Parallelization of a Modern CFD
Incompressible Turbulent Flow Code

J. M. McDonough *, Tianliang Yang,

Department of Mechanical Engineering, University of Kentucky
Lexington, KY 40506-0503, USA

M. Sheetz
Computing Services, University of Kentucky
Lexington, KY 40506 USA

Abstract

We outline the numerical analysis and turbulence modeling of a new large-eddy simulation technique for incompressible flows. It is shown that many opportunities for parallelization occur for the chosen algorithmic structure, and reasonably good speedups with increasing number of processors is observed through 32 processors for a MPI implementation on a HP SuperDome, even with only partial parallelization.

Key words: Incompressible flow, parallelization, turbulence
PACS: 02.70, 05.45Gg, 47.11.+j, 47.27.Eq

1 INTRODUCTION

At this point in the beginning of the 21st Century there are already numerous reliable, inexpensive and widely-used commercial CFD codes. Fluent, STAR-CD and Flow3D are representative examples. In light of this, one might question the need for developing yet another incompressible flow code. But it is important to recognize that in many of the most successful commercial codes the underlying Navier-Stokes (N.-S.) solution procedures and the numerical analysis used to implement them are quite old, and we contend, in some cases outdated. In particular, most (but not all) commercial CFD software is based

* 151 RGAN BLDG, University of Kentucky, Lexington, KY 40506-0503, USA

Email addresses: jmmcd@uky.edu (J. M. McDonough), tlyang@engr.uky.edu (Tianliang Yang), rmshee0@email.uky.edu (M. Sheetz).

Preprint submitted to Elsevier Science 13 August 2003
on one form or another of the SIMPLE algorithm (Patankar [1]). These basic approaches are not efficient for time-dependent simulations that are becoming increasingly more widely performed as computing power continues to improve. Furthermore, although most CFD code vendors now claim the ability of their product to employ large-eddy simulation (LES) for turbulence calculations, one essentially always finds that the manner in which LES has been implemented is far from what would be considered a true, theoretical LES, and in fact is what is sometimes termed “very large eddy simulation” (VLES), or essentially equivalently, unsteady Reynolds-averaged Navier-Stokes (URANS) modeling. Finally, and in many respects the most important point with regard to the present paper, is the fact that parallelization of these commercial codes has in almost all cases been done long after the codes were originally constructed; i.e., they were not designed to be parallelized at the time they were initially coded.

We contend that despite the clear success of many commercial CFD codes, the shortcomings noted above are of such significance to warrant development of a new code—maybe even a new generation of codes—at least for research purposes, but probably even for commercial applications in the longer term. The work to be reported here has been underway for much of the past decade, and various portions of it have previously been reported (several times in Parallel CFD Proceedings volumes). But it has only been in the past year that this work has reached the maturity needed for complete implementation of a new code. Thus, results to be reported here, especially those regarding parallelization, must be viewed as preliminary; nevertheless, they are by this time fairly substantial.

2 GOVERNING EQUATIONS AND IMPLEMENTATION

The equations being treated are the incompressible N.–S. equations:

$$\frac{DU}{Dt} = -\nabla P + \nu\Delta U + F_B \quad \text{on} \quad \Omega \subset \mathbb{R}^d, \quad d = 2, 3,$$

(1)

with the divergence-free constraint $\nabla \cdot U = 0$. Here, $U \equiv (u, v, w)^T$ is the velocity vector; $P$ is pressure divided by constant density; $F_B$ is a similarly-scaled body force vector, and $\nu$ is kinematic viscosity. The differential operators, all of which can be assumed to be expressed in generalized coordinates, are: the substantial derivative $D/Dt$, the gradient and divergence, $\nabla$ and $\nabla \cdot$, respectively, and the Laplacian $\Delta$.

