Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXIX, págs. 3093-3105 (artículo completo)
Eduardo Dvorkin, Marcela Goldschmit, Mario Storti (Eds.)
Buenos Aires, Argentina, 15-18 Noviembre 2010

# PERFORMANCE OF A NUMERICAL THREE-DIMENSIONAL TRANSONIC AND SUPERSONIC FLOW ALGORITHM USING FINITE ELEMENTS IN A CLUSTER OF PERSONAL COMPUTERS

**João R. Masuero[a], Gustavo Bono[b] and Armando M. Awruch[c]**

[a]*Centro de Mecânica Aplicada e Computacional (CEMACOM), Departamento de Engenharia Civil, Universidade Federal do Rio Grande do Sul, Osvaldo Aranha 99, Porto Alegre, Brasil, joao.masuero@ufrgs.br, http://www.ufrgs.br/cemacom*

[b]*Departamento de Engenharia Mecânica, Universidade Federal de Pernambuco, Av. Prof. Moraes Rego, 1235 – Cidade Universitária, Pernambuco, Brasil, bonogustavo@gmail.com, http://www.ufpe.br/ctg*

[c]*Centro de Mecânica Aplicada e Computacional (CEMACOM), Departamento de Engenharia Civil, Universidade Federal do Rio Grande do Sul, Osvaldo Aranha 99, Porto Alegre, Brasil, amawruch@ufrgs.br, http://www.ufrgs.br/cemacom*

**Keywords:** Parallel Computing, Computational Fluid Dynamics, Finite Element Method, High Performance Computing, Compressible Flow.

**Abstract**. The performance of an algorithm to simulate three-dimensional (3-D) high compressible transonic and supersonic flows using the Finite Element Method, which is implemented for distributed memory and hybrid shared-distributed memory parallel configurations in a cluster of personal computers, is presented in this work. An explicit one-step Taylor-Galerkin scheme is used for time integration and both tetrahedral and hexahedral meshes are employed in the spatial discretization. Task division is achieved through a technique based on nodal ordering to obtain two distinct configurations: the first one minimizes the number of neighbor sub-domains in the partitioned mesh, minimizing the number of communication operations among the cluster nodes, and the second one minimizes the number of common elements among sub-domains, minimizing the amount of data exchanged by the cluster nodes through the network. The influence of the meshes size and type (structured or unstructured), the task division employed as well as the number of cores or processors of each cluster node is analyzed through two examples in terms of speed-up and parallel efficiency. The results obtained from these examples show the importance of using a task division suited to the hardware configuration of the cluster in the efficiency of parallel solutions, and the viability of using personal computers in clusters as an alternative to reach relatively high performance computing with cheap resources.

## 1 INTRODUCTION

Over the past fifty years, an intense research activity in the numerical simulation of compressible flows was developed, especially by the aerospace industry, with its requirements for highly accurate solutions at minimal computational cost. The demand to solve finely detailed models with realistic configurations for transonic and supersonic flows has challenged many researchers to come up with new and efficient algorithms. Specifically, computational power and memory have always been the main constrains for size and degree of detail of the numerical models employed. Vector supercomputers have traditionally been used to match these requirements, but their high acquisition and maintenance costs limited their use to few research centers. Recently, parallel computers based on high performance derivatives of personal computer processors have been the most common configuration used to achieve the performance needed for 3-D flow simulation with lower costs. Clusters of personal computers connected by fast usual networks have been used as a cheap, versatile and scalable alternative to obtain the computational power and main memory needed to simulate realistic engineering scale problems.

In Finite Elements Flow Simulation, time integration may be performed in one of the two classical approaches, explicit or implicit techniques. Implicit methods are computationally more expensive in terms of computer memory, but they have less stringent stability bounds. Explicit methods are relatively simple to implement, and they are easily cast in a form suitable for efficient parallel codes, but they are limited by very small time steps due to the Courant-Friedricks-Lewy (CFL) stability condition, which depends on the elements dimension. In many cases, it is necessary to use very small elements to accurately capture some phenomena and, consequently, small time steps must be also used. Adaptive meshes avoid the use of fine meshes over the entire spatial domain, saving computational effort. Adaptive time integration techniques are frequently used to minimize the impact over the performance of the CFL stability condition, especially in unsteady transient flows, but for detailed and complex models the demand for computational power, higher than a single personal computer may offer, still stands. Parallel computing is employed in this work providing the necessary computational power that allows the use of explicit approach in detailed and realistic meshes, preserving its advantages of small computer memory requirements and easy implementation.

## 2 THE GOVERNING EQUATIONS

Let $\Omega \subset R^3$ and (0,T) be the spatial and temporal domains, respectively, and let $\Gamma$ to be the boundary of $\Omega$. The spatial and temporal coordinates are denoted by **x** and t, respectively. We consider the Navier-Stokes equations with no source terms, governing unsteady compressible flows, written here in their dimensionless form as follows:

$$\frac{\partial \mathbf{U}}{\partial t}+\frac{\partial \mathbf{F}_i}{\partial x_i}+\frac{\partial \mathbf{G}_i}{\partial x_i}=0\,,\qquad \mathbf{U}=\begin{Bmatrix}\rho\\ \rho v_i\\ \rho e\end{Bmatrix},\qquad \mathbf{F}_i=\begin{Bmatrix}\rho v_j\\ \rho v_i v_j+p\,\delta_{ij}\\ v_j\left(\rho e+p\right)\end{Bmatrix},\qquad \mathbf{G}_i=\begin{Bmatrix}0\\ -\tau_{ij}\\ -\tau_{ji}v_i-q_j\end{Bmatrix} \tag{1}$$

with *i, j = 1,2,3,* where **U** is the unknown vector of the conservation variables, $\mathbf{F}_i$ and $\mathbf{G}_i$ are, respectively, the convective and diffusive flux vectors. Here $v_i$ is the velocity component in the direction of the coordinate $x_i$, $\rho$ is the specific mass, $p$ is the thermodynamic pressure, $\tau_{ij}$ are the components of the viscous stress tensor, $q_j$ is the heat flux vector, $e$ is the total specific energy and $\delta_{ij}$ is the Kronecker delta function. Dimensionless scales are used.

