SAND2007-2603C

Parallel In-Line Finite-Element Mesh Generation Library

David Hensinger

Sandia National Laboratories

Department 1431

Computational Shock and Multi-Physics

Sandia National Laboratories

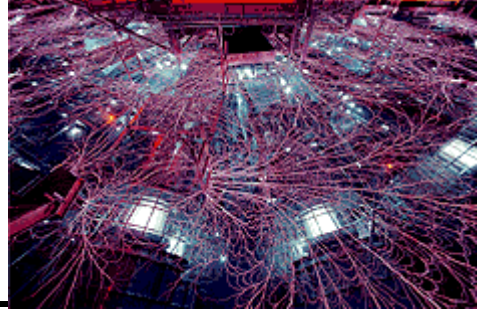
Parallel In-Line Finite-Element Mesh Generation Library

Replaces pre-processed (decomposed) unstructured finite element mesh specification files for MP simulations.

Sandia National Laboratories

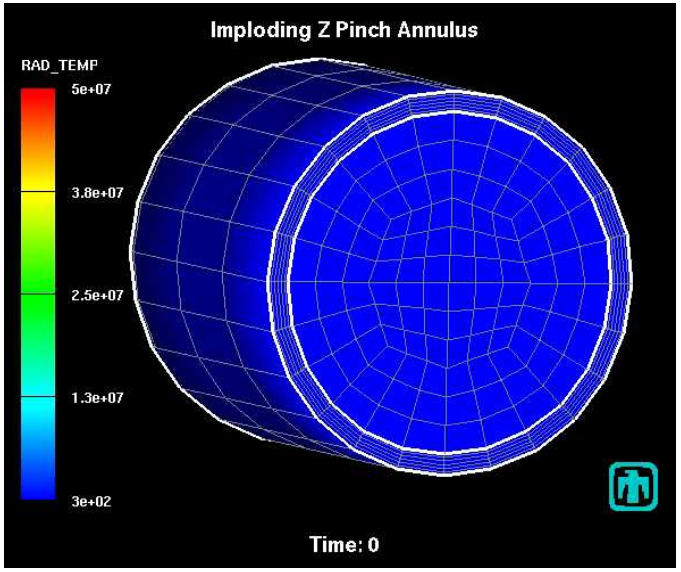
What does it do?

- Generates an unstructured mesh on each processor during start of simulation
- Relies on information available to each processor (input deck)
- Allows simple BC specification
- Allows simple refinement
- Supports several simple geometries
- Produces, in memory, the equivalent of a nemesis file



Why was this done?

- Serial mesh generation strategies were unable to supply analysts' demands for:
  - Billions of Elements having
  - Graded Mesh on
  - Simple Geometries with
  - Quick Turn-around
- Analysis capabilities are leaving pre-processing tools behind



R.S. 12960 nodes
>= 2Gb/node

Purple, 1532 nodes
32Gb/node

Sandia National Laboratories

Requirements for Unstructured Mesh Simulation in Parallel

- Each processor needs to know about itself
  - What elements and nodes it has
  - What the elements topology is
  - Where the nodes are located
- Each processor needs to know who it shares nodes and element faces with
  - This can be in the form of consistently ordered lists on each processor
  - No need to know the id of a neighboring element on another processor

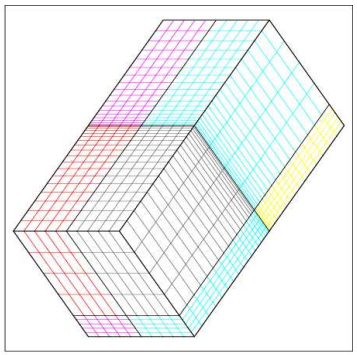
Approach

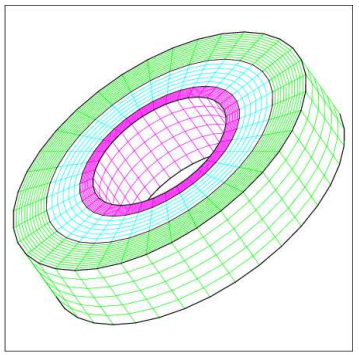
- Make the mesh generation inherently parallel (each processor makes its own mesh)
- Forbid Communication (ensure scaling?)
- Answer mesh related queries by referring to the fundamental description of the mesh (minimize creation of intermediate objects)
- Require mesh be fully determined at input time (no paving?)

