Mesh Generation on a T3D

Joan Carles Mestres, International Center for Numerical Methods in Engineering, Barcelona, Spain.

ABSTRACT: *The goal of this paper is to show the performance of a parallel unstructured mesh generator as well as the improvements in terms of speed-ups in front of the equivalent results achieved without parallelization. The mesh generator allows the partition of domains in a number of subdomains corresponding to the number of processors to be used in a parallel computer, the mesh generation for each subdomain performed by each processor, and lastly, the assembly of the individual outcomes in different files in order to apply the corresponding finite element solvers.*

Introduction

Nowadays the necessity for fast generation of large unstructured meshes for finite elements or finite volume computations is common to many fields in science and engineering and is showing its importance in structural analysis and computational fluid mechanics, where this generation has represented many times a bottleneck for the overall process.

In some applications, the mesh needs to be regenerated either locally or globally many times, even hundreds of times, like in the transient simulations with moving bodies [1]. The possibility of carrying out numerical simulations in parallel computers has helped to significantly reduce the required computational time in many applications. An important pending task will be to discretize the problem efficiently to port unstructured mesh generation to parallel machines. Despite its importance, very few references of work on this subject have been reported [2-5].

To be efficient in the fields where the parallelization of the mesh generation has proved its enhancements, the generation has to be able to handle complex geometries. It seems advisable, for this reason, the use of triangular elements when dealing with 2D meshes and tetrahedra for 3D meshes.

To achieve the maximum efficiency of the parallelization, obtained by avoiding as much as possible the transfer of information among processors, thus allowing great savings of CPU time, the aim of a parallel mesh generation is to work as independently as possible in each particular subdomain. In this work, this has been possible using the advancing front technique, once a common segment of the initial front for different processors -the shared part of the common boundaries-, is defined.

The parallel mesh generation program developed performs the partition of domains in a number of subdomains corresponding to the number of processors to be used in the subsequent analysis computations. The mesh generation for each subdomain is performed for each of the processors, so the most CPU time consuming process, with the introduction of all nodes and elements, what involves a large amount of operations, can take advantage of the parallelism. Lastly, the program assembles the individual outcomes in different output files, depending on the finite element solver to be used.

The paper displays the steps that enable the parallelization of the mesh generation once the problem has been broken up to be solved into pieces, how to handle each processor a piece of the problem and how to make it compatible with all the rest.

The outline of this program is showed in the flow chart of figure 1, pointing out the parts of the program that are executed by a single processor (sequential execution) and the parts that are performed by all processors (parallel execution), as well as the section of the program whose CPU time is measured, which corresponds to the explicit mesh generation. Finally, when the generation is over, the numerical aspects that arise from the parallelization and lead to the overall improvement, represented in the examples, are analysed.

All the technical concepts involved in the advancing front technique that are not specific to the parallelization of the program (generation of a background mesh, construction of the interior points of the boundaries and the initial advancing front, creation of the interior elements, introduction of the elongation to get stretched elements if desired, searching elements algorithms and smoothing operations) can be gotten from the bibliography [7-12]. All numerical values in the program have been adopted from these criteria [10].

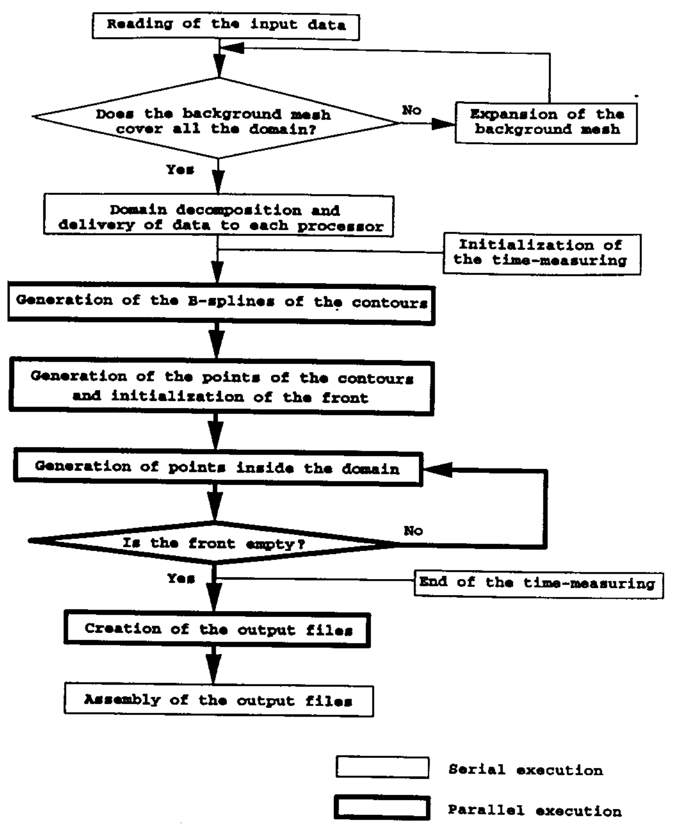
Reading of the input data
Does the background mesh cover all the domain?
No
Expansion of the background mesh
Yes
Domain decomposition and delivery of data to each processor
Initialization of the time-measuring
Generation of the B-splines of the contours
Generation of the points of the contours and initialization of the front
Generation of points inside the domain
Is the front empty?
No
Yes
End of the time-measuring
Creation of the output files
Assembly of the output files
Serial execution
Parallel execution

