LOW COST DISTRIBUTED COMPUTATIONAL IMPEMENTATION OF A FINITE ELEMENT CODE FOR FLOW ANAYSIS USING WINDOWS ENVIRONMENT

João Ricardo Masuero
Centro de Mecânica Aplicada e Computacional – CEMACOM, Programa de Pós-Graduação em Engenharia Civil – PPGEC, Universidade Federal do Rio Grande do Sul – UFRGS. Av. Osvaldo Aranha 99. 3º andar, Porto Alegre, RS, Brazil.
masuero@cpgec.ufrgs.br

Armando Miguel Awruch
Centro de Mecânica Aplicada e Computacional – CEMACOM, Programa de Pós-Graduação em Engenharia Civil – PPGEC, Universidade Federal do Rio Grande do Sul – UFRGS. Av. Osvaldo Aranha 99. 3º andar, Porto Alegre, RS, Brazil.
amawruch@cpgec.ufrgs.br

Abstract. *Numerical analysis of 3-D flow problems usually involve models with large amount of unknowns and high computing power demand. In order to solve these problems, engineers and scientists have used both supercomputers and clusters of workstations. Supercomputers have high acquisition and maintenance costs, and become obsolete in a short period. Clusters of workstations are usually associated with specific network hardware and software, and special implementation techniques are necessary.*

The aim of this work is to test the viability of obtaining high performance computing using usual low cost PCs connected by low cost network hardware forming a temporary cluster, using neither special operational system nor special programming languages (in this case FORTRAN 90). The functionality of the computers for usual tasks are preserved; the computers became unavailable for the laboratory staff only while cluster is formed.

An application implemented in this computational environment is a Finite Element code for 3-D compressive flow analysis, using an explicit Taylor-Galerkin scheme. Time integration is accomplished using Taylor series while spatial discretization is carried out applying the Galerkin method and employing an eight-node hexahedral finite element. Element matrix integration is analytically calculated considering undistorted elements and using one point quadrature.

Results of scalability of solution time with respect to the number of PCs used and the size of the problem are shown. Initial results show the viability of this approach, regarding both the time of solution and the size of the problem that can be solved. Efficiency of the parallelization scheme employed is evaluated.

Keywords. *Flow analysis, Finite Elements, Distributed processing, Clusters.*

1. Introduction

Numerical solution of Solid Mechanics, Fluid Mechanics and other Engineering fields using the Finite Element Method or other numerical techniques demands frequently high performance computational systems, with both high speed processors and large main and secondary memory systems for large data structures storage.

An usual approach for this needs of computational power is the use of supercomputers with several vector processors and shared memory. This is usually the most powerful solution, but its acquisition and operational costs are extremely high, demanding special facilities, its maintenance is complex and specialized and in few years its computational power is reached and surpassed by much cheaper machines. Its use is only viable in open centers serving several users. Besides, only optimized codes (with respect to vectorization and parallelization) achieve acceptable performances.

Another approach is the use of conventional clusters of machines connected by high performance networks and using specific operational systems, programming languages and code libraries (Baloch et all, 2002). These clusters seldom allow the use of each machine as an ordinary computer for other common activities like text processing, graphic processing, code development and test. The operational system management and programming techniques are much more complex than those corresponding to single computers.

A new approach to high performance computing is the use of temporary clusters through internet-based personal computers (Kim and Lee, 2002). Following this approach, this work presents results of a temporary cluster formed by ordinary computers, operational systems, programming languages and network hardware used in scientific computing laboratories, without specific components of software and hardware. By this way, operational and maintenance costs are very low using standard non-expensive components, avoiding the need of personal training in specific operational systems and programming languages and allowing a full time use of the different computers, both as cluster for high performance numerical processing and as isolated workstations for general computer activities.

Even if the performance of the solutions using common tools employed in high performance computing is not reached, initial results points that the use of temporary clusters is very attractive for mid range computational problems, due to its good flexibility, easy operation and low costs.

2. Methodology

For the implementation of the temporary cluster, it was used only common hardware and software, usually available in many scientific computing laboratories.

The workstations chosen to form the temporary cluster are mid range personal computers of AMD Athlon class. These computers has high computing power, standard components and low cost, being usual in scientific laboratories for multiple purposes (text publishing, graphic publishing, CAD, numerical analysis and code development). Each computer has a 100 Mbps standard Ethernet network card, and the cluster is connected by a low cost Switch with 100 Mbps ports.

The operational system used in each computer is the Microsoft Windows 98SE. Although Linux has native support for clusters, the choice for Windows workstations is based in the fact that it was the operational system used in the computers of CEMACOM laboratory were the temporary cluster was implemented, and all personnel works with this operational system. Then, no additional training is necessary for the use of temporary cluster under Windows environment.

