Advances in Applied Mathematics and Mechanics
Adv. Appl. Math. Mech., Vol. **3**, No. 3, pp. 370-388

DOI: 10.4208/aamm.10-10s2-07
June 2011

High Level Languages Implementation and Analysis of 3D Navier-Stokes Solvers

Valerio Grazioso[1,*], Carlo Scalo[1], Giuseppe de Felice[2] and Carlo Meola[2]

[1] *Department of Mechanical and Materials Engineering, Queen's University, 130 Stuart Street, Kingston, Ontario, Canada*

[2] *Dipartimento di Energetica Termofluidodinamica applicata e Condizionamenti ambientali (DETEC), Università degli Studi di Napoli 'Federico II', P.le Tecchio 80, 80125 Naples, Italy*

Received 29 January 2010; Accepted (in revised version) 28 October 2010
Available online 28 February 2011

Abstract. In this work we introduce PRIN-3D (PRoto-code for Internal flows modeled by Navier-Stokes equations in 3-Dimensions), a new high level algebraic language (Matlab®) based code, by discussing some fundamental aspects regarding its basic solving kernel and by describing the design of an innovative advection scheme. The main focus was on designing a memory and computationally efficient code that, due to the typical conciseness of the Matlab coding language, could allow for fast and effective implementation of new models or algorithms. Innovative numerical methods are discussed in the paper. The pressure equation is derived with a quasi-segregation technique leading to an iterative scheme obtained within the framework of a global preconditioning procedure. Different levels of parallelization are obtainable by exploiting special pressure variable ordering patterns that lead to a block-structured Poisson-like matrix. Moreover, the new advection scheme has the potential of a controllable artificial diffusivity. Preliminary results are shown including a fully three-dimensional internal laminar flow evolving in a relatively complex geometry and a 3D methane-air flame simulated with the aid of libraries based on the Flamelet model.

AMS subject classifications: 65M10, 78A48, 76M99.

Key words: Matlab, high level algebraical languages, segregation, preconditioning, flamelet.

1 Introduction

The vast majority of present CFD codes have been developed in what would be con-

*Corresponding author.
URL: `http://me.queensu.ca/people/piomelli/research/TSMLab.php`
Email: graziosov@me.queensu.ca (V. Grazioso), cscalo.ca@gmail.com (C. Scalo), giudefel@unina.it (G. de Felice), cmeola@unina.it (C. Meola)

sidered nowadays as low-level programming languages like C or Fortran. This allows for faster performances, and sometimes, for the creation of machine specific highly efficient executables. On the other hand, high-level or very high-level languages allow for higher level of abstraction from machine language. Important features include runtime interpretation and debugging, handling of more generic data structures, symbolic math toolboxes and intermediate code files. The well known drawbacks fall into the class of the so called abstraction penalties: slower execution speed, higher memory consumption, larger binary program size [4]. However, as computational resources get more sophisticated but yet more available and affordable, newly developed high-level programming languages become faster and easier to use and have the potential to be considered a valid candidate for code developing.

In order to design a CFD code that would enable the user to perform several tasks ranging from algebraic analysis of the equations' structure to modular implementation of virtually any kind of fluid dynamic model, it is therefore convenient to adopt high-level algebraic languages (like Python, Scilab, Octave, etc). For our purposes we considered Matlab ® as being the most appropriate option. This choice was made considering the opportunities that this kind of programming language offers in terms of easy user interfacing, fast libraries, packages integration (i.e., NAG, LAPACK, UMFPACK, etc) and increasing usage of natively multi-threaded functions (such as the fftw-based FFT). In comparison with low-level programming languages (like C or Fortran) many pre-compiled subroutines are available to the user and a very high-level of abstraction from the details of the computer is possible with, for instance, structures, cell-arrays and object oriented programming. The most important feature of Matlab remains its natural and native handling of fundamental linear algebra objects like matrices, and the extensive amount of libraries and functions available for a lot of simple and complex algebraic operations such as matrix multiplication, inversion or eigenvalues extraction. Moreover, Matlab has been developing parallelization capabilities (distributed and shared memory) by providing specifically designed toolboxes (MPI based) and packages (like Star-P®).

The choice of algebraic solving strategies is strongly dependant on the choice of the Navier-Stokes solver which is determined by the class of fluid dynamic problems one intends to solve. A first general distinction may be made among fully implicit, semi-implicit and fully explicit time discretization strategies. For steady laminar flows the transient evolution from an initial condition is usually not being resolved and large time-steps are used when possible; this requires methods that penalize time accuracy but that are, possibly, unconditionally stable in time allowing to reach steady state in the least amount of computational time. The best choice in this case would be SIMPLE-based solving techniques like PISO, SIMPLEC [2] in which both diffusive and (linearized) convective terms are treated implicitly allowing to remove CFL time constraints as well as viscous ones. For eddy resolving turbulent flow simulations the constraints on the time-step are determined by the need to accurately resolve the time scale of the smallest resolved eddies. The required time step is typically smaller than what determined by viscous and CFL constraints and, therefore, fully implicit