A Parallel Compressible 3D Navier-Stokes Solver Using Unstructured Meshes

Mark LORIOT[*] Loula FEZOUI[◇]

[*] SIMULOG, Les Taissounières HB2, Route des Dolines, 06560 VALBONNE, FRANCE.
[◇] CERMICS/INRIA Sophia-Antipolis, 2004, Route des Lucioles, 06560 VALBONNE, FRANCE.

Abstract

We describe in this paper a strategy for parallelising a 3D compressible Navier-Stokes solver using unstructured meshes on a class of message-passing MIMD machines. The performance results obtained on two parallel machines, the Meiko Concerto and Intel iPSC860 are presented and compared to those obtained on CRAY.

Introduction

Defining a way to implement a given serial algorithm on a parallel machine, depends of course on the level of parallelism of the algorithm as well as on some of the characteristics of the selected machine, but also on some previously set objectives. As an example, in some of our previous works ([6], [11], our goal was to reach a maximum efficiency, accepting thus to introduce a large set of modifications in the existing algorithm and sometimes to write the program again completely. This proved to be necessary for some machines (the Connection Machine CM-2, for instance) and remains feasible for "small" programs which correspond here to the one- and two-dimensional cases.
In the present paper, we deal with an existing industrial code, NSTC3D (3D Compressible Navier-Stokes [7] which already runs on scalar and vector machines. A minimum of modifications in the program source is required when building a parallel version, in order to be able to easily maintain and upgrade this program. Moreover, we want the resulting parallel program to be as little "machine-dependant" as possible. That is, it must run on both serial and parallel machines (at least of the same kind) with no further modification. From this viewpoint, the highest efficiency to be achieved on any particular machine is left to an optimisation phase which will not be discussed here.
Only a certain kind of multi-processor machines will be addressed in this paper, although the methodology we use also successfully applies to shared memory systems [10]. The parallel machines addressed here are of MIMD distributed memory type with a message-passing communication model. Such a choice is not so restrictive since a lot of existing parallel machines belong to this class (Intel hypercube, Paragon, Meiko, Ncube, etc..).

The main characteristics of NSTC3D is that it is based on finite volume schemes using finite element type grids (here tetrahedra), which results in complex data structures. A simple way to achieve a parallel implementation is to split the mesh among the processor memories. This is done here via a "preprocessor" which generates also all the necessary data to perform communications between the split domains (ie the processors). A two-dimensional version of this preprocessor with a set of experiment results may be found in [4],[5].
We recall in the first section of this paper the equations to be solved, and present briefly the numerical method used. In the second section we describe the mesh-decomposition software. Numerical solutions and performance comparisons are presented in the last section .

1 The FEM/FVM Navier Stokes solver

In the sequel we shall describe briefly the method used here to solve numerically the Navier-Stokes system. This method is based on a finite volume formulation applied to finite element type discretisation of computational domains. For more details on the method, one may refer to [2] and [7] and the references included.
The conservative form of the 3D Navier-Stokes system describing the compressible and viscous flow of a perfect gas writes as follows:

$$\frac{\partial W}{\partial t} + \vec{\nabla}.\vec{\mathcal{F}}(W) = \frac{1}{Re}\vec{\nabla}.\vec{\mathcal{R}}(W) \tag{1}$$

where W is the vector of the conservative variables which are respectively the density ρ, the momentum vector $\rho\vec{V}$ and the total energy E. Re is the Reynolds number.
The pressure P is related to the other variables through the perfect gas state equation:

$$P = (\gamma - 1)\left(E - \frac{1}{2}\rho(2 + v^2 + w^2)\right)$$

where γ is the ratio of specific heats (1.4 for air).

The definition of the convective fluxes $\vec{\mathcal{F}}$ and the viscous fluxes $\vec{\mathcal{R}}$ may be found in any classical fluid mechanics book and one may also refer to [7] for the model used here.

1.1 The Spatial Discretisation

Since we are interested in using finite volume approximations on unstructured meshes, we have to construct a partition of the computational domain Ω, made of control volumes. Unstructured meshes result from a standard finite element triangulation (2D) or tetrahedrisation (3D). The following figure (1) shows a control cell in 2D and 3D space dimensions.