Figure 1: Flow chart. This diagram shows the processes performed in parallel and in serial execution and the section of the program whose CPU time is measured.

UNSTRUCTURED MESH GENERATION PROGRAM

A background mesh

A background mesh is a mesh that usually covers all the domain to be meshed. At the nodes of this mesh the user defines the desired element size, element stretchings, and stretching directions. The corresponding values for every new created point are obtained by interpolation of the nodal values of the background triangle that contains it. With the method developed by the authors, it is not even necessary that the background mesh covers all the domain, as the program checks this eventuality, before delivering the subdomains to be meshed by each processor, and in case that the domain to be meshed is not completely covered, one of the processors expands the background mesh in order to achieve the whole covering.

This fact enforces the program to run the generation of the background mesh in a sequential way, as the partition in subdomains before defining the corresponding background mesh for each one does not provide any clear computational advantage, but oppositely it would represent a greater computational cost, as well as an unnecessary check of conformity among the corresponding meshes subsequently generated by each individual processor.

To achieve this goal in a quick way, with the minimum transfer of communication among processors, it is imposed that the nodes that belong to the boundary of a subdomain shared by another one must be the same also for the latter. To clarify these ideas and to show the results in a simplified way, henceforth all the concepts and examples will make reference to the 2D advancing front technique.

Once the boundaries are defined, the program will split the domain into the desired number of subdomains. There are different ways of doing that [13-18]; in order to allow the user to take control over the domain generation, he himself can define the boundaries between subdomains.

Generation of the subdomains

The subdomain's boundary will be composed by the combination of a series of closed curves that will limit the outer space, as well the inner space in case that there exist holes inside the domain. Obviously, there will exist interior lines to divide materials or representing loads, for example. All these curves will be defined through the interpolation of series of points entered as data. The order of the introduction of these points will define the sense of the curves.

The exterior boundaries will be oriented in clockwise sense, whilst the interior ones -these limiting interior holes- will be oriented in anti-clockwise sense. For the interior curves separating different subdomains, the sense will be defined accordingly to the subdomain to be considered.

Every processor will then generate, independently in every subdomain, the intermediate points accomplishing the corresponding spacing between two consecutive points, according to the definitions of the background mesh. Later on, the initial front created, everyone will make up the interior unstructured meshes, and once all the individual output files are assembled, the program will perform the smoothing in a sequential way, keeping unmoved the boundaries and the explicitly constrained points.

Results

The examples will consist in the generation of a mesh over a square domain with a square hole inside, run sequentially and in parallel with four (figure 2) and sixteen processors (figure 3), for six different spacings.

The first reference model, called BIG1, has the spacings shown in figure 4, what will be divided by 2, obtaining the model called BIG2, and will be divided by 5, obtaining the so-called BIG5. Successively, dividing by 10, 20 and 50, there will be obtained the models SMALL1, SMALL2 and SMALL5.

With all the required information, the program creates the front of the boundaries, equaling the generated points in the common interior boundaries for the two initial fronts related to the subdomains that share the curves, although being defined in inverse sense to allow the mesh generation in the interior areas of both subdomains. In this way, the corresponding meshes for the six models to be compared are obtained. The number of elements and nodes are shown in Tables 1 and 2.

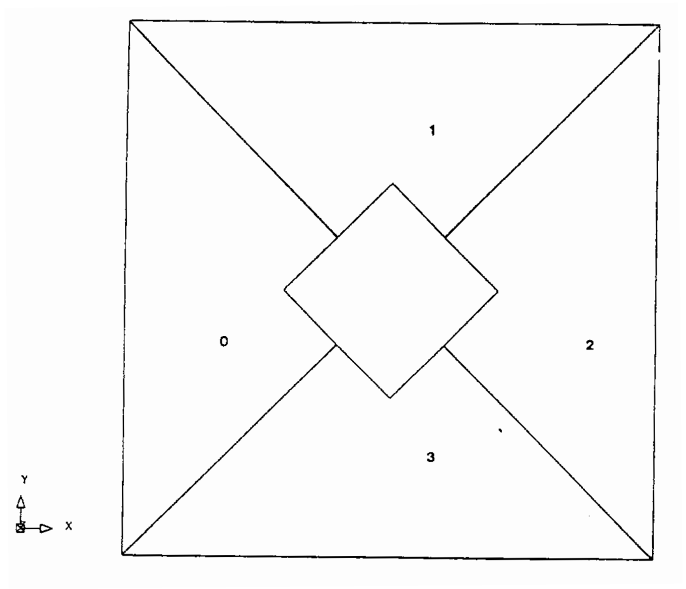

Figure 2: Original domain split into four subdomains.

