains a slice of the computational domain in the wall-normal direction. The necessity of a hub or switch is eliminated, increasing simplicity, communication bandwidth, and cost-effectiveness at the same time, and the transposition of the flow field between computing nodes is never required, thanks to the design of the algorithm.

PERFORMANCE OF THE DIRECT CODE

The direct code has been designed with the primary aim of computational efficiency. Its memory requirements are significantly lower that those of similar codes documented in the literature. For example the memory space used by the code described in (Kim *et al.*, 1987) and the recently optimized code described in (Alvelius and Skote, 2000) and (Skote, 2001) is 40% larger than that required by our code.

Efficiency in terms of execution time is more difficult to evaluate, owing essentially to the lack of complete reference data, but we have estimated that single-processor execution times

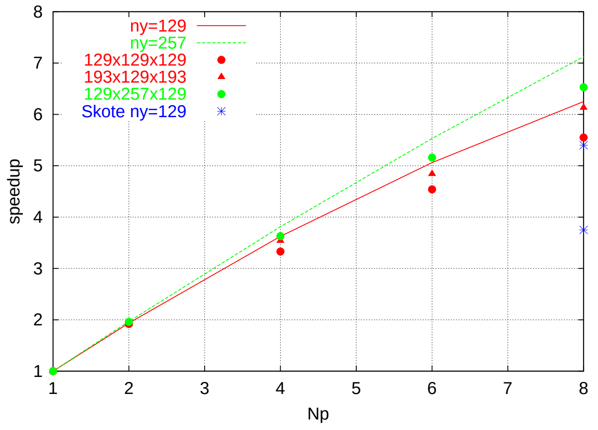

Figure 2. PARALLEL SPEEDUP VS. THE NUMBER N_P OF COMPUTING NODES. LINES WITHOUT SYMBOLS ARE THE IDEAL SPEEDUP. PARALLEL SPEEDUPS INFERRED FROM (Skote, 2001) ARE ALSO SHOWN.

for a given problem size are fully competitive with the available information. Despite the commodity networking hardware, and the use of standard operating system networking services (unix sockets and the TCP/IP protocol) for handling the internode communication, the communication time is low enough to have a very moderate impact on efficiency. Figure 2 illustrates the measured parallel performance of the code run on the PC cluster, for two problem sizes. Speedups are compared with the maximum allowed speedup (lines without symbols). The latter is slightly less than linear owing to the boundary planes duplicated in neighboring slices of the computational domain, but this is not a significant problem as long as the number of computing nodes is much less than the number of collocation points in wall-normal direction, and is planned to be overcome anyway in a future version of the code. Figure 2 also shows a comparison with speedups reported by (Skote, 2001) with a DNS code running on supercomputers with expensive high-bandwidth communication hardware. It is seen that the communication time, which is the main reason of the difference between the maximum allowed speedup and the actual measured speedup, has an effect on performance which is both marginal and decreasing with the number of Fourier modes in the homogeneous directions. Also the penalty deriving from the duplication of some planes is at all acceptable with 8 computing nodes.

Using a single processor of one node of the cluster, a full time step (i.e. 3 Runge-Kutta substeps) for a problem with 129 Fourier modes in the longitudinal and spanwise directions, and 129 collocation points in the normal direction, requires 50 CPU seconds. Time decreases to approximately 6.5 seconds when the entire cluster is used. The RAM requirements for such a case are 103MB.

ADJOINT DNS

A basic decision to be taken at the start of developing an adjoint code is whether to discretize the adjoint of the continuous differential problem or take the adjoint of the already discretized problem. Both are in principle possible: if the adjoint is interpreted as the derivative of some output parameter (for instance the mean friction of a turbulent flow) with respect to external actions (known terms in either the equations or boundary conditions), this definition is applicable to either the continuous problem, where the adjoint will be a functional (Fréchet) derivative, or to the discrete problem where the adjoint will be an ordinary gradient with respect to each discretized variable. There are advantages and disadvantages in either choice, and both can be found in the literature. Here we take the adjoint of the discretized problem. Doing so offers, among other benefits, the possibility of a foolproof test of the computer code: being the adjoint a derivative of the output of the complete program with respect to its input, it must give (for a specified test input) the same result as a finite difference between two direct numerical simulations run with slightly different input parameters.