The programming language used for the temporary cluster implementation is FORTRAN 90/95, due to its high performance in numerical simulations, large use in scientific computing and many numerical codes already implemented using this language. Besides, almost all personnel in scientific computing laboratories use FORTRAN 90/95, being a natural choice for easy portability of cluster routines to existing codes.

Although the chosen programming language has no native support for network communication, no other language or library besides FORTRAN 90/95 was used to implement the routines of message passing and synchronization necessary for cluster operation. The information sharing among the several computers of the cluster was done using two important features of the operational system: a) local folders and files can be shared through the network for remote access; b) shared folders can be mapped as network units and consequently seen by FORTRAN (and any other programming language) as normal local disk units. Then, to get information from or to send information to a remote computer, FORTRAN reads or writes information in a file on a mapped network unit, making the routines for data sharing and synchronization extremely easy to be implemented and understood, since it corresponds to normal FORTRAN file access.

To send data or messages from a computer to another computer corresponds to write data in a disc file. To receive data or message in a computer from another computer corresponds to read data from a disc file. To wait for a message corresponds to read data continuously from a given record of a disc file, until other computer writes this record with the corresponding data or the waited message.

Each computer has *(n-1)* mapped network units, where *n* is the number of computers in the cluster. Trhough these network units, data sharing are performed by FORTRAN file access. This configuration is shown in Fig.(1).

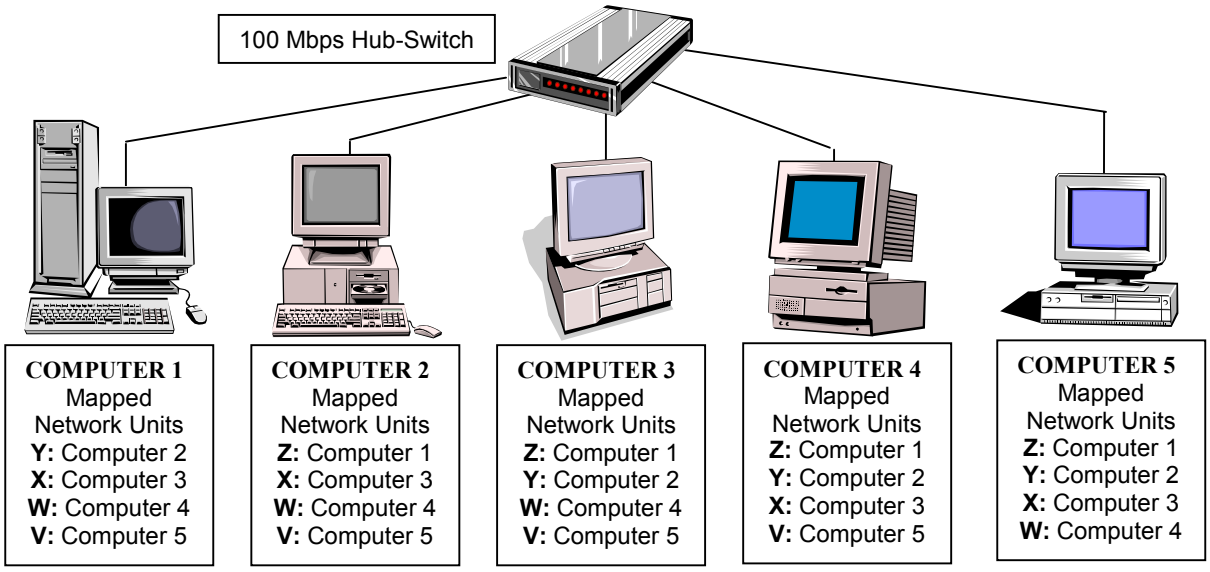

Figure 1. Typical configuration of a temporary cluster

Message passing process and small data sharing are performed by the computer which is sending the message writing data in remote files mapped by network units for all computers belonging to the cluster. This method, although implies in *(n-1)* writing operations for each message, avoid bandwidth saturation, since the receiving message process is performed by a continuous file reading operation by each computer in its own local disc unit. The time cost of multiplying writing operations is acceptable because the amount of data to be written is small. This approach is schematized in Fig.(2).

Otherwise, large data sharing is done through an opposite approach: the computer that is sharing data writes files in a local disc unit, and all other computers present at the cluster read these files remotely through the network. Since the

amount of data is large, the cost in time of writing *(n-1)* files on all remote units is too heavy to be supported with efficiency. The waiting state for this case does not affect network bandwidth because it is performed for a message (small amount of data) indicating that the file with large amount of data is available, as described previously. This approach is schematized in Fig.(3).

