A Multi-platform Shared- or Distributed-Memory Navier-Stokes Code

F. Chalot[a], Q.-V. Dinh[b], M. Mallet[b], A. Naïm[a], and M. Ravachol[b]

[a]Dassault Aviation,
Direction Technique Aéronefs/Modélisation Aérodynamique,
78, quai Marcel Dassault, 92214 Saint-Cloud, France

[b]Dassault Aviation,
Direction Prospective/Département des Etudes Scientifiques Amont,
78, quai Marcel Dassault, 92214 Saint-Cloud, France

This paper describes the implementation of a finite-element Navier-Stokes code on two parallel architectures: the IBM SP2 and the NEC SX-4. Although these implementations are based on two different memory representations, the shared- and the distributed-memory paradigms, the actual source codes are extremely similar, thanks to the finite-element structure of the program. Two industrial applications exemplify the use of parallel computers in the Aerodynamic Modelization Department at Dassault Aviation.

1. INTRODUCTION

Since its beginning in the seventies, Computational Fluid Dynamics has always been an avid consumer of computing resources, and thus developed concurrently with the progresses in computer hardware. Born with the good old mainframes, CFD came to maturity with the vector architectures of the eighties (Cray, Convex, IBM, NEC). The late eighties and the early nineties saw the emergence of RISC-based workstations which outgrew the vector supercomputers in power and ease of programming. But only the new massively parallel architectures were candidates to meet the TeraFLOPS challenge of the nineties [10] and the ever increasing computing power demands of CFD (complex 3-D geometries, unsteady Navier-Stokes computations, ...).

Dassault Aviation was no exception in its experience of CFD with the evolution of computer hardware and went through the necessary code alterations. Dassault Aviation early dedication to unstructured meshes and to finite element type methods induced a natural interest towards domain decomposition and parallel processing. At some point a version of the code was ported to the Connection Machine series of Thinking Machines (CM-2, CM-200, and CM-5) [6]. However, Dassault Aviation's own experience with parallelism was still limited to automatic procedures provided by compilers on IBM and Cray architectures.

In 1992, `VIRGINIE`, the Navier-Stokes code in use at Dassault Aviation, was successfully ported onto the Intel IPSC 860 at ONERA [3]. The first in house massively parallel architecture dedicated to CFD was an IBM SP1, soon upgraded to an SP2. The same

message passing techniques as those used for the IPSC were applied to the SP2 version.

Dating back from the time of the Hermès program, Dassault and the NLR have cooperated on Navier-Stokes computations using NLR's NEC SX-3 [4]. The high level of vectorization of Dassault-Aviation's Navier-Stokes code enabled performances in the 500 MFLOPS range. Since the replacement of NLR's SX-3 with an SX-4, the code had to be revisited to take advantage of the parallel capabilities.

We describe here the implementations of `VIRGINIE` on both the IBM SP2 and the NEC SX-4.

2. DESCRIPTION OF THE CODE

Dassault Aviation's Navier-Stokes code, called `VIRGINIE`, uses a finite element approach, based on a symmetric form of the equations written in terms of entropy variables. The advantages of this change of variables are numerous: in addition to the strong mathematical and numerical coherence they provide (dimensionally correct dot product, symmetric operators with positivity properties, efficient preconditioning), entropy variables yield further improvements over the usual conservation variables, in particular in the context of chemically reacting flows (see [1]).

2.1. The symmetric Navier-Stokes equations

As a starting point, we consider the Euler and Navier-stokes equations written in conservative form:

$$\boldsymbol{U}_{,t} + \boldsymbol{F}^{\text{adv}}_{i,i} = \boldsymbol{F}^{\text{diff}}_{i,i} \tag{1}$$

where $\boldsymbol{U}$ is the vector of conservative variables; $\boldsymbol{F}^{\text{adv}}_i$ and $\boldsymbol{F}^{\text{diff}}_i$ are, respectively, the advective and the diffusive fluxes in the i^{th}-direction. Inferior commas denote partial differentiation and repeated indices indicate summation.

Equation (1) can be rewritten in quasi-linear form:

$$\boldsymbol{U}_{,t} + \boldsymbol{A}_i \boldsymbol{U}_{,i} = (\boldsymbol{K}_{ij}\boldsymbol{U}_{,j})_{,i} \tag{2}$$

where $\boldsymbol{A}_i = \boldsymbol{F}^{\text{adv}}_{i,\boldsymbol{U}}$ is the i^{th} advective Jacobian matrix, and $\boldsymbol{K} = [\boldsymbol{K}_{ij}]$ is the diffusivity matrix, defined by $\boldsymbol{F}^{\text{diff}}_i = \boldsymbol{K}_{ij}\boldsymbol{U}_{,j}$. The $\boldsymbol{A}_i$'s and $\boldsymbol{K}$ do not possess any particular property of symmetry or positiveness.