If the entire direct code is seen as just a very complicated compound function linking its input and output parameters, computing the adjoint is tantamount to a successive application of the chain rule. This can be done in a more or less efficient way depending on the order given to the factors to be multiplied. If the number of output parameters (typically only one) is smaller than the number of input parameters (typically the individual values of a discretized function along the boundary), it is much more convenient to start from the output and work out backwards through the product of chained derivatives (Giles, 1999). If the underlying system of differential equations is parabolic, the numerical integration of the adjoint system will thus proceed backwards in the parabolic variable. The Navier–Stokes system of differential equations is parabolic in time, and its numerical solution proceeds in successive steps starting from given initial and boundary conditions. Whereas in principle its adjoint can be derived from the technique of Lagrange multipliers as one big system of linear equations coupling the values of the discretized adjoint variables at all points in space and times, it is much more efficient to solve it through the chain rule by moving backwards step-by-step in time.

The challenge for us was to develop an adjoint code that mimicked the structure of the direct program enough to retain its speed of execution and parallel organization. In fact, in principle an adjoint code can run in about the same time as the direct code, and this was our objective. For this purpose the modular structure of the direct computation has been crucial: the adjoint of each individual routine was developed separately, and tested against a finite difference of the corresponding direct routine.

In particular, the adjoint of a discrete Fourier transform is just another Fourier transform with suitably conjugated and/or reversed input array; it can therefore be computed through the same Fast-Fourier-Transform algorithm as the corresponding direct stage and with the same amount and technique of parallelization. The adjoint of each quadratic form that appears in the convective terms is just the easy derivative of a quadratic function. Thus, the adjoint of the routine that builds up the nonlinear explicit contribution to the time-integration scheme can be constructed with a comparable computational time and degree of parallelization as the direct code, and tested against the corresponding direct routine.

The implicit part of time integration only involves linear terms, and only one Fourier mode at a time, so that the direct computation consists in writing the matrix of coefficients of this linear system of equations and inverting it separately for each mode. As the adjoint of a matrix is just its transpose, the corresponding adjoint computation requires building exactly the same matrix for each mode and applying it with reversed indices. Again, this operation involves the same computation time and amount of parallelization as the direct code.

Finally, the recombination of the explicit and implicit stages to produce the next time step (which in the adjoint will be at a previous time) is handled by a separate routine, just as in the direct code, so as to make it possible to change the time-advancement scheme by replacing this routine only.

A problem specific to the adjoint calculation, which did not exist in the development of the direct code, is posed by storage requirements. In fact, the adjoint equations contain the direct variables in their coefficients. Whereas both the direct variables in the direct calculation and the adjoint variables in the adjoint calculation are only required at one (or a few, depending on the time-advancement scheme) adjacent time steps, which will be in the past for the direct and in the future for the adjoint computation, the direct variables can only be utilized during the adjoint calculation if the complete time history of the direct solution is stored. For a direct numerical simulation of turbulence, this can be a huge amount of disk space. To overcome this difficulty, we have devised a scheme where the number of time steps whose storage is required is reduced to the square root of its original value at the expense of a doubled direct computation time. Denoting by N^2 the total number of time steps foreseen in the simulation, we store the direct solution only every N-th step on the first run; then we start from the final time with the adjoint, and preliminary to each block of N time steps recompute the corresponding direct solution. The end result is that the complete direct solution has been used without ever storing more than $2N$ snapshots simultaneously.

RESULTS

A one-to-one test of each adjoint routine of the code against the corresponding direct routine has been performed, by first running a direct case starting with an initial condition made of random numbers, and a further direct case started from an initial condition which gives a small relative increment to the same random numbers. A finite difference of the two results is then compared against the product of the result of the adjoint computation times the initial increment, which should turn out to be the same number to within the truncation error brought about by the finite difference. This test, once positive, has then been gradually and successfully extended to the entire structure of the program, including the time integration algorithm. After tuning the amount of the relative increment given to the initial data so as to minimize the truncation error, we have verified that the difference between the two estimates can be reduced to the level permitted by rounding errors (at most half the number of digits of the floating-point hardware for a first-order difference, or 2/3 for a central difference).

Then we turned our attention to the problem of flow control, on which we only report here preliminary results. We considered the case where a suction/blowing is applied at one channel wall and the effect of this on the mean drag is to be evaluated. This suction blowing/action can be imposed through a non-zero wall boundary condition for the velocity components which preserves the condition of zero net mass flux in the wall-normal direction. The sensitivity to this suction/blowing of the time- and space-averaged wall friction is then obtained through the adjoint calculations.

