Building a Better Astrophysics AMR Code with Charm++: Enzo-P/Cello (or more adventures in parallel computing)

Prof. Michael L Norman
Director, San Diego Supercomputer Center
University of California, San Diego

Supported by NSF grants SI2-SSE-1440709, PHY-1104819 and AST-0808184.
I am a serial code developer...

• I do it because I like it
• I do it to learn new physics, so I can tackle new problems
• I do it to learn new HPC computing methods because they are interesting
• Developing with Charm++ is my latest experiment
My intrepid partner in this journey

• James Bordner
• PhD CS UIUC, 1999
• C++ programmer extraordinaire
• Enzo-P/Cello is entirely his design and implementation
My first foray into numerical cosmology on NCSA CM5 (1992-1994)

Thinking Machines CM5

Large scale structure on a 512^3 grid
KRONOS run on 512 processors
Connection Machine Fortran
Enzo:
Numerical Cosmology on an Adaptive Mesh


- Adaptive in space and time
- Arbitrary number of refinement levels
- Arbitrary number of refinement patches
- Flexible, physics-based refinement criteria
- Advanced solvers
Enzo in action
Berger & Collela (1989) Structured AMR

Gas density

Refinement level

$z = 99.000$  $t = 0.018$ Gyr

$z = 99.000$  $t = 0.018$ Gyr
Application: Radiation Hydrodynamic Cosmological Simulations of the First Galaxies
Enzo: AMR Hydrodynamic Cosmology Code
http://enzo-project.org

- Enzo code under continuous development since 1994
  - First hydrodynamic cosmological AMR code
  - Hundreds of users
- Rich set of physics solvers (hydro, N-body, radiation transport, chemistry,...)
- Have done simulations with $10^{12}$ dynamic range and 42 levels
Enzo’s Path

1994
NCSA SGI Power Challenge Array
Shared memory multiprocessor

2013
NCSA Cray XE6 Blue Waters
Distributed memory multicore

Lots of computers in between
Birth of a Galaxy Animation
From First Stars to First Galaxies
Extreme Scale Numerical Cosmology

- Dark matter only N-body simulations have crossed the $10^{12}$ particle threshold on the world’s largest supercomputers

- Hydrodynamic cosmology applications are lagging behind N-body simulations

- This is due to the lack of extreme scale AMR frameworks

1 trillion particle dark matter simulation on IBM BG/Q, Habib et al. (2013)
Enzo’s Scaling Limitations

• Scaling limitations are due to AMR data structures

• Root grid is block decomposed, each block an MPI task

• Blocks are much larger than subgrid blocks owned by tasks

• Structure formation leads to task load imbalance

• Moving subgrids to other tasks to load balance breaks data locality due to parent-child communication

Refinement level

- Each block an MPI task
- OMP thread over subgrids
Hierarchical Timestepping

"W cycle"

Serialization over level updates also limits scalability and performance
Hierarchical Timestepping

“W cycle”

Deep hierarchical timestepping is needed to reduce cost
Adopted Strategy

• Keep the best part of Enzo (numerical solvers) and replace the AMR infrastructure
• Implement using modern OOP best practices for modularity and extensibility
• Use the best available scalable AMR algorithm
• Move from bulk synchronous to data-driven asynchronous execution model to support patch adaptive timestepping
• Leverage parallel runtimes that support this execution model, and have a path to exascale
• Make AMR software library application-independent so others can use it
Software Architecture

- Numerical solvers
- Scalable data structures & functions
- Parallel execution & services (DLB, FT, IO, etc.)
- Hardware (heterogeneous, hierarchical)
Software Architecture

- Enzo numerical solvers
- Forest-of-octrees AMR
- Charm++
- Hardware (heterogeneous, hierarchical)
Software Architecture

- Enzo-P
- Cello
- Charm++
- Charm++ supported platforms
Forest (=Array) of Octrees
Burstedde, Wilcox, Gattas 2011