Assuming that air behaves as a calorically perfect gas, the pressure (which is calculated by the equation of state) and internal energy $u$ are given by the following equations in terms of dimensionless variables:

$$p = (\gamma - 1)\rho\, u\,, \qquad u = c_v\, T = e - \frac{1}{2} v_i\, v_i \tag{2}$$

where $T$ is the temperature and the specific heat ratio $\gamma = c_p / c_v$, relating specific heat coefficients at constant pressure and constant volume, is assumed to be constant and equal to 1.4. The dynamic viscosity and the coefficient of thermal conductivity in the heat flux depend on the temperature and therefore are modeled using Sutherland's law. The Euler equations are obtained eliminating the diffusive flux vector in Eq. (1). Initial and boundary conditions must be added to Eq. (1) in order to define uniquely the problem.

## 3 TAYLOR-GALERKIN FORMULATION

The one-step Taylor-Galerkin scheme employed in this work is similar to that presented by Donea (1984). Expanding the conservation variables $\mathbf{U}$ at $t = t^{n+1}$ in Taylor series including the first and second derivatives, the following final expression is obtained:

$$\Delta \mathbf{U}_{J+1}^{n+1} = \Delta t \left[ -\frac{\partial \mathbf{F}_i{}^n}{\partial x_i} - \frac{\partial \mathbf{G}_i{}^n}{\partial x_i} + \frac{\Delta t}{2} \frac{\partial}{\partial x_k} \left( \mathbf{A}_k{}^n \frac{\partial \mathbf{F}_i{}^n}{\partial x_i} \right) \right] + \frac{\Delta t}{2} \left[ -\frac{\partial\, \Delta \mathbf{F}_{iJ}{}^{n+1}}{\partial x_i} - \frac{\partial\, \Delta \mathbf{G}_{iJ}{}^{n+1}}{\partial x_i} + \frac{\Delta t}{2} \frac{\partial}{\partial x_k} \left( \mathbf{A}_k{}^n \frac{\partial\, \Delta \mathbf{F}_{iJ}{}^{n+1}}{\partial x_i} \right) \right] \tag{3}$$

with $\Delta \mathbf{U}^{n+1} = \mathbf{U}^{n+1} - \mathbf{U}^n$, $\Delta t^{n+1} = t^{n+1} - t^n$ is the time step, where $n$ and $n+1$ indicates $t$ and $t+\Delta t$, respectively, $J$ is an iteration counter, $\Delta \mathbf{F}_i^{n+1} = \mathbf{F}_i^{n+1} - \mathbf{F}_i^n$, $\Delta \mathbf{G}_i^{n+1} = \mathbf{G}_i^{n+1} - \mathbf{G}_i^n$ and $\mathbf{A}_i$ is the convection Jacobian defined as $\mathbf{A}_i = \partial \mathbf{F}_i / \partial \mathbf{U}$. This iterative process converges fast (no more than two or three iterations per time step are necessary).

Applying the classical Bubnov-Galerkin weighted residual method in the context of the FEM to Eq (3), the spatial discretization is obtained. In this work linear tetrahedral and hexahedral elements are used. Tetrahedral meshes are well suited for problems with complex geometry, hexahedral meshes need fewer elements to reach the same accuracy, and both elements can be exactly integrated without numerical quadrature. The consistent mass matrix is substituted by the lumped mass matrix, and then Eq. (3) is solved with an explicit scheme, which is conditionally stable, and its local stability condition is given by

$$\Delta t_E = SC \frac{L_E}{a + (v_i\, v_i)^{1/2}} \tag{4}$$

where $E$ denotes a specific element, $L_E$ is its characteristic dimension, $a$ is the sound speed and $SC$ is a safety coefficient (which is equal or less than 1.0). In this work, $SC$ = 0.1 is adopted for all numerical examples.

At transonic and supersonic speeds, an additional numerical damping is necessary to capture shocks and to smooth local oscillations in their vicinities. An artificial viscosity model, as proposed by Argyris et al. (1990), is adopted in this work due to its simplicity and efficiency in terms of CPU time. The term representing the artificial viscosity is added explicitly to the non-smoothed solution as follows:

$$\mathbf{U}_s^{n+1} = \mathbf{U}^{n+1} + \mathbf{M}_L{}^{-1} \mathbf{d}\,, \qquad \mathbf{d} = \sum_{\text{ele}} \text{CFL CAF } S_{ele} \left[ \mathbf{M} - \mathbf{M}_L \right]_{ele} \mathbf{U}_{ele}^n \tag{5}$$

where $\mathbf{M}$ and $\mathbf{M}_L$ are the consistent mass matrix at element level and the assembled lumped mass matrix, respectively. $\mathbf{U}_s^{n+1}$ and $\mathbf{U}^{n+1}$ are the smoothed and non-smoothed

solutions at $t+\Delta t$, respectively. The index *ele* in vector **d** is referred to a specific element, CFL = $\Delta t/\Delta t_E$ is the local Courant-Friedricks-Lewy number, CAF is an artificial damping coefficient given by the user (in this work, CAF = 1.0 was adopted, and it must be specified with care in order to avoid undesired interference of the artificial viscosity on the physical viscosity), $S_{ele}$ is a pressure sensor at element level obtained as an average of nodal values $S_i$. Values of $S_i$ are components of the following assembled global vector in Eq.(6) where **p** is the pressure vector of a specific element, and the symbol | | indicates absolute values:

$$S_{\mathrm{i}} = \sum_{\mathrm{ele}} \frac{\left|\left(\mathbf{M}-\mathbf{M}_L\right)_{ele}\mathbf{p}_{ele}\right|_i}{\left(\left|\mathbf{M}-\mathbf{M}_L\right|_{ele}\mathbf{p}_{ele}\right)_i} \tag{6}$$

## 4 PARALLEL IMPLEMENATION

Equation (3) corresponds to a set of uncoupled equations regarding the variables $\rho$, $v_i$ and $e$, within each iteration of each time step. Thus, if the values of these variables are known in time $t$, the new value of any nodal variable, in the $J^{th}$ iteration of time $t+\Delta t$, can be evaluated in an independent way. For each nodal equation, the needed information from the previous iteration or the previous time step is only the values of the variables corresponding to nodes belonging to the elements connected to the node where the nodal equation is being evaluated.

In this work, a logical processor corresponds to a CPU core in clusters formed by both single-core single-CPU computers and multi-core single-CPU computers.

In the parallel algorithm, groups of nodal equations can be independently processed by different logical processors within each iteration of each time step. At the end of each iteration, values of the variables corresponding to nodes belonging to elements connected to nodes processed by other logical processors are informed to these logical processors and vice-versa, and the algorithm may be continued. This concept can be used for distributed memory configuration, shared memory configuration or for a combination of both alternatives (hybrid configuration). The use of an artificial viscosity model, particularly the computation of sensors $S_{ele}$ and $S_i$ in equations (5) and (6), demands one extra communication (exchange of variable values) per time step among logical processors. Thus, the one-step Taylor-Galerkin scheme employed in this work, in its parallel implementation, requires, for each time step, one communication among logical processors for the evaluation of the artificial viscosity model and one communication for each iteration into the time step to communicate the problem variables $\rho$, $v_i$ and $e$.

