European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS 2000
Barcelona, 11-14 September 2000

PARALLEL, THREE-DIMENSIONAL FINITE ELEMENT MESH GENERATION BASED ON OCTREE DATA STRUCTURES

[1,2]A.J Barragán*, [1]J. S. Reeve

[1]Electronics and Computer Science Department, Southampton University, Southampton, UK

[2]Escuela de Ingeniería Mecánica, Universidad Central de Venezuela, Caracas, Venezuela
Email: ajbg96r@ecs.soton.ac.uk

Key words: Octree, Mesh Generation, Finite Element, Parallel Computing, Message Passing, Linux, Cluster computing.

Abstract. *This paper describes the implementation of a three-dimensional mesh generation program based on octree data structures, capable of generating large meshes, operating in a parallel, distributed memory computing under the Single-Program, Multiple-Data model of computation. In order to compensate for the limitations in the standard octree technique, some enhancements to it are developed and implemented, based on the authors' previous work[12] for two-dimensional problems. Also, communication routines are developed for the exchange of information among processors. Using both the enhanced data structures and the communication routines developed, algorithms are then created for performing the octree mesh generation. Finally, some test runs are carried out on sample meshes in a cluster of Linux workstations using the MPICH library and GNU C compiler, and graphs of computing time vs number of processors are obtained.*

1 INTRODUCTION

Finite element mesh generation has reached a respectable degree of maturity since its beginnings, having evolved from just manual procedures for feeding input data to finite element programs, to becoming integral part of the solution process in which the design of the mesh is linked to the goodness of the finite element solution through error indicators, in iterative process called adaptivity. Nowadays, mesh generation is also linked to sophisticated methods for representing complex three-dimensional models in an attempt to wholly automate the design-analysis process.

An important requirement is the need for bigger meshes. While it may be argued that adaptivity aims at refining a given mesh only where it is required and thus achieve an "economical" mesh, the very nature of the finite element method states that as the number of elements grows (appropriately), the behaviour of the discretised model will approach (asymptotically) the behaviour of the exact solution. On the other hand, extremely complex geometries, may require a very high level of resolution to be properly represented, which again calls for very large numbers of elements.

More recently, the availability of high performance computing (HPC) has opened the door to the use of bigger meshes. This may allow for more accurate geometric representations in some cases, and more precise iterative methods in others. HPC has become more available for two main reasons: On one side, the increasing ratio performance/cost in hardware, and in the other, the availability of parallel computing, once the domain of highly specialised (and equally expensive) hardware, now with the existence of programming tools such as the Message Passing Interface Library (MPI), has been made portable to environments such as networks of workstations or PCs, making such platforms appealing to those interested on solving this type of problems.

2 RELATED WORK

Spatial subdivision techniques, pioneered for finite element mesh generation by Yerry and Shephard[1,2], require its own approach to parallelism, as discussed in Simone and co-workers[10,11]. They discuss on tools and techniques for handling octree-based mesh generation in a distributed computing environment. Issues addressed are load balancing procedures based on the partitioning of the data and the migration of the octants from highly loaded to lightly loaded processors.

Saxena and Perucchio[4] give definitions of algorithmic parallelism in automatic meshing in the context of Recursive Spatial Division (RSD). They argue that the properties of RSD can be exploited to design a meshing algorithm capable of operating in parallel processing environments. They present a definition of a meshing operator for parallel processing and also algorithms for various stages of the automatic meshing.

Lohner, Canberros, and Merriam[3] developed a system for parallel unstructured grid generation based on the background mesh concept, using a relatively coarse starting mesh whose elements are distributed to the processors for further meshing. Likewise, Hodgson and Jimack[9] use also the same approach in which elements of the background mesh are assigned to each processor to be meshed individually. Load balancing is accomplished by estimating the number of vertices that will be generated within each background element, and splitting the mesh accordingly.

deCougny et al[5] give some details of a parallel mesh generator based on the RSD approach in a MIMD environment. The approach used for load balancing is a bisection-based algorithm that splits portions of the data according to geometric co-ordinates. The main data structures are the mesh and the tree, which must be built initially in a single processor to be distributed afterwards over all the processors.