2 x 2 x 2 trees

6 x 2 x 2 trees

refined tree

unrefined tree
**p4est** weak scaling: mantle convection

Burstedde et al. (2010), Gordon Bell prize finalist paper

---

**Fig. 5.** Weak scaling for a dynamically adapted dG solution of the advection equation (1) from 12 up to 220,320 cores. The mesh is adapted and repartitioned, maintaining 3200 tricubic elements per core. The maximum number of elements is $7.0 \times 10^8$ on 220,320 cores, yielding a problem with $4.5 \times 10^{10}$ unknowns. The top bar chart shows the overhead imposed by all AMR operations, which begins at 7% for 12 cores and grows to 27% for 220,320 cores. The bottom bar chart demonstrates an end-to-end parallel efficiency of 70% for an increase in problem size and number of cores by a factor of 18,360.
What makes it so scalable?

Fully distributed data structure; no parent-child

**Fig. 1.** One-to-one correspondence between a forest of octrees (left) and a geometric domain partitioned into elements (right), shown for a 2D example with two octrees $k_0$ and $k_1$. The leaves of the octrees bijectively correspond to elements that cover the domain with neither holes nor overlaps. A left-to-right traversal of the leaves through all octrees creates a space-filling z-curve (black “zigzag” line) that imposes a total ordering of all octants in the domain. For each octree the z-curve follows the orientation of its coordinate axes. In this example the forest is partitioned among three processes $p_0$, $p_1$, and $p_2$ by using the uniform partitioning rule (2.5). This partition divides the space-filling curve and thus the geometric domain into three process segments of equal ($\pm 1$) octant count.

Burstedde, Wilcox, Gattas 2011
Parallel Languages/Paradigms:
Charm++

Parallel Programming with Migratable Objects

Relevant links: exascale relevance, the manual, mini-apps, downloads, charmplusplus.org

Charm++ is a machine independent parallel programming system. Programs written using this system will run unchanged on MIMD machines with or without a shared memory. It provides high-level mechanisms and strategies to facilitate the task of developing even highly complex parallel applications.

Charm++ programs are written in C++ with a few library calls and an interface description language for publishing Charm++ objects. Charm++ supports multiple inheritance, late bindings, and polymorphism.

Platforms: The system currently runs on IBM's Blue Gene/P, Blue Gene/L, Cray XT3, XT4, XT5, Infiniband clusters such as Ranger, LoneStar and Abe, clusters of UNIX workstations and even single-processor UNIX, Solaris and Windows machines. It also runs on accelerators such as the Cell BE and GPGPUs.

The design of the system is based on the following tenets:
(7.1) What is Charm++?
Charm++ parallel programs: collections of asynchronously-interacting objects

- Charm++ program
  - Decomposed by objects
  - Charm++ objects called *shares*
  - invoke *entry methods*
  - *asynchronous*
  - communicate via *messages*

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

- Additional features
  - checkpoint/restart
  - dynamic load balancing
  - fault-tolerance

(Laxmikant Kale et al. PPL/UIUC)
(7.1) What is Charm++?

Charm++ collections of chares

**Chare Arrays**
- distributed array of chares
- migratable elements
- flexible indexing

**Chare Groups**
- one chare per processor (non-migratable)

**Chare Nodegroups**
- one chare per node (non-migratable)
Charm++ powers NAMD

NAMD, recipient of a 2002 Gordon Bell Award and a 2012 Sidney Fernbach Award, is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. Based on Charm++ parallel objects, NAMD scales to hundreds of cores for typical simulations and beyond 500,000 cores for the largest simulations. NAMD uses the popular molecular graphics program VMD for simulation setup and trajectory analysis, but is also file-compatible with AMBER, CHARMm, and X-PLOR. NAMD is distributed free of charge with source code. You can build NAMD yourself or download binaries for a wide variety of platforms. Our tutorials show you how to use NAMD and VMD for biomolecular modeling.

