PYRAMID: An Object-Based Library for Parallel Unstructured Adaptive Mesh Refinement

Charles D. Norton, John Z. Lou, and Thomas A. Cwik
National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology
http://hpc.jpl.nasa.gov/APPS/AMR
Modern... Simple... Efficient... Scalable...

*Technology Description*
An advanced software library supporting parallel adaptive mesh refinement in large-scale, adaptive scientific & engineering simulations.

*State-of-the-Art Design!*
- Efficient object-oriented design in Fortran 90 and MPI
- Automatic mesh quality control & dynamic load balancing
- Scalable to hundreds of processors & millions of elements

*Application Arena*
- Computer Modeling & Simulation Applications with complex geometry
- Electromagnetic and semiconductor device modeling
- Structural/Mechanical/Fluid dynamics applications

**John Z. Lou, Charles D. Norton, & Thomas A. Cwik**
High Performance Computing Systems and Applications Group
http://hpc.jpl.nasa.gov/APPS/AMR

Initial geometry courtesy of SCOREC (Rensselaer)
Pyramid Package Components

- **Components**
  - Parallel mesh I/O, partitioner, logical and physical adaptive refiners, mesh migration, and visualization

- **Development Structure**
  - Fortran 90: Core data structures and internals
  - C: Interface to ParMetis graph partitioner
  - MPI: Distributed-Memory communication
Development Issues

- **Parallel Unstructured AMR Scheme**
  - Logical AMR: Iterative scheme defining refinement pattern (with mesh quality control)
  - Physical AMR: Locally refine coarse elements based on logical refinement

- **Parallel Dynamic Load Balancing Strategy**
  - Repartition weighted logical mesh, migrate coarse elements, and perform local physical refinement

- **Modular Design**
  - Performance and abstraction features of Fortran 90
Our Parallel AMR Process

- **Organization**
  - Partitioning, Adaptive Refinement, Load Balancing, Mesh Migration, and Element Quality Control

```
Initial Mesh Partitioning

Application Computation

Estimated Error > Tolerance?  
  Y
   
   Adaptive Refinement (logical phase)

  N
   
   Load Balancing?  
     Y
```

```
Addressed by AMR Quality Control

Mesh Improvement

Mesh Smoothing?  
  N
   
   Adaptive Refinement (physical phase)

  Y
   
   Mesh Repartition and migration
```
Technology

- Fortran 90/95 Features Modernize Programming

  Modules
  Encapsulate data and routines across program units

  Generic Interfaces
  One call can perform different actions based on types

  Array Syntax
  Simplifies whole array, and array subset, operations

  Use-Association
  Controls access to module content

  Derived Types
  User-defined types supporting abstractions in programming

  Pointers/Allocatable Arrays
  Supports flexible/dynamic data structures

  Backward compatible with Fortran 77

FOR MORE INFO...

Fortran 90 Programming. Ellis, Philips, & Lahey; Addison Wesley, 1994
http://hpc.jpl.nasa.gov/PEP/nortonc/oof90.html
• A Minimal PYRAMID Program
  – Initialization Section
    • Optional arguments override defaults

PROGRAM pyramid_example
USE pyramid_module
implicit none
  ! Statements omitted...
type (mesh), dimension(2) :: meshes
call PAMR_INIT()
call PAMR_LOAD_MESH_PARALLEL( meshes(1), in_file )
call PAMR_REPARTITION( meshes(1) )
  ! Adaptive refinement loop...
call PAMR_ELEMENT_COUNT( meshes(2) )
call PAMR_VISUALIZE( meshes(2), "visfile.plt" )
call PAMR_FINALIZE( mpi_active = .true. )
END PROGRAM pyramid_example
- A Minimal PYRAMID Program
  - Adaptive Refinement

PROGRAM pyramid_example
  ! Adaptive refinement loop...
  do i = 1, refinement_level
    call PAMR_ERROR_EST( meshes(1), &
                         meshes(2) )
    call PAMR_LOGICAL_AMR( meshes(1) )
    call PAMR_REPARTITION( meshes(1) )
    call PAMR_PHYSICAL_AMR( meshes(1), meshes(2) )
  end do
END PROGRAM pyramid_example