The main idea behind the mixed finite element/finite volume approximation used here is the use of a different approach for each side of the system (1). The left hand side (Euler part) is approximated by means of a finite volume Galerkin (FVG) approximation. A description of such an approach in two space dimensions may be found in [3]. The extension to the 3D case is straightforward. A classical finite element Galerkin (FEG)

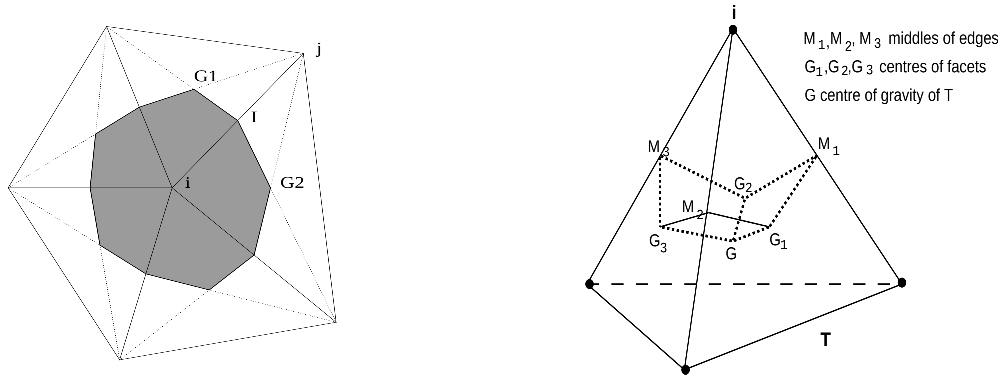

G1
j
I
i
G2
i
M_1,M_2, M_3 middles of edges
G_1,G_2,G_3 centres of facets
G centre of gravity of T
M_3
M_1
G_2
M_2
G_3
G
G_1
T

Figure 1: 2D cell (left) and contribution to 3D cell (right)

approximation is used to discretise the right hand side (viscous part) of (1).
This results in the following semi-discrete system:

$$vol(C_i)\frac{dW_i(t)}{dt} + \int\limits_{\partial C_i} \vec{\mathcal{F}}(W).\vec{\eta} d\sigma + \int\limits_{\partial C_i \cap \partial\Omega} \vec{\mathcal{F}}(W).\vec{\eta} d\sigma = -\frac{1}{Re}\sum_{T, i\in T} \iiint\limits_{T} \vec{\mathcal{R}}(W).\vec{\nabla}\varphi_i^T \, d\omega$$

where $\vec{\eta}$ is the outward normal to the boundary ∂C_i and φ_i^T is the restriction to the element T of the P_1 basis function of node i.

Convective Fluxes

The approximation of the Euler part of the system is done in two stages: first we apply any monotonic upwind scheme extended to the case of unstructured meshes [3] and then perform the second order accuracy using a linear interpolation around each grid vertex.

First Order Explicit Upwind Fluxes

We split the boubondary of an internal cell C_i into an union of boundary interfaces $\partial C_i \cup \partial C_j$ where j denotes a neighbouring node of i:

$$\int\limits_{\partial C_i} \vec{\mathcal{F}}(W).\vec{\eta} d\sigma = \sum_{j\in K(i)} \int\limits_{\partial C_i \cup \partial C_j} \vec{\mathcal{F}}(W).\vec{\eta} d\sigma$$

where $K(i)$ is the set of neighbouring nodes of node i.

$$\int\limits_{\partial C_i \cup \partial C_j} \vec{\mathcal{F}}(W).\vec{\eta}_{ij} d\sigma = \Phi(W_i, W_j, \vec{\eta}_{ij}) = \Phi_{ij}$$

The upwind numerical approximation selected here is based on van Leer's Flux function ([9]). One may use however any other upwind centered scheme such as Roe's or Osher's scheme for example.

$$\Phi_{ij} = F^+(W_i) + F^-(W_j)$$

The definition of the split fluxes may be found in [9].

Second order extension

Following the MUSCL (Monotonic Upwind Scheme for Conservation Laws) introduced by van Leer ([8]), one may use, to enhance the accuracy, the same numerical flux function (as in the first order scheme) applied now to certain values W_{ij}, W_{ji} of the vector W at the interface of cells C_i and C_j.

$$\begin{aligned}\Phi_{ij} &= \Phi(W_{ij}, W_{ji}, \vec{\eta}_{ij}) \\ W_{ij} &= W_i + \frac{1}{2}(\vec{\nabla} W)_i.\vec{ij} \\ W_{ji} &= W_j - \frac{1}{2}(\vec{\nabla} W)_j.\vec{ij}\end{aligned}$$

The gradient of W at node i may be defined as follows:

$$\text{vol}(C_i).(\vec{\nabla} W)_i = \sum_{T, i\in T} \frac{\text{vol}(T)}{4} \sum_{k=1}^{4} W_k.\vec{\nabla}\varphi_k^T$$

We shall not use here any limitation technique for we apply the van Leer scheme (which has a non negligible amount of numerical viscosity) to viscous flow calculations at relatively low Mach numbers. We may found however on [3] and [1] how to define and use such limitation techniques.