Vidwans et al[7] develop an algorithm for load balancing three-dimensional unstructured meshes in adaptive processes for fluid dynamics. Their approach employs migration of cells from heavily loaded processors to less loaded neighbouring processors. Later, Globisch[8] presents a program for the parallel generation of tetrahedral meshes in bodies with symmetry of revolution. The strategy is based on the initial decomposition of the region into simply connected sub-domains and then scattering them among the processors.

3 DESCRIPTION OF THE TECHNIQUE

To generate a mesh with this technique, first an octree representation is obtained of the region of interest. As a result of this step, the shape of the region is approximated by a set of cubic regions of varying size called octants, stored in a eight-node tree data structure. In what follows, the octants will be termed “nodes”, to use the tree nomenclature. Nodes that have been divided are termed “branch” and those not divided “leaf” ones.

As the tree is created, the subdivision is applied only to the nodes that intersect the boundary of the region (ON nodes). Unless otherwise specified, the inside nodes are not subdivided to the same level, resulting in adjacent nodes of different size. As a consequence, a given node may have one or more of its corners lying along the sides/faces of an adjacent one. From a finite element point of view, the location of a neighbour’s corner along the side of a node must meet some limitations, so a maximum difference of level equal to unity is usually enforced. As a consequence, location of intermediate nodes will be at the middle of a side/face, which is allowed.

The following step is to transform the tree structure in such a way to make the ON nodes conform to the boundaries. To achieve this, they have to be modified to make them “fit” to the boundaries of the region. Figure 3.1 shows a two-dimensional region and its tree subdivision.

4 IMPLEMENTATION

The parallelisation strategy chosen in this work is based in the master-worker paradigm, which in turn is based in a Single-Program, Multiple-Data (SPMD) model of computation. This will allow the resulting program to be run in a variety of computing platforms ranging from multiple-processor machines to clusters of workstations which could be of heterogeneous type. In this type of computing environment, interprocessor communication can be very expensive, and as a consequence, a desirable feature of the software to be developed is that such communications are reduced as much as possible.

The main feature of the parallelisation based on the master-worker model of programming is its impact on the design of the program. Initially, the program has been based on recursive procedures which in turn are based on a data structure that in itself is recursive, as is the octree.

The standard octree technique relies on the use of dynamic memory allocation and being pointer-based, requires absolute memory addresses. These, being referred to a machine’s memory, are local by nature and lose all of its meaning when sent out of the machine. After an investigation into the nature and the types of data that can be handled by the MPI standard, it was concluded that an adequate message passing could be achieved using static data structures, that is, arrays. This led to the definition of the main data structures for the parallel program which are based in the octant as the fundamental type, but now arrays of such structures are used in addition to the tree. This way, the tree is stored in an array of octants.

This array grows as the subdivision proceeds until the resolution level is reached. That is, is a dynamic data structure, whose growth comes from the new IN or ON sons that are added to it. The role of the master processor is to create an initial array, and to split this array into sub-arrays, which are sent to the worker processors. This way, every processor works on a local queue where the more computationally intensive operations, (e.g., classifying every new octant against the model boundary) are carried out.

Key to the successful implementation of the software is the identification of those steps in the mesh generation process in which some kind of geometric search is required. This is so because in those cases it is very likely that some amount of inter-processor communications will become unavoidable, and it will have to be handled with the proper care.

Those stages of mesh generation that require geometric search have been identified to occur during neighbour finding and during octant modification. In the first case, the neighbours of each boundary octant have to be located. The second case arises when the boundary octants are being fitted to the boundary, and nodes have to be moved, created or deleted. These operations involve locating adjacent octants, which may not have been allocated in the same processor.

