MICSc, a PETSc-based parallel code for Large Eddy Simulation

A. Cubero[1], J.E. Román[2], V. González[2], N. Fueyo[1], and G. Palau-Salvador[2]

[1]Universidad de Zaragoza, Spain
[2]Universidad Politécnica de Valencia, Spain

Abstract

The paper presents a new open-source Large Eddy Simulation (LES) code -MISCs- which solves the Navier-Stokes equations using a coupled and implicit method. The code is oriented to solve incompressible (constant density) flows and also compressible flows at low Mach numbers. The code has been validated by using the open-channel flow investigated by Moser *et al.* [1] whose DNS data show a good agreement with the results of our current LES simulation. The system of equations has been solved by the Portable, Extensible Toolkit for Scientific Computation (PETSc) and the performance of different combinations of linear solvers and preconditioners has been tested. The combination of Block Jacobi with either BiCGStab or GMRES solvers shows the fastest convergence. The parallel efficiency of the whole simulation is very satisfying up to 64 processors.

Keywords: CFD, LES, coupled, implicit, parallel, PETSc, computational.

1 Introduction

Computational Fluid Dynamics (CFD) is the analysis of fluid flow, heat transfer and associated phenomena (such as chemical reactions, sediment or scalar transport). The technique is very powerful and spans a wide range of industrial and non-industrial applications: aerodynamics, hydrodynamics, environmental fluid mechanics, mechanical engineering or combustion [2]. From the 1960s, the research of CFD has evolved principally from aerodynamics to many other fields. The first approaches were based

in steady-flow assumptions. However, there is nowadays an increasing demand for high fidelity, unsteady CFD capabilities for turbulent flows. Conventional Reynolds Averaged Navier Stokes (RANS) solvers based on various turbulence models often fail to capture unsteady flow physics accurately. This is not surprising in view of the fact that most of these models were developed with the goal of solving steady flow problems. Alternative methods are needed for unsteady CFD analyses in industrial applications.

As computer power becomes more affordable, Large Eddy Simulation (LES) has emerged as a viable and powerful alternative tool in turbulence computations. In recent years, LES has been applied to an increasing number of problems of engineering relevance. This was made possible through the use of parallel computing. The challenge in carrying out LES is that a three-dimensional, unsteady calculation must be carried out on a grid capable of resolving the larger scales of the motion; for flow geometries and Reynolds numbers of engineering interest, this implies that the grid is usually large. Hence, the CPU time required is substantially larger than that for an analogous RANS calculation. Moreover, LES applications have been presented in studies of jets [3], flow around obstacles [4], sediment transport [5] or scalar transport [6]. On the other hand, the Portable, Extensible Toolkit for Scientific Computation (PETSc) has successfully demonstrated that the use of modern programming paradigms can ease the development of large-scale scientific applications. The software has evolved into a powerful set of tools for the numerical solution of partial differential equations and related problems on high-performance computers. The applications of PETSc [7] in engineering has shown as a very strong and efficient tool in fields so spread as Nano-simulations, Biology/Medicine, Fusion, Geoscience, Environmental flows, CFD or Optimization.

The objective of the present project is to develop an open-source code for LES applications using the PETSc numerical library to solve the Navier-Stokes equations. The Standard Smagorinsky model is used to resolve the subgrid scale motions (SGS) which was then tested by application to a fully developed channel flow [1].

2 Governing equations and discretization algorithm

The present version of the MICSc code can solve incompressible (constant density) flows and also compressible flows at low Mach number (with density depending, typically, on temperature). The Navier-Stokes equations for this kind of flows can be written as follows:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_i}{\partial x_i} = 0 \qquad ; \tag{1}$$

$$\frac{(\partial \rho v_i)}{\partial t} + \frac{\partial (\rho v_i v_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau'_{ij}}{\partial x_j} - \rho g_i \qquad ; \tag{2}$$

$$\frac{(\partial \rho T)}{\partial t} + \frac{\partial (\rho v_j T)}{\partial x_j} = \frac{\partial}{\partial x_j}(\kappa \frac{\partial T}{\partial x_j}) \qquad , \tag{3}$$

where g_i is the i-component of gravity acceleration, κ is a diffusion coefficient (assuming the specific heat capacity is constant). The viscous stress tensor τ'_{ij} is:

$$\tau'_{ij} = \mu \left\{ \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial v_l}{\partial x_l} \right\} \qquad , \tag{4}$$

with μ being the fluid viscosity.

Furthermore, conservation equations for an arbitrary number of transported variables (scalars) can be added to the system (for modelling, *e.g.*, pollutant transport in river beds or chemical species present in combustion processes).

MICSc applies the finite volume approach for discretizing these equations. This method is based on the partition of the problem domain to form a computational grid and the application of the conservation equations (in integral form) to each cell in the grid. Advantages such as versatility or easy of incorporation of physical models make it one of the most frequently used methods in CFD codes (*e.g.* [8]).