Viscous Fluxes

To compute the numerical viscous fluxes, we use a classical Galerkin approximation:

$$[\text{Flux}]_i^v = \sum_{T, a_i\in T} \iiint_T \vec{\mathcal{R}}(W).\vec{\nabla}\varphi_i^T d\omega$$

$$[\text{Flux}]_i^v = vol(T)\left(R_1(T)\frac{\partial \varphi_i^T}{\partial x} + R_2(T)\frac{\partial \varphi_i^T}{\partial y} + R_3(T)\frac{\partial \varphi_i^T}{\partial z}\right)$$

Where $R_1(T)$, $R_2(T)$ and $R_3(T)$ are defined as follows:

$$R_l(T) = \frac{1}{4}\sum_{k\in T} R_{l,k}, \quad l = 1,3$$

$R_{l.k}$ is the value of R_l at vertex k of triangle T.

Boundary Conditions

To solve numerically system (1) one has to add some boundary conditions which are in general of two types: a Dirichlet condition on the velocity and temperature on the wall in the case of external flows around a body, and in-out conditions for internal flows. A free stream flow condition is also considered for external flows on the artificial boundary (infinity). We refer to ([7], 89-97) for a detailed description of the boundary conditions used and their numerical treatment.

Time Integration

The spatial approximation beeing defined, one obtains the following semi-discrete system:

$$\frac{\partial W}{\partial t} + \Psi(W) = 0$$

As in our previous works ([11],[5],[6]), we choose again explicit schemes for they are easy to parallelise and require a low storage of data. For second order computations we use a 3-stage Runge-Kutta or a predictor-corrector scheme whereas RK1 (1-stage) is used when computing first-order accurate (in time and space) solutions.

RK3 Scheme

$$\left\{\begin{array}{ll} W^{(0)} & = W^{n} \\ W^{(k)} & = W^{(0)} - \dfrac{\Delta t}{4-k}\Psi\left(W^{(k-1)}\right) \\ W^{(n+1)} & = W^{(3)} \end{array}\right.$$

Predictor-Corrector Scheme

An alternative to Runge-Kutta schemes is a predictor-corrector scheme which is second-order accurate only and less stable than the RK3 but also much cheaper in terms of CPU costs (this is often a requirement of industrial codes). The scheme used here was suggested by Hancock and presented by Van Leer.

First we predict a state $\tilde{W}^{n+\frac{1}{2}}$ using the Euler equations:

$$\tilde{W}_i^{n+\frac{1}{2}} = \tilde{W}_i^{n} - \frac{\Delta t}{2}\frac{\partial \vec{\mathcal{F}}}{W}(W)\vec{\Delta}\tilde{W}_i)$$

In the second phase (correction), the fluxes are evaluated using the predicted state:

$$W_i^{n+1} = W_i^{n} - \Delta t \Phi(\tilde{W}_{ij}^{n+\frac{1}{2}}, \tilde{W}_{ji}^{n+\frac{1}{2}}, \vec{\eta}_{ij}) + [Flux]_i^v$$

Local Time Stepping

We use a local time stepping technique to speed up the convergence when simulating steady flows. This is achieved using a CFL-like condition on each cell C_i:

$$\left[(\|\vec{V}\| + c)h_i + \frac{2\gamma\lambda}{\rho Pr Re}\right]\frac{\Delta T}{h_i^2} \leq 1$$

where c is the local speed of sound, Pr is the Prandtl number and h_i is the largest altitude in the elements surrounding the node i.

Algorithm

The method described above leads to an iterative algorithm which writes, when using RK1, as follows:

At each time step, Compute:

(1) Local or global Time-step
(2) Gradients at Nodes
(3) Diffusive Fluxes
(4) Convective Fluxes
(5) Update the Solution

In a serial implementation, stages (1), (2) and (3) result in loops on tetrahedra, while step (4) and step (5) result in loops on edges and nodes respectively. This mixed data structure comes from the mixed finite element/finite volume formulation.
We recall that a vector mode implementation needs colouring techniques for the scatter vector operations involved in the calculation above.

2 MIMD PARALLELISATION

2.1 Mesh decomposition

We chose to parallelise our code using a *data partitioning* approach. This leads to the decomposition of the initial finite element mesh into several submeshes, each of them being assigned to a processor. This decomposition phase, as well as the generation of all the communication data which are used by the parallel code, is performed by a preprocessor. This preprocessor runs on a workstation.

2.2 Overlapping of subdomains

The best way to achieve one of our main goals, that is, make as little modification to the original serial code as possible, is to reduce as much as possible communications. This is done by providing all the data indispensable to go through a whole time step locally, even if this induces extra memory storage and redundant computations. This leads to overlapping subdomains, where all the information needed form the neighbours is fetched only once at each time step. The size of the interface shared by two neighbouring subdomains depend on the order of the spatial approximation. This choice induces only tiny changes to the original non-parallel code, at the cost of having some computations done twice.

