Available online at www.sciencedirect.com
ScienceDirect
Advances in Engineering Software 38 (2007) 328–337
ADVANCES IN ENGINEERING SOFTWARE
www.elsevier.com/locate/advengsoft

Parallel simulation of three-dimensional complex flows: Application to two-phase compressible flows and turbulent wakes

B. Koobus [a,c,*], S. Camarri [b], M.V. Salvetti [b], S. Wornom [c], A. Dervieux [c]

[a] *Département de Mathématiques, CC 051, 34095 Montpellier Cedex 5, France*
[b] *Dipartimento di Ingegneria Aerospaziale, Universitá di Pisa, 56122 Pisa, Italy*
[c] *INRIA, 2004 Route des Lucioles, BP. 93, 06902 Sophia-Antipolis, France*

Received 12 September 2005; accepted 3 August 2006
Available online 10 October 2006

Abstract

In this paper, we present parallel simulations of three-dimensional complex flows obtained on an ORIGIN 3800 computer and on homogeneous and heterogeneous (processors of different speeds and RAM) computational grids. The solver under consideration, which is representative of modern numerics used in industrial computational fluid dynamics (CFD) software, is based on a mixed element-volume method on unstructured tedrahedrisations. The parallelisation strategy combines mesh partitioning techniques, a message-passing programming model and an additive Schwarz algorithm. The parallelisation performances are analysed on a two-phase compressible flow and a turbulent flow past a square cylinder.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Computational fluid dynamics; Two-phase flow; Turbulent flow; Hybrid RANS/LES model; Parallel computing

1. Introduction

In the last two decades computer technology has grown rapidly, especially parallel architectures [1]. Numerous configurations of parallel platforms are installed worldwide,[1] either through parallel supercomputers or through computational grids (homogeneous or heterogeneous). The combination of these with mature parallel algorithms allows the very efficient solution of academic or industrial problems like those encountered by the computational fluid dynamics (CFD) community.

In this paper, we present parallel simulations of three-dimensional complex flows using a CFD software which combines mesh partitioning techniques and a domain decomposition method (Schwarz algorithm). We propose to investigate the behaviour and performance of such a parallelisation strategy on homogeneous and heterogeneous computational grids, as well as on a parallel supercomputer.

The parallelisation strategy adopted in this study has been successfully applied in both a two-dimensional and a three-dimensional case by the university of Colorado at Boulder and INRIA in a cooperative program [2–4]. This has then been used intensively, in particular by the universities of Montpellier, Pisa and Pau in several CFD research studies (see for example [5–7]). The mesh partitioning techniques are combined with a message-passing programming model [2–4] using the MPI communication library which ensures software portability from one parallel system to another. The mesh partitioning algorithms and the generation of the corresponding communication data structures are computed in a preprocessing step. Because mesh partitions with overlap incur redundant floating-point operations, non-overlapping mesh partitions are chosen in our applications. It has been shown in [4] that the latter option

* Corresponding author. Address: Département de Mathématiques, CC 051 34095 Montpellier Cedex 5, France.
E-mail addresses: koobus@math.univ-montp2.fr (B. Koobus), s.camarri@ing.unipi.it (S. Camarri), mv.salvetti@ing.unipi.it (M.V. Salvetti), stephen.wornom@inria.fr (S. Wornom), dervieux@sophia.inria.fr (A. Dervieux).
[1] http://www.top500.org.

0965-9978/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2006.08.009

is more efficient though it induces additional communication steps.

The fluid solver *AERO* (and its two-phase version *AEDIF*) used in this work, which is representative of modern numerics employed in industrial CFD codes, is based on a mixed element-volume formulation which applies on unstructured tedrahedral meshes [8,9].

The first set of performance results presented in this paper concerns the simulation of a jet in crossflow and a shock wave propagation respectively on homogeneous and heterogeneous computational platforms. The second set of test cases, performed on an ORIGIN 3800 computer, concerns the prediction of a blast wave–bubble interaction and of a bluff-body flow dominated by massive separations. All these simulations must discretise the fluid equations on large three-dimensional meshes with small time steps. Therefore they require intensive computational resources (in terms of CPU and memory) and parallel computation is of particular interest for such applications.

The remainder of the paper is organised as follows. In Section 2 we describe the numerical and parallel features of the CFD softwares used in our applications. In particular, we present the software improvements in terms of memory allocation that have been achieved in order to perform efficient parallel computations on homogeneous and heterogeneous computational platforms. These improvements are highlighted by some performance results. In Section 3, we describe the two-phase flow application which concerns a blast wave–bubble interaction. In Section 4, we present the parallel simulation of a bluff-body flow with a hybrid Reynolds Averaged Navier–Stokes (RANS)/Large Eddy Simulation (LES) model. Finally, conclusions are given in Section 5.

2. Features of the CFD Software *AERO*

2.1. Numerical approximation

The spatial discretisation is based on a mixed element-volume method which applies to unstructured tetrahedrisations [8,9]. The adopted scheme is vertex centred, i.e. all degrees of freedom are located at the vertices. P1 Galerkin finite elements are used to dicretise the diffusive terms.

A dual finite-volume grid is obtained by building a cell C_i around each vertex i through the rule of medians. The convective fluxes are discretised on this tessellation, i.e. in terms of fluxes relative to the common boundaries shared by neighboring cells.