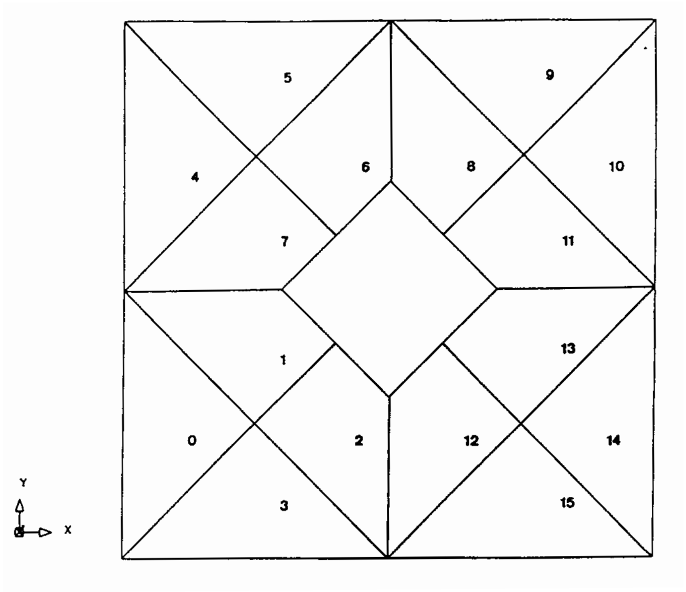

Figure 3: Original domain splitted in sixteen subdomains.

In these examples, the time needed for each processor to generate the corresponding interior mesh has been calculated. This represents to measure the time spent since the processor begins to simulate its own boundary by B-splines (initialization of the advancing front) until it finishes the mesh (empty front).

Two values are considered. The first value -henceforth, called the CPU time-, expresses the addition of the CPU time spent by each processor in the respective procedure, it is, the whole amount of CPU spent.

The second value -henceforth, called the elapsed time-, expresses the waiting time since the process has begun until it finishes; as the different processors run synchronously, it will

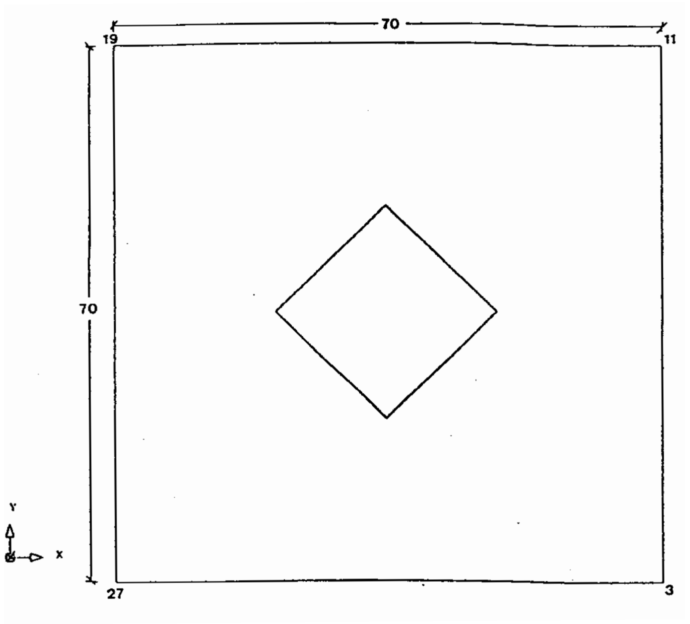

Figure 4: Spacings in Model BIG1.

represent for the parallel procedures the time spent by the processor that takes longer.

All the examples have been run in a CRAY-T3D, and all times are expressed in milliseconds.

The different obtained times are shown in table 3, for models BIG1, BIG2 and BIG5 and in table 4 for models SMALL1, SMALL2 and SMALL5.

For a comparison among the different values, let's assume a relationship of the type y=kx$^{\alpha}$being y the value of the CPU time and x the number of nodes.

Taking logarithms on both sides, the expression gives

$$\mathrm{Log}_{10}\, y = \mathrm{Log}_{10}\, k + \alpha \mathrm{Log}_{10}\, x$$

and making $y = 10^{\psi}$, $k = 10^{\kappa}$ and $x = 10^{\xi}$

the following expression is obtained:

$$\psi = \mathrm{K} + \propto \xi$$

Using the latter, it is therefore possible to represent the sought relationship in a double logarithmic scale as a straight line.