As shown in Equation 5, a discretized equation for a generic transported variable (ϕ) on a given cell (P) involves the summation of convective and diffusive fluxes over all cell faces (neighboring, nb) and volumetric accumulation due to temporal evolution, external forces or sources. Figure 1 shows notation for the cell (with capital letters referring to cell center and small letters to cell faces).

$$\left.\frac{\partial \rho \phi}{\partial t}\right|_P V_P + \sum_{nb(P)} \dot{m}_{nb}\phi_{nb} - \sum_{nb(P)} \Gamma^{\phi}_{nb} A_{nb} \left.\frac{\partial \phi}{\partial x_{nb}}\right|_{nb} = S^{\phi}_P V_P \qquad . \tag{5}$$

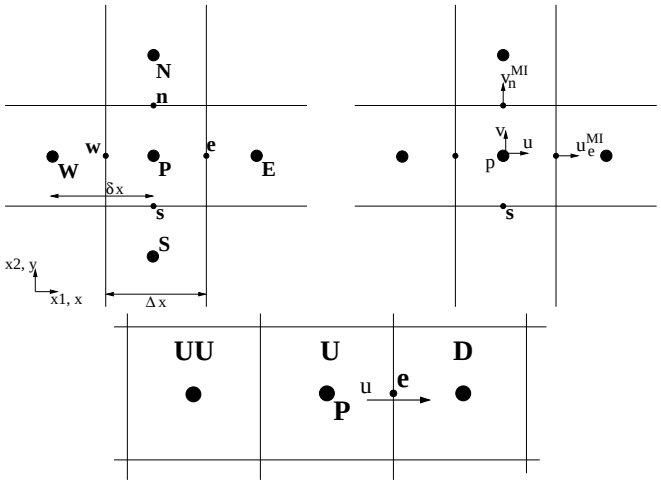

Figure 1: Cell notation (left), variable arrangement (right) and labelling scheme for the nodes involved in the High Resolution Convection Schemes (HRCS, below).

The primitive variables (velocity components, pressure and, if applicable, other scalars) are chosen as unknowns and all of them are stored at the cell centers (collocated-grid approach). For brevity, only the most relevant aspects of the discretization algorithms implemented in MICSc are described in this paper.

The face value of a generic transported variable (ϕ_{nb}) is related to neighboring cell centers (unknown nodes) by means of a convection scheme. Although a linear interpolation is usually recommended for LES, MICSc also provides (apart from the first order Upwind scheme) a variety of Higher Resolution Convection Schemes (such as for example SMART, MUSCL, QUICK, Van-Leer or Notable). The unified flux-limiter approach reported in [9] is applied to build bounded schemes; the approach takes into account the direction of the flow and involves three neighbouring cells (the upwind and downwind nodes and one more node located upwind to the upwind cell; see Figure 1).

On the other hand, in order to avoid chessboard-like pressure contours, an undesirable numerical phenomenon arising on collocated grids, the momentum interpolation procedure (originally proposed by Rhie and Chow [10]) is used to calculate the convecting velocities (those involved in the mass-flow rates $\dot{m}_{nb}$ across the cell faces; u_e^{MI}, v_n^{MI} in Figure 1). MICSc makes use of an improved formulation (Compact Momentum Interpolation, [11]) which avoids these spurious pressure oscillations when solving unsteady flows at very small time steps (as is commonly the case for Large Eddy Simulations of turbulent flows).

Regarding the temporal integration of solving unsteady flows, implicit discretization schemes are implemented. This choice avoids restrictions to the maximum time step imposed by numerical stability requeriments (as in explicit temporal schemes). Thus, some computational time might be saved in Large Eddy Simulations of turbulent flows (technique described in Section 3) since a larger time step may be chosen by just considering physical temporal scales, and simulation of compressible flows at low Mach number can be made computationally affordable. MICSc allows using multi-point implicit temporal schemes, such as first and second order Euler schemes, and also second and third order Adams-Moulton approximations.

3 Large Eddy Simulation for turbulent flows

When solving turbulent flows, a direct numerical simulation as described in the previous section becomes computationally unaffordable for the majority of engineering problems. Time and spatial discretization must be extremely fine to capture the large range of structure sizes in turbulent phenomena, from the largest scales of the particular problem to the smallest ones, where transport phenomena at the molecular level becomes most relevant. Some modeling must be therefore incorporated in order to simulate turbulent flows. While the classic Reynolds-average approach is a practical design tool to ascertain general trends, the technique known as Large Eddy Simulation (LES) is arising as an accurate predictive tool, potentially usable for industrial

applications [12].

Large-Eddy Simulations solved the filtered the Navier-Stokes equations, so that onlu the large scales of motion are directly simulated, while the effect on them of the rest of (small) scales is modeled. Nevertheless, some difficulties remain that prevent a wider application of this technique to industrial flows, such as, for example, the use of filtering in unstructured grids, wall-modeling strategies or its high computational cost. The latter is brought about because Large-Eddy Simulations require solving unsteady flows with (still) rather fine time steps and spatial grids.

When the finite volume method is used, the filtering process is usually carried out implicitly by the grid. An equation system similar to Equations 1-3 for the filtered variables $\overline{\phi}$ is resolved, except for an extra term, known as sub-grid scale stress $\tau_{ij} = \overline{v_i v_j} - \bar{v}_i \bar{v}_j$, which must be modeled.

