Enzo-E/Cello astrophysics and cosmology
Adaptive mesh refinement astrophysics using Charm++

James Bordner, Michael L. Norman

University of California, San Diego
San Diego Supercomputer Center

18th Annual Workshop on Charm++ and Its Applications
2020-10-20/21
Scientific questions in astrophysics and cosmology

- Cosmological volumes, galaxy clusters [750 Mpc]
- Turbulence [scale-free]
- Stellar Common Envelope [10^{-6} pc]
- First Star Formation [0.01 pc]
- Sample of first galaxies and reionization [5 Mpc]
- First galaxies [10 kpc]

[John Wise]
Simulations require modelling multiple physics phenomena

**Physics Equations:** *mathematical models*

- Gravity ($\nabla^2 \Phi = 4\pi G \rho$)
- Hydrodynamics (Euler equations)
- Chemistry/cooling
- MHD
- Cosmological expansion . . .

**Numerical Methods**

- Linear solvers (Krylov subspace, multigrid, composite)
- modified PPM
- Grackle chemistry/cooling
- VL+CT MHD . . .

**Data Structures**

- Adaptive mesh refinement (array-of-octrees)
- Eulerian fields
- Lagrangian particles

**Parallel Runtime System**

- dynamic task scheduling
- data-driven execution
- asynchronous
The Enzo-E/Cello AMR Charm++ astrophysics application

Numerical methods are required for solving the physics equations

**Physics Equations:** mathematical models

- Gravity ($\nabla^2 \Phi = 4\pi G \rho$)
- Hydrodynamics (Euler equations)
- Chemistry/cooling
- MHD
- Cosmological expansion . . .

**Numerical Methods** approximate and solve Enzo-E

- Linear solvers (Krylov subspace, multigrid, composite)
- modified PPM
- Grackle chemistry/cooling
- VL+CT MHD . . .

**Data Structures** computer representation Cello

- Adaptive mesh refinement (array-of-octrees)
- Eulerian fields
- Lagrangian particles

**Parallel Runtime System** distribute data and computation Charm++

- dynamic task scheduling
- data-driven execution
- asynchronous
The Enzo-E/Cello AMR Charm++ astrophysics application

Parallel methods are enabled by distributed data structures

Physics Equations: mathematical models
- Gravity ($\nabla^2 \Phi = 4\pi G \rho$)
- Hydrodynamics (Euler equations)
- Chemistry/cooling
- MHD
- Cosmological expansion

Numerical Methods: approximate and solve
- Linear solvers (Krylov subspace, multigrid, composite)
- modified PPM
- Grackle chemistry/cooling
- VL+CT MHD

Data Structures: computer representation
- Adaptive mesh refinement (array-of-octrees)
- Eulerian fields
- Lagrangian particles

Parallel Runtime System: distribute data and computation
- dynamic task scheduling
- data-driven execution
- asynchronous
Charm++ provides the support for running on large-scale HPC platforms

Physics Equations: mathematical models

- Gravity ($\nabla^2 \Phi = 4\pi G \rho$)
- Hydrodynamics (Euler equations)
- Chemistry/cooling
- MHD
- Cosmological expansion . . .

Numerical Methods approximate and solve

- Linear solvers (Krylov subspace, multigrid, composite)
- modified PPM
- Grackle chemistry/cooling
- VL+CT MHD . . .

Data Structures computer representation

- Adaptive mesh refinement (array-of-octrees)
- Eulerian fields
- Lagrangian particles

Parallel Runtime System distribute data and computation

- dynamic task scheduling
- data-driven execution
- asynchronous
Cosmological simulations with Enzo-E/Cello

- array-of-octrees
- blocks of data
- chare array

- dark matter
  - particle data
  - collisionless
  - CIC gravity

- baryonic matter
  - field data
  - PPM hydro
  - flux-correction
Cosmological simulations with Enzo-E/Cello

- adaptive mesh
  - array-of-octrees
  - blocks of data
  - chare array

- dark matter
  - particle data
  - collisionless
  - CIC gravity

- baryonic matter
  - field data
  - PPM hydro
  - flux-correction
Cosmological simulations with Enzo-E/Cello

- Adaptive mesh
  - array-of-octrees
  - blocks of data
  - chare array

- Dark matter
  - particle data
  - collisionless
  - CIC gravity

- Baryonic matter
  - field data
  - PPM hydro
  - flux-correction
Field and particle data communication