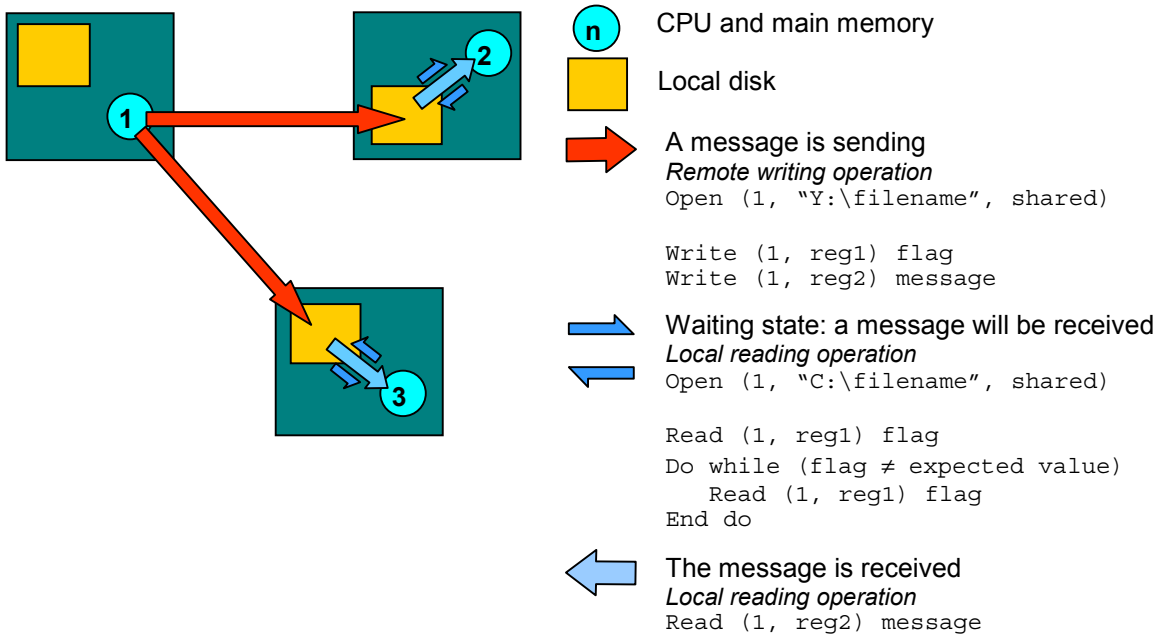

Figure 2. Message passing and small amount of data sharing scheme

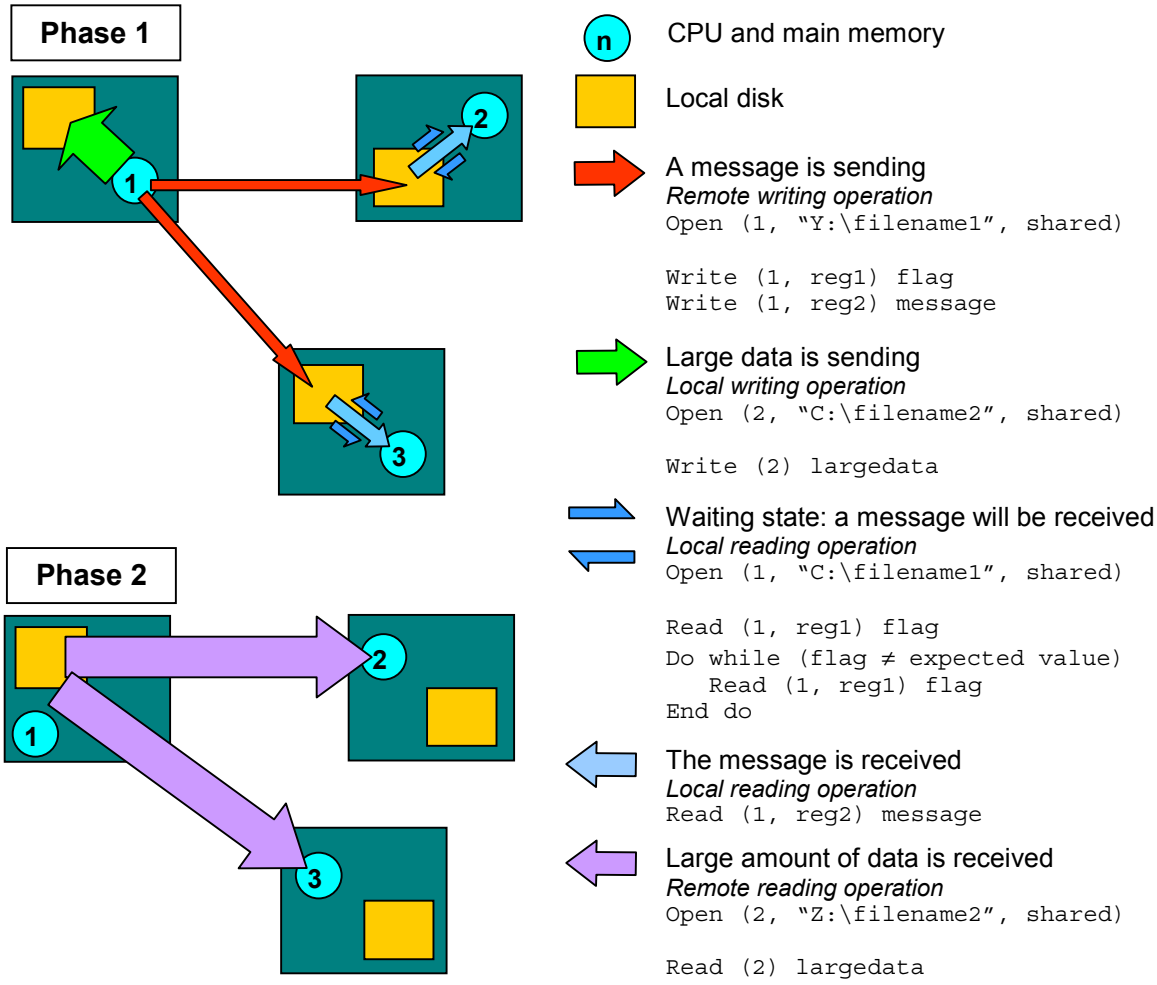

Figure 3**.** Large amount of data sharing scheme

A possible bottleneck in the performance of this approach is that sharing information allies two classes of slow operations: network communication (limited by a bandwidth of 100 Mbps for all simultaneous access to a given computer) and disk file access (which is one of the slower operations in modern computers). These characteristics lead to cluster implementation algorithms that minimize data sharing between computers in the cluster.

The network bandwidth available is shared by all the simultaneous accesses to a given computer. If there is only one computer B accessing data from a computer A, the communication occurs at full network bandwidth, 100 Mpbs. If there are two computers, B and C, accessing data from a computer A simultaneously, the communication between A and B, and A and C occurs at half network bandwidth, 50 Mbps. Several simultaneous communications cause network saturation and performance degradation. Otherwise, if computer B access data from computer A, and computer D access data from computer C, and computer F access data from computer E simultaneously, due to the presence of a switch connecting all computers, each communication occurs at full network bandwidth (100 Mbps) and no network saturation or performance degradation happens. To maximize de use of network bandwidth, all large amount of data are read by each computer i in the following order: from computer $i+1$ to n and from computer 1 to $i-1$. This rule to read a large amount of data is easy to implement and usually avoids the simultaneous access of data from one computer by more than one other computer.

3. Implementation

