Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 00 (2016) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable

"Parallel Uniform Mesh Refinement Geometry Association Scalability"

"Brian Carnes, Madison Brewer" *

Sandia National Laboratories, PO Box 5800, Albuquerque, NM 87185, USA
University of Houston – Clear Lake, 2700 Bay Area Blvd, Houston, TX 77058

Abstract

This paper concerns mesh refinement in parallel. Particular topics include: the scalability of reading and associating a complex mesh and geometry, and a cursory explanation of our implementation details. The following implementation utilizes the Sierra ToolKit mesh environment and ACIS geometry. Scalability testing revealed that while the general refinement process scaled quite well, certain key pre-refinement processes have negative relationship with mesh size and processor count.

Peer-review under responsibility of the organizing committee of IMR 25.

Keywords: "refinement; association; parallel; geometry; decomposition"

1. Introduction

1.1 Need for Uniform Mesh Refinement in Parallel

The majority of meshing algorithms are, by nature, inherently serial. Therefore, meshing a geometry coarsely in serial then scaling its elements in parallel is often used to speed the mesh generation process. However, merely refining the mesh doesn't necessarily yield the desired results. Refinement on a coarse mesh will create a fine mesh that still only roughly approximates its associated geometry. Thus, a process of refining to geometry becomes necessary to ensure the resulting mesh approximates its geometry as closely as possible.

Determining which geometric entities should be associated with which portions of mesh is a key issue in refining to a given geometry. Without geometric associativity, ensuring that new nodes created in the process of refinement are moved an appropriate geometric entity is nearly impossible. The following implementation builds off of existing

* Corresponding author. Tel.: +1-713-822-8343; fax: N/A.
E-mail address: mbrewer1337@gmail.com

Peer-review under responsibility of the organizing committee of IMR 25.

serial methods for geometry association and parallel communication for uniform mesh refinement (UMR) in order to optimally associate a given mesh in parallel then refine it to the associated geometry.

1.2 Association Requirements and the STK Mesh Environment

Refining to geometry requires three primary inputs, a mesh, its associated geometry, and finally, the associativity information between the two. This particular implementation concerns an unstructured finite element mesh (ExodusII format), a CAD geometry file (ACIS format), and finally a database that contains mesh-to-geometry associativity information, which in our case is stored in a text file (.m2g file). The mesh is segmented into files with approximately equal number of elements that will individually be read and refined by each processor.

The parallel mesh model in this implementation is maintained by the Sierra Tool Kit (STK). In this distributed multi-processor environment, processors read their corresponding mesh segments and STK manages communication through specialized mesh Parts. A mesh Part is a subset of the entire processor-distributed mesh domain [1]. Upon reading in the mesh, STK creates four primary Parts, the universal part, the locally-owned part, the globally-shared part, and finally the aura-part. Their definitions will be elaborated on later. In addition to these parts, custom parts can be created by STK users.

In STK, certain information is communicated along processor boundaries through two processes: ghosting and sharing. Nodes on the edge of a processor boundary will only be owned by one processor. However, another processor on that boundary may need that node for an operation. Thus, STK automatically shares these nodes from the owning processor to any other processor along that boundary. Mesh entities shared from one processor to another and vice versa reside in each process's globally-shared part [1].

Ghosting allows for certain operations on nodes that neither fall into locally-owned nor globally-shared parts [1]. In the STK mesh environment, a one element thick ghost layered exists along processor boundaries. Figure 1, adapted from the STK user manual [1], thoroughly describes the ghosting process. Mesh entities visible to a processor only through the ghosting process lay in the aura-part. Finally, the universal part refers to entire processor distributed mesh domain [1].

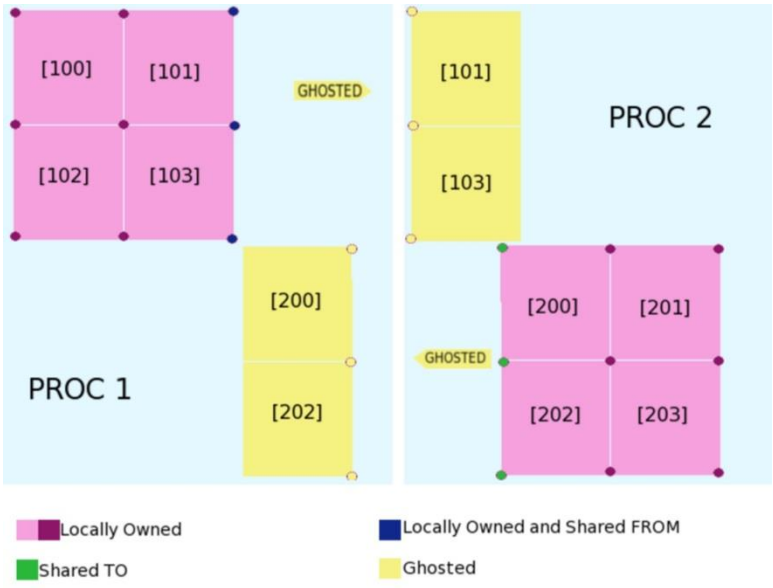

Fig. 1

In the STK mesh environment, Parts possess a persistent nature in concern to the modification of their entities. Consider refining a face on some part, all four new faces created from the subdivision of that face would become members of the Part as would their downwardly connected entities, i.e. the newly created edges and nodes. Once the mesh is refined, this implementation leverages the dynamic nature of parts.