- Field data exchange
  - send Field face data when available
  - count face data messages received
  - last receive triggers computation

- Particle migration
  - scatter across $4^3$ pointer array
  - send to associated neighbors
  - gather incoming particles
Field and particle data communication

- **Field data exchange**
  - send Field face data when available
  - count face data messages received
  - last receive triggers computation

- **Particle migration**
  - scatter across $4^3$ pointer array
  - send to associated neighbors
  - gather incoming particles
Field and particle data communication

**Field data exchange**
- send Field face data when available
- count face data messages received
- last receive triggers computation

**Particle migration**
- scatter across $4^3$ pointer array
- send to associated neighbors
- gather incoming particles
Recent work has focused on scalable linear solvers

- **Krylov subspace methods**
  - CG (symmetric), BiCG-STAB (nonsymmetric)
  - easy to implement (basic linear algebra)
  - poor algorithmic scalability w/o preconditioning
  - communication intensive

- **Multigrid methods**
  - MG V-cycle
  - harder to implement (involves coarse blocks)
  - better algorithmic scalability
  - method limited to uniform meshes

- **Composite methods**
  - HG (Reynolds): multigrid-preconditioned Krylov
  - DD (Norman): domain-decomposition
Recent work has focused on scalable linear solvers

- **Krylov subspace methods**
  - CG (symmetric), BiCG-STAB (nonsymmetric)
  - easy to implement (basic linear algebra)
  - poor algorithmic scalability w/o preconditioning
  - communication intensive

- **Multigrid methods**
  - MG V-cycle
  - harder to implement (involves coarse blocks)
  - better algorithmic scalability
  - method limited to uniform meshes

- **Composite methods**
  - HG (Reynolds): multigrid-preconditioned Krylov
  - DD (Norman): domain-decomposition
Recent work has focused on scalable linear solvers

- **Krylov subspace methods**
  - CG (symmetric), BiCG-STAB (nonsymmetric)
  - easy to implement (basic linear algebra)
  - poor algorithmic scalability w/o preconditioning
  - communication intensive

- **Multigrid methods**
  - MG V-cycle
  - harder to implement (involves coarse blocks)
  - better algorithmic scalability
  - method limited to uniform meshes

- **Composite methods**
  - HG (Reynolds): multigrid-preconditioned Krylov
  - DD (Norman): domain-decomposition
Recent work has focused on scalable linear solvers

- **Krylov subspace methods**
  - CG (symmetric), BiCG-STAB (nonsymmetric)
  - easy to implement (basic linear algebra)
  - poor algorithmic scalability w/o preconditioning
  - communication intensive

- **Multigrid methods**
  - MG V-cycle
  - harder to implement (involves coarse blocks)
  - better algorithmic scalability
  - method limited to uniform meshes

- **Composite methods**
  - HG (Reynolds): multigrid-preconditioned Krylov
  - DD (Norman): domain-decomposition
Issue #1: scalable gravity
The Enzo-E domain decomposition solver DD

1. **EnzoSolverMg0** for root-level solve
   - demonstrated good parallel scalability
   - tested to $N_0 = 2048^3$ on $P = 131K$ BW fp-cores

2. **EnzoSolverBiCgStab** for “tree-solves”
   - use root-level solution for boundary conditions
   - no communication between root block domains

3. **EnzoSolverJacobi** for smoothing
   - smooths discontinuities across domain boundaries
   - previously available as multigrid smoother
Issue #1: scalable gravity
The Enzo-E domain decomposition solver DD

1. **EnzoSolverMg0** for **root-level solve**
   - demonstrated good parallel scalability
   - tested to $N_0 = 2048^3$ on $P = 131K$ BW fp-cores

2. **EnzoSolverBiCgStab** for “tree-solves”
   - use root-level solution for boundary conditions
   - no communication between root block domains

3. **EnzoSolverJacobi** for **smoothing**
   - smooths discontinuities across domain boundaries
   - previously available as multigrid smoother
1. EnzoSolverMg0 for root-level solve
   - demonstrated good parallel scalability
   - tested to $N_0 = 2048^3$ on $P = 131K$ BW fp-cores

2. EnzoSolverBiCgStab for “tree-solves”
   - use root-level solution for boundary conditions
   - no communication between root block domains

3. EnzoSolverJacobi for smoothing
   - smooths discontinuities across domain boundaries
   - previously available as multigrid smoother
Issue #1: scalable gravity
The Enzo-E domain decomposition solver DD