Since the governing equations are uncoupled within each iteration or time step, a hostless program model can be used to minimize the amount of data transferred through the network among logical processors, avoiding the undesirable high network data traffic characteristic of master-slave model that negatively impacts the performance. This way, all logical processors execute the same code, and the values of the problem variables are transferred to a specific computer only at determined intervals in order to be stored.

The MPI (*Message Passing Interface)* library was chosen to provide communication between computers in this work because it is suitable to the parallel approach proposed here, it is efficient in distributed memory, shared memory and hybrid configurations, it has a standard which warrants portability of the same code across different computational platforms, and can be coupled to both, Fortran and C++, the most common programming languages in engineering analysis and simulations. The OpenMP library is equally portable and can be coupled to Fortran and C++ too, but its limited to shared memory configurations. In this work it was used together with MPI in clusters formed by multi-core single-CPU

computers.

To estimate the speed-up provided by the parallel implementation, the relative speed ($rs_i$) of each logical processor $i$ and the obtained relative speed ($ors_P$) of a set of $P$ logical processors can be evaluated, respectively, by:

$$rs_i = \frac{t_{ref}}{t_i}, \qquad ors_P = \frac{t_{ref}}{t_P} \tag{7}$$

where $t_{ref}$ is a reference solution time and $t_i$ is the solution time of logical processor $i$ and $t_P$ is the solution time obtained by the set working in parallel. Considering that the entire solution algorithm can be parallelized, the theoretical relative speed ($trs_P$), the parallelization efficiency $Eff_P$ and the *speed-up*$_P$ of the set of $P$ processors can be evaluated, respectively, by:

$$trs_P = \sum_P rs_i, \qquad Eff_P = \frac{ors_P}{trs_P}, \qquad speed-up_P = P.Eff_P \tag{8}$$

## 5 TASK DIVISION AND LOAD BALANCE

The efficiency of computational parallel algorithms using the Finite Element Method in distributed memory configurations is highly dependent of the way the task division is performed. Mesh splitting in sub-domains and Graph partitioning are commonly used to accomplish task division, resulting in a partition of the problem equations among the different processors employed in the parallel solution. Schloegel et al (2000) provide a comprehensive overview of the techniques employed in graph partition for high performance scientific simulations.

Since the governing equations of the problem are uncoupled nodal equations, a task division among the logical processors based on mesh nodes is used in this work: the computational effort of each logical processor used in the solution of the problem is considered as proportional to the number of mesh nodes allocated to that processor. The mesh nodes (and the corresponding nodal variables and equations) are divided in as many groups as the number of logical processors, proportionally to the individual relative performance of each processor. In the absence of a better parameter, the core frequency of each processor is used to estimate the individual performance, allowing efficient parallelization even in clusters formed by heterogeneous nodes.

When performing element loops, each logical processor acts over all the elements connected to its own group of nodes. Some elements on the mesh, called here as *common elements,* are connected to nodes that belong to different groups, and must be processed by more than one logical processor. Thus, overlapping in computational efforts by different logical processors related to these elements is inherent of the adopted task division.

The values of the variables related to nodes of *common elements* and that belong to a node group (logical processor) are the information which is necessary to others node groups (processors) that also include nodes of these *common elements*. This way, the number of *common elements* in a mesh defines the overlapped computational effort due to parallel implementation, and the number of nodes connected to these elements defines the amount of data that must be communicated among the logical processor in each time step. Considering that network communications are a severe bottleneck to the performance in distributed memory configurations due to the relative low speed of network when compared to memory or local storage accesses, and that computational overlapping impacts performance in both, distributed memory and hybrid configurations, a task division that leads to a minimal number of *common elements* is highly desirable.

When the latency is the main factor in the communication performance through the network, minimizing the number of communication operations among the logical processors (or the number of sub-domains that are neighbors each other) is more important than minimizing the amount of data being communicated through the network (Hendrickson and Leland, 1995). This may be obtained by dividing the mesh in several "slices" across its smaller dimension, in order that each sub-domain has no more than 2 neighbor sub-domains (or each logical processor communicates data to no more than other 2 processors). This task division can be obtained geometrically by the Inertial Stripwise Partitioning method (Dorneles et al., 2005) or, in a much more simple way, by organizing the nodes in order to minimize the matrix bandwidth of the corresponding system of equations.

Despite the fact that equations are uncoupled and the bandwidth of the system has no influence over the performance of the solver, a nodal reordering was used to minimize both, communications among logical processors and computational work overlapping. The algorithm used here consists in an initial element reordering using Silvester and Auda (1984) criteria to minimize *Front*, followed by a nodal reordering using the Profile Front Minimization of Hoit and Wilson (1983). The resultant ordered list of nodes is sequentially divided in as many groups as the number of logical processors present in the cluster, proportionally to the relative computational power of each processor. The load balance is made processing some time steps of the code, registering the CPU time spent individually by each logical processor and re-evaluating the number of mesh nodes allocated to each processor (inversely proportional to CPU-time). This process is repeated until all the individual CPU times are equal, or differing less than a given tolerance, and it is easily performed by re-dividing the ordered list of mesh nodes, without any further geometric consideration or nodal reordering.

This algorithm is called SNR (single nodal reordering) and an example of its application for tetrahedral and hexahedral meshes is shown in Fig. 1(a) and (b), respectively. The number of borders between sub-domains is minimal.

