Data Structures and Methods for Unstructured Distributed Meshes

Roy H. Stogner

The University of Texas at Austin

May 23, 2012
Introduction

Data Structures
- Unstructured Meshing
- Distributed Meshing

Algorithms
- Parallel Utilities
- Parallel Mesh Manipulation

Applications
- Performance
- Challenges

Conclusions
Context and Motivations

libMesh design

- C++ library classes
  - Hybrid parallel FEM calculations
  - Unstructured adapted meshes
  - Abstract base classes like Elem, FEBase
  - Concrete leaf classes like Hex27, FEHierarchic

ParallelMesh design goals

- Eliminate serialized mesh bottleneck
- $O \left( \frac{N_E}{N_P} \right)$ memory use
- Interchangeable with Mesh (now SerialMesh)
- Incremental design and testing
Unstructured Mesh Data: Elements, Nodes

- Heap memory allocation for mixed geometric, finite element types
- Mesh provides iterators, random access
- Each element provides pointers to neighbors, nodes
- Adaptive refinement adds parents, children
- Boundary adds null neighbors
Unstructured Mesh Data: Degrees of Freedom

Indexing Requirements

- Multiple systems of equations on each Mesh
- Multiple variables in each system
- (Varying) multiple components to each variable
- Extra components on some hanging node types
- Second-order geometric nodes give topological storage for higher-order finite elements
Mesh Class Hierarchy

MeshBase
+boundary_info: AutoPtr<BoundaryInfo>
+elem(unsigned int)
+node(unsigned int)
+elements_begin/end()
+nodes_begin/end()
+active_local_elements_begin/end()
+read()
+write()

CartesianMesh

UnstructuredMesh
+find_neighbors()
+read/write()
+add/delete_elem()

SerialMesh
+elements_begin()

ParallelMesh
+_elements: mapvector<Elem>*
+allgather()
+delete_remote_elements()

Object-Oriented Design

• Predicated iterators for traversing subsets of elements, nodes
• Abstract iterator interface hides mesh type from most applications
• UnstructuredMesh "branch" for most library code
• ParallelMesh leaf implements distributed storage, synchronization
## Distributed Mesh Data Structures

### SerialMesh
- Contiguous vector of pointers to Elem, Node
- \( \text{elements}[i] \rightarrow \text{id}() == i \)
- \( \mathcal{O}(1) \) sequential traversal
- \( \mathcal{O}(1) \) random access

### ParallelMesh
- map (binary tree) or unsorted map (hash table)
- \( \text{elements}[i] \rightarrow \text{id}() == i \)
- \( \mathcal{O}(1) \) (sequential or unsorted) traversal
- \( \mathcal{O}(\log \frac{N_E}{N_P}) \) or \( \mathcal{O}(1) \) random access
RemoteElem

- Boundary neighbors still point to NULL
- Neighbors owned by remote ranks are “ghost” elements with full data
- Neighbors which exist only on remote ranks point to remote_elem
- $O(1)$ memory, tests
- Code “walking” past ghost elements triggers runtime errors
Utility Functions - Parallel::

MPI interface

```cpp
std::vector<Real> send, recv;
...
if (dest_processor_id == libMesh::processor_id() &&
    source_processor_id == libMesh::processor_id())
    recv = send;
#else
else
{
    unsigned int sendsize = send.size(), recvsize;
    MPI_Status status;
    MPI_Sendrecv(&sendsize, 1, datatype<unsigned int>(),
                 dest_processor_id, 0,
                 &recvsize, 1, datatype<unsigned int>(),
                 source_processor_id, 0,
                 libMesh::COMM_WORLD,
                 &status);

    recv.resize(recvsize);

    MPI_Sendrecv(sendsize ? &send[0] : NULL, sendsize, MPI_DOUBLE,
                 dest_processor_id, 0,
                 recvsize ? &recv[0] : NULL, recvsize, MPI_DOUBLE,
                 source_processor_id, 0,
                 libMesh::COMM_WORLD,
                 &status);
}
#endif // HAVE_MPI
```
Utility Functions - Parallel::