Using the time measures, it is possible to estimate α. Taking the values obtained in the sequential procedures,

$$\alpha = \frac{\mathrm{Log}\ 3530231 - \mathrm{Log}\ 82}{\mathrm{Log}\ 51521 - \mathrm{Log}\ 38} = \frac{6.5477 - 1.9138}{4.712 - 1.58} = 1.48$$

Figure 5 shows that this value is very well represented along all the curve, as there are not problems of balancing among processors. For the parallel procedures, the slope of the straight line becomes well represented for numbers of nodes big enough, because the influence of linear factors (generation of the points on the boundaries) is more negligible in front of the rest when

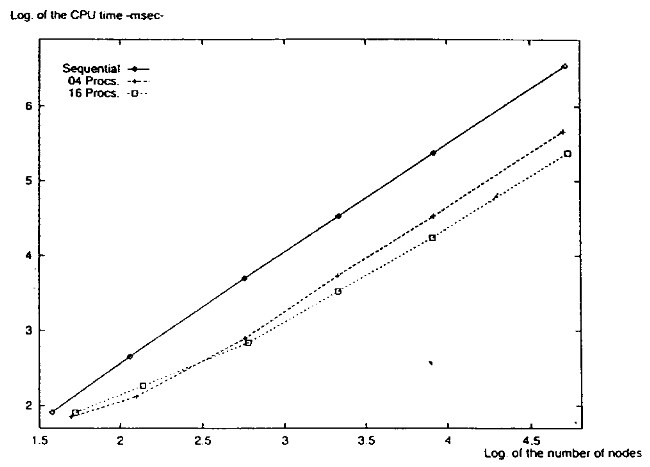

Figure 5: Relationship between the logarithm of the CPU time in milliseconds and the logarithm of the number of nodes - $\psi = K + \propto \xi$ - for all procedures.

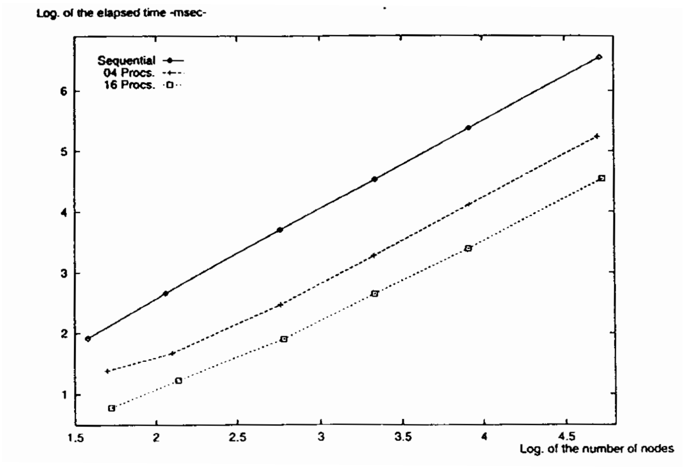

Figure 6: Relationship between the logarithm of the elapsed time in milliseconds and the logarithm of the number of nodes - $\psi = K + \propto \xi$ - for all procedures.

the number of interior nodes is bigger in front of the boundary ones.

With these considerations in mind, the exponent for the different parallel procedures has been calculated. For the procedure with four processors, an approximate value of 1.426 (calculating the slope from the model SMALL5 -458764 milliseconds and 49893 nodes- to the model BIG5 -789 milliseconds and 574 nodes) is obtained and for the procedure with sixteen processors, the approximate value is 1.326 (calculating the slope from the model SMALL5 -235843 milliseconds and 53253 nodes- to the model SMALL1 -3323 milliseconds and 2138 nodes).

When it comes to the elapsed times instead of the total CPU times, the considerations about the importance of a good balance becomes decisive to get the smallest possible time.

Taking as reference the same models considered in the estimation of the slope for the CPU times, the slope value for the four processors procedure is 1.428 -approximately the same as the obtained for the ensemble of the processors, what coincides with the appreciation of a good sharing.

The slope for the sixteen processors procedure of 1.361, (obviously, the value for the sequential procedure will be the same as the CPU times and elapsed times are the same) that has a bigger difference respect the slope obtained for the ensemble of the processors, what could be explained by a greater gap between the fastest and slowest processors, because of the difference between the number of elements generated by the two extremes in both cases.

Figure 8 shows on the Y-axis the logarithm of the elapsed time spent in milliseconds for the three different procedures: sequential, with four processors and with sixteen processors, while on the X-axis shows again the logarithm of the number of nodes.

Two final tables with their corresponding graphics are included to present the enhancements of parallelization, as a synthesis of these results.

