An Unstructured Mesh Infrastructure for Massively Parallel Adaptive Simulation

S. Seol, T. Xie, M. Mubarak,
D. A. Ibanez, M. S. Shephard

Scientific Computation Research Center
Rensselaer Polytechnic Institute
Outline

Distributed Mesh Data Structure
- Distributed Mesh Representation
- Two-Level Mesh Partitioning
  - An approach to combine message passing and threading
- Migration, Dynamic Load Balancing and Ghosting

Applications
- Boundary Layer Mesh Adaptation
- SPR Error Estimator

Design and Implementations
- Generic Components
- Parallel Control Utility
Geometry-Based Analysis

- Geometry, Attribute: analysis domain
- Mesh: 0-3D topological entities and adjacencies
- Field: distribution of solution over mesh
- Common requirements: data traversal, arbitrarily attachable user data, data grouping, etc.
- Complete representation: store sufficient entities and adjacencies to get any adjacency in $O(1)$ time – critical for evolving meshes

**Background**

[Diagram of geometric model and mesh]
FMDB – A FASTMath iMesh/iMeshP Implementation

- Capability to partition mesh to multiple parts per process
Distributed Mesh Data Structure

Each part $P_i$ assigned to a processor
- Consists of mesh entities assigned to $i^{th}$ part.
- Uniquely identified by handle or id plus part number.
- Treated as a serial mesh with the addition of **part boundaries**
  - *Part boundary*: groups of mesh entities on shared links between parts.
  - *Part boundary entity*: duplicated entities on all parts for which they bound with other higher order mesh entities.
  - *Remote copy*: duplicated entity copy on non-local part.
Distributed Mesh Data Structure

Partition Object

- Basic unit to assign destination part id in mesh migration
  - Individual mesh entities
  - P-set - mesh entity set on part

- For partition object $x$, residence part operator $\mathcal{P}(x)$ returns a set of part id’s where $x$ exists based on adjacencies. E.g. $\mathcal{P}[M_1^0] = \{P_0, P_1, P_2\}$

- **Partition object graph**: weighted graph $G(V, E)$
  - Node $V$: partition object
  - Edge $E$: dependencies between graph nodes identified by adjacencies
  - Node and edge weights

A mesh part with 3 P-Sets  Partition object graph
Partition Model

- Partition Model: a conceptual model existing between a geometric model and distributed mesh
- **Goal**: mesh partitioning representation in topology for efficient mesh-based parallel operation support
- **Partition (model) entity**: a topological entity in the partition model, $P_i^d$, representing a group of mesh entities of dimension $d$ with the same residence parts.

![Partition model of previous mesh](image1)

![Partition classification in arrows](image2)
**Mesh Migration**

**Purpose:** Moving mesh entities between parts
- Dictated by operation - in swap and collapse it’s the mesh entities on other parts needed to complete the mesh modification cavity
- Entities to migrate are determined based on adjacencies

**Issues**
- A function of mesh representation w.r.t. adjacencies, P- set and arbitrary user data attached to them
  - Complete mesh representation can provide any adjacency without mesh traversal - a requirement for satisfactory efficiency
- Performance issues
  - Synchronization, communications, load balance and scalability
  - How to benefit from on-node thread communication (all threads in a processor share the same memory address space)
goals: localizing off-part mesh data to avoid inter-process communications for computations

ghost: read-only, duplicate entity copies not on part boundary including tag data

imeshp ghosting rule: triplet (ghost dim, bridge dim, # layers)
  - ghost dim: entity dimension to be ghosted
  - bridge dim: entity dimension used to obtain entities to be ghosted through adjacency
  - # layers: the number of ghost layers measured from the part boundary

E.g., to get two layers of region entities in the ghost layer, measured from faces on part boundary, use ghost_dim=3, bridge_dim=2, and # layers=2 (source: FASTMath iMeshP.h)
Ghosting

Ghosting Steps with Full Representation

- Ghost Dim: 2
- Bridge Dim: 1
- Num. Layers: 1

Ghosting rule

(A) Initial mesh

(B) Get entities to ghost

(C) Mark destination part id

(D) Exchange entities

(E) Final mesh
**Ghosting**

1-Layer Ghosting Scalability

- AAA mesh loaded onto single part per process
- H/W: Hopper Cray XE6 and KAUST BG/P
- Scaling factor decreases as inter-process comm. increases

*Abdominal Aortic Aneurysm (AAA) Mesh*
**Ghosting**

- **N-Layer ghosting creation on 1K parts with 1K processes**

<table>
<thead>
<tr>
<th>Test case</th>
<th>Machine</th>
<th>n=1</th>
<th>n=2</th>
<th>n=3</th>
<th>n=4</th>
<th>n=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA 133M</td>
<td>Cray XE6</td>
<td>E/ghosted</td>
<td>17M</td>
<td>37.9M</td>
<td>57.6M</td>
<td>76.5 M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time(s)</td>
<td>8.86</td>
<td>16.2</td>
<td>21.55</td>
<td>26.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$E_n$</td>
<td>1</td>
<td>1.21</td>
<td>1.15</td>
<td>1.52</td>
</tr>
<tr>
<td>BG/P</td>
<td>E/ghosted</td>
<td>17M</td>
<td>37.9M</td>
<td>57M</td>
<td>76M</td>
<td>93.7M</td>
</tr>
<tr>
<td></td>
<td>Time(s)</td>
<td>35.44</td>
<td>64.80</td>
<td>87.1</td>
<td>105.67</td>
<td>122.70</td>
</tr>
<tr>
<td></td>
<td>$E_n$</td>
<td>1</td>
<td>1.19</td>
<td>1.36</td>
<td>1.49</td>
<td>1.59</td>
</tr>
</tbody>
</table>