For each curve or surface of a geometry, a custom Part is created that contains its edges or faces. After refinement, mesh nodes are queried and checked against these parts to determine which portion of geometry to attach itself to. If a node resides in multiple parts and one or more of those parts are a curve part, then a comparison is made between those curve parts as which is the best to move – or "snap" – the node to. In this case, the best option is defined as the option in which the node makes the smallest movement from its original position. If there are no curve parts that contain the node, the same comparison and snapping process is done for the surface or surfaces in which the node possesses membership.

2. The Mesh to Geometry File

2.1 How .m2g Works

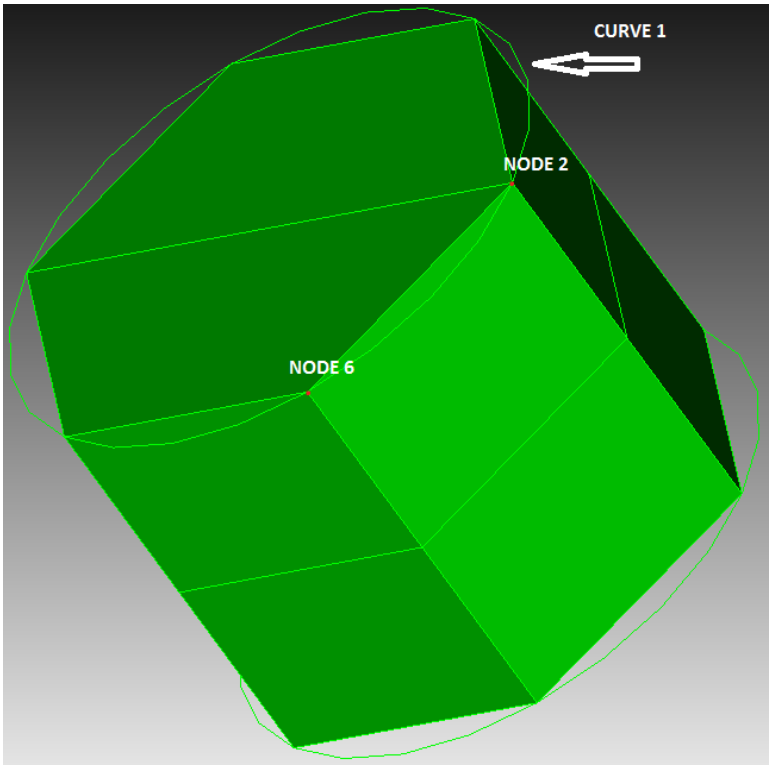

Fig 2

The .m2g file contains nodal ownership information as it pertains to each surface curve and volume [2]. For our purposes, strictly internal node to volume information is never read from the file since the snapping process only concerns nodes created on a surface or a curve. The following coarsely meshed cylinder exemplifies the information found in an .m2g file.

The cylindrical surface in figure 2 contains twelve faces and eighteen nodes. Each circular surface contains two faces and six nodes. Finally, each curve bounding each circular surface contains six edges and six nodes. In this case, the .m2g file describes that curve one, for example, contains nodes one through six and it also denotes the edge connectivity information between the nodes -- I.e. there is an edge between nodes two and six.

2.2 Parallel Issues with the .m2g File

Due to mesh segmentation among processors, the .m2g file possesses a plethora of unwanted mesh information when read in parallel. For an example, if a mesh is segmented among 100 processors, each processor should contain approximately one percent of the mesh. This results in a large loss of computer time spent searching for nodes not in

the processor's mesh portion. For an example, consider a portion of the mesh that is purely internal, none of its nodes lay on surfaces or curves. All efforts to query nodes found in the .m2g file against this mesh are wasted effort. Even if some of the nodes on the processor were on surfaces and curves, only a fraction of the .m2g file would contain any information relevant to the processor. This, too, results in a large loss of time spent in querying nodes. Two potential solutions follow. Note, they are both predicated on two things: first, querying a mesh for a node must take constant time; second, querying groups of nodes for faces and edges is an iterative process.

First solution: segment the .m2g file into smaller files that only contain data relevant to that processor. The second solution involves having each processor read the entire file but filter out unneeded mesh information with a constant time node filtering function.

When segmenting the .m2g file, the process must make use of a node filtering function and read the .m2g file. After filtering, it then must write out an intermediate file then read that intermediate file. In other words, it creates redundant steps after filtering. Therefore, we ascertained that simply loading a copy of the file for each processor then filtering unneeded node date would be more efficient. However, if the .m2g is excessively large, segmenting it may prove worthwhile from a memory conservation standpoint.