We now introduce a new set of variables,

$$\boldsymbol{V}^T = \frac{\partial \mathcal{H}}{\partial \boldsymbol{U}} \tag{3}$$

where $\mathcal{H}$ is the generalized entropy function given by

$$\mathcal{H} = \mathcal{H}(\boldsymbol{U}) = -\rho s \tag{4}$$

and s is the thermodynamic entropy per unit mass. Under the change of variables $\boldsymbol{U} \mapsto \boldsymbol{V}$, (2) becomes:

$$\widetilde{\boldsymbol{A}}_0 \boldsymbol{V}_{,t} + \widetilde{\boldsymbol{A}}_i \boldsymbol{V}_{,i} = (\widetilde{\boldsymbol{K}}_{ij}\boldsymbol{V}_{,j})_{,i} \tag{5}$$

where

$$\widetilde{\boldsymbol{A}}_0 = \boldsymbol{U}_{,\boldsymbol{V}} \tag{6}$$
$$\widetilde{\boldsymbol{A}}_i = \boldsymbol{A}_i \widetilde{\boldsymbol{A}}_0 \tag{7}$$
$$\widetilde{\boldsymbol{K}}_{ij} = \boldsymbol{K}_{ij} \widetilde{\boldsymbol{A}}_0. \tag{8}$$

The Riemannian metric tensor $\widetilde{\boldsymbol{A}}_0$ is symmetric positive-definite; the $\widetilde{\boldsymbol{A}}_i$'s are symmetric; and $\widetilde{\boldsymbol{K}} = [\widetilde{\boldsymbol{K}}_{ij}]$ is symmetric positive-semidefinite. In view of these properties, (5) is referred to as a symmetric advective-diffusive system.

For a general divariant gas, the vector of so-called (physical) entropy variables, $\boldsymbol{V}$, reads

$$\boldsymbol{V} = \frac{1}{T} \begin{Bmatrix} \mu - |\boldsymbol{u}|^2/2 \\ \boldsymbol{u} \\ -1 \end{Bmatrix} \tag{9}$$

where $\mu = e + pv - Ts$ is the chemical potential per unit mass; $v = 1/\rho$ is the specific volume. This expression for the $\boldsymbol{V}$-variables must be contrasted with that for the conservative variables which can be written in the following form:

$$\boldsymbol{U} = \frac{1}{v} \begin{Bmatrix} 1 \\ \boldsymbol{u} \\ e + |\boldsymbol{u}|^2/2 \end{Bmatrix} \tag{10}$$

Taking the dot product of (5) with the vector $\boldsymbol{V}$ yields the Clausius-Duhem inequality, which constitutes the basic nonlinear stability condition for the solutions of (5). This fundamental property is inherited by appropriately defined finite element methods, such as the one described in the next section.

2.2. The Galerkin/least-squares formulation

The Galerkin/least-squares (Galerkin/least-squares) formulation introduced by Hughes and Johnson, is a full space-time finite element technique employing the discontinuous Galerkin method in time (see [8]). The least-squares operator ensures good stability characteristics while retaining a high level of accuracy. The local control of the solution in the vicinity of sharp gradients is further enhanced by the use of a nonlinear discontinuity-capturing operator.

We consider the time interval $I =]0, T[$, which we subdivide into N intervals $I_n =]t_n, t_{n+1}[$, $n = 0, \ldots, N-1$. Let $Q_n = \Omega \times I_n$ and $P_n = \Gamma \times I_n$ where Ω is the spatial domain of interest, and Γ is its boundary. In turn, the space-time "slab" Q_n is tiled by $(n_{\text{el}})_n$ elements Q_n^e. Consequently, the Galerkin/least-squares variational problem can be stated as

Within each Q_n, $n = 0, \ldots, N-1$, find $\boldsymbol{V}^h \in \mathcal{S}_n^h$ (trial function space), such that for all $\boldsymbol{W}^h \in \mathcal{V}_n^h$ (weighting function space), the following equation holds:

$$\int_{Q_n} \Big(- \boldsymbol{W}_{,t}^h \cdot \boldsymbol{U}(\boldsymbol{V}^h) - \boldsymbol{W}_{,i}^h \cdot \boldsymbol{F}_i^{\text{adv}}(\boldsymbol{V}^h) + \boldsymbol{W}_{,i}^h \cdot \widetilde{\boldsymbol{K}}_{ij} \boldsymbol{V}_{,j}^h \Big)\, dQ$$
$$+ \int_{\Omega} \Big(\boldsymbol{W}^h(t_{n+1}^-) \cdot \boldsymbol{U}(\boldsymbol{V}^h(t_{n+1}^-)) - \boldsymbol{W}^h(t_n^+) \cdot \boldsymbol{U}(\boldsymbol{V}^h(t_n^-)) \Big)\, d\Omega$$

$$
\begin{aligned}
&+ \sum_{e=1}^{(n_{\mathrm{el}})_n} \int_{Q_n^e} \Big(\mathcal{L}\boldsymbol{W}^h\Big) \cdot \boldsymbol{\tau}\Big(\mathcal{L}\boldsymbol{V}^h\Big)\, dQ \\
&+ \sum_{e=1}^{(n_{\mathrm{el}})_n} \int_{Q_n^e} \nu^h g^{ij} \boldsymbol{W}^h_{,i} \cdot \widetilde{\boldsymbol{A}}_0 \boldsymbol{V}^h_{,j}\, dQ \\
&= \int_{P_n} \boldsymbol{W}^h \cdot \Big(- \boldsymbol{F}_i^{\mathrm{adv}}(\boldsymbol{V}^h) + \boldsymbol{F}_i^{\mathrm{diff}}(\boldsymbol{V}^h)\Big) n_i\, dP. \qquad (11)
\end{aligned}
$$