1. **EnzoSolverMg0** for **root-level solve**
   - demonstrated good parallel scalability
   - tested to $N_0 = 2048^3$ on $P = 131K$ BW fp-cores

2. **EnzoSolverBiCgStab** for **“tree-solves”**
   - use root-level solution for boundary conditions
   - no communication between root block domains

3. **EnzoSolverJacobi** for **smoothing**
   - smooths discontinuities across domain boundaries
   - previously available as multigrid smoother
Issue #1: scalable gravity
One cycle of a DM-only cosmology simulation using DD
## Issue #2: robust refresh

Implementing DD Solver uncovered communication issues

<table>
<thead>
<tr>
<th>Previous Refresh</th>
<th>Current Refresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>- send face data when available</td>
<td>- sends face data when available</td>
</tr>
<tr>
<td>- copy data to ghosts when received</td>
<td>- 1. receives but not ready (\Rightarrow) copy data to buffer</td>
</tr>
<tr>
<td>- race condition: may not be ready!</td>
<td>- 2. becomes ready (\Rightarrow) copy buffers to ghosts</td>
</tr>
<tr>
<td>- extra synchronization added for correctness</td>
<td>- 3. receives and ready (\Rightarrow) copy data to ghosts</td>
</tr>
</tbody>
</table>
Issue #2: robust refresh
Implementing DD Solver uncovered communication issues

**Previous Refresh**
- send face data when available
- copy data to ghosts when received
- race condition: may not be ready!
- extra synchronization added for correctness

**Current Refresh**
- sends face data when available
- 1. receives but not ready ⇒ copy data to buffer
- 2. becomes ready ⇒ copy buffers to ghosts
- 3. receives and ready ⇒ copy data to ghosts
Issue #2: robust refresh
Implementing DD Solver uncovered communication issues

**Previous Refresh**
- send face data when available
- copy data to ghosts when received
- race condition: may not be ready!
- extra synchronization added for correctness

**Current Refresh**
- sends face data when available
- 1. receives but not ready $\Rightarrow$ copy data to buffer
- 2. becomes ready $\Rightarrow$ copy buffers to ghosts
- 3. receives and ready $\Rightarrow$ copy data to ghosts
AMR cosmology simulations with DM + gas

- Enzo-E can run DM-only AMR simulations
- and DM + gas non-AMR simulations
- but DM + gas with AMR leads to non-physical behavior

clue: always at refinement-level boundaries
AMR cosmology simulations with DM + gas

- Enzo-E can run DM-only AMR simulations
- and DM + gas non-AMR simulations
- but DM + gas with AMR leads to non-physical behavior

clue: always at refinement-level boundaries
AMR cosmology simulations with DM + gas

- Enzo-E can run DM-only AMR simulations
- and DM + gas non-AMR simulations
- but DM + gas with AMR leads to non-physical behavior

clue: always at refinement-level boundaries
Issue #3: flux-correction

Conserved quantities were not conserved at refinement jumps

- consider two fine blocks adjacent to a coarse block
- hydro computations depend on ghost values
- ghost values available from previous refresh
- fluxes at both fine and coarse faces
- conservation requires consistent fluxes
- not consistent in general
- apply a “flux-correction” step
- update coarse values along face so fluxes match
- requires coarse-to-fine block communication
Issue #3: flux-correction
Conserved quantities were not conserved at refinement jumps

- Consider two fine blocks adjacent to a coarse block.
- Hydro computations depend on ghost values.
- Ghost values available from previous refresh.

- Fluxes at both fine and coarse faces.
- Conservation requires consistent fluxes.
- Not consistent in general.
- Apply a “flux-correction” step.
- Update coarse values along face so fluxes match.
- Requires coarse-to-fine block communication.
Issue #3: flux-correction
Conserved quantities were not conserved at refinement jumps

- Consider two fine blocks adjacent to a coarse block
- Hydro computations depend on ghost values
- Ghost values available from previous refresh

- Fluxes at both fine and coarse faces
- Conservation requires consistent fluxes
- Not consistent in general

- Apply a “flux-correction” step
- Update coarse values along face so fluxes match
- Requires coarse-to-fine block communication
Issue #3: flux-correction
Conserved quantities were not conserved at refinement jumps

- consider two fine blocks adjacent to a coarse block
- hydro computations depend on ghost values
- ghost values available from previous refresh

- fluxes at both fine and coarse faces
- conservation requires consistent fluxes
- not consistent in general