The present version of MICSc uses the Smagorinsky model, frequently featured in industrial applications (mainly due to its simple implementation and a low additional cost). It is based on the definition of a turbulent viscosity, ν_T, that represents the dissipation of the energy of the large structures by the non-resolved scales:

$$\tau_{ij} - \frac{\delta_{ij}}{3}\tau_{kk} = -2\nu_T \overline{S}_{ij} \qquad , \tag{6}$$

where $\overline{S}_{ij}$ is

$$\overline{S}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) \qquad , \tag{7}$$

and

$$\nu_T = (C_S \Delta)^2 |\bar{S}| = (C_S \Delta)^2 (2\overline{S}_{ij}\overline{S}_{ij})^{1/2} \qquad . \tag{8}$$

Here Δ is the filter size (usually defined as $\Delta = \Delta x \Delta y \Delta z^{1/3}$) and C_S is the Smagorinsky constant, ranging from 0.065 to 1.1 depending on the type of turbulent flow.

4 Coupled and implicit solution algorithm

The discretization method described in Section 2 results in an algebraic equation system for each conservation equation. Special iterative solvers, suitable for this kind of large and sparse matrices are available [13]. Nevertheless, the Navier-Stokes equations are a coupled and non-linear system and some matrix processing is required before applying a linear iterative solver.

Velocity and pressure are coupled in momentum conservation equations, where the pressure gradient is acting as an external force. The equations governing incompressible flows, however, lack a specific equation for pressure since the continuity equation does not explicitly contain the pressure variable. As a consequence, the coefficient matrix presents zeros on the main diagonal and solving the coupled system becomes a challenge. Traditional algorithms (SIMPLE-like methods) apply a segregated procedure for overcoming this difficulty [8]. Momentum conservation equations are solved

separately, assuming an initial pressure field, and an iterative procedure (including relaxation) is required until mass conservation is assured.

As interest from industry in CFD increases, more robust algorithms are required; and the current trend is towards developing coupled algorithms, able to solve the Navier-Stokes equations [14] simultanously. MICSc includes such a type of algorithm. Some of the fundamentals of the main MICSc (Momentum Interpolation based Coupled Solver, using PETSc) algorithm are briefly outlined in the following (a detailed description can be found in [15]).

A Poisson-like equation for pressure is derived from the continuity equation. Mass-flow rates at faces are calculated according to the Compact Momentum Interpolation mentioned in Section 2. The resulting pressure equation (without considering, for simplicity, relaxation or temporal terms) can be expressed as:

$$\sum_{nb(P)} \rho_{nb} A_{nb} \lfloor v \rfloor_{nb} + \sum_{nb(P)} \rho_{nb} A_{nb} \left\{ \left\lfloor \frac{1}{a} \right\rfloor_{nb} f_{nb} - \left\lfloor \frac{f}{a} \right\rfloor_{nb} \right\} = 0 \qquad , \qquad (9)$$

where $\lfloor x \rfloor$ represents a linear interpolation, f accounts for pressure forces ($f_{nb} = -A_P p_P - A_{NB} p_{NB}$) and a is the coefficient on the main diagonal of the corresponding momentum equation. In this way, the main goal of removing zeros on the main diagonal of the coefficient matrix can be achieved.

MICSc copes with the non-linear momentum equations by applying a successive substitution (or Picard) linearization procedure. A large linear system must be solved for a number of non-linear iterations (until convergence), and (when solving an unsteady flow) for each time step. Although MICSc makes use of the parallel, efficient and robust Krylov-subspace solvers (and preconditioners) provided by the PETSc toolkit (described in Section 5), some additional techniques are incorporated in order to improve the conditioning of the coupled system matrix.

Diagonal dominance (a sufficient condition for the convergence of stationary iterative linear solvers) is enforced by applying pseudo-temporal terms (also known as inertial relaxation) to all conservation equations. Interestingly, in the case of the pressure equation, this technique is analogue to the combination of pseudo-artificial compressibility and dual-step approaches used in algorithms for compressible flows when solving problems at low Mach numbers. For the same purpose (diagonal dominance), MICSc implements High Resolution Convection Schemes (mentioned in Section 2) in a deferred way, so that only the Upwind part of the interpolation is included in the matrix coefficient (the other terms are calculated from values stored at the previous iteration and deferred to the right-side-hand vector). Finally, a deferred implementation is also applied in the Poisson-equation (Equation 9) to pressure values not located at adjacent nodes, so that the matrix structure is not banded with additional non-zero diagonals. At convergence of the non-linear iterations, the original expressions are retrieved.

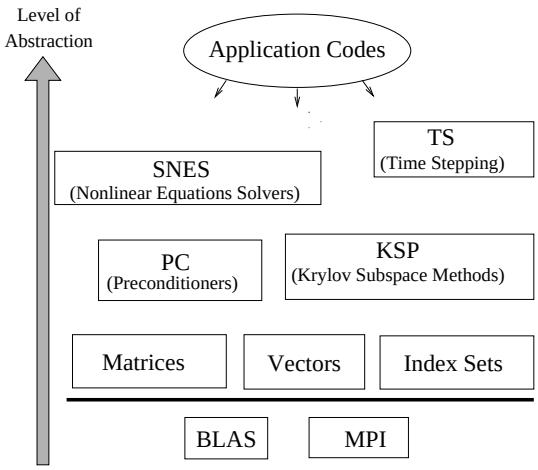

Figure 2: Diagram showing the different abstraction levels covered by PETSc components.

5 Linear solvers and parallelization strategy: PETSc

As in most CFD codes, the high computational demands make parallelization necessary. Our approach is to rely on MPI [16], the standard for message-passing parallel programming in distributed-memory platforms. The benefits of MPI are well known, such as portability to virtually all parallel computers, scalability to thousands of processors, and a rich and flexible application programming interface.