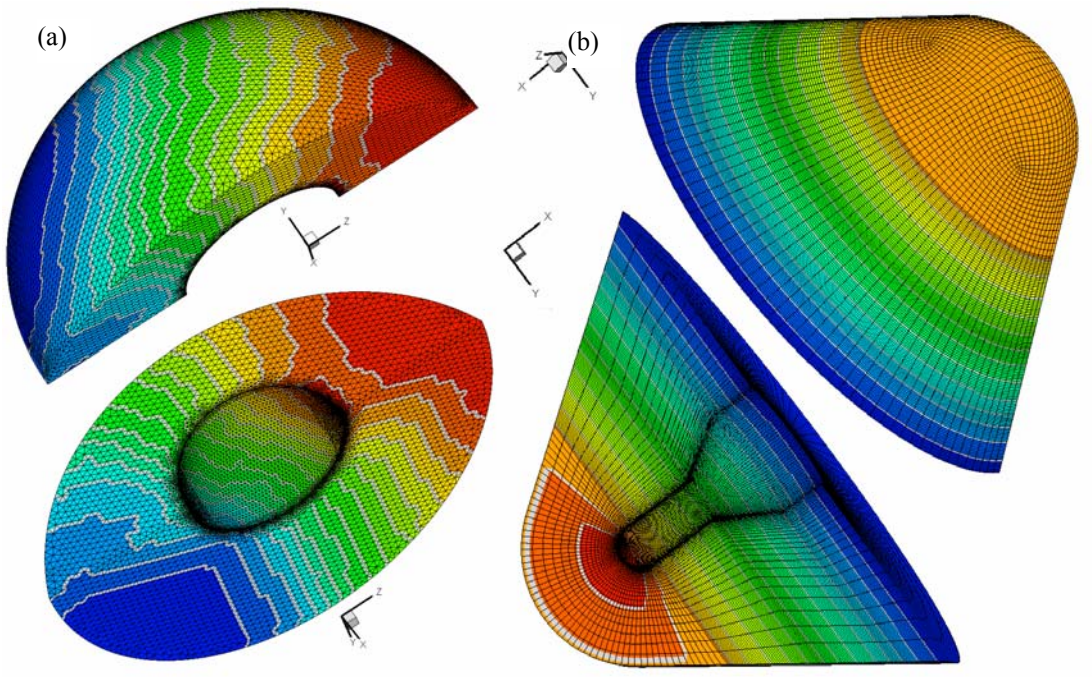


Figure 1: Tetrahedral (a) and hexahedral (b) elements meshes divided in groups or sub-domains by the algorithm SNR.

In Fig. 1 each color corresponds to a sub-domain or group of mesh nodes allocated to a logical processor, and the *common elements* are shown in grey. When the mesh is divided in many groups of nodes following a single initial nodal reordering, the optimal division regarding effort overlapping and data communication is not reached, since a change in the division direction is needed. A better division is obtained geometrically by the Inertial Recursive Coordinate Bisection method, where, after each division, the "narrowest" dimension of each resulting sub-domain is determined, indicating the direction of the next division.

In order to avoid intricate geometric considerations, this kind of division can be performed substituting the determining of "narrowest" dimension by a nodal reordering for matrix bandwidth minimization of the resultant sub-domains after each division. This method is called RNR (recursive nodal reordering) and its application is shown in Fig. 2(a) and (b) for tetrahedral and hexahedral meshes, respectively. The number of neighbor sub-domains is greater than with SNR method, but the amount of data communicated among logical processors through the network (equivalent to the borders length) is considerably smaller. Load balancing can be reached as in SNR, but a new recursive nodal reordering is necessary any time the number of nodes assigned to each processor is changed.

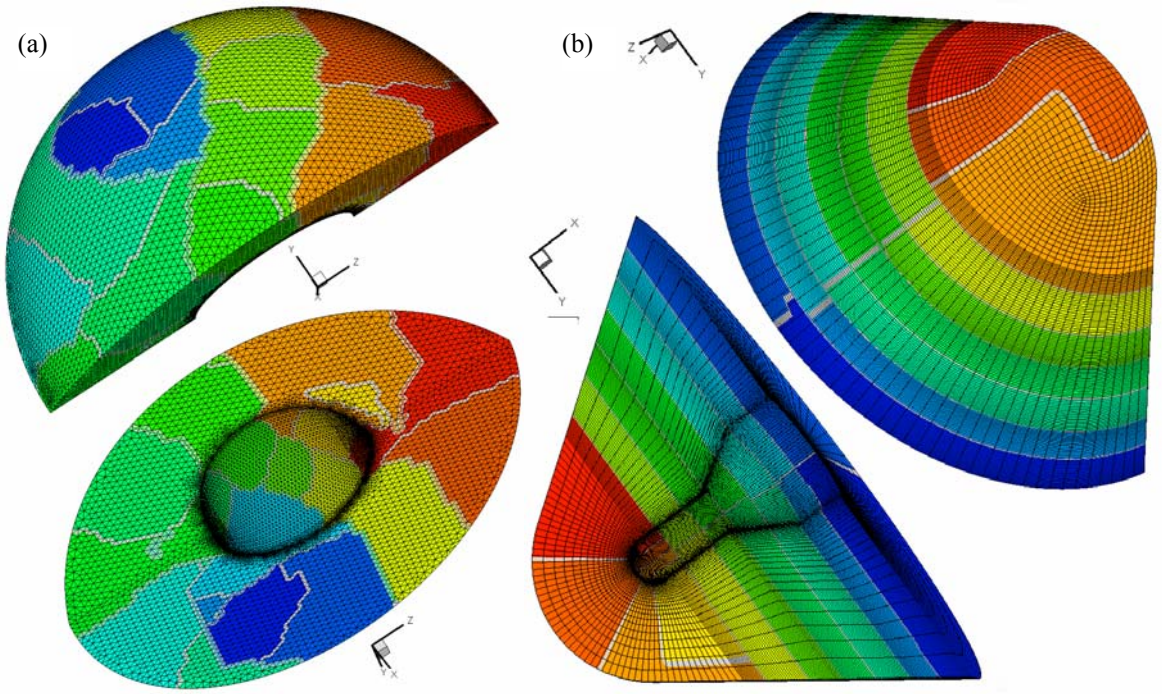


Figure 2: Tetrahedral (a) and hexahedral (b) elements meshes divided in groups or sub-domains by the algorithm RNR.

A complex mesh of a generic Canard-Wing-Body configuration, shown in Fig. 3, is used to numerically evaluate the impact of SNR and RNR partitioning processes over the amount of data communicated through network and the computational overhead. The mesh has 1.5 millions of tetrahedral elements and 272 thousands of nodes, and is partitioned among 16 identical processors. Three mesh partitions based in nodal order are used: the first one uses the original nodal order obtained from the mesh generation process to assign nodes to the processors, the second uses the SNR approach and the last one the RNR approach. Data dependency and computational overhead of these partitions can be seen in Fig. 4.