2.3 Creating Edges and Faces from an .m2g File

In order to keep the mesh as memory-light as possible, the only mesh entities existing upon import in the STK environment are nodes and their associated elements. This incurs the need to create face and edge entities that will possess the new nodes after refinement. These new nodes will be used to snap to the geometry post-refinement. The logic for creating these edges and faces without recreating them redundantly on multiple processors follows:

- Obtain validated node information from .m2g file for a given entity
- Find the common processors that either own or have these nodes shared to them
- Among these processors, find the one with the lowest ID
- Designate this processor as responsible for creating that given entity
- Create the entity then add it to its corresponding curve or surface part
- STK will notify other processors of the new edges/faces via sharing

3. Performance Data and Scalability Issues

Below is time in seconds data on an 8.5M element mesh with approximately 632Kfaces among its associated geometry's surfaces

Table 1

Number of processors	4	8	16	32
Funct: Import ACIS file (cmp. time, sec.)	Total: 3.198 Average: 0.800	Total: 6.277 Average: 0.785	Total: 12.615 Average: 0.788	Total: 25.710 Average: 0.803
Funct: Import .m2g file (cmp. time, sec.)	3.680 Average: 0.920	6.750 Average: 0.844	13.028 Average: 0.814	23.675 Average: 0.740
Total Comp Time (cmp. time, sec.)	Total: 15565.472 Average: 3891.368	14217.142 Average: 1777.143	14702.533 Average: 918.908	15333.746 Average: 479.180
ACIS file size 1.8mb	7.2 mb	14.4 mb	28.8 mb	57.6 mb
.m2g file size 20mb	80 mb	160 mb	320 mb	640 mb
ACIS and .m2g overhead	87.2 mb	174.4 mb	348.8 mb	697.6 mb

As a whole, the meshing process scales quite well. The total computer time fluctuates slightly as the number of processors increases. However, the total real runtime decreases almost by a factor of 0.5 as the number of processors on this job doubles. Despite the general scalability, certain functions actually have a negative computer time correlation with the number of processors used and mesh size, namely importing the geometry file and the .m2g text file.

The data shows that at the number of processors doubles, the percentage of net computer time spent reading a geometry file also approximately doubles. For an example, if we were to extrapolate the current trend to 1024

processors, the computer time spent reading the ACIS file should reach about 800 seconds. Assuming the total computer time remains a roughly constant 16000 seconds, the percentage of computer spent reading should be about five percent. When compared to the initial percentage at four processors of .021 percent, the trend starts to seem unnecessarily costly.

However, importing the ACIS file is hardly the least scalable function. While in this particular example, importing the .m2g file and reading the ACIS file seem to move in lock and step, the reality is that importing the .m2g is potentially much costlier. While the ACIS import increased by a factor of n, where n = number of processors used, the .m2g import cost increases by n by m, where m is the number of surface nodes read out of the .m2g file. This stems from the structure of the .m2g file and the process of parsing it.

The .m2g file structure is such that for any given surface or curve, all the nodes listed must be read and validated. This wastes computer time when reading nodal information of a face or edge that the current process doesn't own. Ideally, the only entity nodes completely parsed would be those owned by the current processor. For an example, if the nodes of a surface quad are being parsed, and an invalid node is encounter, the parsing process for that particular entity is stopped and the parsing process for the next entity is started. This also possess the potential to reduce the number of times the validation function is called. Consider the following case: the current process owns none of the nodes for a given entity, something that would happened quite often on a highly processor-segmented mesh. With the current read process, each node is parse and validated. In an improved process, the parsing process would end on the first node of the quad entity. Thus, no computer time gets wasted on parsing and validating the rest of the invalid nodes.

4. Conclusion

While low cost in many cases, the pre-refinement association process doesn't scale well when a large number of processors are utilized. The following research topics represent potential avenues of improvement for the scalability of this process. In order to reduce overhead of importing complex geometry models, work can be done to segment the geometry such that only the needed portions – i.e. the geometry associated with a segment of mesh – are imported. Alternatively, or additionally, instead of each processor reading the same geometry from the disk, a designated fetching processor could retrieve it and store it in memory, then neighboring processors could query a copy when the job calls for it. This would reduce repeated trips to the disk for reads, which is monumentally slower than reading from memory. Another more vital fix involves restructuring the format of the .m2g file such that not every node gets parsed and validated. Any format that supports the short-circuiting logic described above should provide a viable solution. Through these and other optimizations, the UMR pre-refinement process can be greatly improved.

References

[1] Sierra ToolKit Development Team, *Sierra Toolkit Manual Version 4.38* Albuquerque, NM Sandia National Laboratories, 2015. 19-26.
[2] Ted Blacker, Steven J. Owen, Matthew L. Staten, Roshan W. Quadros, Byron Hanks, Brett Clark, Ray J. Meyers, Corey Ernst, Karl Merkley, Randy Morris, Corey McBride, Clinton Stimpson, Michael Plooster, Sam Showman, *Cubit Geometry and Mesh Generation Toolkit 15.2 Documentation*, Albuquerque, NM Sandia National Laboratories, 2016. 572.