The Roe scheme [10] represents the basic upwind component for the numerical evaluation of the convective fluxes $\mathscr{F}$:

$$\Phi^R(W_i, W_j, \vec{n}) = \frac{\mathscr{F}(W_i, \vec{n}) + \mathscr{F}(W_j, \vec{n})}{2} - \gamma_s \left[|\mathscr{R}(W_i, W_j, \vec{n})| \frac{W_j - W_i}{2} \right] \quad (1)$$

in which $\Phi^R(W_i, W_j, \vec{n})$ is the numerical approximation of the flux between the ith and the jth cells, W_i is the solution vector at the ith node, $\vec{n}$ is the normal to the cell boundary and $\mathscr{R}$ is the Roe matrix. The parameter γ_s, which multiplies the stabilisation part of the scheme, collected within square brackets in Eq. (1), allows a direct control of the numerical viscosity, leading to a full upwind scheme for $\gamma_s = 1$ and to a centred scheme when $\gamma_s = 0$. The spatial accuracy of this scheme is only first order. To increase the order of accuracy of the Roe scheme, the Monotone Upwind Schemes for Conservation Laws linear reconstruction method (MUSCL), introduced by Van Leer [11] is employed. This is obtained by expressing the Roe flux between two cells centred on two generic nodes i and j, as a function of the reconstructed values of W at their interface: $\Phi^R(W_{ij}, W_{ji}, \vec{n}_{ij})$, where W_{ij} is extrapolated from the values of W at nodes i and j. A reconstruction using a combination of different families of approximate gradients (P1-elementwise gradients and nodal gradients evaluated on different tetrahedra) is adopted, which allows us to obtain a numerical dissipation made up of sixth-order space derivatives. This MUSCL reconstruction is described in detail in Ref. [12].

Either implicit or explicit schemes can be used to advance the equations in time by a line method, i.e. time and space are treated separately. In the explicit case a N-stage low-storage Runge–Kutta algorithm is applied. At each stage, the flux evaluation is performed avoiding redundant operations. These operations represent about 2K flop per unknown. Data transfers are done only once per flux evaluation and concern a data volume which represents in average 10% of the system unknowns number (for a subdomain size of about 30K nodes). An implicit time marching algorithm is also available in the code, based on a second-order time-accurate backward difference scheme. The non-linear discretised equations are solved by a defect-correction (Newton-like) method [13] in which a first-order semi-discretisation of the Jacobian is used. At each time-step, the resulting sparse linear system is solved by a restricted additive Schwarz preconditioned iterative method [14]. The non-zero (5×5 and 2×2) blocks of the system is stored at each time step. The most internal loop is a local block-ILU(0) iteration used for preconditioning the Schwarz one. It represents about 1K flop per unknown. Then Schwarz starts a data transfer slightly smaller than in the explicit flux assembly. As a result, the communication/computation ratio in the implicit mode is typically 50% larger than for the explicit case. This implicit scheme is linearly unconditionally stable and second-order accurate.

More details on the numerical ingredients used in this paper can be found in Refs. [5,15].

2.2. Implementation on parallel platforms

2.2.1. Parallelisation in the AERO code

The *AERO* code implements parallel computing using mesh partitioning that is, dividing a large mesh into smaller partitions so that the work for each partition can be computed by a different processor. The partitioning can be

homogeneous (equal size) or heterogeneous (unequal size). Using mesh partitioning is a very popular approach for parallel computing using CFD codes. Once the mesh has been partitioned, parallelisation is implemented using the Message Passing Interface (MPI) system applied over the partitions.

The source code (including the MPI calls) is compiled with the F77 compiler and linked with the MPI library. Using MPI, each processor executes a copy of the executable, reads a data file containing data common to all the processors,[2] a file that contains the size of the full mesh and the maximum size of the partitions, etc. However, each processor reads only the mesh data for its specific partition. The partition files contain the number of vertices, tetrahedra, and external faces (triangles); the coordinates of the vertices, the connectivity for the tetrahedra and external faces (triangles), the type of external boundary conditions to be applied on the external faces, the vertices that must exchange data with neighboring partitions, an identifier to indicate whether a boundary vertice is active or not and finally the global indices of the local vertices.

2.2.2. Computational platforms

For machines like the ORIGIN 3800 with shared memory at the Centre Informatique National de l'Enseignement Supérieur (CINES)[3] with 768 processors, homogeneous partitioning with the executable compiled with F77 works well as all the processors are identical in speed and RAM.

When computing on parallel platforms that have processors with different speeds and RAM, efficiency is lost when homogeneous partitions are used; the slower processors spend more time computing than the faster processors. This is the scenario that often occurs on computational grids. For examples the cluster of INRIA[4] and the MecaGRID[5] both having a mixture of processors of 2 GHz and 1 GHz and 2 GB and 1 GB RAM. Numerical tests using the MecaGRID with as many as 64 processors were reported by Wornom [16]. The MecaGRID is a grid in the classical definition in that the MecaGRID clusters belong to different Virtual Organisations using a Virtual Private Network or tunnel to pass messages between the different MecaGRID clusters.

In order to maximise efficiency on computational grids with processors of different speeds and RAM, dynamic memory allocation is required in order to give more computational work to the faster processors so that all the processors finish their work at approximately the same time. Dynamic memory allocation cannot be achieved with the present F77 code since parameters statements are used and the parameters are fixed not by the size of the local partition but by the size of the largest partition.

2.2.3. Dynamic memory allocation

Several programming languages use dynamic memory allocation, F90, C, C++, and JAVA are examples. Choosing either C, C++, or JAVA as the new language will require changing 100% of the F77 coding.