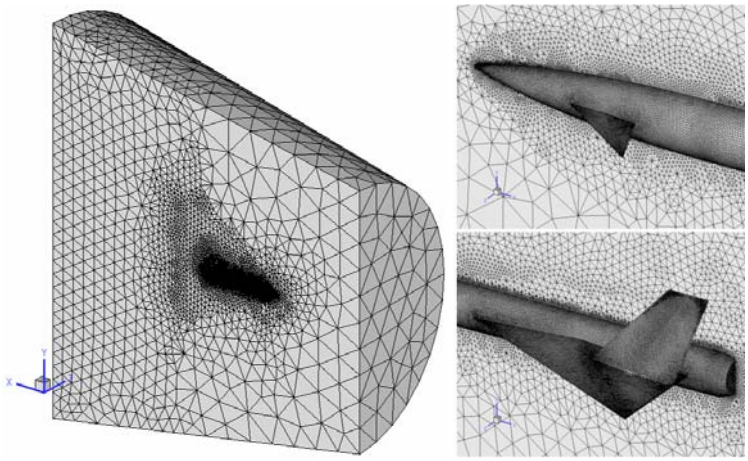

Figure 3: Generic Canard-Wing-Body model.

**Original nodal order**

| Processors | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 2,8 | 1,3 | 3,4 | 3,8 | 3,6 | 4,1 | 5,5 | 5,3 | 5,3 | 4,4 | 5,0 | 5,6 | 5,8 | 4,3 | 4,3 |
| 2 | 3,5 | | 3,2 | 4,2 | 5,0 | 5,8 | 6,0 | 5,4 | 5,0 | 5,4 | 6,7 | 7,3 | 6,8 | 6,5 | 5,2 | 6,7 |
| 3 | 1,7 | 3,0 | | 4,8 | 3,8 | 2,9 | 3,1 | 2,6 | 2,3 | 2,4 | 1,6 | 1,4 | 1,7 | 1,8 | 4,4 | 5,0 |
| 4 | 9,8 | 11,5 | 4,9 | | 7,3 | 6,1 | 5,5 | 5,0 | 3,8 | 3,4 | 4,4 | 3,2 | 2,3 | 1,6 | 1,3 | 1,5 |
| 5 | 10,1 | 11,9 | 3,8 | 7,5 | | 6,7 | 5,9 | 5,4 | 4,3 | 4,1 | 4,8 | 3,7 | 2,8 | 2,0 | 1,2 | 1,7 |
| 6 | 8,7 | 13,0 | 3,5 | 6,1 | 6,8 | | 6,2 | 5,9 | 4,8 | 5,0 | 5,0 | 3,9 | 3,1 | 2,7 | 1,9 | 2,1 |
| 7 | 9,3 | 12,4 | 3,5 | 5,5 | 6,1 | 6,3 | | 7,0 | 5,7 | 5,2 | 4,7 | 4,0 | 3,5 | 2,8 | 2,2 | 2,2 |
| 8 | 12,2 | 10,3 | 3,0 | 5,0 | 5,3 | 5,8 | 7,1 | | 6,5 | 5,4 | 5,2 | 4,7 | 4,0 | 3,5 | 2,6 | 2,4 |
| 9 | 11,4 | 9,3 | 3,0 | 3,6 | 3,9 | 4,6 | 5,6 | 6,4 | | 6,2 | 4,8 | 4,8 | 4,6 | 4,5 | 3,3 | 2,3 |
| 10 | 11,3 | 9,6 | 2,8 | 3,1 | 3,8 | 4,6 | 5,0 | 5,3 | 6,3 | | 5,2 | 5,3 | 5,2 | 5,3 | 4,0 | 2,9 |
| 11 | 9,7 | 12,9 | 1,7 | 4,3 | 4,6 | 5,0 | 4,8 | 5,4 | 5,2 | 5,4 | | 9,8 | 7,4 | 5,5 | 2,7 | 3,3 |
| 12 | 10,8 | 12,2 | 1,6 | 3,0 | 3,5 | 3,7 | 3,8 | 4,6 | 4,7 | 5,4 | 9,4 | | 9,8 | 7,9 | 3,9 | 4,0 |
| 13 | 11,4 | 11,3 | 1,9 | 2,2 | 2,7 | 2,9 | 3,3 | 3,8 | 4,4 | 5,0 | 7,4 | 9,7 | | 10,3 | 5,7 | 4,8 |
| 14 | 11,8 | 10,7 | 2,0 | 1,5 | 1,9 | 2,6 | 2,6 | 3,5 | 4,3 | 5,3 | 5,6 | 7,9 | 10,3 | | 9,0 | 6,1 |
| 15 | 7,2 | 7,5 | 7,3 | 1,2 | 1,1 | 1,8 | 2,0 | 2,4 | 3,0 | 3,7 | 2,6 | 3,7 | 5,2 | 7,5 | | 7,4 |
| 16 | 7,9 | 11,4 | 5,5 | 1,6 | 1,9 | 2,4 | 2,7 | 3,0 | 2,9 | 3,5 | 3,6 | 4,5 | 5,6 | 7,0 | 8,5 | |
| Nodes | 137 | 150 | 49 | 57 | 62 | 65 | 68 | 71 | 69 | 71 | 75 | 79 | 78 | 75 | 60 | 57 |
| Elements | 308 | 399 | 208 | 209 | 216 | 225 | 229 | 235 | 223 | 228 | 253 | 260 | 258 | 254 | 215 | 211 |

**SNR partition**

| Processors | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 2,8 | | | | | | | | | | | | | | |
| 2 | 3,5 | | 4,3 | | | | | | | | | | | | | |
| 3 | | 5,1 | | 5,5 | | | | | | | | | | | | |
| 4 | | | 6,6 | | 6,5 | | | | | | | | | | | |
| 5 | | | | 7,8 | | 7,8 | | | | | | | | | | |
| 6 | | | | | 9,6 | | 9,1 | | | | | | | | | |
| 7 | | | | | | 10,7 | | 9,6 | | | | | | | | |
| 8 | | | | | | | 10,9 | | 9,0 | | | | | | | |
| 9 | | | | | | | | 9,7 | | 8,9 | | | | | | |
| 10 | | | | | | | | | 9,9 | | 9,2 | | | | | |
| 11 | | | | | | | | | | 10,7 | | 9,4 | | | | |
| 12 | | | | | | | | | | | 10,6 | | 9,8 | | | |
| 13 | | | | | | | | | | | | 10,7 | | 9,4 | | |
| 14 | | | | | | | | | | | | | 10,3 | | 8,1 | |
| 15 | | | | | | | | | | | | | | 8,6 | | 6,1 |
| 16 | | | | | | | | | | | | | | | 5,9 | |
| Nodes | 4 | 8 | 11 | 13 | 16 | 18 | 20 | 19 | 19 | 20 | 20 | 20 | 20 | 18 | 14 | 6 |
| Elements | 110 | 121 | 129 | 137 | 145 | 152 | 156 | 154 | 145 | 147 | 147 | 147 | 144 | 135 | 123 | 94 |