The above mentioned poses the need for a data structure that can reduce the inter-processor communications to a minimum in view of the limitations of the SPMD model used. In the present work, this situation as been addressed by the use of procedures that involve sorting of the queue created in the first stages, combined with an allocation strategy that splits the queue according to a histogram. First, the queue is sorted according to ascending values of the geometric coordinates of one of the octant corners. (Fig 4.1)

Next, a histogram is created from these coordinate values, which allows the queue to be split among the processors according to discrete ranges of the coordinates (Fig. 4.2). This may create a slight imbalance in the number of octants allocated to each processor, but on the other hand, it does produce an organisation of the data that reduces the amount of inter-processor communications.

The combination of sorting plus splitting according to the histogram produces a situation in which each processor receives a sorted sub-queue in which only those octants “close” to the ends will be likely to have adjacent octants in another processor’s local queue, but by virtue of the sorted nature of the local queues, it will only be in those processors containing the immediately adjacent queues.

As a consequence of all this, a topology is induced among the processors, which mirrors closely the distribution of octants in the sorted queues. That is, each processor will only require communications with its two adjacent neighbouring processors (Fig 4.3).

The adjustment of differences of level between adjacent octants proceeds according to the following considerations: As the octants are divided to different levels, there can result in adjacent octants having an arbitrary difference of level between them. From the finite-element point of view this can lead to unacceptable configurations, since intermediate nodes along an arbitrary position on element edges sides might appear, thus violating rules about the positioning of mid-side nodes or requiring formulation of non-standard interpolation functions.

To prevent this from happening, steps must be taken to insure that the difference of level is at most unity. This is achieved by locating the neighbours of each leaf node. For each of such neighbours, if the difference of levels exceeds unity, the bigger octant is divided. This division creates new sons that must be added to the data structure and which in turn must be checked for difference of levels. The process is repeated until no more new octants are created.

Key for the adjustment of differences of level is the technique used for locating the octants adjacent to a given one. In a sequential, pointer-based octree this is achieved by traversing the father-son links. In a parallel, distributed memory environment, the lack of absolute memory addresses makes that not possible and alternative methods must be used. In this work, use is made of the spatially sorted nature of the arrays of octants (as mentioned above) to reduce the search space for neighbours, thereby reducing the amount of inter-processor communication.

5 RESULTS AND DISCUSSION

To assess the performance of the program, test runs were made on two different geometries, which are shown in the figures 5.1 and 5.2. The computing environment is a cluster made of commodity components, consisting of 8 single processor Athlon (AMD-K7) machines running Linux Red Hat 6.0. The system has an aggregate peak performance of 900 MegaFlops, with 2 GigaBytes of RAM memory and 100 GigaBytes of disk space. The programming tools are the MPICH software on top of the GNU C compiler.

For the examples the subject of the tests was to see how the required time was influenced by the number of processors, and so a number of runs were done for each one with a number of processors ranging from three to the full eight processors available. The results are shown in the graph of figure 5.3.

From the curves shown, it can be seen that there is a marked decrease in time as the number of processors grows, as it was expected. Also is important to note that the shape of the curves is roughly the same, although example 2 displays a steeper decrease in required time in the low range of processors. Another feature worth of mention is the decrease in the slopes of the curves as the number of the processors increases. It is also worthy of notice how the balance of load per processor, that is, the ratio computation/communication ranges from predominant computation for low numbers of processors, to predominant communication as the number of processors grows and the computing load per processor becomes lighter.

REFERENCES

[1] M.A. Yerry and M.S. Shephard, "Automatic mesh generation for three dimensional solids" *Computers & Structures*, **20,** 31-39 (1985).

[2] M. S. Shephard and M.K. Georges "Automatic three-dimensional mesh generation by the finite octree technique". *IJNME*, **32** ,704-749 (1991).