2.3 Parallel Algorithm and communications

The parallel algorithm is quite similar to the algorithm implemented on scalar machines. There is only a 6^{th} stage added, which is :

(6) Update the Interface

This stage requires communications between two subdomains sharing the same interface (for example: processor 1 sends the new values of its solution at its interface points to processor 2, and vice versa). However, there is also sometimes a need for global communications. This happens if a global time step is used (unsteady calculations), or to make a global evaluation of the residue in order to analyse the convergence (such global residue evaluations are in practice performed only from time to time)

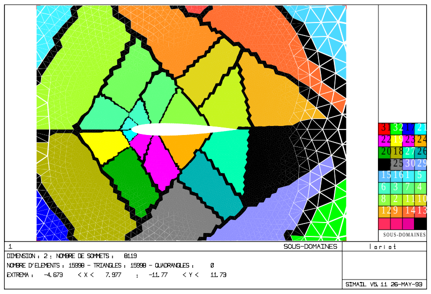

SOUS-DOMAINES
1
SOUS-DOMAINES
loriot
DIMENSION : 2 ; NOMBRE DE SOMMETS : 8119
NOMBRE D'ELEMENTS : 15998 - TRIANGLES : 15998 - QUADRANGLES : 0
EXTREMA : -4.673 < X < 7.977 ; -11.77 < Y < 11.73
SIMAIL V5.11 26-MAY-93

Figure 2: 2D Inertia Axis decomposition into 32 subdomains - zoom close to the body - first order interface

2.4 use of a portable communication library : PARMACS

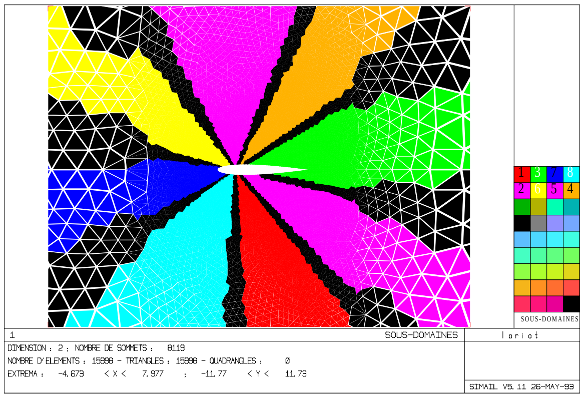
We described above how an easy "migration", that is, port a serial code to a parallel machine, was achieved. Another problem remains in an industrial context: port the code from one parallel platform to another. This is solved by using a portable message-passing library such as PARMACS, which ensures maximum portability and provides mapping tools.

2.5 Automatic mesh splitting

Automatic splitting algorithms are used in the preprocessor. They are based on efficient heuristics, and achieve good load balance and efficient communication graphs very fast. These algorithms may generally be used for both 2D and 3D meshes. They are based on the coordinates (median, sector, inertia axis...) or on the connectivity (greedy). We may refer to [5] and [4] for more details about these algorithms.
We give on fig.(2) and fig.(3) examples of 2D decompositions of a mesh representing the computational domain around a naca0012 airfoil(first order interface); on fig.(4), (5) and fig.(6) are examples of 3D decompositions of a mesh representing the computational domain inside an engine combustion chamber (first order interface).

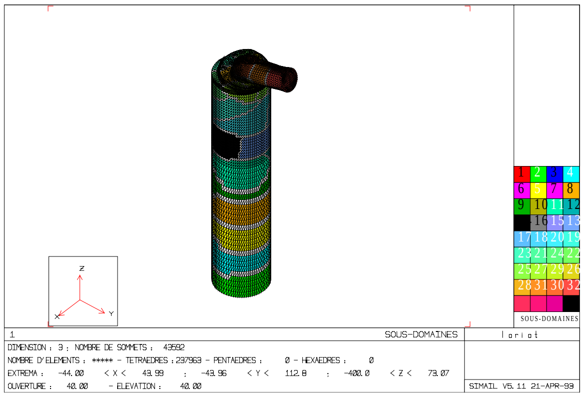
3 Numerical results

This section consists of two parts. We give in the first part the results obtained for two computations: the first is the simulation of a 2D external flow around a NACA0012 airfoil, the second is the simulation of a 3D internal flow inside the combustion chamber of an engine.
In the second part, performance results for both 2D and 3D computations are presented for a wide range of meshes, and discussed. Both the 2D and 3D codes used are fully vectorised, by using whenever necessary colouring techniques.