Parallel:: interface

```cpp
std::vector<Real> send, recv;
...
Parallel::send_receive(dest_processor_id, send,
source_processor_id, recv);
```

Advantages

- Recompilable without MPI
- MPI implementation optimizable “under the hood”
  - MPI probe for vector lengths
  - Asynchronous I/O
- Data type independent
- Container type independent
Utility Functions - Parallel::

Parallel:: algorithms

```cpp
template <typename Iterator,
          typename SyncFunctor> void
sync_dofobject_data_by_id
(const Iterator& range_begin,
 const Iterator& range_end,
 SyncFunctor& sync);
```
Utility Functions - Parallel::

Advantages

- Generic communication for Elements, Nodes
- SyncFunctor::act_on_data updates local mesh subset
- Communications algorithm is independent of functor
Utility Functions - Parallel::

Round-Robin

- Processor $P$ sends to processor $P + K$ while receiving from $P - K$
- Old data discarded after each step - $O\left(\frac{N_G}{N_P}\right)$ in transit
- $O\left(N_G\right)$ worst-case execution time
Utility Functions - Parallel::

Future optimizations

- Precalculate, skip non-neighboring processors
- Queue multiple asynchronous sends at a time
Mesh Distribution

Mesh Creation

- File 1-to-N input, serial generation
- File N-to-N input
- Parallel generation
Mesh Distribution

Mesh Creation

- File 1-to-N input, serial generation
  - Remote elements must be removed
  - One pass: flag nodes on local elements
  - Second pass: delete disconnected elements, replace with remote_elem

- File N-to-N input

- Parallel generation
Mesh Distribution

Mesh Creation

• File 1-to-N input, serial generation
  ▶ Remote elements must be removed
  ▶ One pass: flag nodes on local elements
  ▶ Second pass: delete disconnected elements, replace with remote_elem

• File N-to-N input
  ▶ Ghost elements, nodes must be gathered from neighboring processors
  ▶ Local boundary node id lists sent to adjacent processors
  ▶ Ghost nodes, elements received in return

• Parallel generation
Mesh Distribution

Mesh Creation

- File 1-to-N input, serial generation
  - Remote elements must be removed
  - One pass: flag nodes on local elements
  - Second pass: delete disconnected elements, replace with remote_elem

- File N-to-N input
  - Ghost elements, nodes must be gathered from neighboring processors
  - Local boundary node id lists sent to adjacent processors
  - Ghost nodes, elements received in return

- Parallel generation
  - \( \text{Ids } \{ i : i \mod (N_P + 1) = p \} \) are owned by processor \( p \)
  - Very algorithm-dependent
Distributed Degree of Freedom Numbering

One-pass indexing

- Index processor $P$ starting from $\sum_{1}^{P-1} N_{Ep}$
- Pass $\sum_{1}^{P} N_{Ep}$ to processor $P + 1$
- $O(NE)$
Distributed Degree of Freedom Numbering

**One-pass indexing**
- Index processor $P$ starting from $\sum_{1}^{P-1} N_{Ep}$
- Pass $\sum_{1}^{P} N_{Ep}$ to processor $P + 1$
- $O(N_E)$

**Two-pass indexing**
- Count processor $P$ indices starting from 0
- Gather $N_{Ep}$ on all processors
- Re-index processor $P$ from $\sum_{1}^{P-1} N_{Ep}$
- Trade ghost indices with remote processors
- $O(N_E/N_P)$ execution time
Distributed Mesh Refinement

Error Estimation

- Local residual, jump error estimators
- Refinement-based estimators
- Adjoint-based estimators
- Recovery estimators
## Distributed Mesh Refinement

### Error Estimation

- Local residual, jump error estimators: embarrassingly parallel
- Refinement-based estimators: use solver parallelism
- Adjoint-based estimators: use solver parallelism
- Recovery estimators: require partitioning-aware patch generation
Distributed Mesh Refinement

