Enzo-P / Cello
Formation of the First Galaxies

James Bordner¹  Michael L. Norman¹  Brian O’Shea²

¹University of California, San Diego
San Diego Supercomputer Center

²Michigan State University
Department of Physics and Astronomy

Blue Waters Symposium 2013
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Introducing Enzo-P / Cello

Our group actively develops two related parallel applications:

**Enzo**: astrophysics / cosmology application
- patch-based adaptive mesh refinement (AMR)
- MPI or MPI/OpenMP
- almost 20 years development

**Enzo-P / Cello**: “Petascale” fork of Enzo code
- “forest of octrees” AMR
- Charm++ or MPI
- \(\approx\) 3 years development
- work in progress–AMR just coming online
Enzo’s strengths

- Spans multiple application domains
  - astrophysical fluid dynamics
  - hydrodynamic cosmology
- Rich multi-physics capabilities
  - fluid, particle, gravity, radiation, . . .
- Extreme resolution range
  - 34 levels of refinement by 2!
- Active global development community
  - \( \approx 25 \) developers

[ John Wise ]
Enzo’s struggles

- Memory usage
  - $\approx 1.5\text{KB/patch}$ (MPI/OpenMP helps)
  - memory fragmentation

- Mesh quality
  - 2-to-1 constraint can be violated
  - asymmetric mesh for symmetric problem

- Load balancing
  - difficulty maintaining parent-child locality

- Parallel scaling
  - AMR overhead dominates computation

[ Tom Abel, John Wise, Ralf Kaehler ]
Enzo’s pursuit of scalability

- Enzo was born in early 1990’s
- “Extreme” meant 100 processors
- Continual scalability improvements
  - MPI/OpenMP parallelism
  - “neighbor-finding” algorithm
  - I/O optimizations
- Further improvement getting harder
  - increasing scalability requirements
  - easy improvements made already
- Motivates concurrent rewriting
  - Enzo-P “Petascale” Enzo fork
  - Cello AMR framework
Enzo-P / Cello design overview

- **Charm++** parallelism
  - asynchronous, data-driven
  - latency tolerant
  - dynamic load balancing
  - checkpoint / restart

- **Octree**-based AMR
  - “forest” for root mesh
  - easier to implement
  - scalability advantages
  - fast neighbor-finding
Some advantages of patch-based AMR

- Flexible patch size and shape
  - improved refinement efficiency
- Larger patches
  - smaller surface/volume ratio
  - reduced communication
  - amortized loop overhead
- Fewer patches
  - reduced AMR meta-data
Some advantages of octree-based AMR

- Fixed block size and shape
  - simplified load balancing
  - dynamic memory reuse
- More blocks
  - more parallelism available
- Smaller nodes
  - reduced AMR meta-data
- Compute only on leaf nodes
  - less communication
Enzo’s AMR data structure

- Patches assigned to MPI processes
- Refinement patches created on root patch process
- Load balancing relocates refinement patches
- Patch data (grid, particle) are distributed
- Replicated AMR hierarchy structure
Enzo-P / Cello’s AMR data structure

- Each block is a Charm++ chare
- Blocks initially mapped to root node process
- Charm++ load balances
- AMR hierarchy structure is fully distributed
Charm++ program structure

- Charm++ program
  - Charm++ objects are *shares*
  - invoke remote *entry methods*
  - communicate via *messages*

- Charm++ runtime system
  - schedules entry methods
  - maps shares to processors
  - migrates shares to balance

- Additional scalability features
  - checkpoint / restart
  - sophisticated DLB strategies
Charm++ collections of chares

**Chare Array**
- distributed array of chares
- migrateable elements
- flexible indexing

**Chare Group**
- one chare per processor (non-migrateable)

**Chare Nodegroup**
- one chare per node (non-migrateable)
Cello implementation options using Charm++

1. Singleton chares
   - unlimited hierarchy depth
   - tedious to program
   - limited Charm++ support

2. Chare array
   - efficient: single access
   - restricted hierarchy depth
Charm++ entities in Enzo-P / Cello

- “mainchare” called at program startup
- Simulation chare group holds global data
- Block chare array defines forest of octrees
Current Enzo-P / Cello control flow

1. Startup
2. Initialize
3. Mesh creation
4. Ghost refresh
5. Computation
6. Mesh adaptation
Charm++ supports user-defined array indices
Default array indices are 3 integers
Cello indexing for Block arrays:
- 10 × 3 bits for forest indices
- 20 × 3 bits for the octree encoding
- 6 bits for the block depth
Up to 1024³ array of octrees
Up to 21 octree levels
## Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- *Quiescence detection* between steps
- Keep track of neighbors and children
Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- *Quiescence detection* between steps
- Keep track of neighbors and children
Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- Quiescence detection between steps
- Keep track of neighbors and children
Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- *Quiescence detection* between steps
- Keep track of neighbors and children
Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- *Quiescence detection* between steps
- Keep track of neighbors and children
Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- *Quiescence detection* between steps
- Keep track of neighbors and children
Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- *Quiescence detection* between steps
- Keep track of neighbors and children
Cello mesh generation

- Begins with the forest root grid
- Proceeds level-by-level
- Blocks evaluate refinement criteria
  - if refine, create child blocks
  - if coarsen, notify parent block
- Refine can violate 2-1 constraint
  - tell coarse neighbors to refine
  - may recurse
- *Quiescence detection* between steps
- Keep track of neighbors and children
Cello AMR ghost zone refresh

Intra-level refresh

1. FaceBlock loads face cells
2. Charm++ entry method send
3. FaceBlock stores ghost cells

Fine-to-coarse refresh

1. FaceBlock coarsens face cells
2. Charm++ entry method send
3. FaceBlock stores ghost cells

Coarse-to-fine refresh

1. FaceBlock loads face cells
2. Charm++ entry method send
3. FaceBlock interpolates ghost cells
## Summary

<table>
<thead>
<tr>
<th>Parallelization</th>
<th>Enzo</th>
<th>Enzo-P / Cello</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMR</td>
<td>MPI/OpenMP</td>
<td>Charm++</td>
</tr>
<tr>
<td>AMR structure</td>
<td>patch-based</td>
<td>tree-based</td>
</tr>
<tr>
<td>Block sizes</td>
<td>replicated</td>
<td>distributed</td>
</tr>
<tr>
<td>×1000 variation</td>
<td>level-parallel</td>
<td>constant</td>
</tr>
<tr>
<td>Task scheduling</td>
<td>patch migration</td>
<td>dependency-driven</td>
</tr>
<tr>
<td>Load balancing</td>
<td>checkpoint/restart</td>
<td>Charm++</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td></td>
<td>Charm++</td>
</tr>
</tbody>
</table>

http://cello-project.org

NSF PHY-1104819, AST-0808184