- Users must specify their error estimation method
- Mesh hierarchies can be defined
Object-Based Access to Data Structure

- Explicit reference to element coordinates is complicated

```fortran
  type (mesh) :: this
  real, dimension(3) :: xyz_pos
  xyz_pos = this%nodes(this%elements(2)%node_index(1))%coord
```

- PYRAMID simplifies such references

```fortran
  type (mesh) :: this
  real, dimension(3) :: xyz_pos
  real, dimension(3,4) :: all_pos
  real, dimension(3,3,4) :: n_normal
  xyz_pos = PAMR_ELEMENTCOORD(this, element_index=2, &
                              node_index=1)
  all_pos = PAMR_ELEMENTCOORD(this, element_index=2)
  ! Access signed local normal basis for all faces
  n_normal = PAMR_FACE_NORMALBASIS(this, element_index=3)
```
Numerous User-Driven Commands Are Included

- Initialization, Mesh I/O, Termination, Adaptive Refinement, Repartitioning, Data Migration, Visualization, Data Structure Access, Mathematical, and Auxiliary
- Almost every command contains optional arguments for use customization

PAMR_CURRENT_TIME  PAMR_FACE_INDEX  PAMR_LOAD_MESH_SERIAL
PAMR_DEFINE_MESH_TERMS PAMR_FACEEDGE_ID  PAMR_LOAD_MESH_PARALLEL
PAMR_ELAPSED_TIME PAMR_FACEEDGE_INDEX  PAMR_LOGICAL_AMR
PAMR_ELEMENT_CENTROID PAMR_FACE_NORMALINDEX
PAMR_ELEMENTCOORD  PAMR_FACEUNITNORMAL
PAMR_ELEMENT_COUNT  PAMR_FINALIZE
PAMR_ELEMENT_ID  PAMR_GET_EDGE_TERMS
PAMR_ELEMENT_VOLUME  PAMR_GET_ELEMENT_TERMS
PAMR_ERROR_BST  PAMR_GET_FACE_TERMS
PAMR_FACE_AREA  PAMR_GET_NODE_TERMS
PAMR_FACE_CENTROID  PAMR_INIT
PAMR_FACE_COORD  PAMR_LOAD_MESH_COMP
PAMRaddError  PAMR_SET_EDGE_TERMS
PAMR_setElementTerms  PAMR_SET_ELEMENT_TERMS
PAMR_SET_FACE_TERMS
PAMR_SET_NODE_TERMS
PAMR_VISUALIZE

- Most commands are generic based on the mesh component applied
Dynamic Load Balancing with ParMetis

- ParMetis gives partitioning, PYRAMID performs migration
- Migration handles irregular communication patterns with a scalable and efficient non-blocking algorithm

- We are investigating Zoltan (Sandia National Labs) as an additional option for partitioning
- **Automatic Mesh Quality Control**
  - Modify coarse element refinement if successive refinements cause poor aspect ratios

- Controls quality at the expense of additional elements
Technology

- Automatic Mesh Quality Control
  - Benefit of quality control applied to triangular elements
Technology

- **Automatic Mesh Quality Control**
  - Benefit of quality control applied to tetrahedral elements

Note: Tecplot shows some edges in the backplane that do not exist in the mesh...
Technology

- Large Scale Parallel Mesh Generation
  - Specify uniform error for generation from coarse meshes
Performance

- Pentium III Beowulf Cluster vs. SGI O2K Parallel AMR
  - O2K scales well although the processor is slower than the 800 Mhz Beowulf PIII
  - Beowulf competes well, but performance is limited by 100 BaseT network
Performance

- Pentium III Beowulf Cluster vs. SGI O2K Parallel AMR
  - O2K outperforms Beowulf across all refinement levels
  - Beowulf shows larger percentage improvement as problem size grows

- O2K is an order of magnitude slower from level 2 to level 3
- Performance will vary based on mesh geometry

Note: Simulation uses 32 processors
New migration algorithms applied
Performance

- Pentium III Beowulf Cluster vs. SGI O2K Parallel AMR
  - Migration algorithm improvements benefit Beowulf significantly
  - Network still hinders Beowulf with increased problem size