The first and last integrals represent the Galerkin formulation written in integrated-by-parts form. The solution space consists of piecewise polynomials which are continuous in space, but are discontinuous across time slabs. Continuity in time is weakly enforced by the second integral in (11), which contributes to the jump condition between two contiguous slabs, with

$$\boldsymbol{Z}^h(t_n^{\pm}) = \lim_{\varepsilon \to 0^{\pm}} \boldsymbol{Z}^h(t_n + \varepsilon). \qquad (12)$$

The third integral constitutes the least-squares operator where $\mathcal{L}$ is defined as

$$\mathcal{L} = \widetilde{\boldsymbol{A}}_0 \frac{\partial}{\partial t} + \widetilde{\boldsymbol{A}}_i \frac{\partial}{\partial x_i} - \frac{\partial}{\partial x_i}\Big(\widetilde{\boldsymbol{K}}_{ij} \frac{\partial}{\partial x_j}\Big). \qquad (13)$$

$\boldsymbol{\tau}$ is a symmetric matrix for which definitions can be found in [8]. The fourth integral is the nonlinear discontinuity-capturing operator, which is designed to control oscillations about discontinuities, without upsetting higher-order accuracy in smooth regions. g^{ij} is the contravariant metric tensor defined by

$$[g^{ij}] = [\boldsymbol{\xi}_{,i} \cdot \boldsymbol{\xi}_{,j}]^{-1} \qquad (14)$$

where $\boldsymbol{\xi} = \boldsymbol{\xi}(\boldsymbol{x})$ is the inverse isoparametric element mapping, and ν^h is a scalar-valued homogeneous function of the residual $\mathcal{L}\boldsymbol{V}^h$. The discontinuity capturing factor ν^h used in the present work is an extension of that introduced by Hughes, Mallet, and Shakib (see [8]).

A key ingredient to the formulation is its consistency: the exact solution of (1) satisfies the variational formulation (11). This constitutes an essential property in order to attain higher-order spatial convergence.

2.3. Linear solver and residual evaluations

Convergence to steady state of the compressible Navier Stokes equations is achieved through a fully-implicit iterative time-marching procedure based on the GMRES algorithm (see [7]).

A low-storage extension based solely on residual evaluations was developed by Johan [5]. It reveals particularly adapted to parallel processing, where the linear solver often constitutes a painful bottleneck.

Thanks to the finite element approach, the integrals of equation (11) which constitute the residual, are first computed on each element; then the global residual is constructed from the local element residuals using the so-called assembly operation:

$$\boldsymbol{R} = \mathop{\mathsf{A}}_{e=1}^{n_{\mathrm{el}}} \boldsymbol{R}_e \qquad (15)$$

This operation is depicted in Figure 1: the residual contribution of element e, denoted $\boldsymbol{R}_e$, is added to the global residual $\boldsymbol{R}$, converting the local element node numbering to the global numbering.

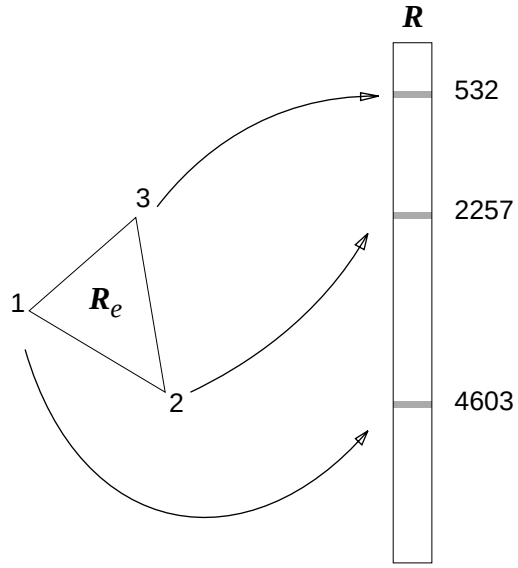

Figure 1. Schematics of the assembly operation.

On a vector architecture, the elements are colored into blocks of disjoint elements in order to avoid recurrence in the assembly operation and to yield the efficient vectorization of the complete residual evaluation process. Element coloring is done through a renumbering into blocks of `lvec` elements. The value of `lvec` is chosen according to the length of the vector registers of the considered computer (typically from 64 to 512).