- apply a “flux-correction” step
- update coarse values along face so fluxes match
- requires coarse-to-fine block communication
Issue #4: improved interpolation

Implementing flux-correction didn’t fix the problem :-(

Ran experiments to narrow down the problem

- Problem only occurs when including gas dynamics
- Internal energies blow up at level interfaces
- Linear interpolation was main suspect
- Got further with injection but grid effects
- Suspected mismatched time-centering
- Using $\alpha = 0.0$ instead of $\alpha = 0.5$ didn’t help
- Reducing order of accuracy only delayed problem
- 2nd order Laplacian and accelerations
Issue #4: improved interpolation
Implementing flux-correction didn’t fix the problem :-(

Ran experiments to narrow down the problem

- problem only occurs when including gas dynamics
- internal energies blow up at level interfaces

- linear interpolation was main suspect
- got further with injection but grid effects
- suspected mismatched time-centering
- using \( \alpha = 0.0 \) instead of \( \alpha = 0.5 \) didn't help
- reducing order of accuracy only delayed problem
- 2nd order Laplacian and accelerations
Issue #4: improved interpolation
Implementing flux-correction didn’t fix the problem :-(

Ran experiments to narrow down the problem:

- **Problem only occurs when including gas dynamics**
- **Internal energies blow up at level interfaces**

**Linear interpolation was main suspect**
- **Got further with injection but grid effects**

- Suspected mismatched *time-centering*
- Using $\alpha = 0.0$ instead of $\alpha = 0.5$ didn’t help
- Reducing order of accuracy only delayed problem
- 2nd order Laplacian and accelerations
Issue #4: improved interpolation
Implementing flux-correction didn’t fix the problem :-(

Ran experiments to narrow down the problem

- problem only occurs when including gas dynamics
- internal energies blow up at level interfaces

- *linear interpolation* was main suspect
- got further with injection but grid effects

- suspected mismatched *time-centering*
- using $\alpha = 0.0$ instead of $\alpha = 0.5$ didn’t help

- reducing order of accuracy only delayed problem
- 2nd order Laplacian and accelerations
Issue #4: improved interpolation
Implementing flux-correction didn’t fix the problem :-(

Ran experiments to narrow down the problem

- problem only occurs when including gas dynamics
- internal energies blow up at level interfaces

- linear interpolation was main suspect
- got further with injection but grid effects

- suspected mismatched time-centering
- using $\alpha = 0.0$ instead of $\alpha = 0.5$ didn’t help

- reducing order of accuracy only delayed problem
- 2nd order Laplacian and accelerations
Issue #4: improved interpolation
Lately have been implementing ENZO’s interpolation scheme

- ideally accurate, monotonic, conservative
- linear interpolation: coarse block values only
- some ghost cells must be extrapolated
- non-monotonic: negative densities, etc.
- ENZO’s SecondOrderA method
- uses an extra layer of coarse zones
- overlaps some other adjacent blocks
- additional communication required
Issue #4: improved interpolation
Lately have been implementing ENZO’s interpolation scheme

- ideally **accurate**, **monotonic**, **conservative**
- linear interpolation: coarse block values only
- some ghost cells must be extrapolated
- non-monotonic: negative densities, etc.

- ENZO’s SecondOrderA method
- uses an extra layer of coarse zones
- overlaps some other adjacent blocks
- additional communication required
Issue #4: improved interpolation
Lately have been implementing ENZO’s interpolation scheme

- ideally **accurate**, **monotonic**, **conservative**
- linear interpolation: coarse block values only
- some ghost cells must be extrapolated
- non-monotonic: negative densities, etc.

- ENZO’s SecondOrderA method
- uses an extra layer of coarse zones
- overlaps some other adjacent blocks
- additional communication required
Issue #4: improved interpolation
New interpolation scheme involves additional blocks

- consider interpolating ghost cells $B_i \rightarrow B_j$
- coarse array overlaps additional blocks $B_k$
- $B_k$ needs to know it participates in $B_i \rightarrow B_j$
- can take advantage of symmetry
- assume fully-balanced octree
- $B_k$ must be same level as $B_i$ or $B_j$
- if same level as $B_i$, then $B_k \rightarrow B_j$
- if same level as $B_j$, then $B_i \rightarrow B_k$
Issue #4: improved interpolation
New interpolation scheme involves additional blocks