Equations (1), together with appropriate initial and boundary conditions needed to constitute a well-posed problem, are solved on bounded domains $\Omega$ using staggered, structured-grid formulations consisting of Gresho’s projection
1 method [2] and standard numerical analytic techniques including trapezoidal integration in time, centered differencing in space, treatment of nonlinearities via the Newton–Kantorovich procedure and use of simple linear filters to treat cell-\(Re\) and aliasing problems (see Yang and McDonough [3]), the last two of which are not generally available—nor is use of projection—in current commercial codes.

3 TURBULENCE MODEL

The turbulence model is an updated version of the Hylin and McDonough approach [4] summarized by Sagaut [5], providing a high-fidelity form of LES that does not require wall models, and yet maintains efficiency (even with no parallelization) comparable to that of \(k-\varepsilon\) models. It is one of several possible forms of so-called “synthetic-velocity” models in which subgrid-scale (SGS) physical quantities are directly modeled, in lieu of modeling their statistical correlations. The particular method employed in the present code is derived from the additive turbulent decomposition (ATD) procedure first proposed by McDonough et al. [6] and investigated in terms of Burgers’ equation in McDonough and Bywater [7], [8]. It is clear from these references that ATD is, in reality, simply a reorganization of direct numerical simulation (DNS). Thus, for practical engineering problems McDonough et al. [9] introduced a form in which the SGS ordinary differential equations were replaced with easily-evaluated discrete dynamical systems (DDSs)—chaotic maps.

This leads to a formulation that begins with the usual LES decomposition of dependent variables:

\[
U(x, t) = \tilde{u}(x, t) + u^*(x, t), \quad x \in \Omega, \quad t \in (t_0, t_f),
\]

where \(\tilde{u}\) is the filtered, resolved-scale velocity vector. We emphasize that it is the dependent variables that are filtered, and not the governing equations as done in typical LES formulations. This eliminates the need to model SGS stresses and motivates direct modeling of the SGS physical variables, denoted here as \(u^*\).

At present the SGS velocities are modeled as

\[
u^*_i = A_i M_i, \quad i = 1, 2, 3,
\]

where \(A_i\) is an amplitude factor constructed at each resolved-scale grid point via a generalized Kolmogorov-like scaling of second-order structure functions (see McDonough [10]), and the \(M_i\) are DDSs first derived by McDonough and Huang [11] and termed the “poor man’s Navier–Stokes” equation in deference
to a similar epithet introduced by Frisch [12] to draw attention to the similarities between simple (algebraic) quadratic maps and the N.-S. equations. The realism that can be achieved with such a modeling procedure is indicated in Fig. 1 which displays streamlines of the small-scale (modeled) velocity field at one instant during a flow corresponding to decay of isotropic turbulence computed on a grid consisting of only $16^3$ points uniformly distributed in a 3-D domain of length $2\pi$ on a side.

Of particular note are the small vortices at various locations, some of which are smaller than can be resolved on the grid being employed. They are generated by a combination of high-pass filtering of the resolved-scale solution, and the model described above.

If it were necessary that large and small scales be computed sequentially, run times would be approximately double those of laminar flow calculations. But, in fact, the two scales can be computed simultaneously by starting each new time step with known results from the previous time step on both scales and combining these according to Eq. (2) at the end of the time step. Hence, with only two processors it is at least formally possible to keep run times below those needed for $k-\varepsilon$ methods.

4 PARALLELIZATION

The most important aspect of this new code with respect to the present study is that it was designed to be parallelized, and even the earliest version of the code (McDonough and Dong [13]) was implemented and tested in parallel.
mode. As discussed in [13], the basic strategy for parallelization has been to employ a re-interpretation of Douglas and Gunn [14] time splitting. In particular, the approach emphasizes planar (rather than line) solves (done in parallel by multiple instances of the 2-D code) to provide sufficient work to keep processors busy long enough to significantly reduce the ratio of communication time to processing time, thus achieving much improved parallel performance.