A residual evaluation can be performed by the FORTRAN loop below:

```
      do 1000 iblk = 1, nelblk
        iel  = lcblk(iblk)
        nvec = lcblk(iblk+1) - iel
        call local (ien(iel), v, ve, ndof)
        call local (ien(iel), x, xe, nsd )
        call getRe (ve, xe, re)
        call globad(ien(iel), r, re, ndof)
1000  continue
```

Loop `1000` consists of a loop over the `nelblk` element blocks. For each block `lcblk(iblk)` gives the number of the first element of the block. `nvec = lcblk(iblk+1) - lcblk(iblk)` is the actual size of the block (`nvec` $\leq$ `lvec`). Quantities such as the entropy variables `v` and the node coordinates `x` then are localized to the elements of block `iblk` (`ien` is the connectivity array which give the global numbers of the nodes in a given element; `ndof` is the number of degrees of freedom per node; and `nsd` is the space dimension). The call to `getRe` contains the actual loop over the elements of the block and evaluates the integrals

of (11) into `re`. Then comes the assembly operation which is performed through the call to `globad` which does the "global add" of `re` into `r`.

This algorithm is the heart of `VIRGINIE` and has proven extremely efficient on many scalar or vector architectures. It is the basis on which all the parallel developments we are going to describe, were built up.

3. A DISTRIBUTED-MEMORY IMPLEMENTATION: THE IBM SP2

The Aerodynamic Modelization Department at Dassault Aviation possesses a 38-processor IBM SP2 with typically from 128 to 256 MBytes of core memory per processor. The SP2 implements a distributed memory paradigm where the interprocessor communications are handled by a high-performance IP-switch.

In order to lay out the data over the processors, the computational domain is partitioned into blocks, each block being affected to a single processor. Blocks are constructed in such a way to balance the number of elements among blocks and to minimize the size of the interface between blocks. The data structure which describes the interface was carefully designed in order to make communications both effective and easy on the programmer. It gives for each block the numbers of blocks which share its interface and, for each of the neighboring block, a list of the nodes along the interface. An important fact is that the order of the nodes in the interface description be the same for two adjacent blocks: this make the exchange of messages transparent with respect to the node numbering local to each block.

Figure 2 shows a typical interface between three two-dimensional blocks made of triangles. One can notice that the node numberings are completely independent from each other. In particular, the node numbers along the interface do not need to (and in fact cannot) be the same from one block to the next: the correspondence is taken care of by the interface description arrays.

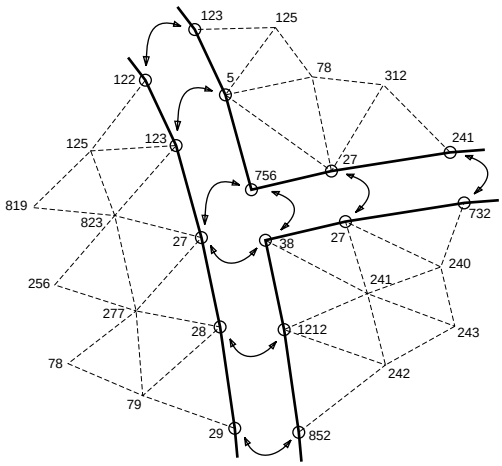

Figure 2. Typical interface between several blocks.

Each processor only sees the piece of the global domain attached to its block. The residual evaluation is performed as in the one-block case, except that the element residuals

must be assembled across the interfaces. The additional assembly operation is performed by the call to `mpassm` in the algorithm below:

```
      do 1000 iblk = 1, nelblk
        iel  = lcblk(iblk)
        nvec = lcblk(iblk+1) - iel
        call local (ien(iel), v, vl, ndof)
        call local (ien(iel), x, xl, nsd )
        call getRe (ve, xe, re)
        call globad(ien(iel), r, re, ndof)
1000  continue
      if ( ntask.gt.1 ) then
        call mpassm(r, ndof, intfbn, intfnn)
      end if
```

As can be seen easily, the call to `mpassm` if `ntask` > 1 (one task is associated with every processor) is the single difference with the scalar code presented in the preceding section. The routine `mpassm` uses the list of block numbers sharing nodes in the interface (`intfbn`) and the list of the interface nodes themselves (`intfnn`). It initializes the reception of incoming messages from all neighboring blocks, gathers the values of the residual on each interface into an outgoing message, sends the outgoing message to the neighboring blocks, and then finally receive the incoming messages and assemble them back to the residual. The send/receive instructions can be performed either by native IBM calls or by MPI primitives.

The extra assembly operation across the block interfaces in the main alteration to the scalar code. GMRES needs a few global scalar product evaluations which are performed separately on each processor using a mask for the interface nodes and then summed over all the processors.

A typical speed-up of 15 can be obtained with 16 processors, which yields a performance in the 500 MFLOPS range.

4. A SHARED-MEMORY IMPLEMENTATION: THE NEC SX-4

NLR's NEC SX-4 is a single-node model composed of 16 processors, which *share* a main memory unit of 4 GBytes. The theoretical peak vector performance of each processor is 2 GFLOPS.