- consider interpolating ghost cells $B_i \rightarrow B_j$
- coarse array overlaps additional blocks $B_k$
- $B_k$ needs to know it participates in $B_i \rightarrow B_j$
- can take advantage of symmetry

- assume fully-balanced octree
- $B_k$ must be same level as $B_i$ or $B_j$
- if same level as $B_i$, then $B_k \rightarrow B_j$
- if same level as $B_j$, then $B_i \rightarrow B_k$
Issue #4: improved interpolation
New interpolation scheme involves additional blocks

- consider interpolating ghost cells $B_i \rightarrow B_j$
- coarse array overlaps additional blocks $B_k$
- $B_k$ needs to know it participates in $B_i \rightarrow B_j$
- can take advantage of symmetry

- assume fully-balanced octree
- $B_k$ must be same level as $B_i$ or $B_j$
- if same level as $B_i$, then $B_k \rightarrow B_j$
- if same level as $B_j$, then $B_i \rightarrow B_k$
Issue #4: improved interpolation
Indexing gets complicated and error-prone: introduced Box class

- define blocks
  - size ($nx, ny$)
  - ghosts ($gx, gy$)
- define neighbor
  - level $L$
  - face ($fx, fy$)
  - child ($cx, cy$)
- variable ghost depths to refresh
- optional ghost depths on send (e.g. mass deposit to total density)
- extra coarse-zone padding ($px, py$) for ENZO interpolation
- easily test block $B_k$ intersection with defined region
Issue #4: improved interpolation
Indexing gets complicated and error-prone: introduced Box class

- define blocks
  - size \((nx, ny)\)
  - ghosts \((gx, gy)\)

- define neighbor
  - level \(L\)
  - face \((fx, fy)\)
  - child \((cx, cy)\)

- variable ghost depths to refresh
- optional ghost depths on send (e.g. mass deposit to total density)
- extra coarse-zone padding \((px, py)\) for ENZO interpolation
- easily test block \(B_k\) intersection with defined region

\[
(\text{nx, ny}) = (10, 10) \\
(\text{gx, gy}) = (4, 4) \\
(\text{gxr, gyr}) = (3, 3) \\
(\text{gx, gy}) = (1, 1) \\
(\text{px, py}) = (1, 1)
\]
Issue #4: improved interpolation

Indexing gets complicated and error-prone: introduced Box class

- **define blocks**
  - size $(nx, ny)$
  - ghosts $(gx, gy)$

- **define neighbor**
  - level $L$
  - face $(fx, fy)$
  - child $(cx, cy)$

- variable ghost depths to refresh
- optional ghost depths on send (e.g. mass deposit to total density)
- extra coarse-zone padding $(px, py)$ for ENZO interpolation
- easily test block $B_k$ intersection with defined region
Conclusions

- Finishing up last steps before production runs
  - 1. scalable gravity
  - 2. buffered refresh
  - 3. flux-correction
  - 4. improved interpolation

- Just a couple remaining loose ends
  - scalable I/O
  - scalable-checkpoint/restart (thanks Ronak!)

- Performance optimization required
  - Enzo-E much slower than ENZO
  - over-refinement relative to ENZO
  - refresh across edges/corners
  - less communication in ENZO’s gravity solver
  - ENZO uses adaptive time-stepping
Conclusions

- Finishing up last steps before production runs
  - 1. scalable gravity
  - 2. buffered refresh
  - 3. flux-correction
  - 4. improved interpolation
- Just a couple remaining loose ends
  - scalable I/O
  - scalable checkpoint/restart (thanks Ronak!)
- Performance optimization required
  - Enzo-E much slower than ENZO
  - over-refinement relative to ENZO
  - refresh across edges/corners
  - less communication in ENZO’s gravity solver
  - ENZO uses adaptive time-stepping
Conclusions

- Finishing up last steps before production runs
  - 1. scalable gravity
  - 2. buffered refresh
  - 3. flux-correction
  - 4. improved interpolation
- Just a couple remaining loose ends
  - scalable I/O
  - scalable checkpoint/restart (thanks Ronak!)
- Performance optimization required
  - Enzo-E much slower than ENZO
  - over-refinement relative to ENZO
  - refresh across edges/corners
  - less communication in ENZO’s gravity solver
  - ENZO uses adaptive time-stepping
Acknowledgements

Funding for development of Enzo-E/Cello has been provided by the National Science Foundation grants OAC-1835402, SI2-SSE-1440709, PHY-1104819, and AST-0808184. http://cello-project.org/