Pamgen, a Parallel Finite-Element Mesh Generation Library

David Hensinger, *Sandia National Laboratories;*

ABSTRACT: *Generating quadrilateral and hexahedral finite-element meshes is a serious bottleneck for large 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 a parallel mesh generation library (Pamgen) has been developed 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 mesh generation strategy used in this implementation will be presented*

KEYWORDS: Finite Element, Mesh Generation

1. Introduction

To overcome the challenge of producing multi-million finite element meshes for simulations using more than 1000 processors a library has been developed that produces each processor's mesh as an early step of the analysis execution. The specification for these meshes is provided by terse instructions within the analysis input decks. This allows analysts to change the resolution of a simulation by altering a few parameters. It also allows them to execute their simulations on an arbitrary number of processors without requiring any pre-processing.

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 a mesh 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 within these shapes. Boundary condition application 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 set of questions:

- What is the total number of elements?
- What processor does any element reside on?
- What is an element's connectivity?
- What are a node's coordinate values?

The answers to these questions can be leveraged to find:

- Which elements are on any particular processor.
- Which elements border an element.
- Which elements are on processor borders.
- Which nodes are shared on processor boundaries.

Resolving these questions inductively, without resolution to communication, is essential for preserving scalability. Once a framework for posing and answering these questions for a particular geometry is established, expanding the capability to support additional geometries is straightforward.

2. Enabling /Limiting Assumptions

All Processors Have Identical Information

Every processor has all the information required to generate the entire mesh. Since each processor has the same information, it is capable of calculating the answer to any mesh related question and communication is unnecessary. This reliance on calculation requires the next assumption.

All Processor are Identical (except for their ids)

If it can be assumed that each processor will produce the same results from identical calculations we can perform calculations identically instead of distributing the result of a single calculation or averaging the results of multiple calculations.

Communication is Unavailable

Assuming communication is unavailable guarantees the parallel scalability of the library. All calculations are made locally without resorting to inter-processor queries. This does limit the use mesh-wide iterative strategies such as of elliptic smoothers. Eschewing communication also significantly aids testing and development. It makes it

possible to test the library in serial by generating the mesh for processor ***n*** of a total of ***m***.

Meshes Consist of Structured Block Topologies

Limiting the topologies of the meshes that could be generated enables algebraic calculations of all mesh topology.

The topologies of the finite element meshes was limited to one or more blocks of elements with a regular or structured topology. For those meshes that consisted of more than one structured block there were a limited number of ways for the blocks to connect with each other.

Within a structured block, calculation of an element's connectivity depended on knowing only the extents of the structured block and the i,j,k indices of the element's primary node.

3. Execution Stages

Since communication is unavailable, the mesh generation process can be discussed in terms of a single processor.

Information Distribution

The mesh generation process begins with each processor receiving a terse description of the mesh's geometry and topology. This information is first used to calculate total quantities for the mesh such as the number of nodes, elements, edges, and faces. From this it can be determined if there is a possibility of exceeding the address space available for generating the mesh.

Decomposition

An identical decomposition calculation is performed on each processor. The most generally successful of the available decomposition strategies performs a bisection decomposition on the topological structured blocks of mesh. The result of this decomposition is a method for determining the processor location of any element of the mesh. Using this method, a list is compiled of all elements local to the processor.

Serial Information Generation

The serial information consists of the nodal geometry and connectivity of each element local to the processor. This is produced by walking the list of local elements and compiling a list of local nodes using information about from the structured block topology and individual element connectivity. This list of nodes is then made unique and the indices of the unique nodes are used to populate the connectivity array of the local element blocks. The coordinates of the local nodes are calculated based on their topological indices in the structured block and the geometric description.

The lists of nodes and element faces to which boundary conditions may be applied are also collected during the serial information generation.

Parallel Information Generation

The parallel information consists of lists of nodes on processor boundaries that correspond to identically ordered lists on neighboring processors.

This information is gathered by visiting the local elements and exploiting the local topology information to calculate the ids of all elements connected to the local element by a face, edge, or corner. These are the element's element neighbors. The decomposition information is then exploited to calculate the processor of each element neighbor. If the element neighbor is local no further action is taken. If the element is on an adjacent processor, then the face, edge, or corner nodes through which that element is connected are placed in a list that corresponds to the neighboring processor. After this process has been completed for all local elements, the lists of nodes are sorted by the nodes' unique global id. This results in corresponding synchronized node lists on adjacent processors.

