A TWO-DIMENSIONAL PARALLEL QUADTREE FINITE ELEMENT MESH GENERATOR

A.J. BARRAGÁN*, J.S. REEVE

Department of Electronics and Computer Science, Southampton University, Southampton, SO17 1BJ, Hants, UK

**Corresponding author, email: ajbg96r@ecs.soton.ac.uk*

In this work we implement a parallel finite element mesh generator for two-dimensional regions, based on recursive spatial division techniques. The main formulation is based on spatial recursive decomposition techniques (RSD) combined with the MPI library. A data structure has been developed aimed at the reduction of inter-processor communication. Results are presented for generation of meshes with a varying number of processors. The extension to the work to 3D regions is also considered.

1 INTRODUCTION

Finite element mesh generation has come a long way since its beginnings, having evolved from being manual procedures for feeding data to finite element programs, to becoming integral part of the solution process. Nowadays, mesh generation is also linked to sophisticated methods for representing complex three-dimensional geometries in an attempt to wholly automate the design-analysis process. Recently, the availability of high performance computing (HPC) has opened the door to the use of more refined 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 increased availability of programming tools such as the Message Passing Interface Library (MPI), that have lightened the task of writing parallel programs for different computing environments.

2 RELATED WORK

Saxena and Perucchio [4] give definitions of algorithmic parallelism in automatic meshing. They argue that the properties of recursive spatial decomposition (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.

Spatial subdivision techniques (see [1,2] and references therein) require its own approach to parallelism, as in Simone et al [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.

Vidwans and co-workers [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

The use of recursive spatial division for finite element mesh generation has been pioneered by Yerry and Shephard [1]. To generate a mesh with this technique, first a quadtree/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 regions of varying size, stored in a tree-like structure. The shape of these regions is square, in 2D problems, and cubic, in 3D. This has given place to the terms “modified quadtree/octree”. In what follows, they will be termed “nodes”, to use the tree nomenclature. Nodes that have been divided are termed “branch” and those not divided “leaf” ones. (See fig. 1 for a two-dimensional example)

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 into a finite element mesh 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 1 shows both a two-dimensional region and its quadtree subdivision.

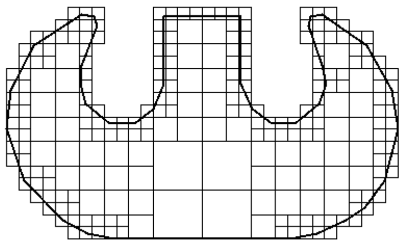

Figure 1. a region and its quadtree representation

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, using the MPI library [6]. 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 data structures based on dynamic memory allocation and being pointer-based, require absolute memory addresses, which are local by nature and lose 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.

This led to the definition of the main data structures for the parallel program which are based in the quadrant as the fundamental type, but now, the tree is stored in an array of structures (quadrants). The role of the master processor is to create the initial array, and then to split it into sub-arrays, which are sent to the worker processors. Thereafter, every processor works on a local queue where the more computationally intensive operations are carried out.

A key issue in the mesh generation process is the identification of those steps in which is required to locate the quadrants adjacent to a given one, because it is very likely that at least some of them may have been allocated to another processor, and consequently, some amount of inter-processor communications will become unavoidable. This situation has been addressed in the present work by sorting of the queue created in the first stages (Figure 2), combined with a splitting of the array (Figure 3) according to discrete ranges of the values of the x coordinate.

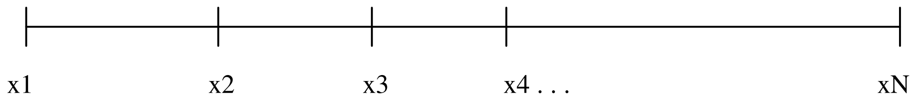

Figure 2 Sorting of the data structure according to ascending values of x-coordinates (increasing from left to right)

This may create a slight imbalance in the number of quadrants allocated to each processor, but on the other hand, it does produce an organisation of the data that reduces the amount of interprocessor communications.