![Graph showing performance comparison between Beowulf and SGI O2K](image)

- Our new migration algorithms are completely non-blocking, scalable, and utilize full-duplex channels when available
- We estimate O2K has a 7 times network speed advantage over 100 BaseT Beowulf

<table>
<thead>
<tr>
<th>Level</th>
<th>Beowulf Refinement</th>
<th>SGI Refinement</th>
<th>Beowulf Migration</th>
<th>SGI Migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>0.15</td>
<td>0.04</td>
<td>3.33</td>
<td>0.23</td>
</tr>
<tr>
<td>Level 2</td>
<td>0.47</td>
<td>0.49</td>
<td>3.7</td>
<td>0.31</td>
</tr>
<tr>
<td>Level 3</td>
<td>16.24</td>
<td>10.24</td>
<td>17.06</td>
<td>5.59</td>
</tr>
</tbody>
</table>

Adaptively Refined Earthquake Mesh on 8 PEs
T3E decommissioned prior to this simulation
Performance

- Irregular Data Communication
  - Migration requires irregular, but predictable, data movement that varies in size and destination

Circular-Shift “MPI_SENDRECV(...)

- All processors inspect all of the data
- “Guarantees” handling of cyclic deadlock dependencies
- Irregular data sizes affect pipelined flow performance
- MPI implicit buffering, due to poor pipeline structure, leads to poor performance

Direct Data Transfers

- Processors send/receive specific messages
- Send continuously while checking for receives
- “Arbitrary” message ordering can flood the network switch, leading to poor performance
Performance

- Irregular Data Communication
  - Reduction schemes can improve performance, if implemented with care...

**MPI Reduce to 0 with Broadcast Scheme**
- Not scalable and very inefficient for large data sets

**MPI Reduce to Leader with Subset Broadcast Scheme**
- More scalable and efficient, but still requires multiple broadcasts at each tree level

**Exchange with Subset Broadcast**
- Same characteristics as above

Note: Broadcasts simplify handling cases where the number of processors is not a power of two

Scatter collective PE data

Gather partial PE data

Exchange within levels

Broadcast within exchange subset
Performance

- Irregular Data Communication
  - Reduction schemes can improve performance, if implemented with care...

Our Algorithm Improvements

- Maximize exchanges at each level without repeated calculations
- Reduce data volume at each level with full-duplex communication
- Much fewer broadcasts are required to support an arbitrary number of processors
- Processors which do not contribute to the calculation at a given level are idle

---

Comparison of Old and New Migration Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Brocaff</th>
<th>Old Alg</th>
<th>New Alg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time</td>
<td></td>
<td>563.1</td>
<td>61.88</td>
</tr>
<tr>
<td>(sec)</td>
<td></td>
<td>442.54</td>
<td>79.23</td>
</tr>
</tbody>
</table>

Maximum number of pairwise exchanges are performed
Performance

- Pentium III Beowulf Cluster vs. SGI O2K Migration
  - Beowulf performs well, but network dominates with increasing processors
  - O2K is also affected for large (>30MB) messages

<table>
<thead>
<tr>
<th>Pentium III Beowulf vs. SGI O2K Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
</tr>
<tr>
<td>2500</td>
</tr>
<tr>
<td>2000</td>
</tr>
<tr>
<td>1500</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Wall Clock Time (sec.)</td>
</tr>
<tr>
<td>4 Pes</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>SGI O2K</td>
</tr>
<tr>
<td>Beowulf</td>
</tr>
</tbody>
</table>

Artery Segment with 1.1 Million Elements

Adaptively Refined Artery Segment with 2 Million Elements

Note: New migration algorithms are applied
Next Generation Features

- **Development is User-Driven**
  - Used for adaptive refinement of multi-scale meshes for active device modeling

- **Additional Work Directions**
  - User-controllable boundary zone definition
  - Interpolation methods among mesh levels
  - Straightforward approaches for incorporating error estimation
  - Coarsening

- **Demonstration Release**
  - hpc.jpl.nasa.gov/APPS/AMR

Note: Functionality is limited in demo release