**RNR partition**

| Processors | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 1,5 | 0,1 | 1,1 | | | | | | | | | | | | |
| 2 | 1,8 | | 1,6 | 0,7 | | | | | 1,9 | | | | | | | |
| 3 | 0,2 | 2,0 | | 1,4 | 0,7 | 1,8 | 0,2 | | 1,1 | | | | | | | |
| 4 | 1,2 | 0,7 | 1,6 | | 0,4 | 0,7 | 1,4 | 0,1 | 0,1 | | | | | | | |
| 5 | | | 0,8 | 0,6 | | 1,4 | 0,8 | 0,3 | 1,1 | | | | | | | |
| 6 | | | 2,0 | 0,9 | 1,8 | | 1,4 | 1,2 | 1,3 | | | | | | | |
| 7 | | | 0,2 | 1,4 | 0,8 | 1,7 | | 0,7 | 0,8 | | | | | | | |
| 8 | | | | 0,1 | 0,2 | 1,5 | 0,8 | | 2,8 | | | | | | | |
| 9 | | 1,8 | 1,0 | 0,1 | 1,2 | 1,6 | 1,0 | 3,5 | | 8,9 | | | | | | |
| 10 | | | | | | | | | 9,9 | | 4,9 | 4,4 | | | | |
| 11 | | | | | | | | | | 5,4 | | 0,4 | 4,5 | | | |
| 12 | | | | | | | | | | 5,3 | 0,5 | | 5,3 | | | |
| 13 | | | | | | | | | | | 4,8 | 6,3 | | 9,4 | | |
| 14 | | | | | | | | | | | | | 10,3 | | 8,1 | |
| 15 | | | | | | | | | | | | | | 8,6 | | 6,1 |
| 16 | | | | | | | | | | | | | | | 5,9 | |
| Nodes | 3 | 6 | 7 | 6 | 5 | 9 | 6 | 6 | 19 | 20 | 10 | 11 | 20 | 18 | 14 | 6 |
| Elements | 107 | 117 | 123 | 114 | 118 | 125 | 112 | 114 | 145 | 147 | 119 | 122 | 144 | 135 | 123 | 94 |

Figure 4: Data dependency and computational overhead for mesh partition following the original nodal order, SNR partition and RNR partition.

The columns represent each logical processor and the lines represent the processors with which it communicates, indicated by the number of nodes (or nodal variables sets) being communicated in thousand of units. In the bottom lines labeled *nodes* and *elements* are shown the total number of nodes communicated and the total number of elements processed by each logical processor, respectively, in thousands of units. The line *elements* include both exclusive elements and *common elements*. The mesh generation process results in a sparse dependency matrix, while SNR approach results in a narrow band shaped matrix. RNR leads to a sparse matrix too, but the amount of data being communicated among processors and the number of communication processes is far smaller than the original mesh and smaller than SNR.

Table 1 shows some efficiency statistics of each partition method. The *redundancy in processing elements* is obtained by the quotient between the total number of elements processed by all logical processors that exceeds the number of elements in the mesh and the number of elements in the mesh. The *efficiency in processing elements* is evaluated by the quotient between the number of elements in the mesh and the total number of elements processed by all logical processors. Finally, the *Data exchange index* is the quotient between the total amount of nodes communicated through the network by all processors and the number of nodes in the mesh. Both SNR and RNR partitioning processes result in much better statistics than the partition based on original nodal order, indicating great reduction in computational overhead and in data communication through the network. The RNR approach has better indexes than SNR, except by the *Number of communication processes*. Increasing the number of communications make the schedule of these communications more complex, and the performance more dependent on the network latency.

| Efficiency statistics | Original nodal ordering | SNR Partition | RNR Partition |
|---|---|---|---|
| Amount of nodes communicated by all processors (x1000) | 1221 | 246 | 166 |
| Amount of elements processed by all processors (x1000) | 3932 | 2186 | 1958 |
| Redundancy in processing elements | 162% | 46% | 30% |
| Efficiency in processing elements | 38% | 69% | 77% |
| Data exchange index | 449% | 91% | 61% |
| Number of communication processes | 240 | 32 | 70 |

Table 1: Efficiency parameters of partition methods.

## 6 RESULTS

The methodology described above was applied in two realistic models, both under steady-state supersonic flows: a generic Space Vehicle (SV) configuration shown in Fig.5 and a generic Canard-Wing-Body (CWB) configuration already shown in Fig. 3. Two meshes, relatively coarse and fine, were used for each configuration in order to evaluate de impact of mesh size in the parallelization efficiency. The characteristics of the meshes are shown in Tab. 2.

| **Mesh** | **Element** | **Number of Elements** | **Number of Nodes** | **Degrees of Freedom** |
|---|---|---|---|---|
| SV1 | Hexahedral | 198750 | 211146 | 1.055.730 |
| SV2 | Hexahedral | 917700 | 949212 | 4.746.060 |
| CWB1 | Tetrahedral | 242979 | 45823 | 229115 |
| CWB2 | Tetrahedral | 1501912 | 271842 | 1359210 |

Table 2: Meshes characteristics.

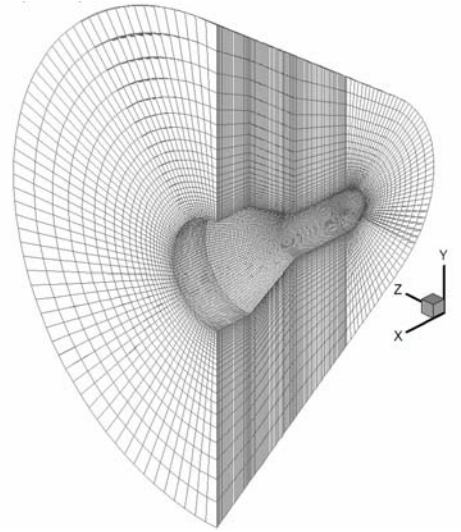


Figure 5: (a) Space Vehicle model.

The four meshes were tested in a heterogeneous cluster formed by 12 quad-core single-CPU Intel Core 2 nodes with core frequency of 3.7 GHz and 8 quad-core single-CPU Intel Core i7 nodes with core frequency of 3.5 GHz. The network used was Gigabit Ethernet and each node had a single network interface.

For hexahedral meshes, the parallel algorithm scales quite well, as shown in Fig.6 (a) and (b) for SV1 and SV2 meshes, respectively. There are no significant differences between the results for SNR and RNR methods. The best result was obtained using MPI alone (as if each core was an independent computer) instead of using hybrid parallelization (OpenMP inside the node and MPI between cluster nodes). As expected, the solution for the larger mesh SV2 scales better than for the smaller mesh SV1. Using 8 cluster nodes, 32 logical processors (cores), the speed-up curve for SV1 presents a slope reduction, pointing to a loss of parallel efficiency for increasing cluster sizes.