Table 5 gives the ratios of the CPU times between the sequential and a four processors procedure, between the four and a sixteen processors procedure and between the sequential and a sixteen processors procedure concerning to all models.

To have a comparative value for the number of elements or nodes in every model, as these numbers did not coincide exactly, it has been chosen a common value for each model. Hence, for model BIG5, the arithmetic mean is 1050.67, obtained from

$$\frac{1028 \quad + 1036 \quad = 1088}{3}$$

consequently, the number 1000 is taken as the common number of nodes for the different procedures.

From this value, all the rest can be calculated easily. When the spacing in all the nodes increases twice, the number of elements increases four times (the square of 2). Analogously, a decrease of 2.5 times the spacing in all the nodes will represent a decrease of 6.25 times (the square of 2.5).

From this reference, the different number of elements considered will be: 40, 160, 1000, 4000, 16000 and 100000 for models BIG1, BIG2, BIG5, SMALL1, SMALL2 and SMALL, respectively.

It is interesting to see the relationship between the obtained values for the different procedures, as it can be an indicator of the goodness of the achieved balance among processors.

In fact, for perfectly balanced work-share among processors, the improvement for four processors in front of the sequential procedure would be the same as the improvement for sixteen processors in front of the four processors procedure.

In this case, the time spent by sequential procedure would be $y = kx^{\alpha}$. For the four processors procedure, the time spent would be

$$y = \sum_{i=1}^{4} k\, x_i^{\alpha} = k\frac{x^{\alpha}}{4^{\alpha-1}}$$

For the sixteen processors procedure, the time spent would be

$$y = \sum_{i=1}^{16} k\, x_i^{\alpha} = k\frac{x^{\alpha}}{16^{\alpha-1}}$$

And the ratio between the two first exposed times and between the two last would be exactly the same: $\left(4^{\alpha-1}\right)^{-1}$.

In the latter, the ratio goes from 1.616 to 1.945 for the three SMALL models, and the ratio between the sequential and with four processors is not too far, as it goes from 6.251 to 7.695, what seems a rather acceptable balance.

It can be appreciated a regular relationship between the two parallel procedures. To consider the previously exposed relationship of $\left(4^{\alpha-1}\right)^{-1}$, for an average value of 1.376, a ratio of 1.684 would be achieved, what can be a representative value for the biggest values (the most reliable) of the shown ratios in the table.

All these values have its graphical representation in figure 7, that represents the relationship between the CPU times for the different procedures in all models, while the X-axis indicates the logarithm of the number that represents the amount of generated elements.

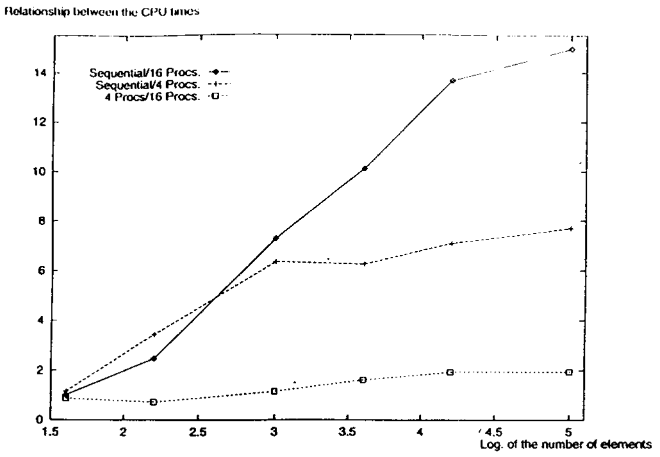

Figure 7: Relationship between the CPU times.

Most of the previous considerations apply for the elapsed times. However, it cannot be expected the same degree of proportionality and, only in case of a perfectly balanced domain decomposition, the ratio between the obtained times for every couple of procedures will have the same simple expression.

In fact, in this case, the time spent for the four processors procedure will be $k\frac{x^{\alpha}}{4^{\alpha}}$ and for the sixteen processors procedure will be $k\frac{x^{\alpha}}{16^{\alpha}}$, with a new common ratio between the consecutive procedures of $\left(4^{\alpha}\right)^{-1}$.

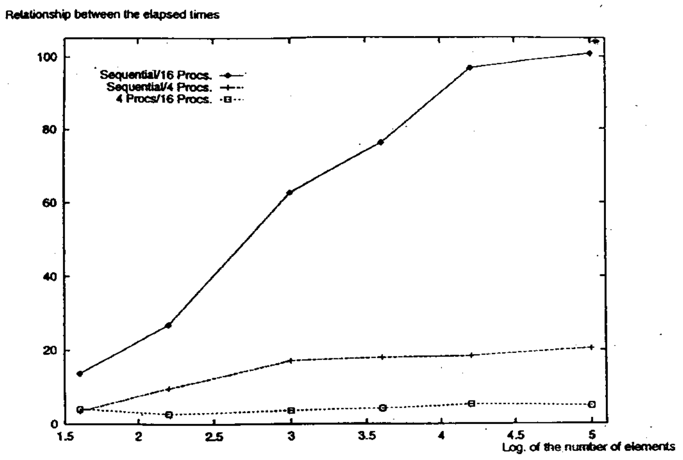

Figure 8: Relationship between the elapsed times.