The methodology described above was first implemented in a code for static analysis of 3D linear elastic structures using hexahedral elements with 8 nodes, 1 point quadrature and hourglass control (Duarte Filho, 2002). The solver used was the conjugated gradients interactive process, with diagonal preconditioners (Alquati, 1991, Huges and Ferencz, 1987). This first implementation showed the viability of the parallelization approach regarding easiness of implementation and performance achieved (Masuero and Awruch, 2002).

In this work, the parallelization approach was applied to a Finite Element code for 3-D compressive flow analysis, using an explicit Taylor-Galerkin scheme. Time integration is accomplished using Taylor series while spatial discretization is carried out applying the Galerkin method and employing the same eight-node hexahedral finite element. Element matrix integration is analytically calculated considering undistorted elements and using one point quadrature. The resultant equations are solved for each degree of freedom in a uncoupled way (Burbridge and Awruch, 2000).

The task division for parallel solution is made based in the allocation of groups of nodes for each computer belonging to the cluster. In Fig.(4), for easier representation, a 2D mesh of quadrilateral elements has its nodes divided in three equal groups, one for each computer. Since several calculations in the code are made in element loops, each computer has to work over all elements connected to its node group. Besides, several element calculations depends on the correct value of the variables corresponding to all its nodes. This way, each computer has to work over its own node group and is necessary to include all nodes belonging to elements connected to its node group, defining the whole work group for each computer.