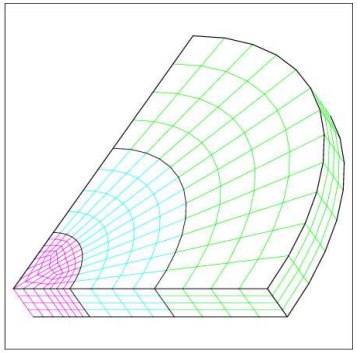
Advantages of this approach

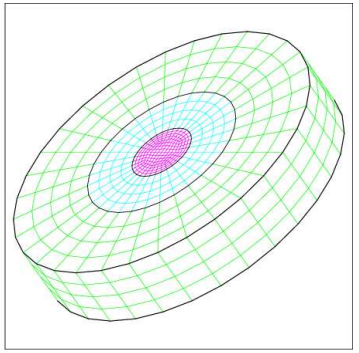
- Can handle huge meshes > 10^9 elements
  - No serial pre-processing step
  - No need to move input files around
- Easy of use
  - No pre-processing
  - Can change number of processors easily
  - Readily increase and decrease size of problem
- Performance
  - Mesh generation is fast
  - Small memory footprint

Inline Parallel Mesh Geometry Zoo









Sandia National Laboratories

Execution Steps

- Answer the global questions about the mesh.
  - How many elements are there?
  - How many nodes are there?
  - How many nodeset and sidesets (BC application regions) are there?
- Answer the serial (one each processor) questions.
  - What elements are on this processor?
    - Partitioning is right here up front.
  - What nodes are on this processor?
    - Answered by looking at the elements on this processor and calculating their global ids.
  - What is the connectivity of this processors' elements?
  - What sideset element faces and nodeset nodes are on this processor?
- Answer the parallel (inter-processor) questions about the mesh.
  - What elements border this element?
  - What processor do neighboring elements reside on?
  - What nodes are shared with neighboring elements on other processors.

Global Questions

- **Answering total number of elements/nodes/bc's… requires a deterministic specification available to all processors**

Serial Information - Decomposition- Element/Node ownership

- Each processor carries out the decomposition.
  - Recursive binary decomposition has been successful with a pre-calculation to determine size of cuts
  - User-specified is quite popular
  - Space filling curve would work well
  - Trivial decomposition (counting off) followed by a second pass would be powerful
- Each processor can determine processor ownership of any element.
  - Element ownership mirrors decomposition

Serial Information – Connectivity, Nodal Coordinates

- Each processor has access to complete specification information
- Connectivity and coordinates may be calculated from deterministic specification

Parallel Information: Border Elements/Nodes

- All processors have access to complete mesh specification
- Face neighbors can be calculated
- Processor ownership can be calculated

Parallel Information: Locating Border Elements and their Processor Counterparts

- walk all local elements
  - visit their neighboring elements through faces/edges/vertices and ask if that neighbor is on my processor
  - if it is on my processor do nothing
  - if it is not on my processor
    - I am a border element
    - all nodes on that face/edge/vertex are border nodes
    - expand lists with neighbor proc ids and topology directions
- Sort the resulting lists by lowest common global id to produce correspondence between lists on all processors

What questions must we answer?

- How many elements/nodes total – depends on initial specification
- Which elements are on this processor – depends on decomposition
- Which nodes are on this processor – depends on which nodes are on this element
- What are the coordinates of this node – depends on initial specification of geometry
- Communication pattern – depends on decomposition
- Which elements are border elements – depends on what elements are my neighbors and what processor do they reside on
- What nodes are border nodes – depends on element neighbor calculation

The questions

Any mesh generation process can be parallelized in this way provided the following questions can be answered without resorting co communication.

- For elements
  - How many are there?
  - To what processor do they belong?
  - What are their local/global global/local ids?
  - What are their nodes?
  - What elements are their face/edge/vertex neighbors?
- For nodes
  - How many are there?
  - What are their local/global global/local ids?
  - What are their coordinates?

Missing Abstraction

Terse specification of mesh is missing due to:

- **Intensely interactive mesh generation process (this is actually mesh geometry decomposition and mesh specification).**
- **Tight coupling between mesh specification interface and mesh generation.**
- **Reliance on sequential operations in mesh generation.**

This approach can be extended provided..

- The data (and required libraries [eg ACIS]) representing the entire mesh fits on each processor (surface meshes, geometry + directives …)
- Objects that correspond to each element/edge/face/node/vertex… are never created.