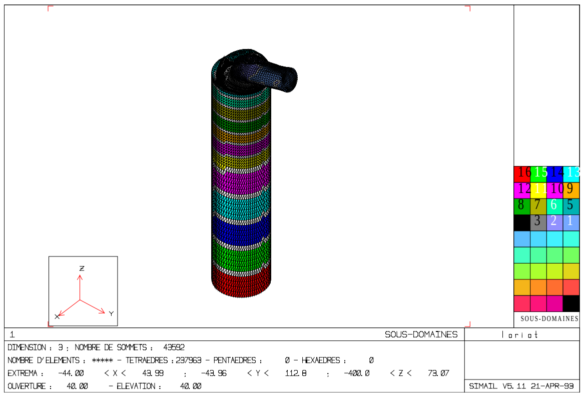
```
1                                                     SOUS-DOMAINES   loriot
DIMENSION : 2 ; NOMBRE DE SOMMETS :   8119
NOMBRE D'ELEMENTS : 15998 - TRIANGLES : 15998 - QUADRANGLES :     Ø
EXTREMA :   -4.673    < X <   7.977    ;   -11.77    < Y <   11.73
                                                                      SIMAIL V5.11 26-MAY-93
```

Figure 3: 2D Sector decomposition into 8 subdomains - zoom close to the body - second order interface



```
1                                                     SOUS-DOMAINES   loriot
DIMENSION : 3 ; NOMBRE DE SOMMETS :  43592
NOMBRE D'ELEMENTS : ***** - TETRAEDRES :237963 - PENTAEDRES :    Ø - HEXAEDRES :    Ø
EXTREMA :   -44.ØØ   < X <   43.99   ;   -43.96   < Y <   112.8   ;   -4ØØ.Ø   < Z <   73.Ø7
OUVERTURE :   4Ø.ØØ    - ELEVATION :    4Ø.ØØ                         SIMAIL V5.11 21-APR-93
```

Figure 4: 3D Inertia Axis decomposition into 32 subdomains - first order interface



```
1                                                     SOUS-DOMAINES   loriot
DIMENSION : 3 ; NOMBRE DE SOMMETS :  43592
NOMBRE D'ELEMENTS : ***** - TETRAEDRES :237963 - PENTAEDRES :    Ø - HEXAEDRES :    Ø
EXTREMA :   -44.ØØ   < X <   43.99   ;   -43.96   < Y <   112.8   ;   -4ØØ.Ø   < Z <   73.Ø7
OUVERTURE :   4Ø.ØØ    - ELEVATION :    4Ø.ØØ                         SIMAIL V5.11 21-APR-93
```

Figure 5: 3D front decomposition into 16 subdomains

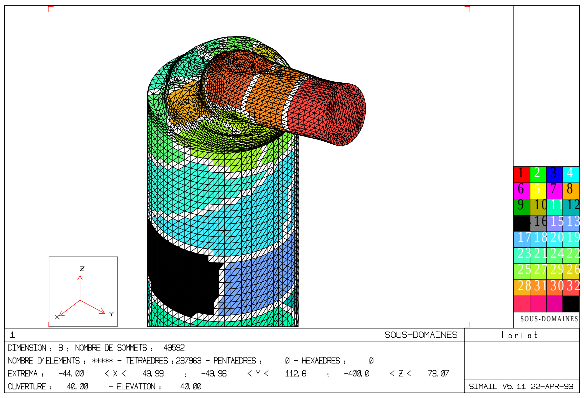


Figure 6: 3D Inertia Axis decomposition into 32 subdomains - zoom close to admission pipe

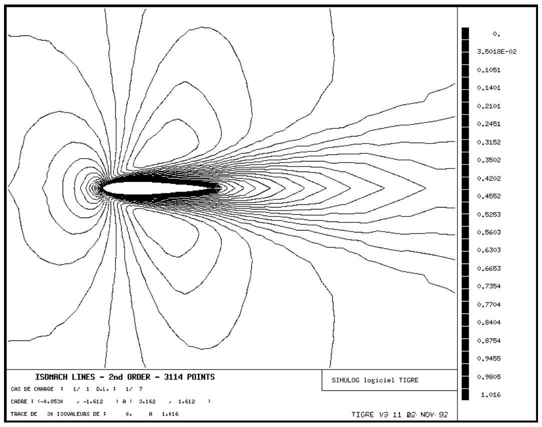


Figure 7: Isomach lines at convergence - zoom close to the body

3.1 2D Navier-Stokes results

We showed on fig. (3) the decomposition used for the second order calculation of the viscous flow around a NACA0012 airfoil. This decomposition was selected because it reduces the number of neighbours to 2, and therefore the number of messages, and this is most important for an explicit calculation. There are 8 processors used in this case. The mesh has 3114 vertices. The calculation was made with no incidence, and a free stream Mach number of .85 . The Mach number is 500, and the body temperature 349 K (see fig. (7)).