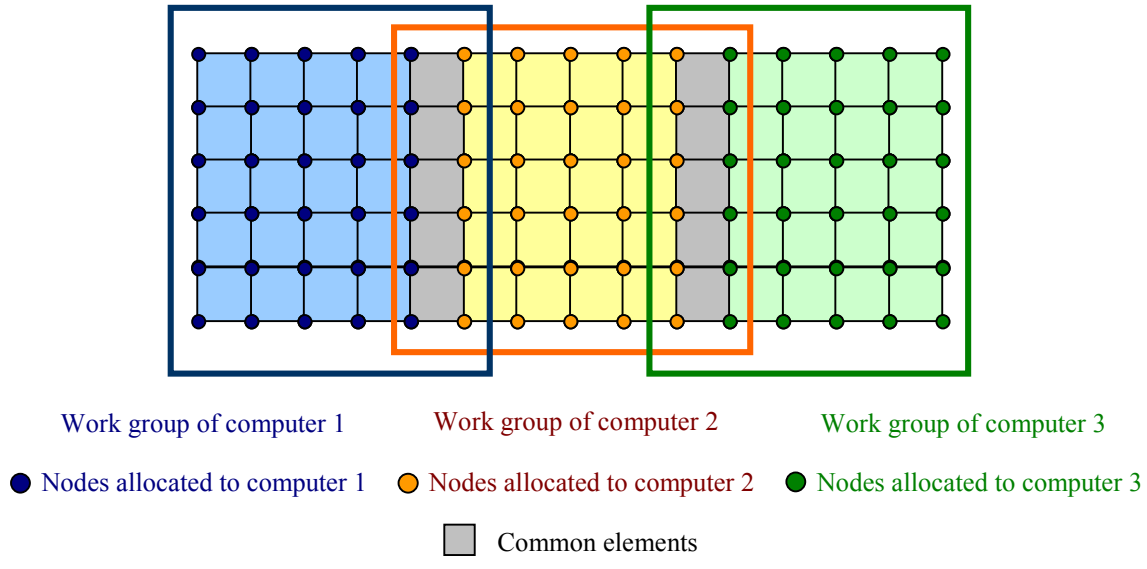

Figure 4. Uniform task division with a good node ordering

In a given time step, each computer calculates only the new values of variables related to its own node group. In order to advance the time integration algorithm, each computer needs the new values of nodes that not belong to its node group, but connected to elements with nodes corresponding to different node groups. This condition defines the amount of data that must be shared through the network among the several computers forming the cluster.

Since there are always redundant work in the task division scheme, represented by the superposition of work groups of the different computers, the efficiency of parallelization is less than 100%. Considering the work associated to elements, the task division in Fig. (4) results in the calculation of 80 elements (25 by computer 1, 30 by computer 2, 25 by computer 3) in a mesh of 70 elements, that corresponds to an efficiency of 87,5% in element calculations. If the complete task was divided in 5 computers, an uniform task division would result in the calculation of 90 elements (15 by computer 1, 20 by computers 2,3 and 4 and 15 by computer 5), resulting in an efficiency of 77,8%. Time spent in data sharing is also indirectly dependent of the number of common elements.

So, regarding efficiency of parallelization, better performance is obtained in task divisions that result in fewer common elements, due to both less data sharing through the network (which is a slow operation) and less redundant work made by the computers belonging to the cluster.

The way that task division was implemented in this work is quite simple: the first computer works over the first *n/m* nodes, where *n* is the total number of nodes and *m* the number of computers in the cluster, the second computer works over the second *n/m* nodes and so on. This implementation is highly dependent of the node ordering in the mesh. The same problem of Fig.(4), but with a different node ordering, is shown in Fig.(5), with an increase of common elements and decreasing of parallelization efficiency.

Considering the work associated to elements, the task division in Fig. (5) results in the calculation of 98 elements (28 by computer 1, 42 by computer 2, 28 by computer 3) in a mesh of 70 elements, that corresponds to an efficiency of 71,4% in element calculations, compared to 87,5% of the former task division, given in Fig.(4).

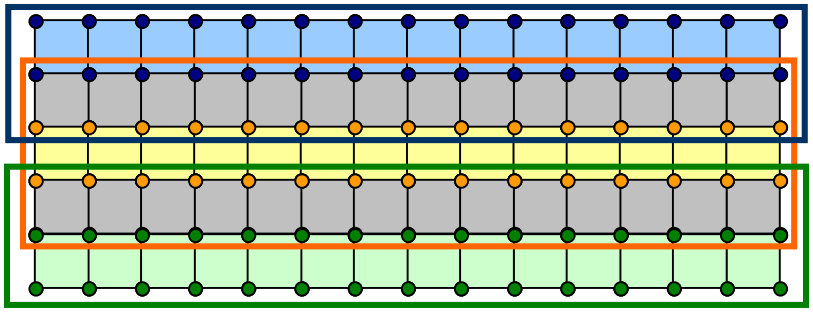

Figure 5. Uniform task division with bad node ordering

Although the solution algorithm is independent of the node ordering, since it works equation by equation in an uncoupled way inside the time step, the minimization of the number of common elements is very similar to bandwidth minimization problem in a direct solver. This way, the bandwidth minimization algorithm presented by Teixeira (1991) was used successfully.

When machines with different computing power work together in a cluster, if a uniform task division is used, the slowest computer determines the performance of the cluster. To avoid loss of efficiency, the task allocated to each computer must be proportional to its computing power. In this work, this is done running a series of benchmark problems using a single computer code that is identical to the cluster code, and determining a performance index for each computer based in the average relative speed. So, the number of nodes allocated to each computer in the cluster is proportional to its performance index. Fig.(6) shows the mesh of Fig.(4) with a task division for three computers with performance indexes of 200 (40 nodes), 150 (30 nodes) and 100 (20 nodes).