Again, the elapsed times are also presented. The ratios between the different procedures are shown in Table 6.

All these values are equally represented in figure 8. The elapsed time is an important value to be considered. It represents, definitely, the difference in waiting time between different procedures, so the effective speed-up that can be achieved. With the synchronism of the parallel processors, the time needed to run model SMALL5 sequentially is more than 20 times the time needed to run it with 4 processors and more than 100 times the time needed to run it with 16 processors.

These figures can be a proof of the conclusions of this study: the improvements presented for the different procedures agree with the goals of parallelization, important speed-ups and minimization of the interprocessor transfers of information.

Numerical example. A real mesh

The mesh generation of a real example is presented. The example represents the front of an aircraft and it has been taken from a workshop on hypersonic flows [19].

Two procedures have been run, the sequential case and with four processors.

In the first case, 315 elements for 183 nodes have been obtained, whereas with four processors there are 305 elements for 179 nodes, although two new nodes on the exterior boundary have been added to better define the subdomains.

Both overall meshes are shown in figures 9 and 10.

Finally, in figure 11, it can be seen the final smoothed mesh, once all the individual meshes have been joined for the smoothing.

For the sequential procedure, it takes 1073 milliseconds to create the mesh, while with four processors, the total CPU time to create the mesh is 219 milliseconds, and the processor that

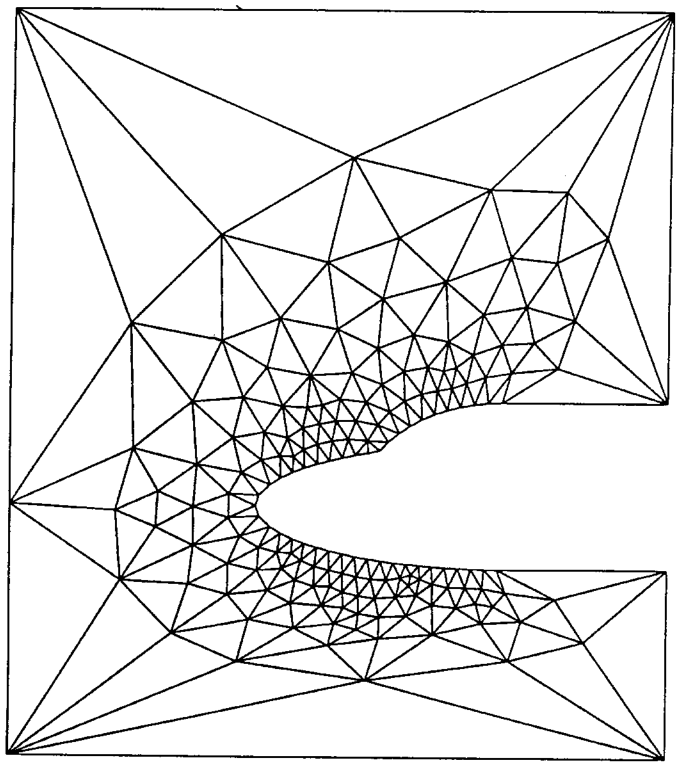

Figure 9: Sequential procedure (315 elements, 183 nodes).

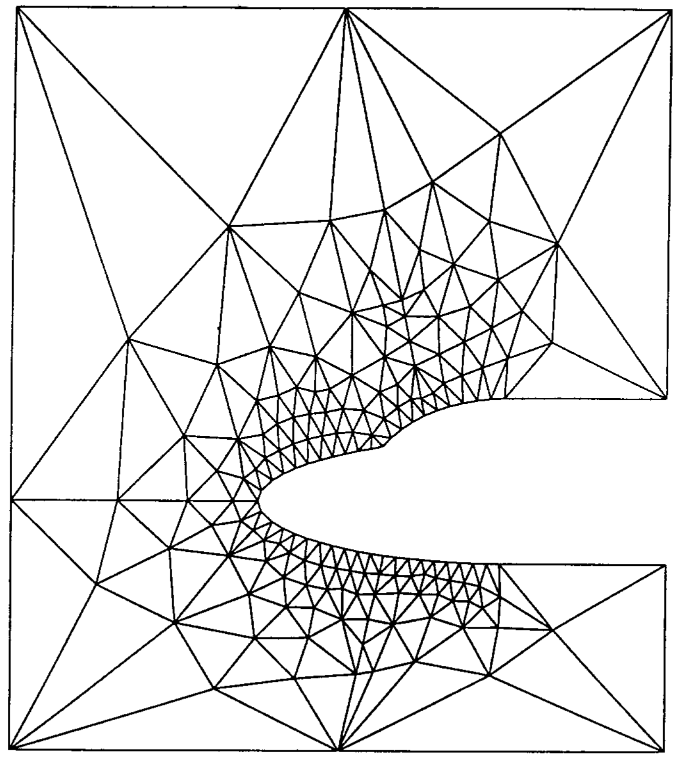

Figure 10: Four processors procedure (305 elements, 179 nodes).