Alternatives (for filling machines like purple) [23Gb 1532 nodes])

- Serial machines and serial code
- Parallel mesh generation on clusters/SM machines with additional post-processing/movement/decomposition of immense files
- Failure (can't make the mesh, can't fill the machine)

What are the limitations

- Complex geometry input
- Addressable memory

Sandia National Laboratories

Logical Next Steps (Comparative Difficulty)

- Generating 3D mesh from 2D mesh by sweeping (walk in park)
  - Every processor reads 2D mesh
  - All questions can be answered from this information
- Generating from boundary mesh of topological cubes (walk in jungle)
- Generating from boundary mesh including paved and swept volumes (walk on moon)

EXTRA SLIDES

Sandia National Laboratories

Abstract

Generation of large finite-element meshes is a serious bottleneck for parallel simulations. When mesh generation is limited to serial machines and element counts approach a billion, this bottleneck becomes a roadblock. To surmount this barrier the ALEGRA shock and multi-physics project has developed a parallel mesh generation library that allows on-the-fly scalable generation of finite element meshes for several simple geometries. It has been used to generate more that 1.1 billion elements on 17,576 processors. The library operates on the assumption (and constraint) that the mesh generation process is deterministic. Each processor in a parallel simulation is provided with a complete specification of the mesh, but it only creates a full representation of the elements that will be local to that processor. Because of this, no inter-processor communication is performed.

The mesh generation proceeds through steps of decomposition, local element creation, and communication information generation. The final product of the library is a data structure that can be passed to an analysis code in the place of one generated from an input file. Currently the library is limited to generating meshes of domains with cylindrical, tubular, and block shapes. Substantial control is allowed over the element density. Boundary condition regions can be specified on the surfaces and interior of the mesh .

Development of this capability revealed that the parallel mesh generation process can be reduced to answering a series of questions: What elements are on this processor? What nodes are on this processor? What is the connectivity of this element? What elements border this element? What processor does this element reside on?... Resolving these questions inductively, without resolution to communication, is essential for preserving scalability. Once a framework is established for posing and answering these questions for a particular geometry is established, expanding the capability to support additional geometries is straightforward.

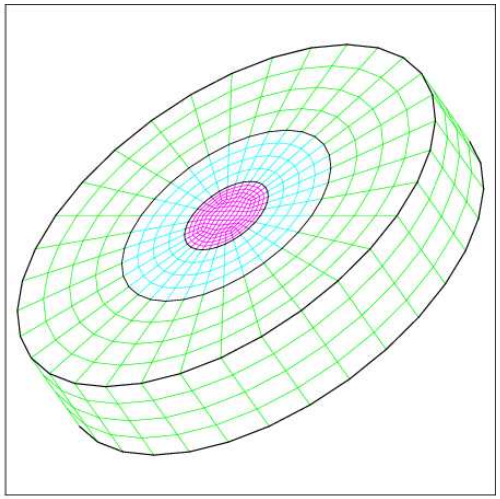
Sandia National Laboratories

Parallel Inline Meshing Library - Characteristics

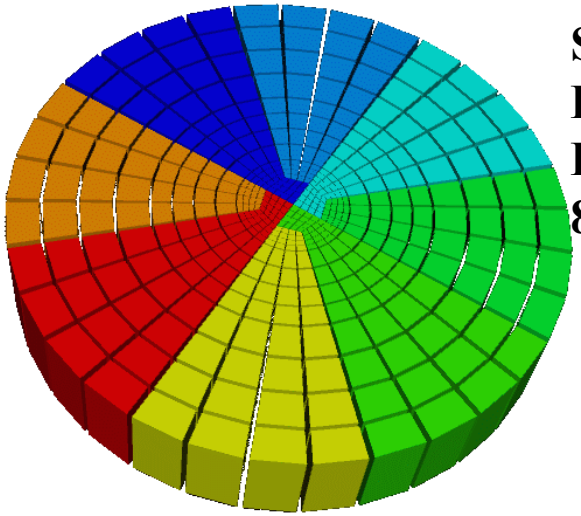
- Creates finite element meshes for processor n of k where n <= k
- Requires no communication (forbids it in fact)
- All operations occur in parallel
- Requires mesh generation be deterministic

Inline Mesh Generation
Bricks and Cylinders

- Parallel and Scalable for Arbitrarily Large Numbers of Elements and Processors (Requires No Communications)
- Driven by input deck
- Decomposition Options
  - Sequential
  - Bisection
  - Processor Layout
- Assigns Nodesets, Sidesets, and Element Blocks



Solid Cylinder Showing Element Blocks



Solid Cylinder Decomposed Decomposed for 8 Processors

Sandia National Laboratories

Parallel In-Line Finite-Element Mesh Generation Library

- What was done?
- What were its limitations?
  - Which are arbitrary?
  - Which are systemic?
- How did it perform?
- What are its failings?
- What was the need?
- Why was it done?
- How was it done?
- How can this approach be extended?
- How much effort did it take.
- What was the approach?
- What makes this useful?
- What makes this unique?
- Who did this?
- What was hard?
- What was easy?
- What was impossible?