The idea behind the SX-4 port was to keep the alterations to `VIRGINIE` at their minimum and to use a code as close as possible to the SP2 version. At first, we expected to be able to use the SP2 version as is, but unfortunately the message passing library MPI/SX was not available at the time of the port. We had then to consider the concept of multitasking as proposed by NEC. FORTRAN 77/SX provides two options: macrotasking and microtasking. Macrotasking concerns the parallelization of large units on a top-down basis, whereas microtasking deals with the parallelization of loop iterations and statement groups. At first, microtasking seemed too cumbersome, having to treat loops one by one and to decide which would parallelize and which would not. Macrotasking might have been the solution, since we wanted to parallelize large chunks of code at the top level

instead of deep down at the loop level. In fact, as we understand it now, macrotasking enables to fork the parent process at some point and to execute child processes on different other processors: this corresponds more or less to the implementation of an MIMD programming model. In contrast, microtasking "*à la NEC*" permits, with the insertion of directives, the parallelization of loops that may even contain subroutine calls. This is indeed what we need: if we can parallelize a loop high enough in the subroutine structure, then we are guaranteed to keep the changes to the code minimal. The only delicate point is to make sure that there exits no data dependency between processors which may lead to "critical section" if detected or to unpredictable results if unnoticed.

The candidate loop is the loop over the colored element blocks described earlier. Here is how we modified it to parallelize on the NEC SX-4:

```
      do 1000 iblk = 1, nelblk
        ielc   = lcblk(iblk)
        ielend = lcblk(iblk+1)
*pdir pardo
        do 1001 iel = ielc, ielend - 1, lvec
          nvecp = min(lvec,ielend-iel)
          call localp (ien(iel), v, ve, ndof,
     &                 nvecp)
          call localp (ien(iel), x, xe, nsd,
     &                 nvecp)
          call getRep (ve, xe, re, nvecp)
          call globadp(ien(iel), r, re, ndof,
     &                 nvecp)
1001    continue
1000  continue
```

The first thing to note is that the blocks were made artificially longer to allow for vectorization *and* parallelization. In practice the blocks are constructed with `nproc*lvec` (instead of `lvec`) unconnected elements, where `nproc` is the number of processors. Then, thanks to the `pardo` directive, loop `1001` will distribute sub-blocks of `lvec` elements on each processor for the usual residual evaluation. The only difference is the following: the actual vector length `nvecp` is passed to the subroutines `localp`, `getRep`, and `globadp` as an argument instead of through a common block as before; this makes sure that each processor has its own copy of `nvecp`. A few other declarations had to be moved out of common blocks to make them local to each processor.

Practically, `VIRGINIE` reaches 800 MFLOPS on one processor, which is quite satisfactory for an unstructured code, and over 10 GFLOPS on 16 processors. In fact the code could be much more finely tuned. For instance smaller loops (e.g., array clearing, ...) could be made parallel. Loop `1001` could also better balance the load over the processors in the case of uncompletely filled colors: in the present implementation all processors but one work with `lvec` elements.

Nevertheless, we satisfied our requirement to modify the source code as lightly as possible. Although much different in principle than the SP2 version, the port onto the NEC

SX-4 turned out much simpler than anticipated and close to the familiar concept of vectorization.

5. NUMERICAL EXAMPLES

Parallel computations are performed on a daily basis in the Aerodynamic Modelization Department at Dassault Aviation. A few civil and military applications are presented in the invited paper authored by Ph. Thomas [9]. We will focus on two spatial examples, for which the calculations were carried out on parallel computers. The first one is a rebuilding of the American Orbiter using an idealized shape called "Halis." The second one concerns the transonic assessment of a Crew Rescue/Crew Transfer Vehicle. These two computations were performed respectively on the IBM SP2 and the NEC SX-4 using the techniques described in the previous sections.

5.1. Halis

The Halis geometry consists of a 1:90 scale-down model of the US Orbiter windward side with an idealized leeward side. It was tested in the high-enthalpy wind tunnel F4 with a 15° body flap deflection. The free-stream conditions correspond to a 4930 m/s flow in thermochemical equilibrium at a static temperature of 795 K. Thermochemical nonequilibrium effects are present and a separated region develops over the body flap.

The idea behind the strategy which was applied to this computation, was to restrict the use of a Navier-Stokes code to the sole region where strong viscous interactions appear; i.e., the aft part of the windward side in the vicinity of the deflected bodyflap. Thus, a nonequilibium Euler computation was performed on the half geometry, followed by a defect boundary layer computation. In the case of a thick viscous layer which interacts with the shock layer, the defect boundary layer approach yields smooth profiles that perfectly match the inviscid solution away from the boundary layer. The Navier-Stokes computation is restricted to a "box" which comprises the bodyflap area. This box, which is depicted in Figure 3, is fed at the inflow with the Euler + boundary layer profiles.