takes longer spends 99 milliseconds (elapsed time) for 79 elements and 52 nodes.

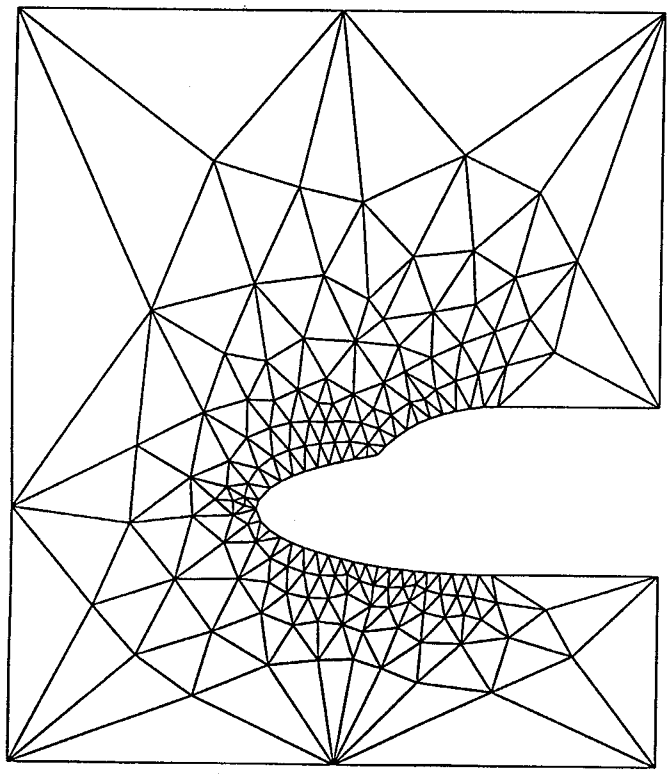

Figure 11: Smoothed mesh.

It is important to point out the great difference in time observed between both procedures, more than a second versus a little more than a fifth. This great difference between both procedures, bigger than what should be expected according to the previous examples, is mainly due to the generation of the boundary of the aircraft, more complicated than the generation in the square domain, what penalizes the procedure that lasts more.

It is important to emphasize this point, the increase of the penalization that represents not to parallelize the algorithms when the complexity of the problem is bigger.

Considering, besides, the tendency of the increasing time in function of the number of elements (what depends on the number of iterations required as well as the average number of tries required to find the host element for each point to be interpolated [20]) generated by the program, it is clear the enhancement represented by parallelization.

To carry out the efforts to continue creating parallel programs capable of creating balanced unstructured meshes, estimating efficiently the errors and quickly readapting the meshes to accurate the results seems to be the challenge for mesh generation.

References

[1] Löhner, R. 1990. Three-Dimensional Fluid-Structure Interaction Using a Finite Element Solver and Adaptive Remeshing. Computational Systems Engineering, 1: 257-272.

[2] Carey, G.F. 1989. Parallel Supercomputing Methods, Algorithms and Applications. Chichester, UK.

[3] Farhat, C. 1993. TOP/DOMDEC: A Software Tool for Mesh Partitioning and Parallel Processing. Center for Space Structures and Controls, CU-CSSC-93-11, Boulder, US.

[4] Farhat, C. 1993. Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics. International Journal for Numerical Methods in Engineering,36: 745-764.

[5] Globish, G. 1995. PARMESH - A Parallel Mesh Generator. Parallel Computing,21:509-524.

[6] Löhner, R., Camberos, J. and Merriam, M. 1992. Parallel Unstructured Grid Generation. Computer Methods in Applied Mechanics and Engineering,95:343-357.

[7] Peraire, J. 1986. A Finite Element Method for Convection Dominated Flows. Ph. D. Thesis, University College of Swansea, UK.

[8] Peraire, J., Morgan, K. and Peiró, J. 1989. Unstructured Finite Element Mesh Generation and Adaptive Procedures for CFD. AGARD FDP: Specialists' Meeting, Loen, Norway.

[9] Frykestig, J. 1994. Advancing Front Mesh Generation Techniques with Application to the Finite Element Method. Department of Structural Mechanics, Chalmers University of Technology, Göteborg, Sweden.