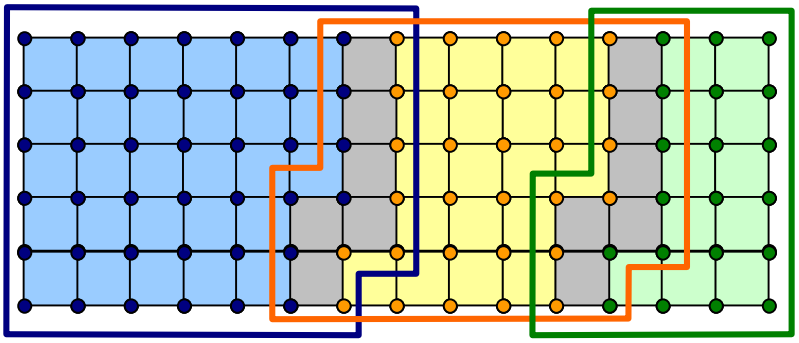

Figure 6. Proportional task division for computers with different computing power.

This approach is based on the assumption that the work is proportional to the number of nodes allocated. This is not really true because several calculations depend on the number of elements, and special considerations like boundary conditions has no direct relation with the number of nodes. However, it is expected that, for large problems, the task distribution follows the number of nodes allocated in each computer.

4. Results

A supersonic flow of a nonviscous compressible fluid past a sphere with radius R=1.0 (Burbridge and Awruch, 2000) was taken as a basic problem from which a family of meshes were created to evaluate the behavior, with respect to performance and efficiency, of clusters regarding the increasing of problem size and number of computers forming the cluster. Figure (7) shows the problem to be analyzed and the appearance of the family of meshes that will be used. Table (1) shows the basic data of the different meshes.

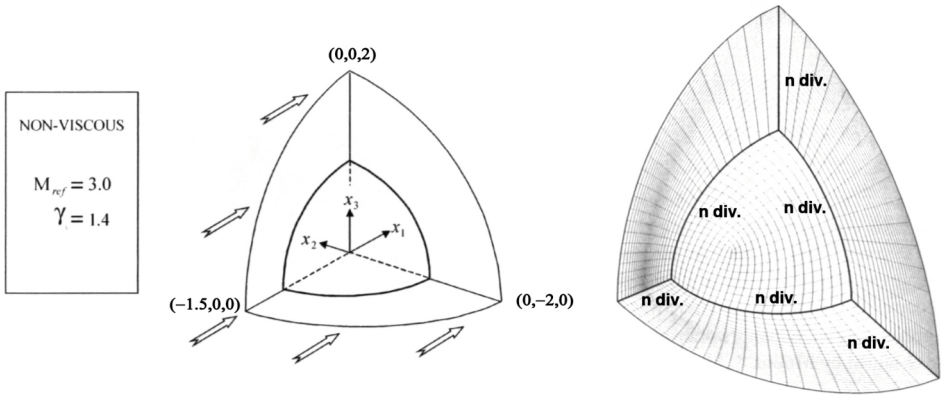

Figure 7. Problem to be analyzed and family of meshes that will be used

Table 1. Basic data of the meshes

n div.	20	30	40	50	60	70	80	90
Nodes	6000	20250	48000	93750	162000	257250	384000	546750
Elements	6951	22351	51701	99501	170251	268451	398601	565201
Equations (DOF)	34755	111755	258505	497505	851255	1342255	1993005	2826005
Memory required (MB)	5	16	38	74	127	201	298	423

The temporary clusters used in this work were formed by up to 6 mid-range personal computers with AMD Athlon processors from 1333 MHz to 1533 MHz, 512 Mbytes of RAM (PC133 or PC266, depending on the computer) and Hard Disks of 7200 rpm class and connected by a low cost Hub-Switch Encore ENH908-NWY+ with eight 10/100 Mbps ports. The individual performances of the computers were evaluated, and the performance indexes were estimated to be equal to 128, 128, 125, 111, 111 and 103, from the fastest to the slowest computer. These indexes indicate the relative speed of the computers and they were used to weight the task distribution in the clusters.

The average speed of each computer and clusters, in Mflops, were estimated through a linear relation of the total time of solution of each problem in a CRAY T94 and in the desired configuration and the average speed in Mflops reported by CRAY´s operation system.

The results obtained are shown in Fig. (8), where "AXXX" indicates a single computer with XXX performance index and "n Comp" is referred to a temporary cluster of *n* computers, gathering always from the fastest to the slowest computer.

The behavior of clusters should be similar to single computers, but Fig.(8) shows that clusters with many computers are not efficient for small problems: the corresponding curves present a curvature which can not be seem in the single computers curves, related to the lack of efficiency in the parallelization approach.

