Towards an Efficient Distributed Geometry for Parallel Mesh Generation

David Martineau, Jeremy Gould, and Jacques Papper

ICON Technology & Process Consulting Ltd., Berkshire House, Thames Side, Windsor, UK.
[d.martineau j.gould j.papper]@iconcfd.com

Abstract. This paper discusses the implementation of a distributed geometry for parallel mesh generation, involving dynamic load-balancing and hence dynamic re-partitioning of the geometry. A novel approach is described for improving the efficiency of the distributed geometry interface when dealing with irregular shaped mesh partitions.

1 Introduction

Whilst efficient parallel flow solvers are now commonly available, the parallelization of the mesh generation process still represents a considerable challenge [1]. In industrial applications, it is often the mesh generation task which therefore limits the scale of the problem that can be simulated.

Attempts to perform mesh generation in parallel generally decompose the original problem into N smaller problems which can then be meshed concurrently using P processors or threads [2]. Parallel mesh generation strategies can be categorized based on the degree of coupling between sub-problems, ranging from completely decoupled [3], to tightly coupled [4].

Decoupled, or loosely coupled techniques are attractive in terms of simplicity, however they present problems in terms of load-balancing, mesh quality at domain boundaries, and the ability to generate contiguous viscous layers throughout the volume. Conversely, tightly coupled techniques are more complicated to implement, but enable dynamic load-balancing. However, the use of dynamic load-balancing presents problems for the interface to the geometry. Storing a copy of the geometry on each processor is one solution, but presents a serious limitation on the size of geometry which can be meshed. The focus of this paper, therefore, is on the implementation of a distributed geometry interface with support for dynamic load-balancing during parallel mesh generation.

2 Parallel Mesh Generation Process

The parallel hexahedral-dominant mesh generator, iconHexMesh, is part of the iconCFD process, an open source-based CFD suite developed by ICON, using OpenFOAM® technology. The mesh generation process in iconHexMesh is illustrated in Fig. 1, and involves refinement of the initial block mesh based on prescribed refinement levels on geometry surfaces and feature lines, creation of a conformal mesh by snapping to geometry surfaces, and insertion of a layer mesh to capture viscous boundary layers. At various points during the mesh generation, the mesh is dynamically re-partitioned to achieve good load-balancing.

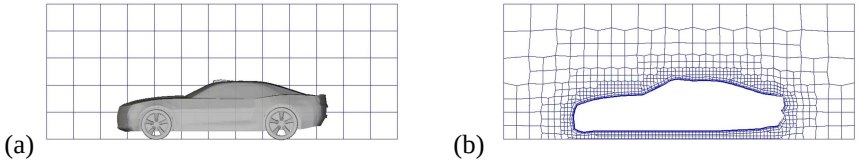

Fig. 1. Mesh generation process in iconHexMesh: (a) input STL geometry and initial block mesh, (b) final mesh.

3 Distributed Geometry Implementation

When performing mesh generation in parallel with a distributed geometry, the geometric surfaces must be re-distributed each time the mesh is re-partitioned, so that each mesh partition has access to the local portion of the geometry. Feature lines, which consist of a series of contiguous edge segments that represent geometric discontinuities, surface intersections or boundaries of user-specified regions, must also be re-distributed when the mesh is re-balanced.

For the re-distribution of geometry surfaces, OpenFOAM® provides the class distributedTriSurfaceMesh, which extends the class triSurfaceMesh normally used to store STL geometry by adding a method to distribute the surface mesh. This is accomplished by exchanging a series of bounding boxes enclosing each of the mesh partitions. Each distributed surface then checks whether any triangles overlap these bounding boxes, and therefore need to be communicated to another processor. Any new triangles received from other processors are then merged with the existing surface mesh, removing any duplicate points and triangles in the process.

However, the redistribution of the geometry is only one aspect of a distributed geometry interface. When the geometry is partitioned, geometric queries have to be handled differently, as illustrated in Fig. 2 for the process of finding the intersection of a line with the geometry.

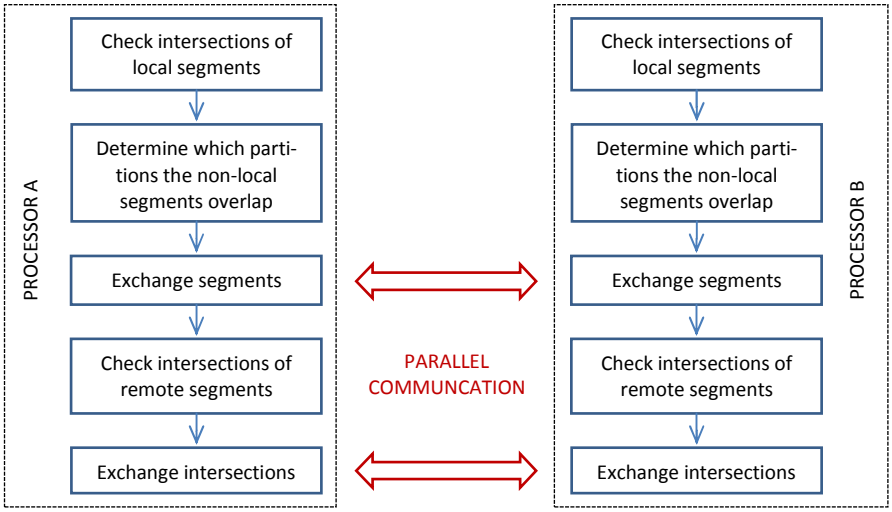

Fig. 2. Distributed geometry implementation of line intersection checks

As a result, it was necessary to modify several parts of the meshing algorithm to avoid single point or edge geometry queries, and replace these with queries for groups of points or line segments which could then be performed synchronously across all processors.