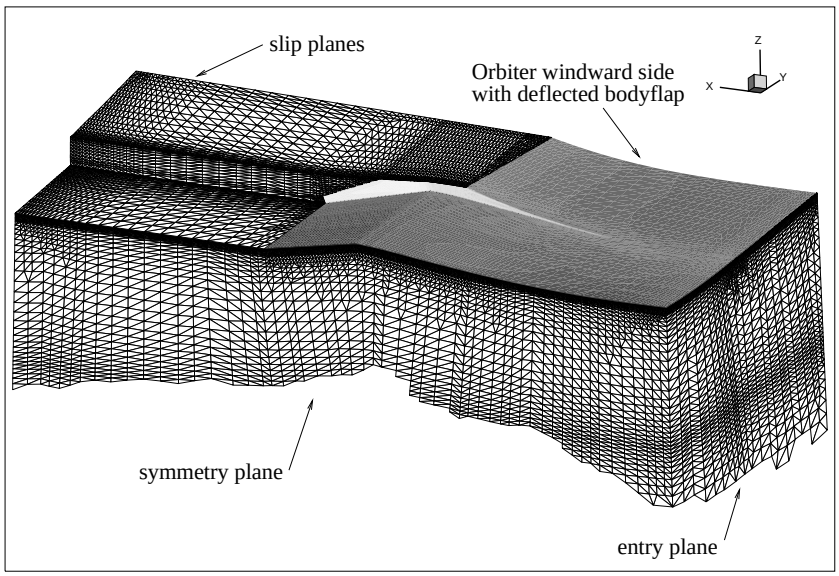

Figure 3. HALIS: partial view of the mesh of the Navier-Stokes box.

The base flow was not computed and splip planes were introduced instead in the wake of the main body.

The nonequilibrium extension of `VIRGINIE`, called `AETHER`, was used for this calculation. It is based on the very same numerical ingredients (finite elements, entropy variables, Galerkin/least-squares, GMRES, ...) and is more extensively described in [1].

Through pressure-coefficient contours, a view of the solution is shown in Figure 4.

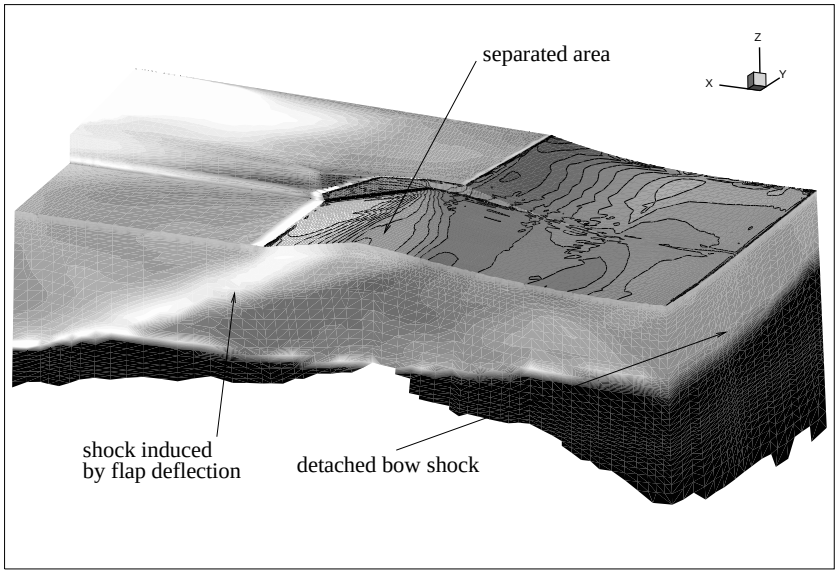

Figure 4. HALIS: pressure coefficient contours.

The trace of the detached bow shock can be seen in the entry plane; it enteracts henceforth with the shock induced by the flap deflection. One can also notice the pressure plateau in the separated area along the hinge of the bodyflap. By comparison with the experiment, the extent of the separation could be accurately reproduced, as well as pressure and heat flux levels. This computation could be performed using 16 SP2 processors in about 40 hours, on a mesh which contained slightly over 200,000 grid points.

5.2. CRV/CTV

Intensive parallel computing was used during the design process of the Crew Rescue/Crew Transfer Vehicle. Starting with the X-24, the final shape has been selected by NASA for its Crew Rescue Vehicle, and may as well serve as the basis for the future European Crew Transfer Vehicle. The transonic optimization of such a spacecraft required numerous detailed computations of the complex flow between the main body and the winglets. Thanks to the NEC SX-4 installed at NLR, key ingredients to the design, such as multi-point lift-versus-drag and pitching-moment-versus-angle-of-attack curves, could be computed overnight. Eight processors were used routinely on meshes made up of about 220,000 nodes for symmetric configurations. The reader is referred to [2] for further details about the design of the CRV/CTV.

For the purpose of illustration, we have selected an unsymmetrical configuration past one of the many spacecraft shapes which were considered in the design iteration process: the free-stream Mach number is 0.95, the angle of attack 20°, and the side-slip angle 5°. A

view of the surface mesh is presented in Figure 5. The complete three-dimensional mesh contains about 500,000 nodes.