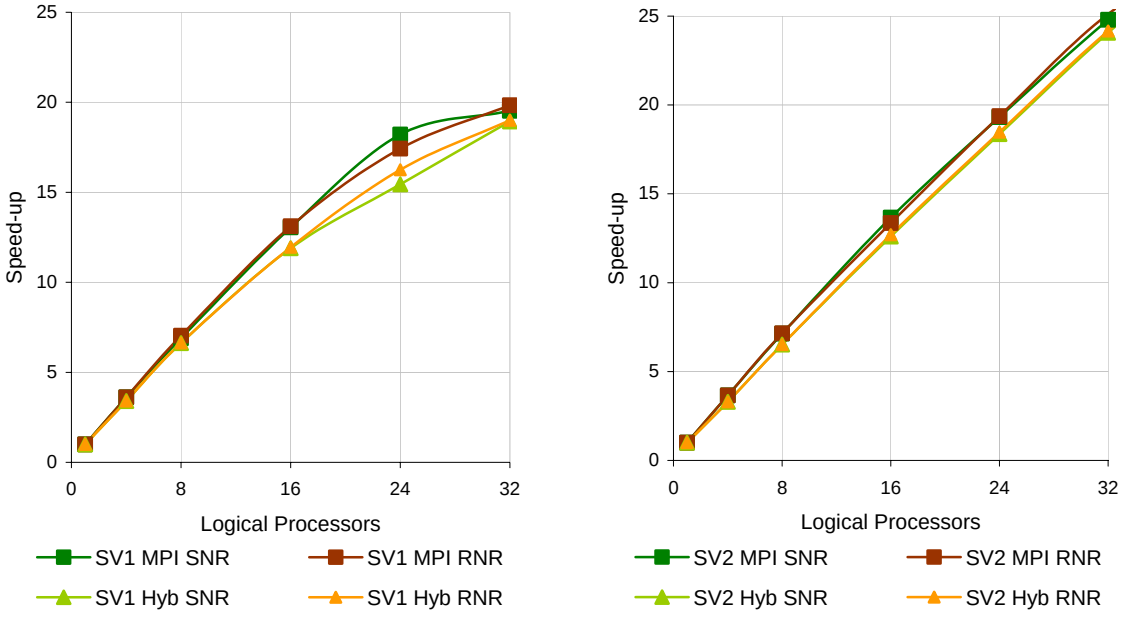


Figure 6: Obtained speed-up in (a) SV1 and (b) SV2 hexahedral meshes

The largest mesh scales well even when all available logical processors in the cluster were used, as shown in Fig. 7. The differences between partition methods and kind of parallelization became less visible here. For structured meshes, even a relatively simple partition algorithm as SNR can lead to good performance results, reaching a parallel efficiency of around 60% for a cluster size of 20 nodes / 80 cores.

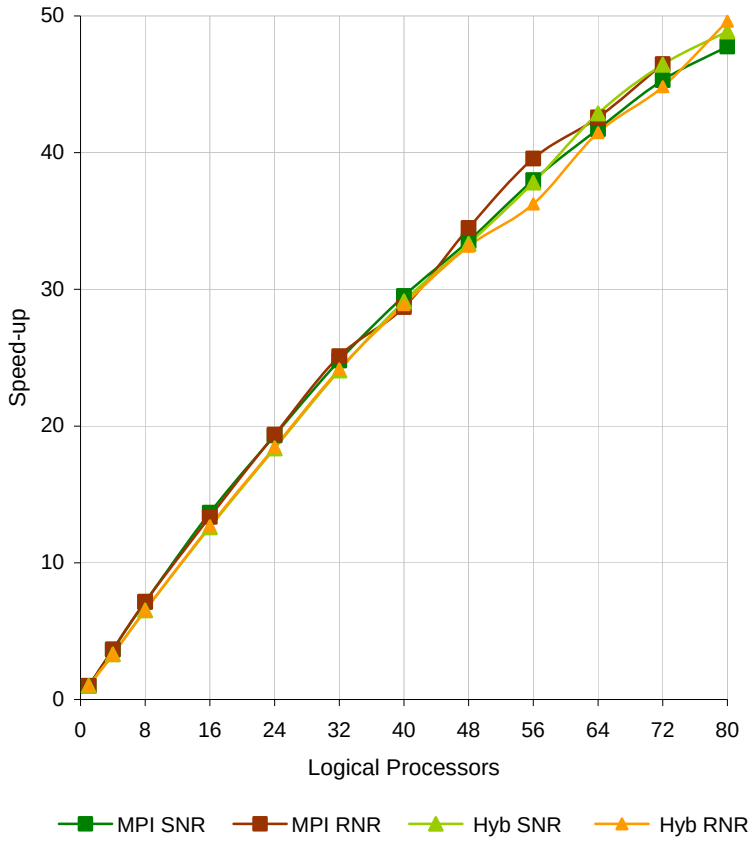


Figure 7: Obtained speed-up in SV2 hexahedral mesh

For tetrahedral meshes, the parallel algorithm scales considerably worse than for hexahedral ones, although significant speed-up values could be reached, as can be seen in Fig. 8 (a) and (b) for CWB1 and CWB2 meshes, respectively. The elevated number of elements connected to each mesh node in tetrahedral meshes results in higher computational effort overlapping and data communication, explaining the worse results, when compared to hexahedral meshes. In a quad-core configuration cluster, the communication among processors through the network results in a severe performance penalty, since a single network interface must attend 4 cores in each cluster node. In this configuration, the use of hybrid parallelization (OpenMP inside de node and MPI between cluster nodes) led to better results than using MPI alone. For the same reason, the SNR method (minimum number of neighbor sub-domains or communication processes) led to better results than RNR (minimum amount of communicated data), since RNR tends to generate cross communications among all logical processors (sparser matrix than the SNR one, shown in Fig. 4). This effect is more evident in the smallest CWB1 mesh when the number of logical processors increases, resulting in a severe loss of parallel efficiency, as shown in Fig.9.