To see how this can be carried out, recall that in Douglas and Gunn time splitting a 3-D coefficient matrix can be decomposed as

$$ A = A_x + A_y + A_z, \quad (4) $$

where subscripts denote grid directions in which differencing has been done, and along which solutions must be computed. This results in three separate sets of tridiagonal systems to be solved over the entire 3-D grid. These tridiagonal solves are all independent of one another in any particular direction, but, especially in 3-D, the corresponding linear systems tend to be quite small making parallelization less straightforward—and less effective. But it is shown in [13] that by defining $A_{x+y} = A_x + A_y$, so that Eq. (4) becomes

$$ A = A_{x+y} + A_z, \quad (5) $$

solution of Eqs. (1) can be accomplished by first performing 2-D solves corresponding to $A_{x+y}$ followed by the usual line solves associated with $A_z$.

MPI has been employed for parallelization of this code; Fig. 2 displays the strategy associated with . Three processors are employed here to demonstrate the parallel computational procedures. Each processor performs calculations for a specific number of planes in each time-split step. The figure shows the schematic of data structure in each processor and the calculation procedure performed by each processor (the third, $x$-direction, is out of the page). At the beginning of each time step, only shaded regions of the three-dimensional arrays have the valid velocity and pressure data. In the second step, it is recognized that $A_{x+y}$ in Eq. (5) can be evaluated, and 2-D solves corresponding to $A_{x+y}$ can be performed, on each $x$-$y$ plane independently. Each processor is employed to do these calculations for a specific number of $x$-$y$ planes, and an intermediate result, $\delta U_1$, is obtained and stored in the corresponding processors. Since $\delta U_1$ is stored separately on three processors, and line solves associated with $A_z$ will be conducted in $x$-$z$ planes, $\delta U_1$ must be redistributed among the processors. In this procedure, each processor sends several sections of $\delta U_1$ to other processors and receives several parts of $\delta U_1$ from other processors (third step in Fig. 2). In the fourth step, each processor is employed to calculate $A_z$ and perform 1-D solves corresponding to this matrix for a specific number of planes to obtain $\delta U$. The fifth step is similar to the third one. The solution vector $\delta U$ is redistributed among the processors and the final distribution of $\delta U$ is shown in Fig. 2. The sixth step is to update $\bar{U}$ to obtain
\( \tilde{U} = U + \delta U \). After this step, convergence of Newton-Kantorovich iterations is tested. If convergence has not been achieved, the procedure returns to step two; otherwise, the projection step will be carried out as follows.

Once \( \tilde{U} \) is calculated, the pressure Poisson equation is solved using red-black ordered SOR. The corresponding processors are used to treat the same \( x-y \) planes treated in performing the 2-D solves corresponding to \( A_{x+y} \) because velocity and geometry data are stored there before starting SOR. Between the red and black iteration steps, the values of \( p \) on the first plane and last plane in each processor are sent to corresponding planes on other neighboring processors to maintain usual red-black SOR iteration in the whole computational domain.

5 RESULTS

Figure 3 presents the speedups obtained up to this point on a 224-processor HP SuperDome operated by the University of Kentucky Computing Center. The data displayed in this figure correspond to four different tests of parallelization employed to solve a 3-D lid-driven cavity problem posed on the unit
cube. On each of the $32^3$ and $64^3$ grids employed, runs were made in both dedicated and non-dedicated modes. Overall, one observes very little difference in performance from one case to the next except in the comparison of dedicated and non-dedicated processing for the $64^3$ grid using 32 processors; in this case, dedicated processing is far superior. Furthermore, we should note that although performance is far from the perfect linear speedup, it nevertheless continues to improve significantly through 32 processors in the dedicated mode of operation. Moreover, it is important to emphasize that these parallel performance results correspond to a laminar flow case. Further improvement, with regard to parallelization, is anticipated for the complete turbulent flow code for reasons noted above.

References


4. E. C. Hylin and J. M. McDonough, “Chaotic Small-Scale Velocity Fields as Prospective Models for Unresolved Turbulence in an Additive Decom-