Developing an efficient message-passing code is not trivial. Issues such as load balancing or minimization of communication overhead must be taken into account. On the other hand, some numerical algorithms such as linear solvers are tricky to implement, especially in parallel, and providing numerically robust implementations requires advanced skills. For these reasons, it is highly advisable to make use of parallel libraries or frameworks that offer tools to help developing parallel numerical simulation codes. One such toolkit is PETSc, which has been used in this work.

PETSc, the Portable Extensible Toolkit for Scientific Computation [7], is a parallel framework for the numerical solution of problems arising in applications modeled by partial differential equations. Its design follows an object-oriented approach in order to be able to manage the complexity of numerical methods for very large and sparse problems on parallel computers [17]. In PETSc all the code is built around a set of objects that encapsulate data structures and solution algorithms, see Fig. 2. The application programmer works directly with these objects rather than concentrating on the underlying data structures. In this way, it is possible to work at a high level of abstraction, leaving most details related to parallelization hidden within the objects. Next, we describe the main PETSc objects and the offered functionality.

The data objects include management of index sets, vectors and sparse matrices in different formats, as well as basic support for structured and unstructured meshes. The vector object is used to store a parallel discrete representation of a field such as the

pressure, the velocity components, or any other magnitude associated to the discretization mesh. In general, PETSc vectors are distributed among MPI processes in a way that a roughly equal number of unknowns is assigned to each process. For the particular case of structured meshes, such as the one discussed in this paper, PETSc provides some additional helping tools to aid in the parallelization. In that case, one can think of a field as distributed in a domain-decomposition style, that is, each process owns the unknowns corresponding to a compact subdomain. Storage is allocated also for ghost values, *i.e.*, a halo of unknowns surrounding the subdomain that stores values belonging to neighboring processes. During the computation, the ghost values are eventually updated, which implies parallel communication among neighboring processes, and this communication is managed automatically by PETSc's internal data structures. Regarding the matrix data structures, they follow a row-oriented distribution, compatible with the distribution of vectors. Matrix-vector products are performed very efficiently, with minimal communication across processes. PETSc also takes care of important implementation details such as memory allocation and automatic assembly of elements during creation of matrix objects.

Built on top of the data objects are various classes of solver objects, such as linear, nonlinear and time-stepping solvers. The nonlinear solvers include various Newton-type methods, with explicit or implicit Jacobian, and the time-stepping solvers include some well-known ODE methods such as Euler or Runge-Kutta. In this work, neither of these have been used since the linearization and time discretization are carried out by the code itself, as described in section 6. Hence, we focus our description on linear solvers.

For solving linear systems of equations, PETSc provides a long list of iterative methods, that can be combined with different preconditioners. All iterative solvers belong to the class of Krylov methods. Examples of such methods are the Conjugate Gradient, GMRES, BiCGStab, and TFQMR. The user can choose among them very easily, via command-line switches, without having to rebuild the program. The performance of these algorithms, *e.g.*, in terms of convergence, can vary widely depending on the numerical properties of the system of equations. It is very important to use an appropriate preconditioner in order to improve convergence. PETSc provides several parallel preconditioners, based on substructuring and domain decomposition, such as block Jacobi or additive Schwarz, with incomplete factorizations within each block. Apart from the mentioned methods, PETSc also provides infrastructure for implementing multi-grid linear solvers.

PETSc is being used around the world in many application areas, including CFD computations, see for instance [18].

6 MICSc parallel code: description

MICSc is a parallel code written in C which uses PETSc to solve the Navier-Stokes equations with the coupled and implicit method explained in section 4. It focuses

on the LES approach for incompressible (and compressible with low Mach number) flows.

The code is able to write output files with the results of the simulation in TecPlot format, as well as binary files. Additionally, an ongoing development effort will confer MICSc the ability to write the output files using HDF5 and SILO technology.

On the other hand, there are two input files for the application. The first one is the parameters file (usually rod.inp), where all the parameters of the problem together with the rest of the logging and extra options are specified. It is also possible to specify these parameters at runtime from the command line and these will override the input file parameters.

The second input file is the mesh (usually grd.inp). This file has an specific format for this application and it consists of the coordinates of the points, the type of mesh (Cartesian or cylindrical) and the boundary conditions, whose type must be one of `WALL`, `MOVING_WALL`, `INLET`, `OUTFLOW`, `OUTPRESS`, `SYMMETRY` and `FIX_VALUE` (used to fix the value of the variables in a certain region of the mesh). Together with the development for the HDF5/SILO output, there is an ongoing work to include MED files created with Salomé.

6.1 Data structures

Most of the MICSc data structures are the aggregation of several pieces of information corresponding to problem elements such as the mesh, variables, etc. The main data structures are those representing the critical information of the problem: the geometry and coordinates of the mesh, the variables to solve, the properties, the boundary conditions and the system of equations. All of them are defined as follows:

St_Bcond: Boundary condition, defined in a region of the mesh (*patch*). Includes the equation and variable where the condition applies and two properties (explained below) used to evaluate the implicit and explicit coefficients for the system matrix.

St_Geom: Geometry properties of the mesh, such as volume, edge length and surface area for each cell of the mesh.

St_Grid: Node information of the mesh: number of nodes and coordinates for each dimension of the problem. It also defines the type of mesh (Cartesian or cylindrical).