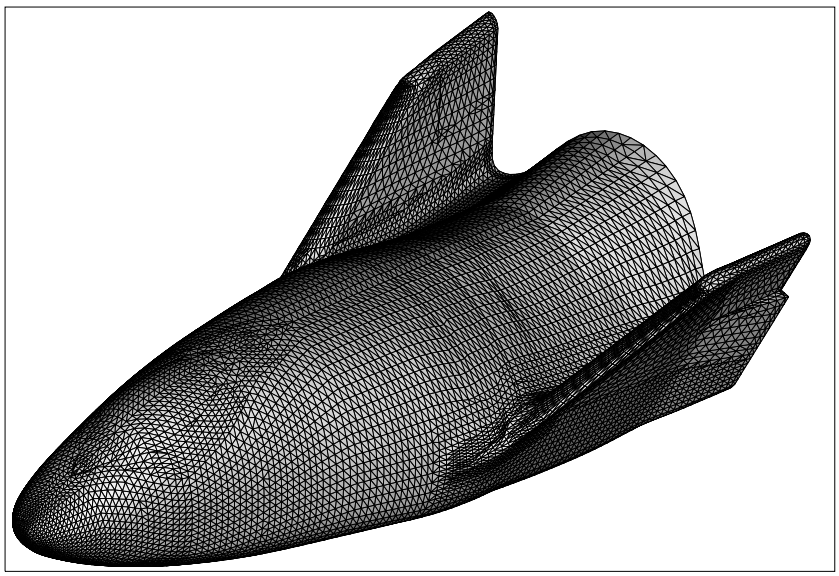

Figure 5. CRV/CTV: surface mesh.

Figure 6 shows the pressure-coefficient contours on the surface of the CRV/CTV; it gives an idea of the complex flow pattern which surrounds the vehicle.

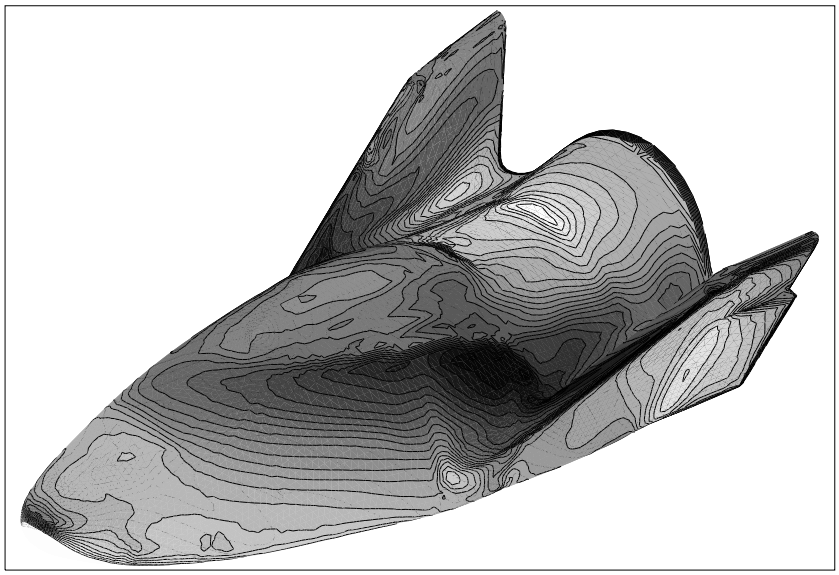

Figure 6. CRV/CTV: pressure coefficient contours.

Finally, we present in Figure 7 a summary of the speed-up which could be achieved on the NEC SX-4. This data is extracted from a typical run of `VIRGINIE` among the many computations which took place during the CRV/CTV design project. The slight drop in efficiency for 10 processors and above, can be explained in two ways: first, the

SX-4 at NLR is shared between users; this could account for the lower efficiency when one calculation requires more than half the total number of processors of the machine. The second reason might be that the considered problem is too small to run efficiently on 10 processors. Nevertheless, the shape of the curve fits reasonably well the theoretical curve obtained assuming a parallel work fraction of 0.945.

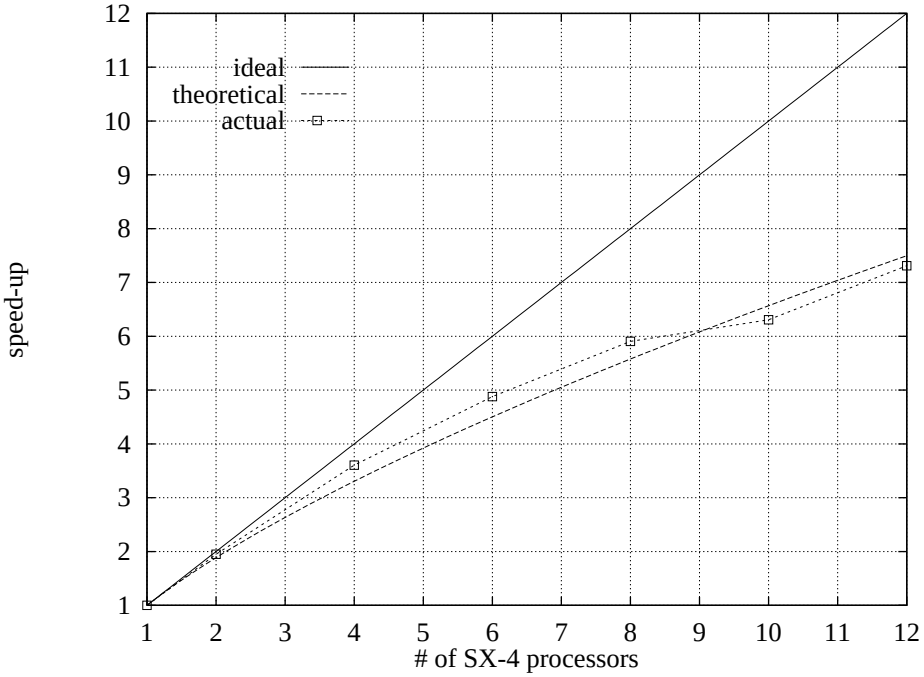

Figure 7. Overall run-time speed-up on the NEC SX-4.