The speed-up, regarding the number of computers forming the cluster, is shown in Fig.(9). The theoretical speed-up is not a straight line at 45^o because the computers have not the same performance. The efficiency of parallelization can be evaluated by the following expression:

$$\eta = \frac{\left(T_{cluster}\right)^{-1}}{\sum_{i=1}^{n}\left(T_i\right)^{-1}} \tag{1}$$

where $T_{cluster}$ is the total time of solution for a given cluster, T_i is the total time of solution for a single computer i and n is the number of computers forming the cluster. The efficiencies obtained in the parallelization approach used in this work are presented in Fig. (10).

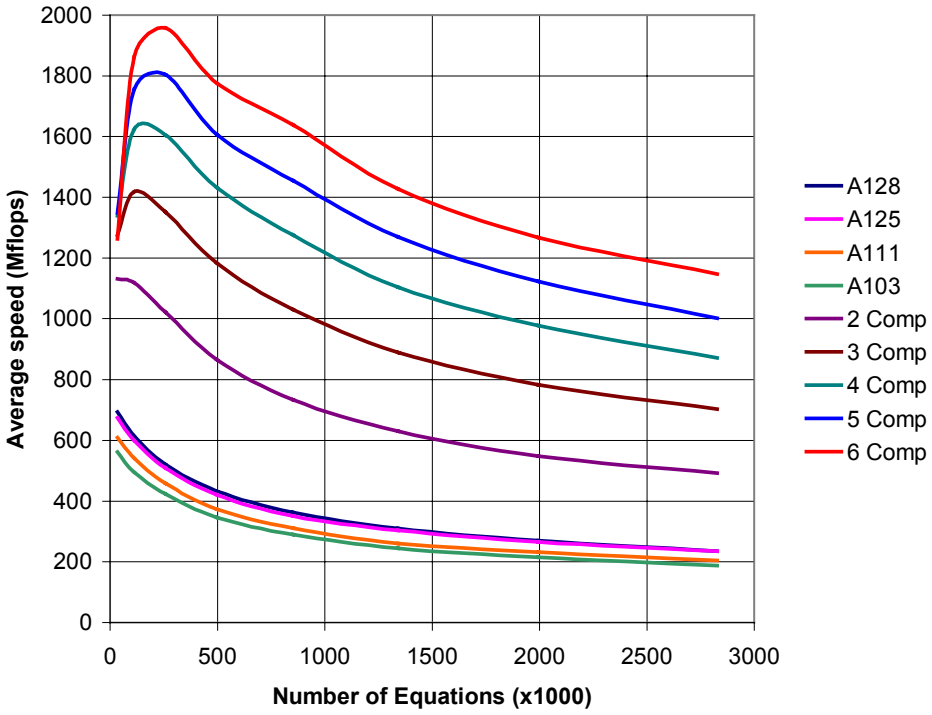

Figure 8. Average speed of single computers and temporary clusters with mid-range computers.

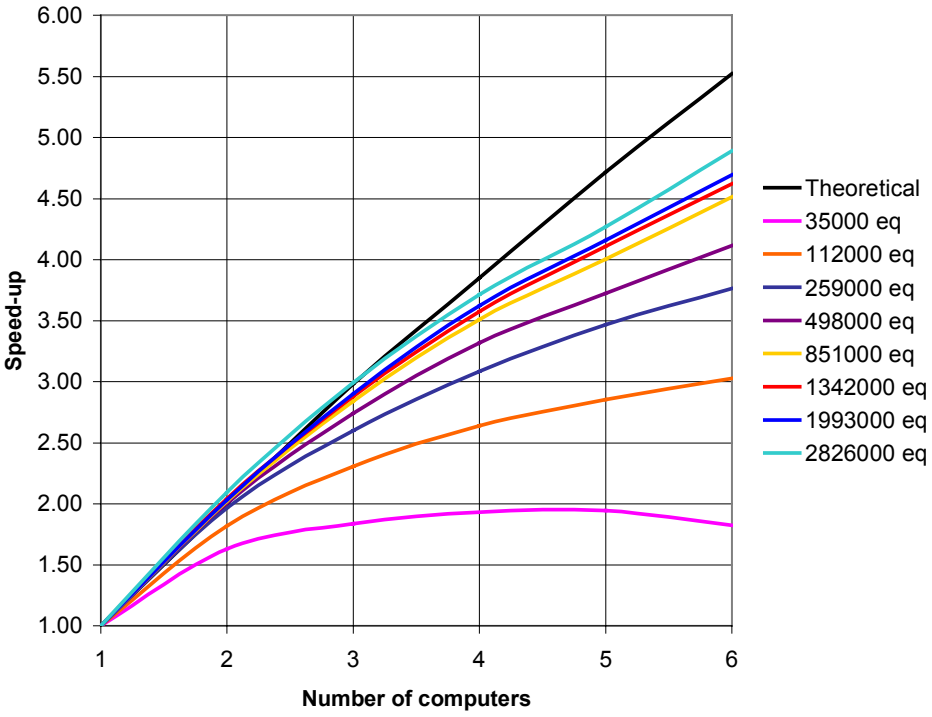

Figure 9. Speed-up of clusters for several problem sizes with mid-range computers