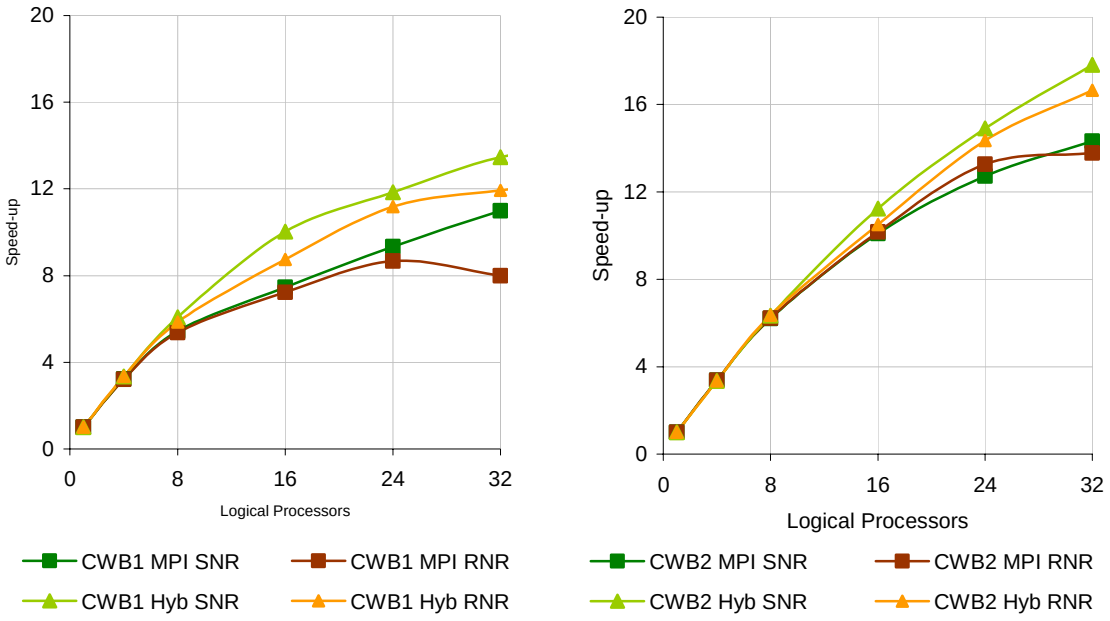


Figure 8: Obtained speed-up in (a) CWB1 and (b) CWB2 tetrahedral meshes

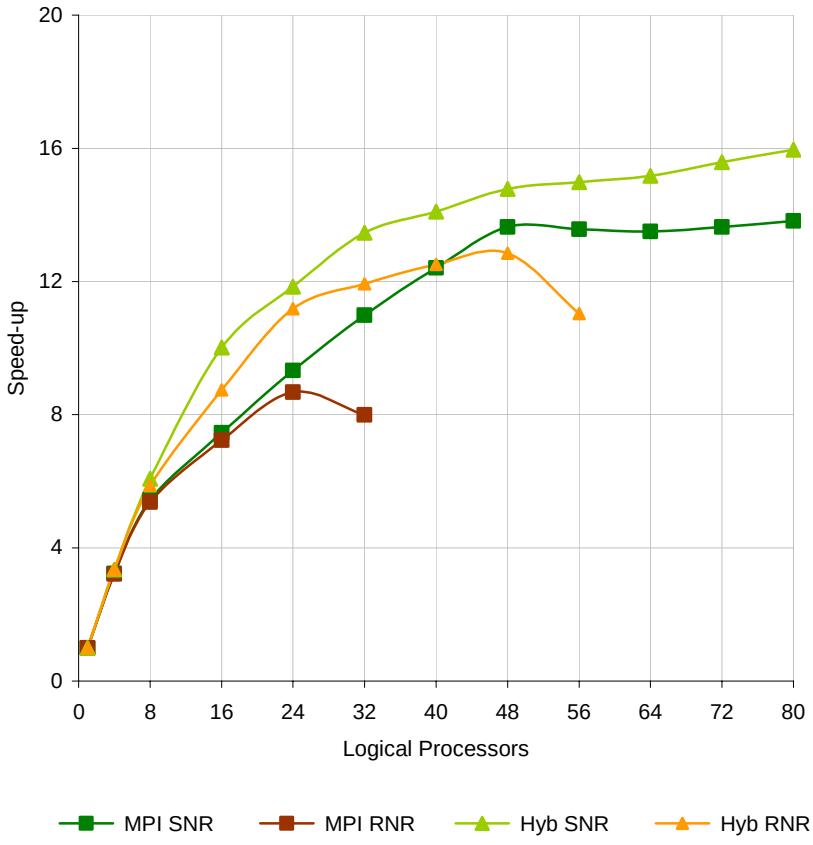


Figure 9: Obtained speed-up in CWB1 tetrahedral mesh

## 7 CONCLUSIONS

The results obtained seem to indicate that it is possible to reach good computational performance in numerical flow simulations using inexpensive clusters of personal computers. Besides, the task division method must be adequate to the hardware configuration of the cluster. Multi-core nodes with a single network interface seem to require a partition method that minimizes the number of communication operations instead the amount of data being communicated between processors, especially in unstructured meshes were the number of elements connected to each mesh node is high.

## 8 ACKNOWLEDGEMENTS

The authors wish to thank CAPES and CNPq for their financial support.

## REFERENCES

Argyris J., Doltsinis I.S., Friz, H., 1990. "Study on computational reentry aerodynamics". Computer Methods in Applied Mechanics and Engineering Vol. 81, pp. 257–289.

Donea J., 1984. " A Taylor-Galerkin for convective transport problems". International Journal for Numerical Methods in Engineering, Vol. 20, pp. 101–119.

Dorneles, R.V.; Rizzi, R.L.; Martinotto, A.L.; Picinin Jr., D.; Navaux, P.O.A.; Diverio, T.A., 2005. "Parallel Computational Model with Dynamic Load Balancing in PC Clusters". Lecture Notes in Computer Science, Springer-Verlag GmbH, Vol. 3402, pp. 468-479.

Hendrikson, B.; Leland, R., 1995. "An improved spectral graph partitioning algorithm for mapping parallel computations". SIAM Journal Scientific Computing, Vol. 16, No. 2, pp. 452-469.

Hoit, M., Wilson, F.L., 1983. "An equation numbering algorithm based on a minimum FRONT criteria". International Journal for Numerical Methods in Engineering, Vol. 16, pp. 225-239.

Schloegel, K.; Karypis, G.; Kumar, V., 2000. "Graph partitioning for high performance scientific simulations". In http://www.cs.umn.edu/research/technical_reports.php?page= report& report _id=00-018, accessed in October 2009.

Silvester, P.P.; Auda, H.A., 1984. "A memory economic frontwidth reduction algorithm". International Journal for Numerical Methods in Engineering, Vol. 20, pp. 733-743.