St_Prop: Property defined in a region of the mesh (*patch*). The type of property can be constant or variable and, if variable, it can depend on the geometry, the variables or the transient terms. The main purpose of the properties is to evaluate them when necessary (depending on the type) and introduce them into the system. Moreover, it is important to notice that properties can be used separately (*e.g.*, density) or associated to boundary conditions or variables (*e.g.*, initial values or diffusion coefficient).

St_Sys: This data structure contains PETSc objects used to store the coefficient matrix (`Mat`) and the right-hand side vector (`Vec`) of the linear system, as well as solver objects (`KSP`, `PC`) that will be used for its resolution.

St_Var: Parameters of a variable (or unknown). Includes values such as relaxation coefficients and maximum/minimum limits or properties for the initial values and diffusion coefficient. It also uses a `Vec` object of PETSc and allocates memory to store the variable values when necessary.

6.2 Execution scheme

The execution scheme used by MICSc to perform the simulations is divided in three different sections, clearly distinguishable and common to most of the scientific applications: initialization, resolution and finalization. The scheme is illustrated graphically in Figure 3.

At the beginning of the execution there is a section where the input files are read, data structures are created and variables are set to their initial values.

Next, in the second section of the flux diagram, we find the core of MICSc, where the Navier-Stokes equations are solved by implementing the time stepping using implicit time integration and the linearization of the equation system via Picard iteration. For the time integration it is possible to choose between several implicit methods such as Euler (first and second order) and Adams-Moulton (second and third order). Moreover, the code is able to solve steady flows where time integration is not needed and it can be specified as `NONE` for the integration method. The implementation of the implicit time integration methods is represented in the source code by the most external loop where the stop condition (for non-steady flows) is given by the final time, specified as input parameter. Inside the time integration loop we have the linearization by Picard iterations (also known as 'outer iterations') where the stop condition is either achieve convergence or reach the maximum number of outer iterations, specified as an input parameter. For each outer iteration, a coupled system of equations is solved using the coupled variables (usually velocities and pressure) although it is also possible to solve a segregated system for each non coupled variable. Since these systems are linear, the KSP and PC objects provided by PETSc are used to solve them in an efficient and transparent manner, also giving the possibility to choose from different Krylov subspace solvers at runtime. However, in order to solve the system, it has to be created previously by evaluating all the properties and introducing all the terms (boundary conditions, relaxations, ...) in the coefficient matrix and the right-hand side vector. Then, after the resolution, the variables are limited to their maximum/minimum values if necessary.

Finally, in the third section, all the finalization tasks are performed, such as finalize PETSc objects or free data structures' memory.