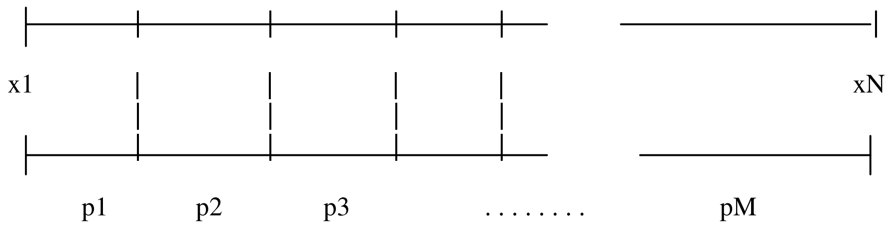

Figure 3. Allocation of the data structure to processors p1 to pM

As a consequence, each processor receives a sorted sub-queue in which only those quadrants “close” to the ends will be likely to have adjacent quadrants 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 (Fig 4).

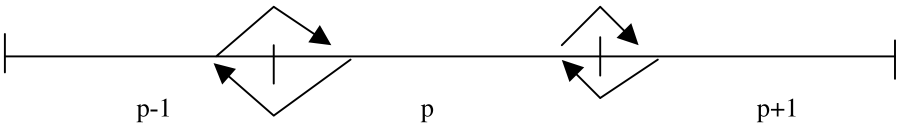

Figure 4. Exchange of information between a processor “p” and its adjacent processors “p-1” and “p+1”. The arrows show the flow of information.

As a consequence of all this, a topology is induced among the processors, according to which each processor will only require communications with its two adjacent neighbouring processors (Fig 4).

5 RESULTS AND DISCUSSION

Several environments were used in the development of the software: In the first stages a SPARC machine with simulated distributed memory and where up to eight processors can be simulated was used. Another machine used as a following step was an Meiko CS2, which is also limited to a maximum of eight processors, and finally, the definitive runs were made on an IBM SP2. This machine consists of 23 nodes, having each one a single 66MHz RS6000 processor.
To assess the performance of the program, test runs were made on three different geometries, with a varying number of boundary points ranging from 12 to 237 and, as it can be seen, their complexity varies accordingly (See figs. 5a-5c). Sample meshes for each region are shown in Figure 7.1 to 7.3.

For all the examples the program was timed to obtain the speed-up as a function of the number of processors, and so a number of runs were done for each example with a number of processors ranging from 7 to 20, and the results are shown in figure 6, where each curve corresponds to each one of the example regions.

From the curves shown, it can be seen 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 3 displays a steeper decrease in required time in the range of processors from seven to around 12. This could make worthwhile to try this region of low number of processors with bigger meshes. Another feature worth of mention is the flattening of the curves as the number of the processors grows to the maximum.

6 FURTHER WORK

The results obtained are considered to be encouraging to extend the scope of the present work consists of the in order to generate three-dimensional meshes. One of the advantages of the tree-based data structures is that the concepts employed in the generation of two-dimensional meshes can be directly applied to three-dimensional ones. However, special care must be given to some critical issues that arise when handling 3D models, basically due to the inherently superior complexity that such models convey.