[3] R. Lohner, J. Camberros, and M. Merriam, "Parallel unstructured grid generation", *CMAME*, **95**, 343-357 (1992).

[4] M. Saxena and R. Perucchio, "Parallel FEM algorithms based on recursive spatial decomposition - I. Automatic mesh generation", *Computers & Structures*, **45**, 817-831 (1992).

[5] H.L. de Cougny, M.S. Shephard and C. Ozturan, "Parallel three-dimensional mesh generation", *Computing Systems in Engineering*, **5**, 311-323 (1994).

[6] W. Gropp, E. Lusk, and E. A. Skjellum, *Using MPI: Portable Parallel programming with the Message Passing Interface*, The MIT Press (1994).

[7] A. Vidwans, and Y. Kallinderis, "Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids", *AIAA Journal*, **32**, 497-505 (1994)

[8] G. Globisch, "On an automatically parallel generation technique for tetrahedral meshes", *Parallel Computing*, **21**, 1979-1995 (1995)

[9] D.C. Hodgson, and P.K. Jimack, "Efficient parallel generation of partitioned, unstructured meshes", *Advances in Engineering Software* **27** , 59-70 (1996)

[10] M.L. Simone, R.M. Loy, M.S. Shephard, and J.E. Flaherty, "A Distributed Octree and its application to Parallel Mesh Generation", *SCOREC Report #23-1996*, Scientific Computation Research Center, Rensselaer Polythecnic Institute, Troy, NY, USA (1996)

[11] M.L. Simone, H. de Cougny and M.S. Shephard, "Tools and Techniques for parallel Grid Generation", *SCOREC Report #5-1996*, Scientific Computation Research Center, Rensselaer Polythecnic Institute, Troy, NY, USA (1996).

[12] A.J. Barragán and J.S. Reeve, "A Two-Dimensional parallel quadtree finite element mesh generator". Parallel Computing 99 Conference, Delft, Holland, August 17-20 (1999).

ACKNOWLEDGEMENTS

The first author wishes to express his gratitude to the Consejo de Desarrollo Cientifico y Humanistico of the Central University of Venezuela, whose support for the realization of his PhD has contributed to the present paper. Also, the cooperation of Mr Panagiotis Melas in the setting up and use of the hardware is highly regarded.

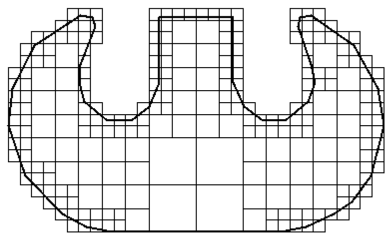

Figure 3.1. a 2-D region and its tree representation

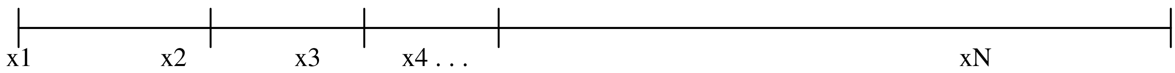

Figure 4.1 Sorting of the data structure according to ascending values of x-coordinates