The 2005 reference paper *Scalable molecular dynamics with NAMD* has over 6,000 citations as of October 2016. NEW

With grit and a supercomputer yield chemical structure of HIV capsid (article referring to NAMD simulations on Blue Waters reported in Zhao et al., *Nature*, 497:843-846, 2013.)

Rapid parameterization of small molecules using the force field toolkit, JCC, 2013.

HPCwire Editors’ Choice Award: Best use of HPC in life sciences

**NAMD Powers Molecules by Theodore Gray** App for iPhone and iPad


Search all NAMD resources: [Search NAMD web site and tutorials](#) Google

Spotlight: Molecular Dynamics - Child’s Play (Nov 2014)
• **Goal:** implement *Enzo’s* rich set of physics solvers on a new, extremely scalable AMR software framework (*Cello*)

• *Cello* implements forest of quad/octree AMR on top of Charm++ parallel objects system

• *Cello* designed to be application and architecture agnostic (OOP)

• *Cello* available NOW at [http://cello-project.org](http://cello-project.org)

*Supported by NSF grants SI2-SSE-1440709*
Enzo AMR

Cello AMR

fields & particles

fields & particles

parallel
parallel
sequential
Demonstration of Enzo-P/Cello

Total energy

V F R K D
H W R N G
Q L V I L
Demonstration of Enzo-P/Cello

Mesh refinement level
Demonstration of Enzo-P/Cello

Tracer particles
Enzo-P/Cello
Dynamic Load Balancing

Charm++ implements dozens of user-selectable methods

$ charmrun +p4 bin/enzo-p input/load-balance-4.in +balancer RefineLB$

http://charm.cs.illinois.edu/manuals/html/charm++/7.html

How does Cello implement FOT?

- A **forest** is an array of **octrees** of arbitrary size $K \times L \times M$.
- An octree has leaf nodes which are **blocks** ($N \times N \times N$).
- Each block is a **chare** (unit of sequential work).
- The entire FOT is stored as a **chare array** using a bit index scheme.
- Chare arrays are **fully distributed data structures** in Charm++.
(7.2) How is Charm++ used in Cello?

- Each leaf node of the tree is a **block**
- Each block is a **chare**
- The forest of trees is represented as a **chare array**
(9.1) How are phases of the computation controlled?

Simulation evolution is controlled in control_charm.cpp
(9.2) Adaptive Mesh Refinement

Mesh refinement proceeds in several steps

1. Apply refinement criteria (Refine)
2. Tell neighbor Blocks your desired level
   - Blocks form a char array
   - remote entry method call to neighbor blocks
3. Receive neighbor level
   - entry method
   - called by neighbors
4. Update own level if needed (goto 2)
5. Exit after quiescence
   - no processor is executing an entry point
   - no messages are awaiting processing
   - and no messages in-flight
(9.2) Adaptive Mesh Refinement

- temporal level jump criterion
- spatial level jump criterion
(9.3) Refresh ghost zones

Neighbor in same refinement level

1. Face data copied to array
   - FieldFace object
2. Array sent to neighbor
   - char entry method
   - array sent as message
3. Array copied to ghost zones

Refresh ends when arrays from all neighbors have been received.
(9.3) Refresh ghost zones

Neighbor in coarser refinement level

1. Face data coarsened to array
   - Restrict object
   - FieldFace array
2. Array sent to neighbor
3. Array copied to ghost zones
(9.3) Refresh ghost zones
Neighbor in finer refinement level

1. Face data copied to array
2. Array sent to neighbor
3. Data interpolated to ghost zones
   - Prolong object
Particles in Cello

Enzo-P/Cello
(1.1) Classes for representing particle data

- **ParticleData**
  - represents state-independent (intrinsic) data
  - associated with **Blocks** (one object per mesh node)
  - stores arrays of particle data

- **ParticleDescr**
  - represents state-dependent (extrinsic) data
  - associated with **Simulation** objects (one per process)
  - describes how to interpret particle data (types, attributes, etc.)