6. CONCLUDING REMARKS AND PERSPECTIVE

We have presented two parallel ports of the same vectorized finite-element Navier-Stokes code onto two different parallel architectures, the IBM SP2 and the NEC SX-4.

The SP2 is conceptually simpler since it implements a paradigm close to the idea of a finite element method. The unique drawback is that more effort must be deployed in order to split every new mesh into blocks. The ease of use of the technique could be improved in several ways: parallel mesh generation methods would avoid the splitting process all together; alternatively, the splitting algorithm could be introduced directly in the Navier-Stokes code, hence rendering parallelism truly transparent to the user.

The SX-4 may feel a little weird at first, but in fact turns out not so complex to work with, since within its microtasking environment parallelization is very similar to vectorization.

We have shown that an appropriately designed finite-element Navier-Stokes code can efficiently be ported on shared- or distributed-memory computers. The distinction between these two memory representations may soon become irrelevant due to the progress in operating systems and network speeds. One may dream of a system which would enable a programmer to use in a transparent fashion any combination of computing power and memory resources of machines distributed over a network and see it as a single computer. The idea of the "mainframe" might be definitely obsolete with the occurrence of

memory-less processing units and processor-less memory units which could be mixed to assemble a "*couture*" computer tailored to one's needs. We may not be that far from the time when, say, a European engineer would be able to run a computation on processors that may reside somewhere in America using a hudge amount of memory spread across different countries in Asia, or vice versa?...

7. ACKNOWLEDGEMENTS

The authors would like to thank the Nationaal Lucht- en Ruimtevaartlaboratorium (NLR) in Amsterdam, The Netherlands, for providing access to their NEC SX-4. They are particularly grateful towards B. Oskam, K. de Cock, and J.J.W. van der Vegt from NLR, and G.A. van der Velde from NEC Netherlands for fruitful discussions.

The Halis and CRV/CTV applications were sponsored by ESA respectively under the MSTP and CRV/CTV programs.

REFERENCES

1. F. Chalot, M. Mallet, and M. Ravachol, "A comprehensive finite element Navier-Stokes solver for low- and high-speed aircraft design," paper #94-0814, *AIAA 32nd Aerospace Sciences Meeting*, Reno, NV, January 10–13, 1994.
2. F. Chalot, J.M. Hasholder, M. Mallet, A. Naïm, P. Perrier, M. Ravachol, Ph. Rostand, B. Stoufflet, B. Oskam, R. Hagmeijer, and K. de Cock, "Ground to flight transposition of the transonic characteristics of a proposed Crew Rescue/Crew Transfer Vehicle," paper #97-2305, *15th AIAA Applied Aerodynamics Conference*, Atlanta, GA, June 23–25, 1997.
3. Q.V. Dinh and T. Fanion, "Applications of dual Schur complement preconditioning to problems in Computational Fluid Dynamics and computational electromagnetics," *DD9 Conference*, Bergen, Norway, June 3–8, 1996.
4. R. Hagmeijer, B. Oskam, K.M.J de Cock, P. Perrier, Ph. Rostand, J.M. Hasholder, and J.W. Wegereef, "Validation of the design of the computational methods used for the design of the canopy of the Hermes spaceplane," paper #94-1865, *AIAA 12th Applied Aerodynamics Conference*, Colorado Springs, CO, June 20–22, 1994.
5. Z. Johan, T.J.R. Hughes, and F. Shakib, "A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids," *Computer Methods in Applied Mechanics and Engineering*, Vol. 87, pp 281–304, 1991.
6. Z. Johan, *Data Parallel Finite Element Techniques for Large-scale Computational Fluid Dynamics*, Ph.D. Thesis, Stanford University, 1992.
7. F. Shakib, T.J.R. Hughes, and Z. Johan, "A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis," *Computer Methods in Applied Mechanics and Engineering*, Vol. 75, pp 415–456, 1989.
8. F. Shakib, T.J.R. Hughes, and Z. Johan, "A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equa-

tions," *Computer Methods in Applied Mechanics and Engineering*, Vol. 89, pp 141–219, 1991.

9. Ph. Thomas, "Usage of parallel computing in aeronautics," *Parallel CFD '97*, Manchester, UK, May 19–21, 1997.
10. "Grand Challenges: High performance computing and communications. The FY 1992 US research and development program," Report by the Committee on Physical, Mathematical, and Engineering Sciences; Federal Coordinating Council for Science, Engineering and Technology; Office of Science and Technology Policy.
11. *IBM AIX Parallel Environment: Parallel Programming Subroutine Reference*, Release 2.1, Second Edition, December 1994.