An alternative choice would be to retain the FORTRAN coding and replace the FORTRAN parameter statements with C, C++, JAVA, PYTHON, or TCL wrappers to achieve the necessary dynamic memory allocation. Although not used in this study, this alternative choice is valid. Here F90 was chosen for the following reasons:

1. In general, the F77 coding is compatible with the F90 compiler. Therefore we can add F90 features to the existing F77 code and it is not necessary to convert 100 percent of the F77 code to the F90 standard. This saves considerable time both in not having to rewrite the F77 code in the F90 standard as well as the additional debugging time that would be necessary.
2. The modification to the F77 code to include the F90 dynamic memory allocation are relatively easy. In fact, most of the necessary changes were completed over a two-day period with an additional week needed for testing and verification.
3. Most of the F90 modifications (replacing .h files with F90 Modules) are transparent to the users, thus the use of the F90 version results in no perturbation to F77 users.
4. The F90 code compiles only options actually used. With F77, all arrays and options are allocated memory at compiling time whether or not they are used at run time. This results in a very large executable that reduces the size of the mesh that can be executed on computational platforms.
5. In contrast to the F77 version of the *AERO* code, the source of the F90 version does not depend on mesh size parameters or on the number of processors; therefore the F90 code is much easier for new and F77 users to use.

2.2.4. Improvements related to F90

Two improvements related to the F90 version of the *AERO* code are illustrated below.

Homogeneous parallel platforms: In this section it is shown that because memory is allocated at runtime, and only allocated for the code options actually used, the size of the F90 executable is much smaller than the F77 version that allocates memory for all options at compile time, whether or not they are used at run time. As a consequence

[2] Time scheme, order of accuracy, CFL number, number of time steps, ... etc.

[3] http://www.cines.fr.

[4] The INRIA cluster is actually comprised of three clusters (two operational). Thus it is appropriate and more descriptive to refer to it as a local computational grid.

[5] The MecaGRID connects the clusters of INRIA Sophia Antipolis, Ecole des Mines de Paris at Sophia Antipolis, and the IUSTI in Marseille. The MecaGRID project is sponsored by the French Ministry of Research (http://www.recherche.gouv.fr/recherche/aci/grid.htm, http://www.recherche.gouv.fr/recherche/aci/grid.htm) through the ACI-Grid program.

many test cases that cannot be run with the F77 version, because the executable is too large for the available RAM, can be easily run with the F90 version.

The problem that motivated the F90 work was the test case of a jet in crossflow [17] involving 400K mesh points. The INRIA cluster is really composed of two clusters, one called *nina* and the other *pf*.[6] Table 1 shows the speed and RAM of the *nina* processors to be twice that of the *pf* processors. Due to the *pf* RAM limit, this test case could not be run using the implicit scheme on the INRIA cluster when less than approximately 32 mesh partitions were used if one of the 32 processors was a *pf* processor because the size of the F77 executable was too large for the *pf* processor (1/2 GB RAM). In order to run the F77 version the user accepted the use of the explicit algorithm.[7]

Using the F90 complete code version, both the explicit and implicit executed successfully using either *nina* or *pf* processors for the 400K mesh with as few as eight total partitions. We also estimate, based on our tests, that, using the F90 version, a mesh of 1.2 million points could be run using only 32-*pf* processors.

In Table 2, we compare, using 32 partitions of the 400K vertices mesh, the performance results (wall time and memory) obtained with the F77 and F90 versions of *AERO* for different parallel systems: 32 *pf* processors, 32 *nina* processors, and 32 R14000 500 MHz processors of an ORIGIN 3800 computer. The results show that with F90 almost 3 times less memory is required than with F77, and that wall times are rather closed for a similar parallel system (F90 is a little less performant due certainly to allocation/deallocation of temporary tables). For the ORIGIN 3800 computer, the reduction of memory (which does not appear in this table) is not crucial, due to its shared memory. This table also shows that, as predicted according to their relative processor performance (see Table 1), the computations performed on 32 *nina* processors are twice as fast as the ones achieved on 32 *pf* processors, and that the best performances are obtained with the "integrated" parallel system (ORIGIN 3800 computer).

MecaGRID: In this section, the benefit of using heterogeneous partitioning is illustrated. The focus here was to determine the efficiency of using dynamic memory allocation and not to evaluate the efficiency of the MecaGRID.

In order to verify the dynamic memory allocation feature of the F90 code, test cases were run on the MecaGRID using homogeneous and heterogeneous mesh partitioning with the Globus software. The test case studied is a plane shockwave propagating in a rectangular tube. The mesh contains 252K vertices. Two partitionings were created: (1) 16 homogeneous partitions and (2) 16 heterogeneous partitions. The mesh partitioner used in this study was the FastMP software developed by Wornom [18]. The FastMP software is written in F90 and executes in parallel using MPI and is reasonably fast. The heterogeneous partitioning for the 252K mesh required 3.8 s.[8] The partitioning for a mesh of 2 million vertices into 32 homogeneous partitions, discussed in Section 3.2, required 30 s.

Table 1
Characteristics of the INRIA cluster

Processor	CPUs	GHz	RAM/node (GB)	LAN speed (Gbps)
nina	32	2	1.00	1.000
pf	32	1	0.50	0.100

We executed the F90 code via the MecaGRID using the GLOBUS software[9] with a total of 16 processors (8-*nina* and 8-*pf*). The number of processors to be used on each cluster can be selected by MecaGRID users in Globus script; this possibility is not available to users of the INRIA cluster. Both the *nina* and *pf* clusters are members of the MecaGRID. Unfortunately the other clusters belonging to the MecaGRID were off line and could not be used for this evaluation. As the bandwidth between *nina* processors is 1 Gbps and 100 Mbps between *pf* processors, the computations made via the MecaGRID may be considered as a metacomputing, rather than a true grid, computation. For a true MecaGRID calculation using, for example, *pf* processors and *iusti* processors in Marseille 180 km distant, latency and bandwidth fluctuating bandwidths become important; these were not important in the calculations here as both the *nina* and *pf* processors were local.