$T_1$: time consumed for ghosting 1 layer
$T_N$: time consumed for ghosting N\(^{th}\) layer
E/ghosted: number of ghosted entities
$E_n$: efficiency of N-layer ghosting ($T_1 \times (1 + \text{increase in E/ghosted})) / T_N$
Dynamic Load Balancing

- **Purpose**: to rebalance load imbalanced mesh during mesh modification
  - Equal “work load” with minimum inter-process communications

- **Two tools being used**
  - Zoltan Dynamic Services supporting multiple dynamic partitioners with general control of partition objects and weights.
  - *ParMA (Partitioning using Mesh Adjacencies)*: See C. Smith’s Presentation in Session MS51
Global and local graphs to support mesh partitions at extreme scale (>=100K parts)

Mesh and its initial partition (partition from 3 parts to 6 parts)
Multiple Parts Per Process

Goals
- Changing number of parts
- Dealing with problems with current graph-based partitioners that tend to fail on really large numbers of processors (See Slides 24-25)
- Architecture-aware two-level mesh partitioning (See Slide 27-28)

Multiple-Parts Per Process contained in Mesh Instance
- For effective manipulation, a mesh instance defined on each processor contains part handles assigned to the process

A 3D mesh in 4 parts per process (16 parts total)

(LEFT) Different color represents different part
(RIGHT) Different color represents different process
Boundary Layer Mesh Adaptation

Boundary Layer stacks in P-sets

- Mesh entities contained in a set are unique, and are not part of the boundary of any higher dimension mesh entities
- Migrate a set and constituting entities to another part together

Before Split

After Split

Split stack of edges
Boundary Layer Mesh Adaptation
A Parallel error estimation procedure based on Super-convergent Patch Recovery (SPR) scheme

Employs a local least square fitting scheme over the whole patch of elements surrounding a node

Requires complete field information of a nodal patch

- Problem: In distributed meshes, complete nodal patch of a part boundary vertex may be missing
- Solution: Create 1-layer of ghost regions to acquire off-node field information (regions with vertex bridge)
- After ghosting is carried out, no further communication among parts is required during error estimation
- Ghosts are thrown away once the error estimation process is complete
SPR Error Estimation

(a) Mesh
(b) Solution field
(c) Mesh after 1-layer ghosting
(d) Error field after SPR procedure applied on ghosted mesh
(e) Final size field
(f) Mesh after ghost deletion
Two-Level Mesh Partitioning

Hardware-Software Correspondence
- Part handles are accessible through mesh instance per node

Hybrid Programming Model
- The message passing (MPI) model between parts on different nodes
- The threads model between parts on the same node
Two Level Mesh Partitioning

Non-threaded FMDB v1.x

Threaded FMDB v2.x

inter-process part boundary

Proc $i$

Proc $j$

intra-process part boundary

$P_0$

$P_1$

$P_2$

off-node part boundary

Node $i$

Node $j$

on-node part boundary

$P_0$

$P_1$

$P_2$
Two Level Mesh Partitioning

Consequence:
- Demand for architecture-aware parallel utility with generic API

Goals of generic Parallel Control Utility (PCU)
- Support for architecture-aware data partitioning and communications
- Support for “mesh operation” - level parallelization
- Enhancing development productivity by hiding low-level parallel coding such as
  - H/W interactions
  - Message buffer management
  - Hybrid mode: MPI and thread models
  - Asynchronous communications within neighboring parts
  - Resource (memory and time) management and monitoring
- Reusability and maintenance
Parallel Control Utility

- Generic parallel utility necessitated by architecture-aware two-level partitioning
- IPComMan extension to account for high core count HPC
  - IPComMan: asynchronous MPI communications within neighboring processes with buffer management and message packing

Current Efforts

- Thread management
  - launching main function as threads in a process
- H/W topology querying and binding
  - Querying hardware topology
  - Binding s/w threads to h/w threads (processing units)
- Message passing between processes
  - All messages packed into buffers: improved MPI performance
  - Inter-node messaging by MPI
FMDB complies to iMesh/iMeshP spec

- Defined by the DOE SciDAC center on Frameworks, Algorithms, and Scalable Technologies for Mathematics to support unstructured meshes on parallel computers
- iMeshP focuses on supporting parallel interactions of serial mesh part (iMesh)
- FMDB represents a FASTMath unstructured implementation that supports parallel mesh adaptation

**Web:** [http://www.itaps.org](http://www.itaps.org)

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy Office of Advanced Scientific Computing Research
Closing Remarks

Status of iMeshP/FMDB
- Parallel mesh data structure with full support for mesh-level operations for adaptive simulations on a massively parallel computers

Future Directions
- Architecture-awareness: node-socket-core-processing unit
- Identifying optimal granularity and major h/w factors for max. scalability
- Interaction with other threaded/non-threaded parallel library
- Minimizing parallel overhead

More Information
- **Web**: [http://www.scorec.rpi.edu/FMDB](http://www.scorec.rpi.edu/FMDB)
- **Email**: fmdb@scorec.rpi.edu