Geometric Transformations

The final step in the mesh generation is the application of arbitrary geometric transformations. While the topological limitations on the mesh are driven by difficulty in calculating element connectivities, the geometric limitations are driven by the difficulty of producing terse descriptions of complex geometries. This limitation on geometric complexity is somewhat alleviated by allowing arbitrary geometric modifications.

The terse mesh description can be associated with a user provided subroutine that is evaluated using each node's original (based on terse specification) coordinates to calculate new nodal coordinates. This general geometric transformation allows distortion of the terse geometry description into arbitrary shapes.

4. Library Interface Implementation

The library is accessed through its API in two stages, mesh creation and mesh query.

Mesh Creation Interface

Meshes are created within the library by the client program through a single function call:

```
int Create_Pamgen_Mesh(char * mesh_description,
                       int dimension,
                       int rank,
                       int num_procs);
```

It creates a representation of the mesh with dimensionality ***dimension*** for the processor of the specified ***rank*** out of the total ***num_procs***. It returns an enumerated error code.

The mesh_description input variable points to a null terminated string that holds a terse description of the desired mesh. Examples of this description are given in a following section.

Mesh Query Interface

A mesh may be queried only after it has been created. All of the mesh query functions of the Pamgen library are based on the EXODUS II [1] and NEMESIS [2] APIs. These APIs were written to standardize a platform independent interface for writing and reading binary mesh specification files. NEMESIS is a parallel extension of the serial EXODUS II API the Pamgen function names are formed by prefixing the EXODUS II or NEMESIS functions names with im_. The remainder of the function signature and operation remains unchanged.

The client program builds up its model of the local finite element mesh and inter-processor communication information through a sequence of query functions. Initial functions provide the number of local nodes, elements, node sets, and side sets. Subsequent functions provide the connectivity of the local elements the local nodes' coordinates. Additional query functions provide inter-processor communication information.

Complete documentation of the query functions are available in reference [3].

5. Capabilities

Topologies

Pamgen can create meshes with the following topologies:

- Cubes
- Solid Cylinders
- Hollow Cylinders

Geometries

Pamgen allows terse descriptions of cube, cylinder, partial cylinder, partial solid cylinder, and solid cylindrical shapes.

Geometry Modification

A user supplied function may be supplied to apply a general transformation to the standard available forms. The function is delimited by double quotes and allows most of the functionality of the C language.

Boundary Conditions

Regions of the mesh may be called out for boundary condition application in the form of node sets and side sets. Node sets are lists of nodes, and side sets are lists of element faces. Node sets and side sets may be specified on any exterior face, edge, or corner of a mesh. They may also be called out on the face edge or corner of material blocks within the a mesh.

Decompositions

There are several decomposition options available within Pamgen. All of them are simple enough to require no communication or iteration.

The default decomposition optimally bisects repeatedly. This decomposition is most successful when the number of processors is the product of multiple prime numbers.

A sequential decomposition is available that assigns an equal number of elements to each processor in their numerical order. This is guaranteed to produce poor decompositions for large numbers of processors, but it may be optimal for small numbers of processors.

A user determined decomposition is available by which the number of processors in each topological direction is supplied by the user. The product of the number of processors in each direction must equal the total number of processors.

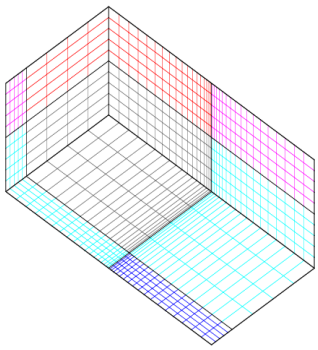
A random decomposition is also available. This strategy assigns elements randomly to processors. It is very useful for testing the robustness of inter-processor communication strategies.

6. Examples

The following examples show the terse mesh description passed to the Pamgen library and an illustration of the resulting mesh. Complete documentation of the Pamgen interface is available in reference [3].