Error Estimation

- Local residual, jump error estimators: embarrassingly parallel
- Refinement-based estimators: use solver parallelism
- Adjoint-based estimators: use solver parallelism
- Recovery estimators: require partitioning-aware patch generation

Refinement Flagging

- Flagging by error tolerance: \( \eta^2_K < \eta^2_{tol}/N_E \)
- Flagging by error fraction: \( \eta_K < r \max_K \eta_K \)
- Flagging by element fraction or target \( N_E \)
Distributed Mesh Refinement

Error Estimation

- Local residual, jump error estimators: embarrassingly parallel
- Refinement-based estimators: use solver parallelism
- Adjoint-based estimators: use solver parallelism
- Recovery estimators: require partitioning-aware patch generation

Refinement Flagging

- Flagging by error tolerance: \( \eta_K^2 < \eta_{tol}^2 / N_E \)
  - Embarrassingly parallel
- Flagging by error fraction: \( \eta_K < r \max_K \eta_K \)
  - One global Parallel::max to find maximum error
- Flagging by element fraction or target \( N_E \)
  - Parallel::Sort to find percentile levels?
  - Binary search in parallel?
  - TBD
Distributed Mesh Refinement

Elem, Node creation
Distributed Mesh Refinement

**Elem, Node creation**

- **Ids** \( \{ i : i \mod (N_P + 1) = p \} \) are owned by processor \( p \)
Distributed Mesh Refinement

Elem, Node creation
- \( \text{Ids } \{ i : i \mod (N_P + 1) = p \} \) are owned by processor \( p \)

Synchronization
- Refinement Flags
- New ghost child elements, nodes
- Hanging node constraint equations
Distributed Mesh Refinement

Elem, Node creation
- \( \{i : i \mod (N_P + 1) = p\} \) are owned by processor \( p \)

Synchronization
- Refinement Flags
  - Data requested by id
  - Iteratively to enforce smoothing
- New ghost child elements, nodes
  - Id requested by data
- Hanging node constraint equations
  - Iteratively through subconstraints, subconstraints-of-subconstraints...
“Typical” PDE example

Transient Cahn-Hilliard, Bogner-Fox-Schmidt quads or hexes

Results

- Parallel codes using SerialMesh are unchanged for ParallelMesh
- Overhead, distributed sparse matrix costs are unchanged
- Serialized mesh, indexing once dominated RAM use
Distributed Correlation Length Postprocessing

Cahn-Hilliard FEM has only semilocal dependencies

\[
\left( \frac{\partial c}{\partial t}, \phi \right)_\Omega = - \left( M_c \nabla f'_0(c), \nabla \phi \right)_\Omega - \epsilon_c^2 \left( \Delta c, \nabla \cdot M_c^T \nabla \phi \right)_\Omega \\
+ \left( \left( M_c \nabla \left( f'_0(c) - \epsilon_c^2 \Delta c \right) \right) \cdot \vec{n}, \phi \right)_{\partial\Omega} + \epsilon_c^2 \left( \Delta c, M_c^T \nabla \phi \cdot \vec{n} \right)_{\partial\Omega}
\]

Correlation lengths involve global integrals

\[
<f(\vec{x})> \equiv \frac{\int_{\Omega} f(\vec{x}) \, d\Omega}{\int_{\Omega} 1 \, d\Omega}
\]

\[
r(\vec{y}) \equiv <c(\vec{x})c(\vec{x} + \vec{y})> - <c(\vec{x})>^2
\]

- Communication of remote data unavoidable
- Communication of remote geometry avoided via coarse mesh integration
Hypersonic Flow Calculations

Flow, transport, shock capturing, reacting chemistry, thermal non-equilibrium, turbulence: all passing regression tests on ParallelMesh at $10^{-9}$ tolerances

SerialMesh-dependent exceptions:
Shock fitting, transition integrals
Ongoing Work

• DofObject serialization, redistribution
• Adaptive coarsening
• Benchmarking, optimization
• Extend Parallel:: APIs from vector, set → map
• APIs for integro-differential terms

Any Questions?