3.2 3D Navier-Stokes results

We showed on fig. (6) the decomposition used for the calculation of the viscous flow inside a combustion chamber. The calculation was performed on a Meiko Concerto, with 16 i860

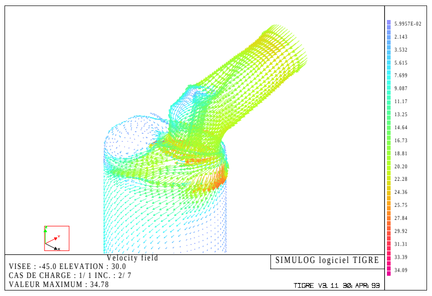


Figure 8: Velocity field at convergence - zoom close to the body

processors used in this case. The grid contains 43,592 vertices and 237,963 tetrahedra. The calculation was made with a incoming Mach number of .05. Temperature on the walls is 294 K, flow rate is imposed for the incoming flow, and the pressure for the outcoming flow is of $.37.10^5$ Pa . We show on fig. (8) a zoom close to the admission pipe.
It is interesting to notice that the preprocessing phase for this computation (including decomposition, regularisation, load balance optimisation and communication map generation) took only 15 minutes on a Sun Sparc 2, the major part of this time (75 %) being taken by the mesh and data structure file I/O.

3.3 Performance results

We give here the performance results for the Navier-Stokes solver. The computations made are second order in time and space accurate, and performed in 32 bit and 64 bit precision.
We compared the performance obtained on the three following machines: a CRAY-YMP and CRay 2 both with one processor, a Meiko-CS1 (Concerto) with 16 i860 processors, and an Intel iPSC860 with 128 i860 processors (only up to 64 processors were used). We will give here only results in single precision, since double precision results on both the Meiko and the iPSC860 may be obtained directly by applying a loss of 30 % in CPU performance to the single precision results.
All the figures given below include the idle and communication times of the processors. All performances were obtained when using the VAST vectorizer and the Greenhills compiler (g860apx) with the -OMA options on the Meiko, and the Portland compiler with the -vector -noieee -O4 on the iPSC860.
PARMACS was used in all the computations as the message passing library.

3.3.1 2D case

For the 2D case, the meshes represent the computational domain around a NACA0012 airfoil. The decomposition algorithms used vary depending on the number of subdomains of the decomposition.
The global grid size varies from 8119 to 68877 vertices. We give on fig. (9) detailed results in CPU time, percentage of communication and performance in terms of Mflops (including communications). One may observe that for a fixed grid size, the global CPU

Nb points	Machine	Nb procs	Mflops	total CPU time s/iter	% commuc.
8119	Meiko CS1	16	126.9	0.383	4.7 %
8119	iPSC860	16	97.5	0.499	8.8 %
8119	Cray 2	1	79.22	0.41	0 %
8119	Cray YMP	1	163	0.2	0 %
16116	Meiko CS1	16	128.4	0.655	4.6 %
16116	iPSC860	16	101.5	0.828	4.4 %
16116	Cray 2	1	81.7	0.774	0 %
16116	Cray YMP	1	164	0.386	0 %
32236	Meiko CS1	16	132.4	1.21	2.2 %
32236	iPSC860	16	101	1.59	2.7 %
32236	Cray 2	1	80.3	1.61	0 %
32236	Cray YMP	1	163	0.80	0 %
68877	Meiko CS1	16	132.6	2.43	1.6 %
68877	iPSC860	16	99.4	3.25	1.9 %
68877	iPSC860	32	196	1.59	5 %
68877	iPSC860	64	386	0.85	6.7 %
68877	Cray 2	1	77.7	3.57	0 %
68877	Cray YMP	1	163.4	1.70	0 %

Figure 9: Compared 2D Performance results with single-precision

time decreases as the number of processors increases at approximatively the same rate. The part due to the communications is small (around 5 %). These results show that the cost of the three-element wide overlapping is also very small (10 % of CPU time in the case of 32 processors, 15 % for 64 processors) when comparing the CPU times to the Mflops, and that one needs a 32-processor hypercube to go faster than a one-processor CRAY YMP when only single precision is needed (however we recall that the Cray uses 64 bit precision)

3.3.2 3D case

For the 3D case, the meshes represent the computational domain inside a rectanguler closed cavity. We chose this simple geometry in order to have an easy scalability for the number of mesh vertices. The global grid size varies from 10,000 to 80,000 vertices. We give on

Nb points	Machine	Nb procs	Mflops	total CPU time s/iter	% comm.
10000	Meiko CS1	16	95	1.0	4.8 %
10000	iPSC860	16	78.6	1.2	4.3 %
10000	Cray 2	1	66.6	0.9	0 %
10000	Cray YMP	1	140.7	0.45	0 %
20280	Meiko CS1	16	98.5	1.85	3.2 %
20280	iPSC860	16	79.8	2.3	3.3 %
20280	Cray 2	1	66.5	1.9	0 %
20280	Cray YMP	1	140.6	0.9	0 %
40500	Meiko CS1	16	99.9	3.3	3.2 %
40500	iPSC860	16	79.1	4.2	3.9 %
40500	Cray 2	1	65.4	4.0	0 %
40500	Cray YMP	1	139.7	1.9	0 %
80000	Meiko CS1	16	101	6.5	2.8 %
80000	iPSC860	16	80.1	8.3	3.2 %
80000	iPSC860	32	147	5.4	8.3 %
80000	iPSC860	64	251	3.7	15 %
80000	Cray 2	1	60.6	8.7	0 %
80000	Cray YMP	1	-	-	-