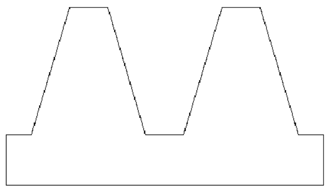

Figure 5.a. Example 1

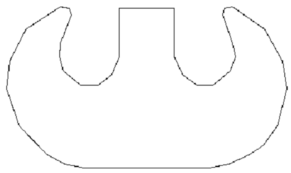

Figure 5-b. Example 2

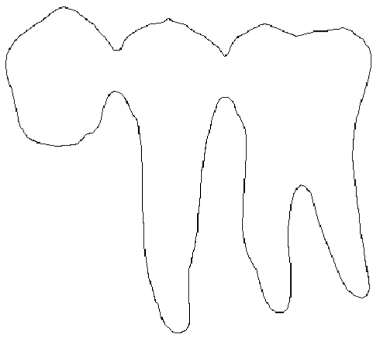

Figure 5-c. Example 3

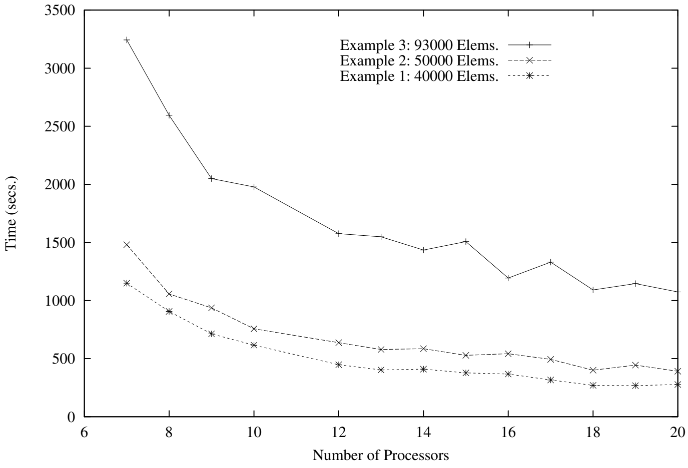

Figure 6. Runtime vs no. of processors

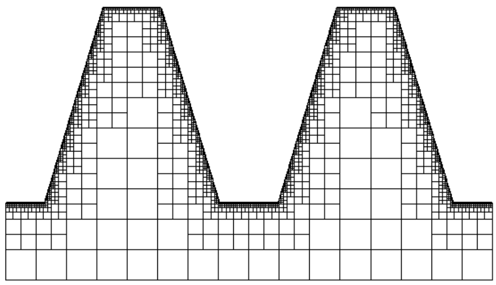

Figure 7.1. Mesh for example 1

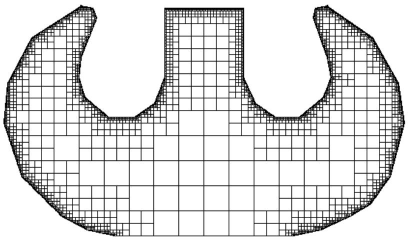

Figure 7.2. Mesh for example 2

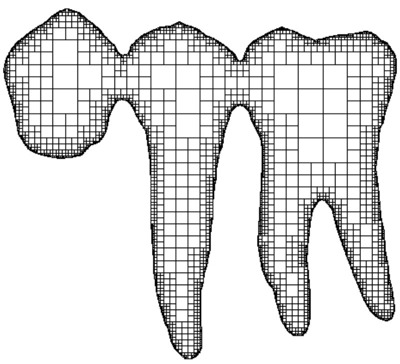

Figure 7.3. Mesh for example 3

REFERENCES

1. Yerry, M.A.; Shephard, M.S.: Automatic mesh generation for three-dimensional solids. Computers & Structures Vol 20 No. 1-3 (1985) pp 31-39.
2. Shephard, M.S.; Georges, M.K.; Automatic three-dimensional mesh generation by the finite octree technique. IJNME Vol 32 (1991) pp 704-749.
3. Lohner, R.; Camberros, J.; Merriam, M.: Parallel unstructured grid generation. CMAME, V95 (1992) pp 343-357.
4. Saxena, M.; Perucchio, R.: Parallel FEM algorithms based on recursive spatial decomposition-I. Automatic mesh generation. Computers & Structures, Vol. 45, No. 5/6 (1992), pp 817-831.
5. de Cougny, H.L.; Shephard, M.S.; Ozturan, C.: Parallel three-dimensional mesh generation. Computing Systems in Engineering, Vol 5, No. 4-6 (1994), pp 311-323.
6. Grop, W.; Lusk, E.; Skjellum, A.: Using MPI: Portable Parallel programming with the Message Passing Interface. The MIT Press (1994).
7. Vidwans, A.; Kallinderis, Y.: Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. AIAA Journal, V. 32 No 3 (1994), pp 497-505.
8. Globisch, G.: On an automatically parallel generation technique for tetrahedral meshes. Parallel Computing, V 21 (1995), pp 1979-1995.
9. Hodgson, D.C.; Jimack, P.K.: Efficient parallel generation of partitioned, unstructured meshes. AES Vol. 27 (1996) pp 59-70.
10. Simone, M.L.; Loy, R. M.; Shephard, M.S.; Flaherty, J.E.: A Distributed Octree and its application to Parallel Mesh Generation. SCOREC Report #23-1996, Scientific Computation Research Center, Rensselaer Polythecnic Institute, Troy, NY, USA.
11. Simone, M.L.; de Cougny, H.; Shephard, M.S.: Tools and Techniques for parallel Grid Generation. SCOREC Report #5-

1996, Scientific Computation Research Center, Rensselaer Polythecnic Institute, Troy, NY, USA.

ACKNOWLEDGEMENTS

The first author acknowledges the support of 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.