[10] Bugeda, G. 1990. Utilización de Técnicas de Estimación de Error y Generación Automática de Mallas en Procesos de Optimización Estructural. Ph. D. Thesis, Universitat Politècnica de Catalunya, Barcelona.

[11] Löhner, R. and Parikh, P. 1988. Generation fo Three-Dimensional Unstructured Grids by the Advancing Front Method.International Journal for Numerical Methods in Fluids,8:1135-1149.

[12] George, P.L. 1991. Automatic Mesh Generation. Application to the Finite Element Methods. INRIA, Institut National de Recherche en Informatique et en Automatique, Domaine de Voluceau, France:137-154.

[13] Farhat, C. 1988. A Simple and Efficient Automatic Finite Element Method Domain Decomposer. Computational Structures, Vol. 28:579-602.

[14] Flower, J., Otto, S. and Salama, M. 1987. Optimal Mapping of Irregular Finite Element Domains to Parallel Processors. Parallel Computations and Their Impact on Mechanics, The American Society of Mechanical Engineers, AMD, Vol. 86:239-252.

[15] Nour-Omid, B., Raefsky, A. and Lyzenga, G. 1987. Solving Finite Element Equations on Concurrent Computers. Parallel Computations and Their Impact on Mechanics, The American Society of Mechanical Engineers, AMD, Vol. 86:209-228.

[16] Vanderstraeten, D., Zone, O., Keunings. R, and Wolsey, L. 1993. Non-deterministic Heuristics for Automatic Domain Decomposition in Direct Parallel Finite Element Calculations. Parallel Processing for Scientific Computing, SIAM:929-932.

[17] Chan, W. and George, A. 1980. A Linear Time Implementation of the Reverse Cuthill McKee Algorithm. BIT, Vol. 20:8-14.

[18] Farhat, C. and Lesoinne, M. 1993. Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics. International Journal for Numerical Methods in Engineering, Vol. 36, No. 5:745-764.

[19] Flow over a Double Ellipsoid; Workshop on Hypersonic Flows for Reentry Problems, Antibes. 1990. INRIA, Institut National de Recherche en Informatique et en Automatique, Paris, France, Vol. 4:III.

[20] Löhner, R. 1994. Robust, Vectorized Interpolation Algorithms for Unstructured Grids. GMU/CSI, the George Mason University Publications, Fairfax, US.

Table 1. Number of elements and number of nodes for models BIG1, BIG2 and BIG5.

	BIG 1		BIG 2		BIG 3	
	Elements	Nodes	Elements	Nodes	Elements	Nodes
Seqtial.	48	38	182	115	1028	570
4 Procs.	67	50	201	126	1036	574
16 Procs.	77	53	222	138	1088	600

Table 2. Number of elements and number of nodes for models SMALL1, SMALL2 and SMALL5.

	SMALL1		SMALL2		SMALL5	
	Elements	Nodes	Elements	Nodes	Elements	Nodes
Seqtial.	4097	2159	16160	8297	101961	8297
4 Procs.	4026	2124	16099	8268	99657	49893
16 Procs.	4053	2138	15874	8156	102916	53253

Table 3. CPU and elapsed time -msec- for models BIG1, BIG2 and BIG5.

	BIG 1		BIG 2		BIG 5	
	CPU	Elapsed	CPU	Elapsed	CPU	Elapsed
Seqtial.	82		456		5015	
4 Procs.	71	24	133	47	789	293
16 Procs.	81	6	185	17	688	80

Table 4. CPU and elapsed time -msec- for models SMALL1, SMALL2 and SMALL5.

	SMALL1		SMALL2		SMALL5	
	CPU	Elapsed	CPU	Elapsed	CPU	Elapsed
Seqtial.	33566		236735		3530231	
4 Procs.	5370	1865	33399	12946	458764	172490
16 Procs.	3324	440	17346	2448	235843	35026

Table 5. Ratios between the CPU times for the different procedures in all models.

	BIG1	BIG2	BIG5	SMALL1	SMALL2	SMALL5
Seqtial./4 Procs.	1.155	3.429	6.356	6.251	7.088	9.132
4 Procs./16 Procs.	0.877	0.719	1.146	1.616	1.925	1.945
Seqtial./16 Procs.	1.012	2.465	7.289	10.101	13.648	17.763

Table 6. Ratios between the CPU times for the different procedures in all models.

	BIG1	BIG2	BIG5	SMALL1	SMALL2	SMALL5
Seqtial./4 Procs.	3.417	9.702	17.116	17.898	18.286	20.466
4 Procs./16 Procs.	4.	2.765	3.662	4.238	5.288	4.925
Seqtial./16 Procs.	13.667	26.823	62.725	76.286	96.705	100.789