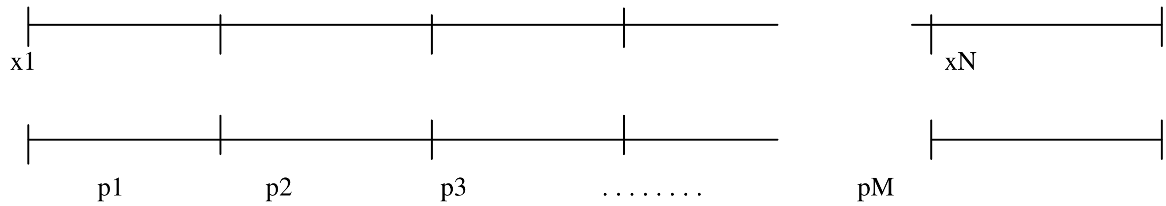

Figure 4.2 Allocation of the data structure to processors p1 to pM

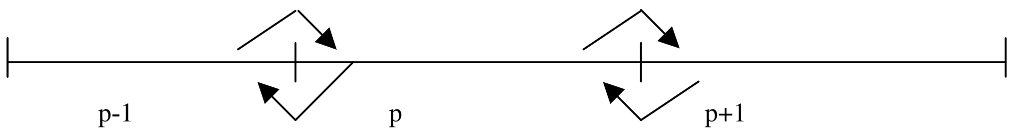

Figure 4.3 Exchange of information between a processor “p” and its adjacent processors