For the homogeneous mesh partitioning, all the partitions were of equal size. For the heterogeneous partitioning, the 8-*nina* partitions contained twice the number of tetrahedra as the 8-*pf* partitions. In Table 3, we give the wall time relative to the homogeneous and heterogeneous partitioning for a simulation of 150 time steps. From this table, we observe the substantial gain in simulation time obtained with the heterogeneous partitioning for which a speedup of 1.45 is reached (the theoretical speedup for this case is 1.5).

2.2.5. Summary: implementation on parallel platforms

F90 offers several features not available in F77 that increase the capabilities and ease of use of the *AERO* code. These include dynamic memory allocation. The F90 version of the *AERO* code is mesh independent and because memory is allocated at runtime, only arrays and options that are used are allocated memory. Therefore the size of the executable is much smaller than the F77 version that allocates memory for all options, whether they are used or not, at run time. As a consequence many test cases (large meshes) that cannot be run with the F77 version can be easily run with the F90 version. This improvement was first checked on homogeneous parallel platforms by considering

[6] See http://www-sop.inria.fr/parallel/.

[7] The 400K grid has mesh refinement near boundaries; therefore the implicit algorithm was preferred to avoid the long computational times needed with the explicit algorithm. Using the explicit algorithm permitted us to remove eight implicit subroutines from the Makefile, and their call statements from the source code, to get the size of the executable $\leqslant$1/2 GB.

[8] Time to read the mesh to be partitioned, partition the mesh and write the partition files.

[9] http://www.globus.org.

Table 2
Implicit computation on the 400K vertices mesh: comparison of memory needed and wall time for the F77 and F90 versions of *AERO*

Parallel system	F90		F77	
	Size (MB)	Wall time (s)	Size (MB)	Wall time (s)
Cluster 32-*pf*	164	2075	431	1945
Cluster 32-*nina*	164	1148	431	1050
ORIGIN 3800, 32 proc	–	798	–	726

Table 3
A mesh containing 252K vertices: wall time comparison between a homogeneous and a heterogenous partitioning using 8-*nina* 8-*pf* processors

Mesh partitioning	Wall time (s)
16 homogeneous partitions	701
16 heterogeneous partitions	484

a mesh of 400K vertices. Then, we show numerical results for a mesh containing 252K vertices running on the Mecagrid using GLOBUS and heterogeneous partitions. This resulted in a speedup of 1.45 relative to the same run using the homogeneous partitioning which compares well with the theoretical speedup of 1.5. This validates the efficiency of the F90 version of the *AERO* code.

3. Application to two-phase flows

Numerical methods for multi-fluid or multiphase flows have a long history and many different methods have become successively popular over the last five decades. Today Eulerian formulations are favoured for solving many problems where the interface shows strong variations of physical properties. The difficulties are not only related to the advection of the interface, but also to the computation of the composite fluid (typically liquid and gas) in mixed cells. The diffuse interface method of Saurel and Abgrall [19] is an Eulerian method which applies particularly to the latter problem. The main principle of this method is to build accurate and stable models for the mixed cells involving fluids with a common law, but with very different constants. Then such methods can be modelled in the same manner either for an interface without mixing or for a region with some local mixing. The five equation two-dimensional model of Guillard–Murrone [20] has been applied in a large set of two-dimensional test cases. This model was extended to three dimensions using unstructured tetrahedra volumes by Wornom et al. [21]. The scheme was implemented in the basic solver in *AERO* code developed at the University of Colorado by Farhat [22] with the collaboration of INRIA – see Dervieux [8], Nkonga and Guillard [23], and Martin and Guillard [13] for fluid flow problems. The two-phase[10] software is called *AEDIF*.

[10] The term "two-phase flows" is used in the literature to describe flows involving two different liquids or gases: for example, liquid and gas, or two different liquids, as in the present report.

3.1. Numerical algorithm

The *AEDIF* software, derived from *AERO* for the calculation of two-phase flows, replaces the Euler equations for a unique, perfect gas with the following two-phase model:

Seven-equation quasi conservative reduced model

$$\frac{\partial}{\partial t}\alpha_k\rho_k + \operatorname{div}(\alpha_k\rho_k u) = 0; \quad k = 1,2 \tag{2}$$

$$\frac{\partial}{\partial t}\rho u + \operatorname{div}(\rho u \otimes u) + \nabla p = 0, \tag{3}$$

$$\frac{\partial}{\partial t}\rho e + \operatorname{div}(\rho e + p)u = 0, \tag{4}$$

$$\frac{\partial}{\partial t}\alpha_2 + u\dot{\nabla}\alpha_2 = \alpha_1\alpha_2\frac{\rho_1 a_1^2 - \rho_2 a_2^2}{\sum_{k=1}^{2}\alpha_{k'}\rho_k a_k^2}\operatorname{div}u \tag{5}$$

with $e = \varepsilon + u^2/2$ and $\rho\varepsilon = \sum_{k=1}^{2}\alpha_k\rho_k\varepsilon_k(p,\rho_k)$, where α_k are the mass fractions of the two fluids, ρ_k the corresponding densities, u, p the velocity and pressure common to the two phases.

The flux splitting for the five equation two phase flows model was developed in [24], extended to three dimensions by Wornom [21] and can be understood as an extension of the acoustic Riemann solver described for instance in [25] for the Euler equations of gas dynamics. This linearised Riemann solver uses the mathematical structure of the model and in particular the continuity of pressure and velocity across a contact discontinuity. Based on several different numerical tests, this acoustic Riemann solver seems to be very robust with respect to the Mach number and especially for interface problems (see [26]).