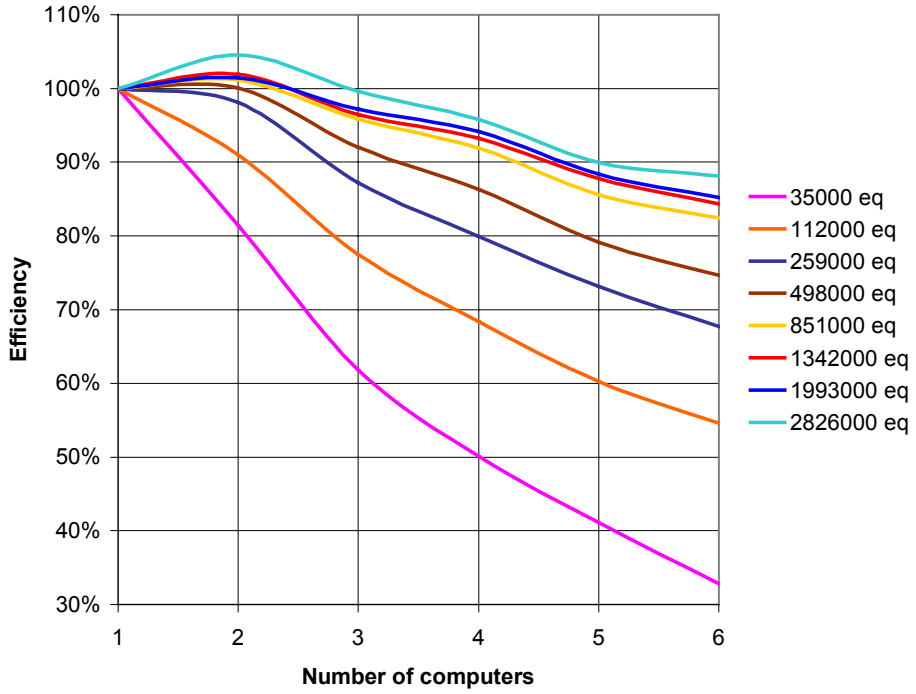

Figure 10. Efficiency of parallelization for several problem sizes with mid-range computers.

Figures (9) and (10) show that the approach used is efficient for large problems (more than 500000 equations), allowing good speed-up and efficiency of parallelization. More computers could be added to the cluster maintaining the efficiency above 80% .

5. Conclusions

The methodology used in the cluster implementation seems to have acceptable efficiency for application to small clusters and problems of reasonable size. The implementation is easy and the use of common programming language is a very desirable characteristic. The possibility of using temporary clusters of low cost hardware make this approach very useful for small research centers

The algorithm of task distribution worked adequately, although improvements can be implemented.

6. References

Alquati, E.L.G., 1991, "Preconditioning of the conjugate gradient method with na element-by-element formulation" (in portuguese: Precondicionamento do método dos gradientes conjugados numa formulação elemento-por-elemento). MSc. Thesis, PPGEC, Universidade Federal do Rio Grande do Sul, 117p.

Baloch, A, Grant, P.W. and Webster, M.F., 2002, "Homogeneous and heterogeneous distributed cluster processing for two- and three-dimensional viscoelastic flows", International Journal for Numerical Methods in Fluids, **40,** pp.1347-1363.

Burbridge, H.P. and Awruch, A.M., 2000, "A finite element Taylor-Galerkin scheme for three-dimensional numerical simulation of high compressible flows with analytical evaluation of element matrices", Hybrid Methods in Engineering, Vol.2, pp. 485-506.

Duarte Filho, L.A., 2002, "Static and dynamic linear and geometrically non linear analysis with eight node hexahedrical elements with one point quadrature" (in portuguese: Análise estática e dinâmica, linear e não-linear geométrica, através de elementos hexaédricos de oito nós com um ponto de integração). MSc. Thesis, PPGEC, Universidade Federal do Rio Grande do Sul, 112p.

Hughes, T.J.R. and Ferencz, R.M., 1987, "Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing ebe preconditioned conjugate gradients", Comp. Meth. Appl. Mech. Engng.,**61**, pp. 215-248..

Kim, S.J. and Lee,C.S., 2002, "Large-scale structural analysis by parallel multifrontal solver through internet-based personal computers", AIAA Journal, **40,** No.2, pp. 359-367.

Masuero, J.R. and Awruch, A.M., 2002, "A distributed computational implementation of a finite element code using fortran in a windows 9x environment". Mecánica Computacional, Vol. XXI, pp.2945-2957.

Teixeira, F.G., 1991, "A node renumbering system to reduce the bandwidth of the stiffness matrix" (in portuguese: Sistema de reordenação nodal para soluções tipo banda), MSc Thesis, PPGEC, Universidade Federal do Rio Grande do Sul, 98p.