Figure 10: Compared 3D Performance results with single-precision

fig. (10) detailed results in CPU time, percentage of communication and performance in terms of Mflops (including communications). No result was available on the CRAY-YMP for the large mesh due to memory shortage.
These performance results show that in the 3D case, the cost of the overlapping is much higher than in 2D. We already get a 20 % extra cost when comparing the Mflops and CPU time for the 40500 node mesh, and this would probably reach 40 % for 64 processors. However, we must point out that we did not try to use an optimal decomposition, nor load balance optimisation in the case of 64 processors, in order to be in a realistic “industrial environment”. We could probably improve these results by taking an optimal number of processors for a given problem.
Here again, the cost of communications is relatively low (less than 15 %)

Performance Comparisons

We give below on fig. (11) and fig. (12) charts allowing comparisons between the supercomputers used. We must point out that the code used (NSTC3D) was *rigorously* the same on the MIMD computers, and only a few communication lines were commented out in order to run the code in sequential mode on the CRAY machines. Only the parallel results obtained with 16 processors are shown, since the others were available only for the largest mesh. Let us emphasize the fact that these results were made in 32-bit precision on the parallel machines, and that the use of 64-bit precision, such as the one used on CRAY machines, induces a loss of 30 % in CPU time on these machines.

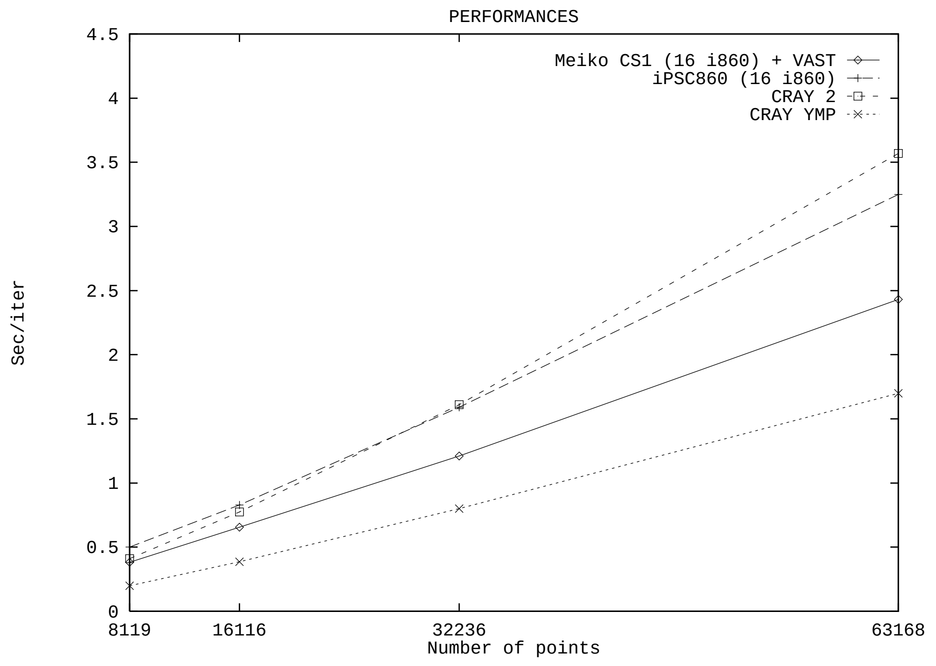


Figure 11: Comparison of CPU times for the 2D computations

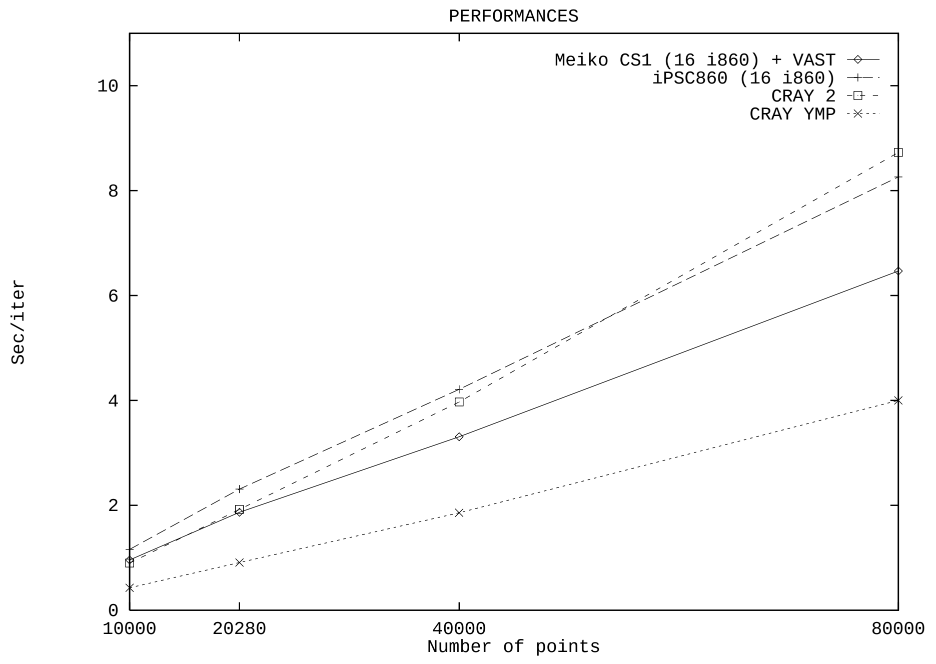


Figure 12: Comparison of CPU times for the 3D computations

4 Conclusion

The presented strategy of parallelisation, based on overlapping mesh decompositions, proved to be a good one to reach the preliminary fixed objectives which were : first to keep the same source code for both serial and parallel codes, secondly : ensure that the resulting program would still run on serial machines (scalar or vector) with no loss in efficiency. The first goal was completely achieved through our strategy since less than 0.05% of the overall program was modified. Concerning the second objective, one only has to comment out the communication statements in the parallel program to get back to a serial version. The use of a portable communication library such as PARMACS helps the porting of the program from one parallel machine to another. However the presented results show that the use of a three element-wide overlapping is much more costly in 3D than in 2D. Let us point out that no optimisation was performed wich could be in opposition with our first goal. Further optimisation would indeed