3.2. Results: blast wave–bubble interaction

This kind of flow has been studied in 2D and 2D axisymmetric computations by Giordano [27]. Very fine 3D calculations are necessary in order to study unstable capillary effects that cannot be modelled using axisymmetric formulations. The calculations shown hereafter were performed on a mesh with 2 million vertices and represent a step in this direction.

The initial position of the shockwave and the densities are shown in Fig. 1.[11] Shown in Fig. 2 are the contours of density and the mass fractions of the two fluids on a symmetry plane after 600 time steps. Fig. 3 shows the

[11] The initial pressure and density ratios were 10/1.

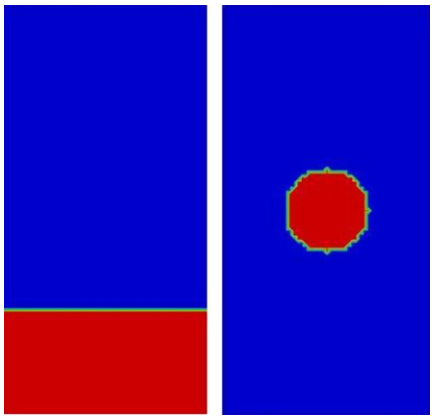

Fig. 1. Symmetry plane contours at time step 600: pressure (left), mass fractions (right).

three-dimensional w-velocity contours after 600 time steps at which time the blast wave has passed through the bubble. The deformation of the initial spherical bubble can be seen.

In order to evaluate the parallel speedup and efficiency of *AEDIF*, the mesh containing 2 million vertices has been decomposed into 24, 32, 48 and 64 partitions, each partition being assigned to one processor of an ORIGIN 3800 computer.

As the total simulation time of the serial program running on a single processor is not available because of memory limits, we have used the following speedup estimation [4]:

$$S(Np) = Np \times \left(1 + \frac{T^{Np}_{\text{comm}}}{T^{Np}_{\text{comp}}}\right)^{-1},$$

where Np is the number of processors, T^{Np}_{comm} and T^{Np}_{comp} are respectively the communication time and the computational time of the parallel application on Np processors, and $S(Np)$ is the resulting speedup.

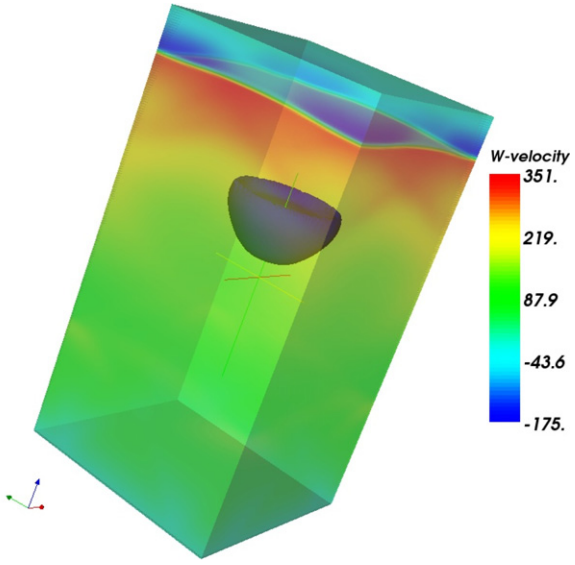

Fig. 3. Blast wave w-velocity contours after 600 time steps.

The efficiency, which is a meaningful measure of the percentage of a processor's time spent in useful computation, is then defined by:

$$E(Np) = \frac{S(Np)}{Np}.$$

Table 4 shows the parallel speedup and efficiency relative to one processor, as well as the computational and communication times for 600 time steps. These performance results prove the good parallel scalability of the software used for these two-phase flow calculations.

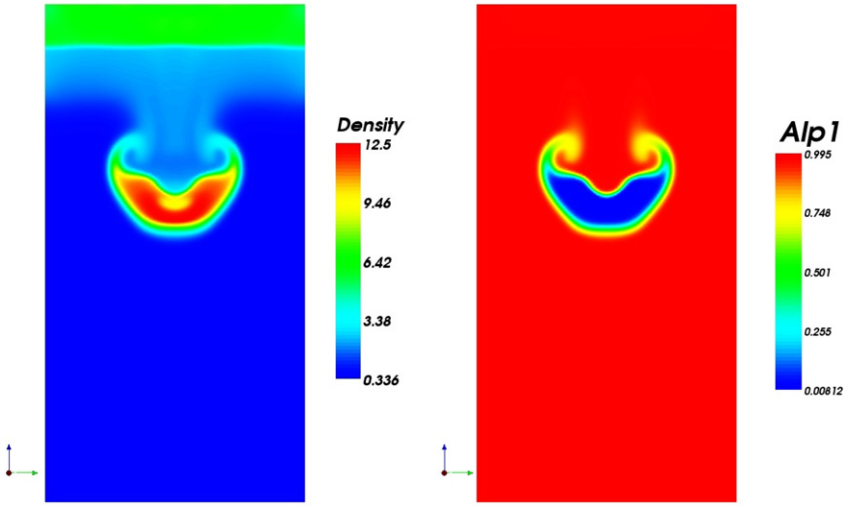

Fig. 2. Centre plane contours at time step 600: density (left), volumic ratio (right).

Table 4
A mesh containing 2 million vertices: speedup $S(Np)$ and efficiency $E(Np)$ vs the number Np of processors of an ORIGIN 3800 computer for 600 time steps

Np	Computational time (s)	Communication time (s)	$S(Np)$	$E(Np)$ (%)
24	14,218	990	22.44	93.49
32	10,626	812	29.73	92.90
48	8372	1194	42.01	87.52
64	6824	1029	55.65	86.95

4. Application to turbulent wakes

The accurate and efficient simulation of turbulent wakes is an important challenge in CFD. It concerns many flow problems, in particular those around bluff-bodies involving vortex shedding. In the present study, we are interested in the turbulent flow which develops around a square cylinder at a relatively high Reynolds number.