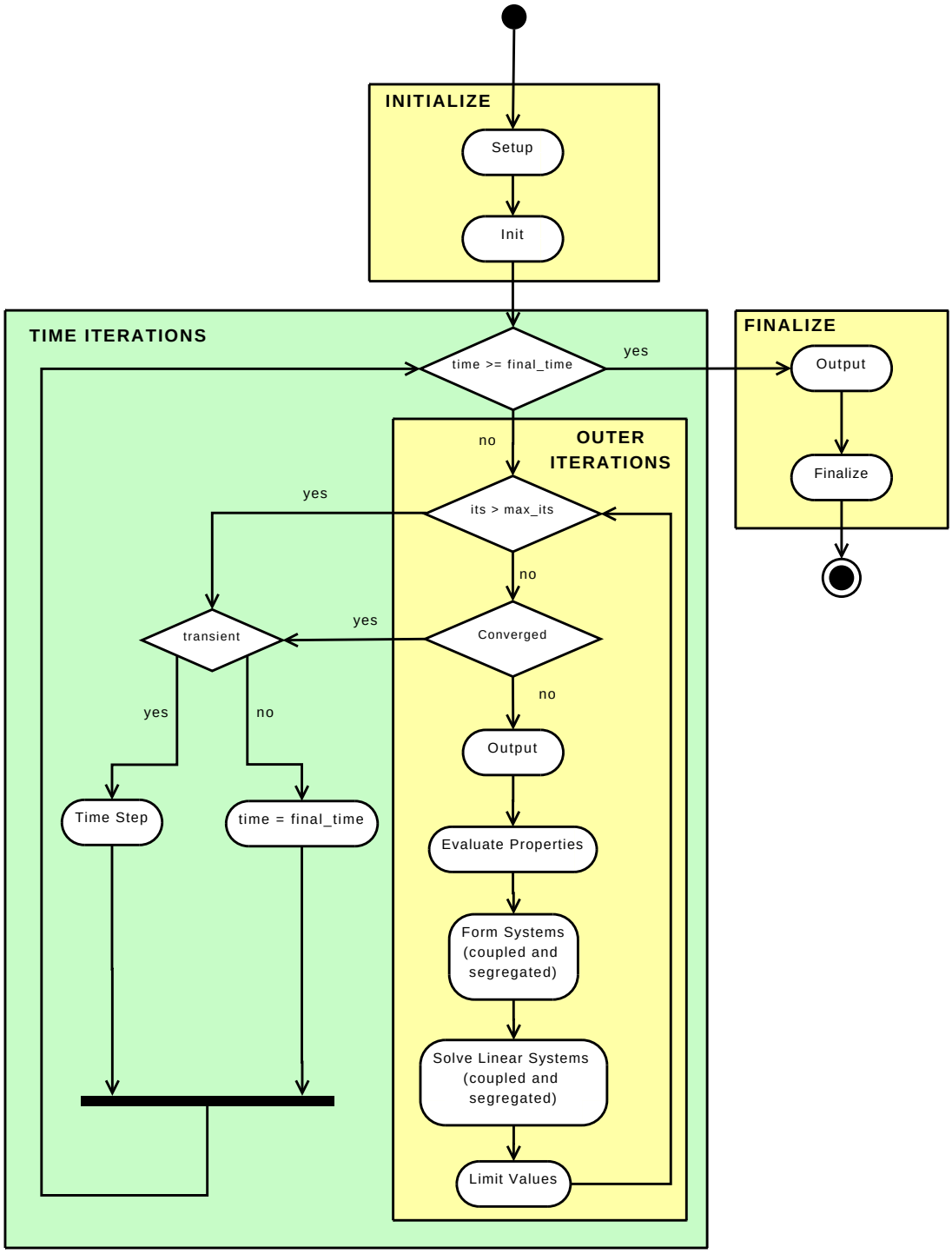

Figure 3: Execution scheme of the MICSc code.

7 Results

This section presents the validation results and the performance analysis of the MICSc code when applied to a large eddy simulation of a channel flow.

7.1 LES validation: periodic channel case

The fully developed turbulent flow in a plane channel is a common validation case for Large Eddy Simulations. At low Reynolds numbers, Direct Numerical Simulations of this wall-bounded flow can be performed and statistical data are available for comparison. In this paper, some preliminary results obtained with MICSc are compared with DNS reference data reported by Moser *et al.* [1].

A grid with $121 \times 121 \times 81$ nodes in respectively the streamwise (x), wall-normal (y) and spanwise (z) directions is used to discretize a computational domain of dimension $2\pi\delta \times 2\delta \times \pi\delta$. Grid stretching has been applied in the normal direction by a hyperbolic tangent function.

Periodic boundary conditions are imposed in x and z directions, and no-slip conditions at the walls. The pressure gradient that drives the flow is forced by means of a (volumetric) source in the x-momentum equation.

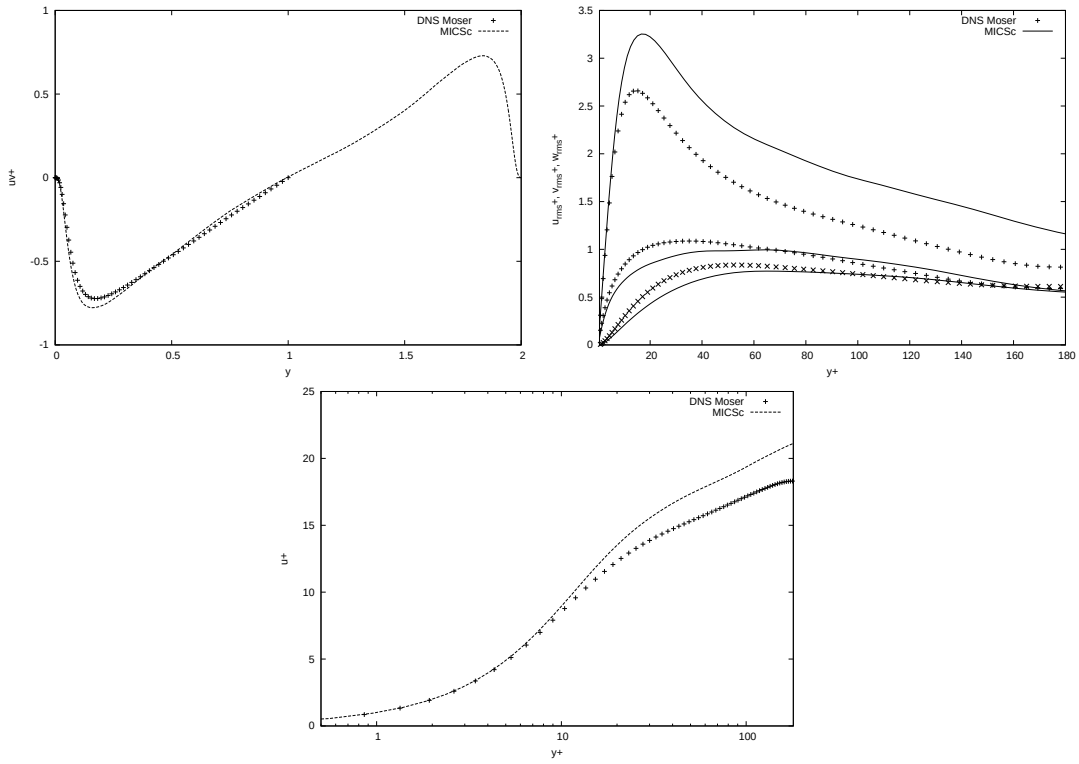

Figure 4: Normal profiles of streamwise velocity (below), Reynolds stress (left) and turbulent intensities (right). Comparison of MICSc results and DNS reference data

Sub-grid stresses are calculated using the Smagorinsky model (described in Section 3), being the $C_S = 0.065$ and the filter size $\Delta = \Delta x \Delta y \Delta z^{1/3}$. The well-known Van-Driest damping function is also applied to reduce the turbulent viscosity near the walls.