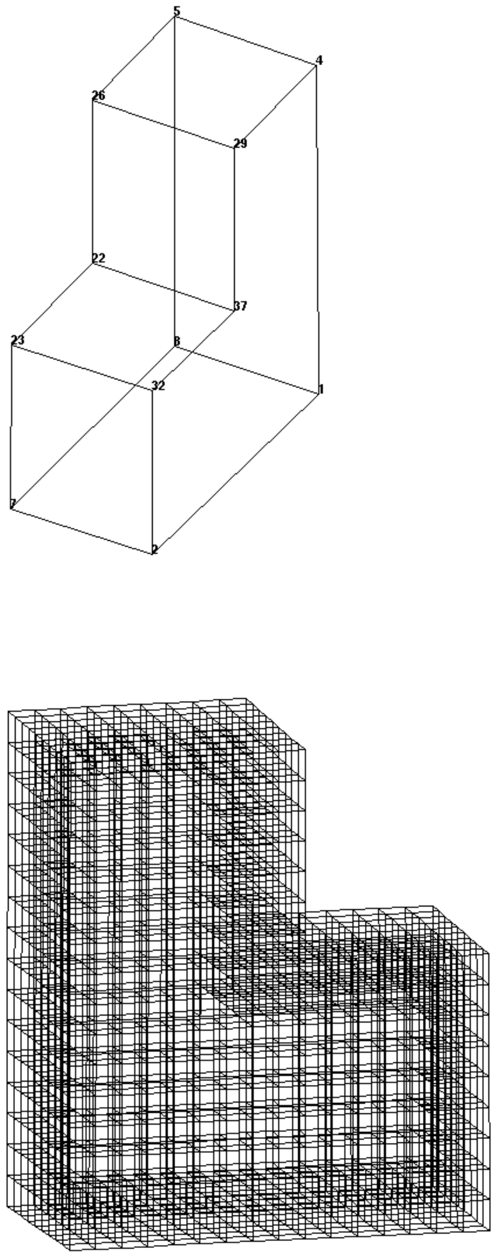

Figure 5.1 Example 1

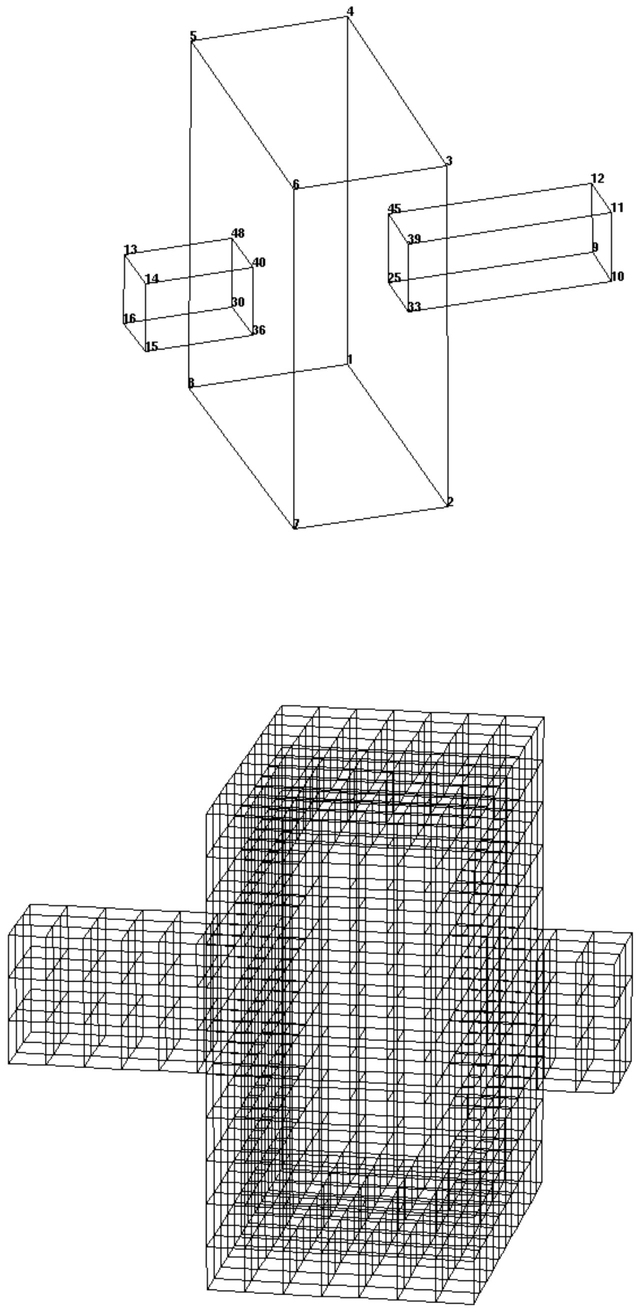

Figure 5.2 Example 2

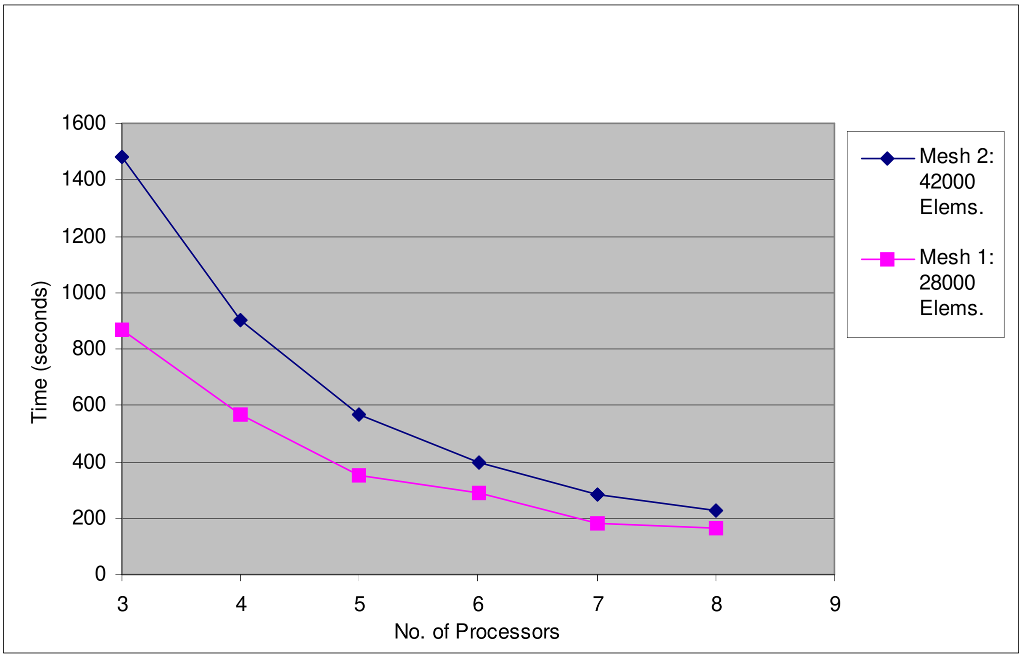

Figure 5.3 Speed-Up graphs. Runtime vs Number of Processors.