Several turbulence models can be used for the simulation of such massively separated flows. Among these models, we can cite two important classical approaches: the one in which the Reynolds Averaged Navier–Stokes (RANS) equations are discretised, and the Large Eddy Simulation (LES). In this work, we use a third and more recent approach: a hybrid RANS/LES model which combines RANS and LES features in order to exploit as much as possible the advantages of the two previous classical approaches: less computational resources compared with LES, and better accuracy than RANS. In the next subsection, we describe briefly the chosen hybrid RANS/LES approach, before presenting some performance results in the next subsection.

4.1. Limited numerical scales (LNS) approach

The basic idea of the LNS model [28] is to multiply the Reynolds stress tensor, given by the RANS closure, by a blending function, which permits us to switch from the RANS to the LES approach.

For the RANS closure, the standard k–ε model [29] is used here, in which the Reynolds stress tensor is modelled as follows, by introducing a turbulent eddy-viscosity, μ_t:

$$R_{ij} \simeq \mu_t \underbrace{\left[\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} - \frac{2}{3}\frac{\partial \tilde{u}_l}{\partial x_l}\delta_{ij}\right]}_{\widetilde{P}_{ij}} - \frac{2}{3}\overline{\rho} k \delta_{ij}, \tag{6}$$

where the tilde denotes the Favre average, the overbar time averaging, δ_{ij} is the Krönecker symbol and k is the turbulent kinetic energy. The turbulent eddy-viscosity μ_t is defined as a function of k and of the turbulent dissipation rate of energy, ε, as follows:

$$\mu_t = C_\mu \frac{k^2}{\varepsilon}, \tag{7}$$

in which C_μ is a model parameter, here set equal to the classical value of 0.09 and k and ε are obtained from the corresponding modelled transport equations [29].

The LNS equations are then obtained from the RANS equations by replacing the Reynolds stress tensor R_{ij}, given by Eq. (6), with the tensor L_{ij}:

$$L_{ij} = \alpha R_{ij} = \alpha \mu_t \widetilde{P}_{ij} - \frac{2}{3}\overline{\rho}(\alpha k)\delta_{ij}, \tag{8}$$

where α is the damping function ($0 \leqslant \alpha \leqslant 1$), varying in space and time.

In the LNS model, the damping function is defined as follows:

$$\alpha = \min\left\{\frac{\mu_s}{\mu_t}, 1\right\} \tag{9}$$

in which μ_s is the SGS viscosity obtained from a LES closure model.

The Smagorinsky SGS model [30] is adopted here; thus, we have:

$$\mu_s = \overline{\rho} C_s \Delta^2 \sqrt{\widetilde{S}_{ij}\widetilde{S}_{ij}}, \tag{10}$$

where C_s is the model input parameter, $\widetilde{S}_{ij}$ is the strain-rate tensor and Δ is a length which should be representative of the size of the resolved turbulent scales. Here, Δ has been selected, for each tetrahedrical element of the grid, as the length of the longest edge [5].

Summarising, wherever the LES SGS-viscosity is lower than the RANS eddy-viscosity ($\alpha < 1$), an expression very similar to the classical Smagorinsky model is obtained for the turbulent stresses by combining Eqs. (6), (8) and (9). The difference with the classical Smagorinsky model is the presence of the diagonal term proportional to k. However, for compressible flows, this can be considered as a model for the isotropic part of the SGS stresses. As discussed in [28], the model should work in the LES mode where the grid is fine enough to resolve a significant part of the turbulence scales, as in LES; elsewhere ($\alpha = 1$), the k–ε RANS closure is recovered.

Note that in LNS R_{ij} is replaced with L_{ij} not only in the momentum and energy equations, but also in the two additional equations in k and ε (omitted here for the sake of brevity). In those regions where $\alpha < 1$, this implies a reduction of the turbulent kinetic energy production, together with a reduction of the turbulent transport of k and ε.

Finally, by construction, the present version of the LNS model is no more time consuming than the RANS k–ε model. Indeed, the extra cost due to the evaluation of the Smagorinsky eddy viscosity is negligible compared to the

overall computation required by the solution of the RANS k–ε equations.

4.2. Results: flow around a square cylinder

The flow around a square cylinder of infinite length is considered. The Reynolds number, based on the cylinder side length and the freestream velocity, is set to 22,000, and the Mach number to 0.1. The unstructured grid used for this simulation contains around 83,000 nodes and half a million tetrahedra (see Fig. 4 for a horizontal and a vertical cut of the mesh). The dimensions of the computational domain are equal to those employed by contributors to the LES workshop [31] except for a larger dimension in the spanwise direction which corresponds to that of the water tunnel used in the experiments of Lyn and Rodi [32,33]. Thus, slip conditions have been applied in the spanwise direction. The Reichardt wall-law is used for an approximate near-wall treatment on the cylinder surface.

The simulations are advanced in time using an implicit scheme, with a maximum CFL number equal to 25. The resulting time step leads to the use of around 150 time iterations per shedding cycles. We have a-posteriori checked that no significant information is lost in time with respect to the case of an explicit four steps Runge–Kutta scheme with a CFL equals to 1.

We show in Fig. 4 the instantaneous Mach number contours in a vertical and horizontal section. These plots highlight the small structures predicted in the wake by the rather coarse unstructured mesh employed in this work, and illustrate the vortex shedding phenomenon which is characteristic of subsonic flows around such bluff-bodies.