A linear interpolation scheme (CDS) is used as for the convection terms, and the the temporal integration is carried out with the implicit, second-order Euler scheme. The flow evolves (from the imposed initial field) towards the statistically steady state, a fully developed turbulent flow at a Reynolds number (in wall units) of $Re_\tau = 224$. Figure 4 shows (time-averaged) wall-normal profiles of mean streamwise velocity ($< u^+ >$), turbulent intensities ($< u^+_{rms} >$, $< v^+_{rms} >$, $< w^+_{rms} >$) and Reynold stress ($< u'v'^+ >$), normalized by the wall shear velocity u_τ. For comparison, DNS profiles by Moser *et al.* of the turbulent channel at $Re_\tau = 180$ (a lower Reynolds number) are also presented. Although some additional simulation time would be required to obtain smoother profiles, the LES results obtained with MICSc are satisfactory: the overestimation of the mean streamwise velocity (and, therefore, also of the fluctuations u_{rms}) at the core of the channel is common in LES; both the location and the values of the peaks in the other turbulent intensities are captured with an error smaller than 15%.

7.2 MICSc code performance

In this section, we present some time measurements in order to discuss the performance of the code, especially in terms of parallel efficiency. In order to avoid long simulations, we have considered only the first 30 time steps, which comprise 360 linear solves in total. The tests are executed on CaesarAugusta, a machine consisting of 256 JS20 blade computing nodes, each of them with two 64-bit PowerPC 970FX processors running at 2.2 GHz, interconnected with a low latency Myrinet network. For the tests, only 64 processors are used due to account limitations.

The first comparison is over different combinations of linear solvers and preconditioners. The linear solvers used are GMRES (*gmres*), BiCGStab (*bcgs*), BiCGStab(ℓ) with $\ell = 2$ (*bcgsl*), and Transpose Free QMR (*tfqmr*), while the tested preconditioners are Jacobi (*jacobi*) and Block Jacobi (*bjacobi*). In the case of Block Jacobi, we use p blocks, where p is the number of processes, and an ILU factorization is built for each block. For details about these methods, the reader is referred to [13].

Table 1 shows the execution time, averaged out over all the linear systems, as well as the average number of iterations. Compared to the Jacobi preconditioner, the use of Block Jacobi reduces the iterations and time needed to solve the system but noticeably increases the setup time of the KSP object. The reason is that Jacobi does not require any initial computation whereas Block Jacobi needs to compute the ILU factorization of the diagonal blocks. Iterations performed with Block Jacobi preconditioning are more expensive, but since the number of iterations performed is reduced, the overall solving time is lower with Block Jacobi preconditioner for all linear solvers. Moreover, the fastest convergence is obtained with the BiCGStab and GMRES solvers (both

KSP	PC	Setup	Solve	Total	Its
gmres	jacobi	0.019	0.692	0.711	12.84
gmres	bjacobi	0.129	0.322	0.451	3.83
bcgs	jacobi	0.019	0.570	0.589	7.25
bcgs	bjacobi	0.127	0.332	0.459	2
bcgsl	jacobi	0.020	0.812	0.832	9.33
bcgsl	bjacobi	0.129	0.556	0.685	3.67
tfqmr	jacobi	0.019	1.052	1.071	11.86
tfqmr	bjacobi	0.130	0.353	0.483	2.03

Table 1: Comparison of linear solvers (KSP) and preconditioners (PC). The total time is divided into setup and solve times. The average number of iterations is also provided.

have virtually the same results), so speedup tests have been performed with the fastests combinations BiCGStab + Block Jacobi, GMRES + Block Jacobi and the default GMRES + Jacobi.

From the performance results shown in Table 2, we can draw the following conclusions. The parallel efficiency of the whole simulation, including the evaluation of the properties and assembly of the new system of equations, is reasonably good (more than 60% with 64 processes). In the case of the linear solver, the efficiency is slightly worse, except in the case of BiCGStab. The worst efficiency is obtained in the case of Jacobi preconditioning, and this can be attributed to the much higher number of iterations performed. The behaviour of Block Jacobi is very good, even though the effectiveness of the preconditioner decays with the number of processes (since the size of the blocks decreases). We can foresee that for cases where convergence is much more difficult to attain, the parallel efficiency of Block Jacobi will be reduced.

8 Conclusion

In this paper, a new open-source LES code (MICSc) is presented. The code solves the Navier-Stokes equations with a coupled and implicit method. The PETSc library is used to solve the system of linear equations and study the best performance over different combinations of linear solvers and preconditioners.

The LES code has been validated successfully with the well-known open channel flow by Moser *et al.*. The results show good agreement with differences lower that 15% in the prediction of the turbulent stresses.

Regarding the performance of the code, especially in terms of parallel efficiency, it has been shown that the best preconditioner is Block Jacobi whereas both BiCGStab and GMRES give the fastest convergence time (with virtually the same results). Also, the parallel efficiency of the whole simulation, including the evaluation of the properties and assembly of the new system of equations, is reasonably good (more than 60% with 64 processes).