The computational domain between the two walls separated by a distance 2δ is a periodic box with streamwise and spanwise size given by $L_x = 2\pi/\alpha_0$ and $L_z = 2\pi/\beta_0$, where α_0 and β_0 are the fundamental wavenumbers in each direction. The reference velocity is chosen to be the centerline velocity U_P of a laminar Poiseuille flow with the same streamwise flow rate, so that a Reynolds number can be defined as $R_P = U_P\delta/\nu$, where ν is the kinematic viscosity of the fluid. A Reynolds number of $Re_P = 4250$ is used in the simulations, corresponding to a Reynolds number of ~ 180 when based on the friction velocity u_τ. The fundamental wavenumbers are $\alpha_0 = 1/\delta$ and $\beta_0 = 2/\delta$, corresponding to $L_x = 2\pi\delta$ and $L_z = \pi\delta$. 129 Fourier modes are used for the discretization of both the streamwise and the spanwise directions. The number of collocation points in the y direction is 129, distributed over an unevenly-spaced mesh. The spatial resolution is relatively high as compared to that commonly employed in similar DNS at the same Reynolds number: $\Delta x^+ \sim 8.8$; $\Delta z^+ \sim 4.4$ and $\Delta y^+ \sim 0.75 - 4.5$. With this preliminary test case we have verified that the computing time needed for the complete calculation of the direct and adjoint solution is of the order of 3 times that needed for the direct calculation alone, corresponding to the fact that the direct solution is effectively run twice and the adjoint solution in turn takes about the same time as one direct computation.

In Figure 3 we report the time-dependence of the sensitivity of the mean friction to perturbations applied at the wall. Our preliminary results show that, after approximately 30-40 δ/U_P, the sensitivities to all three velocity components diverge expo-

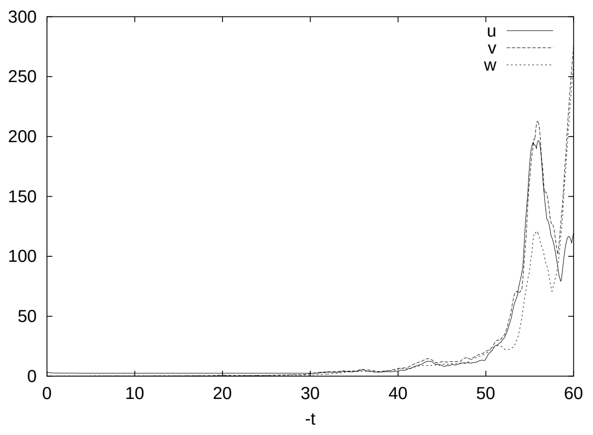

Figure 3. TIME BEHAVIOUR OF THE SENSITIVITY OF THE MEAN WALL FRICTION TO 3 VELOCITY COMPONENTS

nentially.

While the divergence of the adjoint computation was unexpected for us until we actually saw it, with hindsight it is a reasonable behavior for a chaotic dynamical system, which a numerical simulation of turbulence undoubtedly is. In fact, in a system whose dynamical phase-space trajectories diverge from each other, the derivative of the final-time solution with respect to initial conditions diverges, a concept which is made rigorous by the introduction of Liapounov exponents. Since it is this derivative that the adjoint code calculates, its explosion is just an indication that the Liapounov exponents are positive, in accord with known estimates of the latter for a turbulent flow. The possible limitations brought about by this divergence for the applications of turbulence control are still to be evaluated. It should be noted that the successful control effort of (Bewley *et al.*, 2001) used a finite horizon, reportedly shorter than the time after which we now find divergence to become noticeable, and therefore may not have incurred in this difficulty.

REFERENCES

Alvelius, K. and Skote, M. *The performance of a spectral simulation code for turbulence on parallel computers with distributed memory.* TRITA-MEK 2000:17 Royal Institute of Technology - Dept. of Mechanics - Stockholm, Sweden, 2000.

Bewley, T.R. *Flow Control: New Challenges for a New Renaissance*. Progress in Aerospace Sciences, 37:21-58, 2001.

Bewley, T.R., Moin, P., and Temam, R. *DNS-based predictive control of turbulence: an optimal benchmark for feedback algorithms* J. Fluid Mech. **447** 179–225, 2001.

Giles, M. *An introduction to the adjoint approach to design*. Invited lecture, ERCOFTAC Workshop on Adjoint Methods, Toulouse (France) ,June 21-23 1999.

Högberg, M. and Bewley, T.R. *Spatially localized convolution kernels for feedback control of transitional flows*. In Proc. 39th IEEE Conf. on Decision and Control, 4:3278–3283

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.

Quadrio, M. and Luchini, P. *A IV-order accurate, parallel numerical method for the direct numerical simulation of turbulence in rectangular and cylindrical geometries*. XV AIMETA Meeting of Applied and Theoretical Mechanics, Taormina (Italy) September 26-29 2001.

Skote, M. *Studies of turbulent boundary layer flow through direct numerical simulation*. PhD Thesis, Royal Institute of Technology, Dept. of Mechanics, Stockholm, Sweden, 2001.