induce other changes in the code (another communication phase would then have to be added) Nevertheless, the preliminary results presented here show that the communications are not costly since they constitute only 5% of the global CPU time in 2D and reach a maximum of 15% in 3D cases. Let us emphasise that these results may be viewed as excellent ones since the program has a complex data structure (finite volumes with unstructured meshes) and is used for undustrial applications. Moreover, this methodology enables easy modifications in terms of physical models and easy migration: it took indeed with this technique one day to migrate a 2D serial Maxwell solver , and two days to migrate an industrial product such as NSTC3D to parallel machines.

Acknowledgments

The authors wish to thank Pierre Leca from ONERA for providing access to the hypercube, and Armel De la Bourdonnaye from CERMICS/ONERA and Didier Chargy from SIMULOG, for their kind help.

References

[1] A. Dervieux, *Steady Euler Simulations Using Unstructured Meshes* Von Karmann Insitute Lecture Series, 1985.

[2] L. Fezoui, S. Lantéri, B. Larrouturou, C. Olivier, *Résolution numérique des équations de Navier-Stokes pour un fluide compressible en maillage triangulaire* Rapport de Recherche Inria N^0 1033, mai 1989

[3] L. Fezoui and B. Stoufflet, *A class of implicit upwind schemes for Euler simulations with unstructured meshes*, J. of Comp. Phys. vol 84, pp. 174-206, (1989).

[4] M. Loriot, L. Fezoui, *FEM/FVM Calculations of compressible flows on a Meiko system* BECAUSE European wokshop - Sophia-Antipolis - France 14-16 Oct. (1992)

[5] L. Fezoui, F. Loriot, M. Loriot and J. Régère, *2-D Finite Volume / Finite Element Euler Solver on a MIMD Parallel Machine*, "High Performance Computing II", Montpellier 1991, M. Durand and F. El Dabaghi Ed., Elsevier Science Publishers, North-Holland, (1991).

[6] L. Fezoui and S. Lantéri, *Parallel Upwind FEM for Compressible Flows*, “Parallel CFD Conference”, Stuttgart 1991, K.G. Reinsch et al. Ed., North-Holland (1992).

[7] SIMULOG, *NSTC3D, Manuel Theorique, Vrsion I.1*, 1992.

[8] B. Van Leer, *Towards the Ultimate Conservative Difference V: a Second-Order Sequel to Godunov's Method*, J. Comp. Phys., Vol. 32, p. 361-370 (1970).

[9] B. Van Leer, *Flux-Vector Splitting for the Euler Equations*, Lectures Notes in Phys., Vol. 170, p. 507-512, (1982).

[10] S. Lantéri, C. Farhat, Viscous flow computations on MPP systems: implementational issues and performance results for unstructured grids, Sixth SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, (1993).

[11] C. Farhat, L. Fezoui and S. Lantéri, Two-dimensional viscous flow computations on the CM-2: Unstructured Meshes, Upwind Scheme and Massively Parallel Computations, Comp. Meth. in Appl. Mech. and Eng.,Vol. 102, No. 1, pp. 61-88 (1993).