Parallel AMR Application Development with the SAMRAI Library

Andy Wissink & Rich Hornung

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
www.llnl.gov/CASC/SAMRAI

February 13, 2005
SIAM CSE Meeting

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.
SAMRAI Project Members

Noah Elliott

Brian Gunney

Rich Hornung

Craig Kapfer

Steve Smith

Photo unavailable

Andy Wissink
- AMR dynamically increases spatial and temporal grid resolution to resolve important local features.

- SAMRAI is an object-oriented C++ framework that supports applications investigating multi-scale phenomena.

- Framework provides high-level reusable code and algorithms shared across a variety of applications.

**SAMRAI**

Structured Adaptive Mesh Refinement Application Infrastructure

- Two beam laser interaction
- ALPS
- Richtmeyer-Meshkov instability
- Hybrid Continuum Particle
- Richtmeyer-Meshkov instability
- ALE-AMR
Structured AMR (SAMR) employs a “patch” hierarchy

- Hierarchy of nested “patch” levels → low overhead mesh description
- Data mapped to patches → simple model of data locality
- Patches cover logically rectangular index space

Berger, Oliger, Colella
SAMRAI manages many of the complexities of SAMR implementations

**SAMRAI Provides:**
- Parallel communication (MPI)
- Dynamic gridding support
- Inter-patch data transfer operations (copy, coarsen, refine, time int, …)
- Solver interfaces for SAMR data (PetSc, hypre, pvode)
- Checkpointing and restart (HDF5)
- Visualization support (VisIt)

**User provides:**
- (serial) numerical routines for individual patches
- Composition of SAMRAI classes to implement desired algorithm.
SAMRAI is an object-oriented “toolbox” of classes for SAMR application development.

A SAMRAI "patch" contains all data on a box region of the computational mesh.
SAMRAI *Variable* and *PatchData* delineate “static” and “dynamic” data concepts

<table>
<thead>
<tr>
<th><strong>Solution algorithms and variables tend to be static</strong></th>
<th><strong>Mesh and data objects tend to be dynamic</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variable object</strong></td>
<td><strong>PatchData object</strong></td>
</tr>
<tr>
<td>— defines a data quantity; type, (centering), (depth)</td>
<td>— represents data on a “box”</td>
</tr>
<tr>
<td>— attributes:</td>
<td>— attributes:</td>
</tr>
<tr>
<td>— name (string)</td>
<td>— box</td>
</tr>
<tr>
<td>— unique instance id (int)</td>
<td>— ghost cell width</td>
</tr>
<tr>
<td>— <em>Variable</em> objects generally persist throughout computation</td>
<td>— Attributes facilitate construction of communication dependencies</td>
</tr>
<tr>
<td></td>
<td>— <em>PatchData</em> objects are created and destroyed as mesh changes</td>
</tr>
</tbody>
</table>
**Comm. Algorithm and Schedule:** “static” and “dynamic” communication concepts

- **Solution algorithms and variables** tend to be **static**
  - **Communication Algorithm**
    - describes data transfer phase of computation
    - expressed using variables, operators, ...
    - independent of mesh
    - typically persists throughout computation
- **Mesh and data objects** tend to be **dynamic**
  - **Communication Schedule**
    - manages details of data movement on mesh
    - created by communication algorithm
    - depends on mesh
    - re-created when mesh changes

**Compare with...**

- **Variable**
  - defines a data quantity independent of mesh
  - usually persists throughout computation
- **PatchData**
  - represents data on a “box”
  - created and destroyed as mesh changes
Communication schedules create and store data dependencies

- Amortize cost of creating send/receive sets over multiple communication cycles

- Data from various sources packed into single message stream
  - supports complicated variable-length data
  - one send per processor pair (low latency)

packStream(...);

message buffer

MPI send
Adaptive Mesh and Algorithm Refinement (AMAR) refines mesh and numerical model

- AMR is used to refine continuum calculation and focus particles
- Algorithm switches to discrete atomistic method to include physics absent in continuum model

Particles resolve molecular-scale dynamics of mixing region

Pre-existing particle data structures coupled to SAMRAI via patch data interface

```
DsmcPatchData* particles = patch->getPatchData( . . . );
particles->advance(dt);
```
ALE-AMR combines ALE integration with AMR

- Advantages of ALE (multiple materials, moving interfaces)
- Advantages of AMR (dynamic addition & removal of mesh points)

Deforming grids in ALE-AMR managed by specializing SAMRAI grid geometry

Manages “index space” coordinates

Manages “physical space” coordinates

PatchHierarchy \rightarrow GridGeometry

CartesianGrid Geometry \rightarrow DeformingGrid Geometry

Utilizes all the other features of SAMRAI:
- parallel communication
- adaptive gridding
- solver interfaces
- etc.
SAMRAI “multiblock” capability supports multiple patch hierarchies

- Multiple index spaces stitched together
  - translations & rotations
  - transparent data communication between hierarchies

Five AMR patch hierarchies meet at single point
SAMRAI supports Cartesian Embedded Boundary grid representations

- Constructing body-fitted logically rectangular grids is tedious and expensive.
- Embedded boundary grids constructed automatically in SAMRAI
  - Built from polygons or from surface triangulation using CUBES

Example embedded boundary mesh constructed with CUBES

M. Berger, Courant Inst./NASA Ames
SAMRAI index data supports embedded boundary as “patch data”

- IndexVariable and IndexData classes manage data quantities on irregular index sets

```cpp
IndexVariable<TYPE> ivar("name")
IndexData<TYPE> idata(Box& box, ghosts)
```

“TYPE”

Required methods

```
TYPE()
TYPE& operator=(const TYPE&)
getDataStreamSize(Box&)
packStream(...)
unpackStream(...)
```

CutCell type describes internal boundary and state information along boundary
AUDIM applying adaptive meshing for CFD Urban Dispersion Modeling

“Shapetile” (Manhattan)

SAMRAI
Adaptive Meshing, embedded boundary representation, parallel AMR support

FEM3MP
FE based CFD Dispersion model

Eleven (Overture)
Polygonal Geometry Representation

polygons

Triangulated surf grid

AMR Emb Boundary Vol Mesh

CUBES (Berger)

Adaptive CFD simulation
Immersed boundary methods model fluid structure interactions

Griffith, Peskin (NYU) are developing an electrical-mechanical heart model combining immersed boundaries and AMR (SAMRAI)
Communication specialized for finite-element based operations

- **Transaction**
  - void packstream()
  - void unpackstream()

- **CopyTransaction**

- **SumTransaction**

- **Ghost Cell Copy**
  - Typically used for Finite Difference and Finite Volume discretizations

- **Node sum**
  - Used for element assembly in Finite Element discretization

- **“Sum” transactions used for finite element calculations**
  - Used across multiple levels for AMR
  - Node & Edge sum available
SAMRAI supports applications on large parallel platforms

Scaled Euler Hydrodynamics
IBM Blue Pacific

Scaled Linear Advection
Linux MCR Cluster

Non-adaptive

Adaptive

Concluding remarks

- AMR is an important technology for large-scale science & engineering problems that require greater resolution of localized features

- New applications require expansion of current AMR methodologies
  - Model refinement in addition to grid refinement
  - Support for variety of data representations and non-Cartesian grids
  - Complex geometries
  - Efficiency on large-scale parallel architectures

- AMR libraries must effectively interoperate with other software packages – solver libraries, grid generation packages, etc.