The bulk coefficients obtained in this LNS simulation are reported in Table 5, together with some reference LES results [31] and experimental data [32,33]. The results summarised in this table show that the LNS approach predicts the bulk coefficients with good accuracy, and on a coarser grid than those employed in the LES workshop [31].

Per shedding cycle, the total simulation time is 1076 s on eight processors of an ORIGIN 3800 computer equipped with R14000 500 MHz chips. In order to evaluate the parallel speedup and efficiency of our software, we have decomposed the mesh into 2, 4, 8 and 16 partitions, each partition being assigned to one processor.

In Table 6, we show the computational and communication time, as well as the speedup and efficiency (defined in Section 3.2) relative to each of these mesh partitionings for the prediction of a complete shedding cycle. From this table, we observe the good speedup and efficiency obtained through this sequence of simulations. This proves the good parallel scalability of the software used in this work for "integrated" parallel systems.

Table 5
Bulk coefficients: numerical results obtained in the LNS simulation together with some reference LES results and experimental data

	C_l'	$\overline{C_d}$	St	l_r
Simulations				
LNS	1.0	2.0	0.127	1.29
Ref. LES [31]	[0.38,1.79]	[1.66,2.77]	[0.07,0.15]	[0.89,2.96]
Experiments				
[32] and [33]	–	2.1	0.132 ± 0.004	1.4
[34]	1.2	2.28	0.130	–

$\overline{C_d}$ is the mean drag coefficient, C_l' is the r.m.s. of the lift coefficient, St is the Strouhal number and l_r is the length of the mean recirculation bubble.

Table 6
Computational time, communication time, speedup $S(Np)$ and efficiency $E(Np)$ vs. the number Np of processors of an ORIGIN 3800 computer for a shedding cycle (83,000 vertices mesh)

Np	Computational time (s)	Communication time (s)	$S(Np)$	$E(Np)$ (%)
2	4229	34	1.98	99.00
4	2053	41	3.92	98.00
8	1020	56	7.58	94.75
16	830	63	14.87	92.93

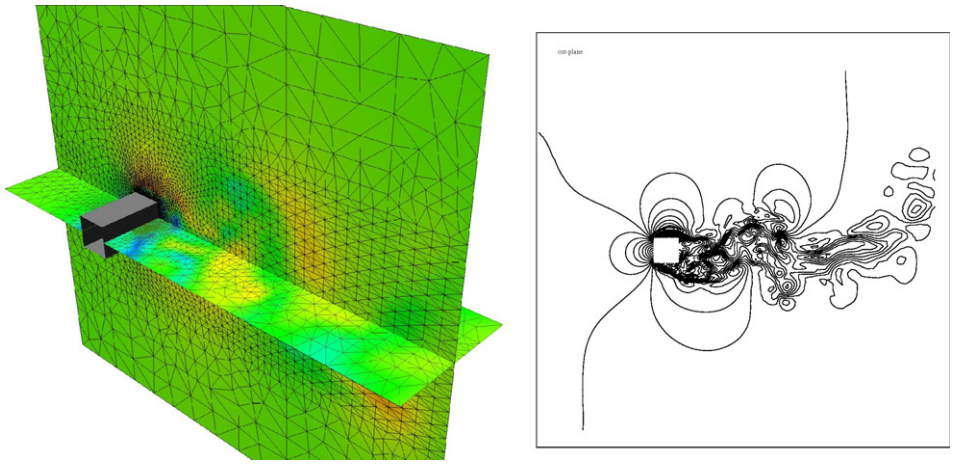

Fig. 4. Mesh and instantaneous Mach field in a vertical (left and right) and horizontal section (left).

5. Conclusion

In this paper, we have presented some parallel simulations of three-dimensional complex flows. The first performance results concern the simulation of a jet in crossflow and the propagation of a shock wave on homogeneous and on heterogeneous computational platforms respectively. For this purpose, and in order to allow dynamic memory allocation, the original F77 software was extended to an F90 version. The second set of simulations were performed on an ORIGIN 3800 computer and concern a blast wave–bubble interaction and the prediction, by a recent hybrid RANS/LES model (namely the LNS approach), of a turbulent flow which develops around a square cylinder. All these simulations are resource intensive, and their computation on one processor would require prohibitive delays, as well as using too much memory. The parallel option, which we have briefly described in this paper, combines mesh partitioning techniques and a message-passing programming model. The performance measurements performed on parallel systems made of identical or different processors (in terms of RAM and speed), have shown that our algorithm has good scalability features and is well adapted to the coarse–grain parallel computing power that is available today on parallel supercomputers and on homogeneous or heterogeneous computational grids.

Acknowledgements

This research has been made possible by the ACI-GRID 2002 Project of the French Ministry of Research[12]. The authors would like to acknowledge the support of Centre Informatique National de l'Enseignement Supérieur (CINES[13]), Montpellier, FRANCE, and the support of PACA[14] region for the cooperation between INRIA and the University of Pisa.