GMRES + Jacobi

Procs	TOTAL			MEANS				
	Solve Total	S_p	E_p	**KSP Setup**	**KSP Solve**	**KSP**	S_p	E_p
1	15309.17	-	-	0.37	12.39	12.77	-	-
2	9926.51	1.5	77.1%	0.29	9.65	9.94	1.3	64.2%
4	5391.1	2.8	71%	0.14	4.95	5.09	2.5	62.7%
8	2791.4	5.5	68.6%	0.073	2.54	2.62	4.9	61%
16	1397.5	11	68.5%	0.038	1.37	1.41	9	56.5%
32	742.55	20.6	64.4%	0.019	0.69	0.71	18	56.1%
64	386.87	39.6	61.8%	0.0099	0.35	0.36	35.3	55.1%

GMRES + Block Jacobi

Procs	TOTAL			MEANS				
	Solve Total	S_p	E_p	**KSP Setup**	**KSP Solve**	**KSP**	S_p	E_p
1	14777.78	-	-	3.79	5.81	9.59	-	-
2	8703.70	1.7	84.9%	2.14	3.94	6.1	1.6	79%
4	4669.2	3.2	79.1%	1	1.93	2.93	3.3	81.8%
8	2387.5	6.2	77.4%	0.54	0.96	1.5	6.4	79.9%
16	1151.7	12.8	80.2%	0.26	0.52	0.78	12.3	76.6%
32	631.86	23.4	73.1%	0.13	0.32	0.45	21.3	66.4%
64	351.56	42	65.7%	0.064	0.17	0.23	41.2	64.4%

BiCGStab + Block Jacobi

Procs	TOTAL			MEANS				
	Solve Total	S_p	E_p	**KSP Setup**	**KSP Solve**	**KSP**	S_p	E_p
1	15312.97	-	-	3.79	7.06	10.86	-	-
2	8860.89	1.7	86.4%	2.10	4.80	6.91	1.6	78.6%
4	4793.98	3.2	79.9%	1.04	2.4	3.44	3.2	79%
8	2559.25	6	74.8%	0.53	1.32	1.85	5.9	73.5%
16	1186.53	12.9	80.7%	0.25	0.64	0.89	12.2	76.4%
32	639	24	74.9%	0.13	0.33	0.46	23.7	73.9%
64	354.49	43.2	67.5%	0.065	0.18	0.24	44.9	70.1%

Table 2: Parallel performance for three different combinations of iterative solver and preconditioner. The left side of the table shows results for the whole simulation, whereas the right side considers only the solution of the linear systems. The columns labeled S_p and E_p represent parallel spedup and efficiency for p processes.

Acknowledgements

The authors thankfully acknowledge the computer resources and assistance provided by the Barcelona Supercomputing Center (BSC).

References

[1] R.D. Moser, J. Kim, N. Mansour, "Direct Numerical Simulation of turbulent channel flow up to $Re_\tau = 590$", *Physics of Fluids*, 11(4): 943–945, 2007.

[2] H. Versteed, W. Malalasekera, *An introduction to computational fluid dynamics. The finite volume method*, Pearson, Prentice Hall, 1995.

[3] T. Xing, S. Frankel, "Effect of Cavitation on Vortex Dynamics in a Two-Dimensional Submerged Laminar Jet", *AIAA Journal*, 40(11): 2266–2276, 2002.

[4] G. Palau-Salvador, T. Stoesser, J. Froelich, M. Kappler, W. Rodi, "Large Eddy Simulation and Experiments of Flow Around Finite-Height Cylinders", *Flow Turbulence and Combustion*, 84(1): 239–275, 2010.

[5] J. Zeng, S. Constantinescu, L. Weber, "A 3D non-hydrostatic model to predict flow and sediment transport in loose-bed channel bends", 46(3): 356–372, 2008.

[6] M. Antonopoulos-Domis, "Large Eddy Simulation of a passive scalar in isotropic turbulence", 104(3): 55–79, 1981.

[7] S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L.C. McInnes, B. Smith, H. Zhang, "PETSc Users Manual", Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[8] J. Ferziger, M. Peric, *Computational Methods for Fluid Dynamics*, Springer; third edition, 2001.

[9] N. Waterson, H. Deconinck, "Design Principles for Bounded Higher-Order Convection Schemes - a Unified Approach", *Journal of Computational Physics*, 224 (1): 182–207, 2007.

[10] C. Rhie, W. Chow, "Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation", *AIAA Journal*, 21(11): 1525, 1983.

[11] A. Cubero, N. Fueyo, "A Compact Momentum Interpolation Procedure for Unsteady Flows and Relaxation", *Numerical Heat Transfer, Part B: Fundamentals*, 52(6): 507–529, 2007.

[12] S. Pope, *Turbulent Flows*, Cambridge University Press, Cambridge, UK, 2000.

[13] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, Second edition, 2003.

[14] S. Acharya, B.R. Baliga, K. Karki, J.Y. Murthy, C. Prakash, S.P. Vanka, "Pressure-Based Finite-Volume Methods in Computational Fluid Dynamics", *Journal of Heat Transfer-Transactions of the ASME*, 129(4): 407–424, 2007.

[15] A. Cubero, N. Fueyo, "Preconditioning Based on a Partially Implicit Implementation of Momentum Interpolation for Coupled Solvers", *Numerical Heat Transfer, Part B: Fundamentals*, 53(6): 510–535, 2008.

[16] W. Gropp, E. Lusk, A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*, MIT press, second edition, 1999.

[17] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries", in E. Arge, A.M. Bruaset, H.P. Langtangen (Editors), *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhaüser, 1997.

[18] W. Gropp, D. Kaushik, D. Keyes, B. Smith, "High-performance parallel implicit CFD", *Parallel Computing*, 27(4): 337–362, 2001.