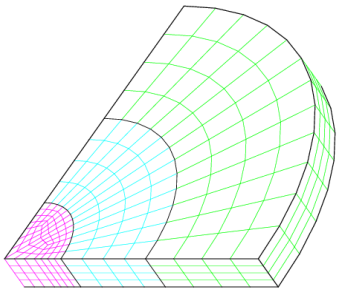
A 3D hexahedral mesh domain consisting of eight different material blocks with element sizes graded in the Y coordinate direction.

```
mesh
  brick
    numz 2
      zblock 1 2. interval 5
      zblock 2 8. interval 4
    numx 2
      xblock 1 5.0 interval 5
      xblock 2 5.0 interval 5
    numy 2
      yblock 1 10. first size 1. last size .1
      yblock 2 10. first size .1 last size 1.
  end
end
```



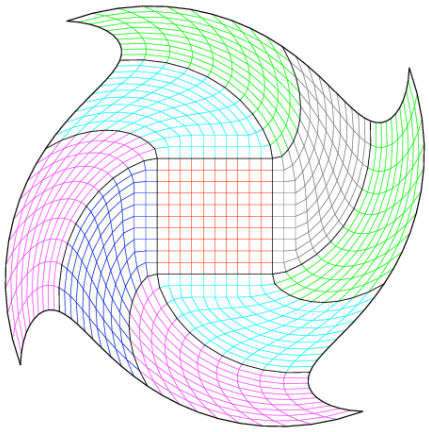
A ninety degree section of a solid cylinder with the center filled by two transition blocks of elements.

```
mesh
  radial trisection
    trisection blocks, 2
    zmin -0.00075
    numz 1
      zblock 1 1. interval 4
    numr 3
      rblock 1 2.0 interval 4
      rblock 2 3.0 interval 4
      rblock 3 4.0 interval 4
    numa 1
      ablock 1 90. interval 12
  end
end
```



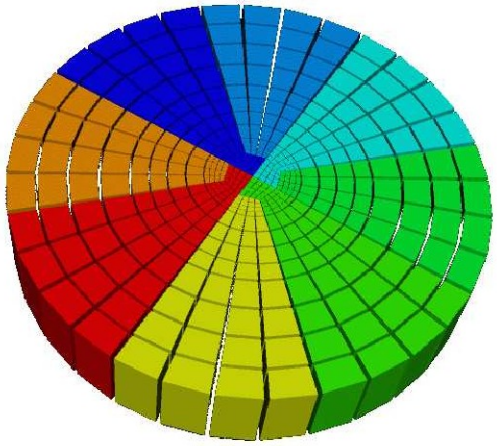
A 2D mesh consisting of nine material blocks that is transformed using a user supplied function.

```
mesh
  rectilinear
    nx = 10
    ny = 10
    bx = 3
    by = 3
    gmin = -1.0 -1.0
    gmax = 1.0 1.0
  end
  user defined geometry transformation
    "
      double r = sqrt(inxcoord*inxcoord
+inycoord*inycoord);
      double theta = atan2(inycoord,inxcoord);
      if(r > 0.5)
      {
  theta = theta + (3.14159 / 4.0)*((r-0.5)/0.5);
      outxcoord = r*cos(theta);
      outycoord = r*sin(theta);
      }
    "
  end
end
```



A 3D solid cylinder that is decomposed for eight processors by explicitly specifying the number of cuts to make in the j topological direction.

```
mesh
  radial trisection
    trisection blocks, 4
    numz 1
      zblock 1 4.0 interval 1
    numr 3
      rblock 1 2. interval 4
      rblock 2 3. interval 4
      rblock 3 5. interval 4
    numa 1
      ablock 1 360. interval 32
  end
  decomposition strategy
    numprocs j, 8
  end
end
```



7. Availability and Distribution

The Pamgen library is available under the terms of the GNU Lesser General Public License. It is distributed as a component under the Trilinos Library Distribution System. It may be found at:

http://trilinos.sandia.gov/packages/pamgen

References

[1] L. A. Schoof and V. R. Yarberry. EXODUS II: A Finite Element Data Model. Technical report SAND92-2137, Sandia National Laboratories, Albuquerque, NM, November 1995.

[2] G. L. Hennigan, M. St. John, and J. N. Shadid. NEMESIS I: A set of functions for describing unstructured finite-element data on parallel computers. Technical report, Sandia National Laboratories, Albuquerque, NM, May 1998.

[3] D. M. Hensinger, R. A. Drake, J. G. Foucar, and T. A. Gardiner. Pamgen, A Library for Parallel Generation of Simple Finite Element Meshes. Technical report SAND08-1933, Sandia National Laboratories, Albuquerque, NM, April 2008

Acknowledgments

Thanks to Chris Garasi for asking forcefully for big quality meshes. Thanks to Tom Gardiner for the bisection decomposition routines in Pamgen.

About the Author

David Hensinger dmhensi@sandia.gov is a staff member at Sandia Nation Laboratories in the Computational Shock and Multi-Physics group. P.O. Box 5800, Albuquerque NM, 87185-0378