References

[1] Van der Steen AJ, Dongarra JJ. Overview of Recent Supercomputers. TOP500 Supercomputer Sites, 2004.
[2] Fezoui L, Loriot F, Loriot M, Regere J. A 2D Finite volume/finite element Euler solver on M.I.M.D. parallel machine. In: Duran M, El Dabaghi F, editors. Proceedings of the high performance computing II conference, Montpellier, 1991.
[3] Farhat C, Lanteri S. Simulation of compressible flows on a variety of MPPs: computational algorithms for unstructured dynamic meshes and performance results. Comp Meth Appl Mech Eng 1994;119:35–60.
[4] Lanteri S. Parallel solutions of three-dimensional compressible flows. Technical Report RR-2594, INRIA, Sophia Antipolis, June 1995.
[5] Camarri S, Salvetti MV, Koobus B, Dervieux A. Large-eddy simulation of a bluff-body flow on unstructured grids. Int J Numer Meth Fluids 2002;40:1431–60.
[6] El Omari K, Schall E, Koobus B, Dervieux A. Turbulence modeling challenge in airship CFD studies. In: Proceedings of the eigth Zaragoza-Pau conference of applied mathematics and statistics, Jaca, 15–17 september, 2003.
[7] Koobus B, Farhat C. A variational multiscale method for the large eddy simulation of compressible turbulent flows on unstructured meshes – application to vortex shedding. Comp Meth Appl Mech Eng 2004;193:1367–83.
[8] Dervieux A. Steady Euler simulations using unstructured meshes. In: Published in partial differential equations of hyperbolique type and applications. In: Geymonat, editor. Lecture series 1985–04. World Scientific (1987), Von Karman Institute for Fluid Dynamics; 1985.
[9] Fezoui L, Lanteri S, Larrouturou B, Olivier C. Resolution numerique des equations de Navier–Stokes pour un fluide compressible en maillage triangulaire. Technical Report 1033, INRIA, Sophia Antipolis, May 1989.
[10] Roe PL. Approximate Riemann solvers, parameters, vectors and difference schemes. J Comp Phys 1981;43:357–72.
[11] Van Leer B. Towards the ultimate conservative scheme. IV: A new approach to numerical convection. J Comp Phys 1977;23:276–99.
[12] Camarri S, Salvetti MV, Koobus B, Dervieux A. A low-diffusion MUSCL scheme for LES on unstructured grids. Comput Fluids 2004.
[13] Martin R, Guillard H. Second-order defect-correction scheme for unsteady problems. Comput Fluids 1996;25(1):9–27.
[14] Farhat C, Cai XC, Sarkis M. A minimum overlap restricted additive Schwarz preconditioner and applications in 3D flow simulations. Contemp Math 1998;218:478–84.
[15] Koobus B, Tran H, Farhat C. Computation of unsteady viscous flows around moving bodies using the k–ε turbulence model on unstructured dynamic grids. Comp Meth Appl Mech Eng 2000;190:1441–66.
[16] Wornom S. Parallel computations of one-phase and two-phase flows using the MecaGRID. Technical Report RT-297, INRIA, Sophia Antipolis, August 2004.
[17] Mariotti V, Camarri S, Salvetti MV, Koobus B, Guillard H, Wornom S. Numerical simulation of a jet in crossflow. Application to GRID computing. Technical Report in preparation, INRIA, Sophia Antipolis, 2005.
[18] Wornom S. FastMP, A fast mesh partitioner designed for GRID computing. Technical report, INRIA, Sophia Antipolis, May 2005.
[19] Saurel R, Abgrall R. A simple method for compressible multifluid flows. SIAM J Sci Comput 1999;21–3:1115–45.
[20] Guillard H, Murrone A. A five equation reduced model for compressible two phase flow problems. Technical Report RR-4778, INRIA – Sophia Antipolis, March 2003.
[21] Wornom S, Koobus B, Guillard H, Murrone A, Dervieux A. Seven-equation, two-phase flow three-dimensional calculations using a mixed-element-volume method. Technical Report RR-5560, INRIA, Sophia Antipolis, April 2005.
[22] Farhat C. High performance simulation of coupled nonlinear transient aeroelastic problems. Special course on parallel computing in CFD. R-807, NATO AGARD Report, October 1995.
[23] Nkonga B, Guillard H. Godunov type method on non-structured meshes for three dimensional moving boundary problems. Comp Meth Appl Mech Eng 1994;113:183–204.
[24] Murrone A, Guillard H. A five equation reduced model for compressible two phase flow problems. J Comput Phys 2005;202:664–698.
[25] Toro EF. Riemann solvers and numerical methods for fluid dynamics. Berlin: Springer Verlag; 1997.
[26] Murrone A. Modèles bi-fluides à six et sept équations pour les écoulements diphasiques à faible nombre de Mach. Ph.D. thesis, University of Aix-Marseille I, to be found at http://tel.ccsd.cnrs.fr/, 2003.
[27] Giordano J. Contribution à la modélisation numérique des problèmes d'interactions fluide-fluide et fluide-structure. Ph.D. thesis, Université d'Aix-Marseille I, 2004.

[12] http://www.recherche.gouv.fr/recherche/aci/grid.htm.
[13] http://www.cines.fr.
[14] Provence-Alpes-Côte-d'Azur.

[28] Batten P, Goldberg U, Chakravarthy S. Interfacing statistical turbulence closures with large-eddy simulation. AIAA J 2004;42(3): 485–92.
[29] Launder BE, Spalding DB. The numerical computation of turbulent flows. Comp Meth Appl Mech Eng 1979;3:269–89.
[30] Smagorinsky J. General circulation experiments with the primitive equations. Monthly Weather Rev 1963;91(3):99–164.
[31] Rodi W, Ferziger JH, Breuer M, Pourquié M. Status of large eddy simulation: results of a workshop. ASME J Fluids Eng 1997;119: 248–62.
[32] Lyn DA, Einav S, Rodi W, Park JH. A laser-doppler velocimeter study of ensemble-averaged characteristics of the turbulent near wake of a square cylinder. J Fluid Mech 1995;304:285–319.
[33] Lyn DA, Rodi W. The flapping shear layer formed by flow separation from the forward corner of a square cylinder. J Fluid Mech 1994; 267:353–76.
[34] Bearman PW, Obasaju ED. An experimental study of pressure fluctuations on fixed and oscillating square-section cylinders. J Fluid Mech 1982;119:297–321.