- **Particle**
  - applications access particle data via **Particle** objects
(1.1) How Particle objects store particle data

- multiple particle types
- particles allocated in batches
  - fixed size arrays
  - fewer new/delete operations
  - efficient insert/delete operations
  - potentially useful for GPU’s
- batches store particle attributes
  - (position, velocity, mass, etc.)
  - 8,16,32,64-bit integers
  - 32,64,128-bit floats

- particle positions may be floating-point or integers
  - floating-point for storing global positions
  - integers for Block-local coordinates
    - solves reduced precision issue for deep hierarchies
    - less memory required for given accuracy
(1.1) How particle data is communicated between Blocks

- Communication is required when particles move outside a block.
- This is done using a 4x4x4 array.
  - Array contains pointers to ParticleData (PD) objects.
  - One PD object per neighbor block.

- Migrating particles are:
  - scatter()-ed to PD array objects.
  - Sent to associated neighbors.
  - gather()-ed by neighbors.

- One sweep through particles.
- One communication step per neighbor.
- Similar for refinement / coarsening.
WEAK SCALING TEST –
HOW BIG AN AMR MESH CAN WE DO?
Unit cell: 1 tree per core
201 blocks/tree, $32^3$ cells/block
Weak scaling test: Alphabet Soup array of supersonic blast waves

<table>
<thead>
<tr>
<th>N trees</th>
<th>Np = cores</th>
<th>Blocks/Chares</th>
<th>Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>1³</td>
<td>1</td>
<td>201</td>
<td>6.6 M</td>
</tr>
<tr>
<td>2³</td>
<td>8</td>
<td>1,608</td>
<td></td>
</tr>
<tr>
<td>3³</td>
<td>27</td>
<td>5,427</td>
<td></td>
</tr>
<tr>
<td>4³</td>
<td>64</td>
<td>12,864</td>
<td></td>
</tr>
<tr>
<td>5³</td>
<td>125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6³</td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8³</td>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10³</td>
<td>1000</td>
<td>201,000</td>
<td></td>
</tr>
<tr>
<td>16³</td>
<td>4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24³</td>
<td>13824</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32³</td>
<td>32768</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40³</td>
<td>64000</td>
<td>12.9M</td>
<td></td>
</tr>
<tr>
<td>48³</td>
<td>110592</td>
<td>22.2M</td>
<td>0.7T</td>
</tr>
<tr>
<td>54³</td>
<td>157464</td>
<td>31.6M</td>
<td>1.0T</td>
</tr>
<tr>
<td>64³</td>
<td>262144</td>
<td>52.7M</td>
<td>1.7T</td>
</tr>
</tbody>
</table>
Largest AMR Simulation in the world

1.7 trillion cells

262K cores on NCSA Blue Waters
Enzo-P Weak Scaling on Blue Waters: Efficiency

Alphabet Soup Test (SMP): 161031

Efficiency

parallel efficiency
memory efficiency

Processes (FP cores)
Enzo-P Weak Scaling on Blue Waters: Time

Alphabet Soup Test (SMP): 161031

Enzo-P solver

Cello fcns

Time (s)

- compute
- refresh time
- adapt time
- refresh sync
- adapt sync
- total

Processes (FP cores)
Enzo-P Weak Scaling on Blue Waters: Time

Alphabet Soup Test (hydro only): 161011

- adapt sync
- refresh sync

Time (s)

Processes (FP cores)
SCALING IN THE HUMAN DIMENSION – SEPARATION OF CONCERNS
(8.2) Design overview
Cello software components

High-level
Data Structures
Middle-level
Hardware-interface

Cello
Object-oriented design implements "separation of concerns", enhancing extensibility, maintainability, understandability.
Adding a Method to Enzo-P is Easy
(As easy as writing a sequential program)

Suppose we wish to add a FE heat equation solver to Enzo-P.