With the modifications described above, it is now possible to generate meshes on models of much greater complexity, as demonstrated by Fig. 3, which shows a detail of the 20M cell mesh generated in 20 minutes on 128 processors for a civil aircraft model comprising of a 1.8GB STL file.

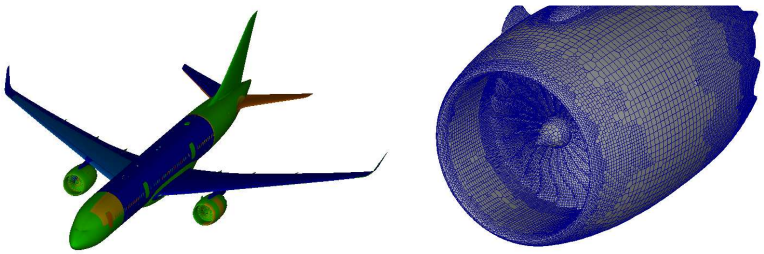

Fig. 3. Geometry (left) and detail of mesh (right) generated on civil aircraft

4 Support for Different Decomposition Methods

The mesh generator iconHexMesh supports various decomposition methods: *simple* or *hierarchical* geometric decomposition methods in which the domain is split by planar cuts in the x, y and z directions, or *ptscotch*, which uses the PT-Scotch graph partitioning library. Although the hierarchical method is most commonly used during parallel mesh generation, the ptscotch method is attractive in terms of providing good load-balancing whilst minimizing communication boundaries.

However, when moving to ptscotch, the speed of the refinement and snapping stages were reduced, because of a reduction in the locality of the geometry queries. The bounding boxes associated with each partition overlap more than those resulting from hierarchical decomposition, causing an increase in the exchange of data during re-distribution, and in the number of non-local geometry queries.

In order to address the problems associated with irregular shaped mesh partitions, the distributed geometry interface provides the ability to describe mesh partitions using multiple bounding boxes. The problem becomes one of determining a minimal set of bounding boxes which will best describe the shape of the mesh partition on each processor.

This problem was solved by creating an octree for the boundary of the mesh partition, and extracting a list of bounding boxes which correspond to the octree at a specified depth. This results in a set of bounding boxes which match the shape of the mesh partition as closely as possible, subject to a user-defined level, as illustrated in Fig. 4. Using multiple bounding boxes to approximate the mesh partitions results in a decrease in the number of triangles exchanged during geometry re-distribution, and improved localization of the geometry queries.

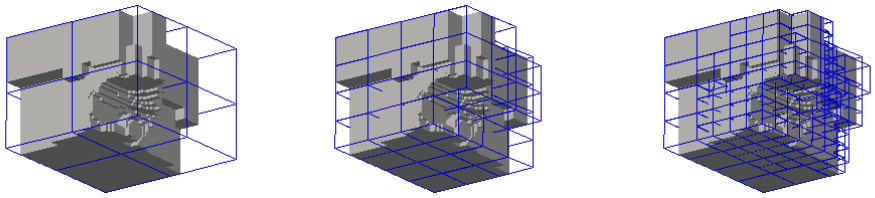

Fig. 4. Multiple bounding boxes used to approximate a mesh partition

5 Future Work

Whilst the use of multiple bounding boxes to approximate a mesh partition offers performance enhancements, it could be further improved by communicating the structure of the octree of the mesh boundary to the specified level, instead of an explicit list of bounding boxes. The octree for each partition could then be re-constructed on remote processors from the root bounding box and a series of markers indicating whether each child node should be included, excluded, or further sub-divided. This would then enable a more efficient octree search to be performed to check for overlapping parts of the geometry.

6 Conclusion

A parallel mesh generation tool has been modified to support a distributed geometry with dynamic load-balancing, enabling generation of meshes for extremely large geometry assemblies.

A novel approach to improving the efficiency of the distributed geometry interface has been implemented for dealing with irregular shaped mesh partitions. Efforts are underway to further improve the efficiency of this approach, and enable better performance scalability.

References

1. T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K-L Ma, and D. R. O'Hallaron. 2006. From mesh generation to scientific visualization: an end-to-end approach to parallel supercomputing. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, New York, NY, USA, Article 91.
2. Nikos Chrisochoides, Parallel Mesh Generation, Chapter in *Numerical Solution of Partial Differential Equations on Parallel Computers*, Springer-Verlag, pp 237-259, 2005.
3. J. Galtier and P. L. George. Prepartitioning as a way to mesh subdomains in parallel. *Special Symposium on Trends in Unstructured Mesh Generation*, pp 107--122. ASME/ASCE/SES, 1997.
4. W. N. Dawes, S. A. Harvey, S. Fellows, N. Eccles, D. Jaeggi, and W. P Kellar. A practical demonstration of scalable, parallel mesh generation. In *47th AIAA Aerospace Sciences Meeting & Exhibit*, 2009, AIAA-2009-0981.