\[ u_t - \alpha \nabla^2 u = 0 \]

### 1. Parameter file: heat.in

```c
% list field variables
Field { list = [ "temperature" ]; };
% define initial conditions
Initial {
    list = [ "value" ];
    value {
        temperature = [ 10.0, (0.3<x && x<0.7) && (0.3<y && y<0.7), 1.0 ];
    }
}
% define method parameters
Method {
    list = [ "heat" ];
    heat {
        alpha = 1.0;
    }
}
```

### 2a. Include file: MethodHeat.hpp

```cpp
class MethodHeat : public Method {
    // Create a MethodHeat object
    MethodHeat ( double a ) : a_(a) {};
    // Apply FE to the block
    virtual void compute(Block *);
    // Compute the CFL restriction
    virtual double timestep (Block *);
    };
```

### 2b. Source code: MethodHeat.cpp

```cpp
MethodHeat::compute(Block * block) {
    // Get field block attributes
    Field field = block()->data()->field();
    iU = field.field_id("temperature");
    U = (double *) field.values (iU);
    field.dimensions (iU,&mx,&my);
    field.size (&nx,&ny);
    field.ghosts (iU,&gx,&gy);
    field.cell_width (&hx,&hy);
    rx = 1.0/(hx*hx); ry = 1.0/(hy*hy);
    // Apply forward Euler method
    for (int iy=gy; iy<ny+gy; iy++) {
        for (int ix=gx; ix<nx+gx; ix++) {
            i = ix + mx*iy;
            Uxx=(U[i-dx]-2*U[i]+U[i+dx])*rx;
            Uyy=(U[i-dy]-2*U[i]+U[i+dy])*ry;
            Unew[i]=U[i] + a_ * dt *(Uxx + Uyy);
        }
    }
```

![Image of a 3x3 grid with varying temperatures]
Voila’, parallel AMR heat conduction
Current Work: Linear Solvers

- Poisson and implicit flux-limited diffusion eqs.
- CG and BiCGStab implemented and functioning in parallel
  - Suffer from poor algorithmic scaling
- HG algorithm (D. Reynolds) under development (multigrid preconditioned BiCGStab)
  - Matlab prototype exhibits excellent algorithmic and parallel scalability
Takeaways

• **Cello** is a software framework for extreme scale AMR simulations

• **Cello** implements the most scalable AMR algorithm known: forest-of-octrees

• Parallelism is handled by **Charm++**, which supports fully distributed AMR data structures, asynchronous execution, dynamic load balancing, and fault tolerance, parallel IO

• Developing applications on top of **Cello** is easy—as simple as writing a sequential program

• It is available NOW at [http://cello-project.org](http://cello-project.org)
Path Forward

• Finish scalable gravity solver (we’re close!)
• Do a 1 trillion cell/particle hydro cosmology simulation as a demonstration
• Implement block adaptive timestepping
  – Exercises Charm++’s dynamic execution capability
• Experiment with Charm’s built-in DLB schemes on real applications
Resources

- Project site: [http://www.cello-project.org](http://www.cello-project.org)
- Source code: [https://bitbucket.org/cello-project](https://bitbucket.org/cello-project)
- Tutorials: on project site

Using and Developing Enzo-P/Cello

Scalable Adaptive Mesh Refinement for Astrophysics and Cosmology

PART I: Using Enzo-P

NSF SI2-SSE-1446796 PHY-1104619 AST-0808104

James Bordner  Michael L. Norman

University of California, San Diego
San Diego Supercomputer Center

Enzo Workshop
28 September–1 October 2015

Using and Developing Enzo-P/Cello

Scalable Adaptive Mesh Refinement for Astrophysics and Cosmology

PART II: Developing with Cello

NSF SI2-SSE-1446796 PHY-1104619 AST-0808104

James Bordner  Michael L. Norman

University of California, San Diego
San Diego Supercomputer Center

Enzo Workshop
28 September–1 October 2015
THANK YOU!