AMReX - a new framework for block-structured adaptive mesh refinement calculations

Andy Nonaka

Lawrence Berkeley National Laboratory

SciDAC-TEAMS Collaboration Meeting: August 22, 2018

BERKELEY LAB

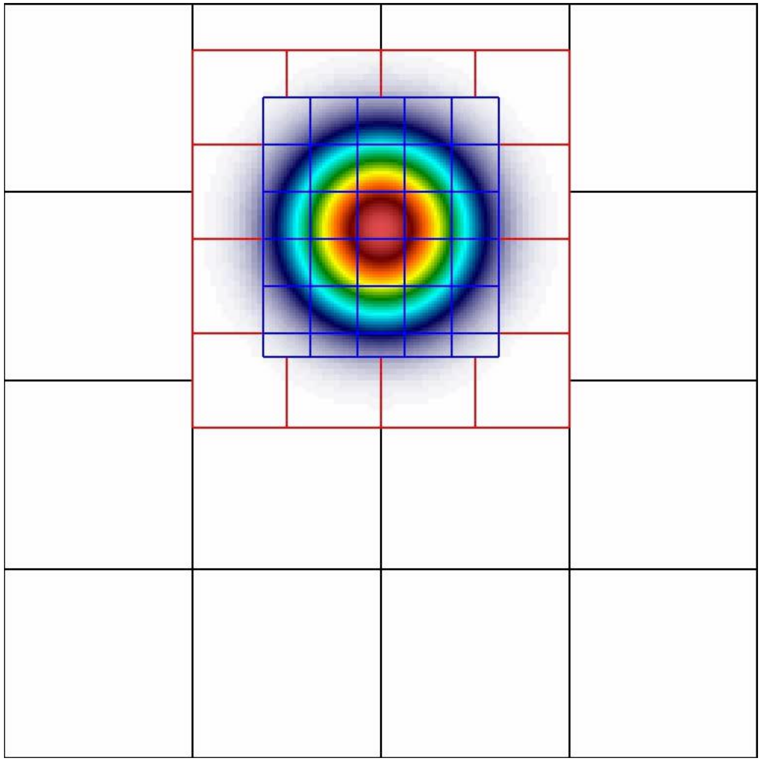
SciDAC Scientific Discovery through Advanced Computing

ECP EXASCALE COMPUTING PROJECT

What is AMReX?

- AMReX is a block-structured Adaptive Mesh Refinement (AMR) framework for solving systems of nonlinear PDEs for a variety of US Department of Energy applications.
  - DOE Exascale Computing Project (ECP) Co-Design Center

- Mission of the Co-Design Center
  - **Support applications**
  - Evaluate new software technologies
  - Interact with vendors
  - Much of the algorithmic methodology is developed as part of the DOE Applied Math Program



Cast

- **Ann Almgren**
- John Bell
- Weiqun Zhang
- Marcus Day
- **Andy Nonaka**
- Brian Friesen
- Kevin Gott
- Andrew Myers
- Steven Reeves
- Tan Nguyen
- Cy Chan
- Sam Williams
- **Anshu Dubey (ANL)**
- Petros Tzeferacos (U Chicago)
- Klaus Weide (U Chicago)
- Ray Grout (NREL)
- Shashank Yellapantula (NREL)

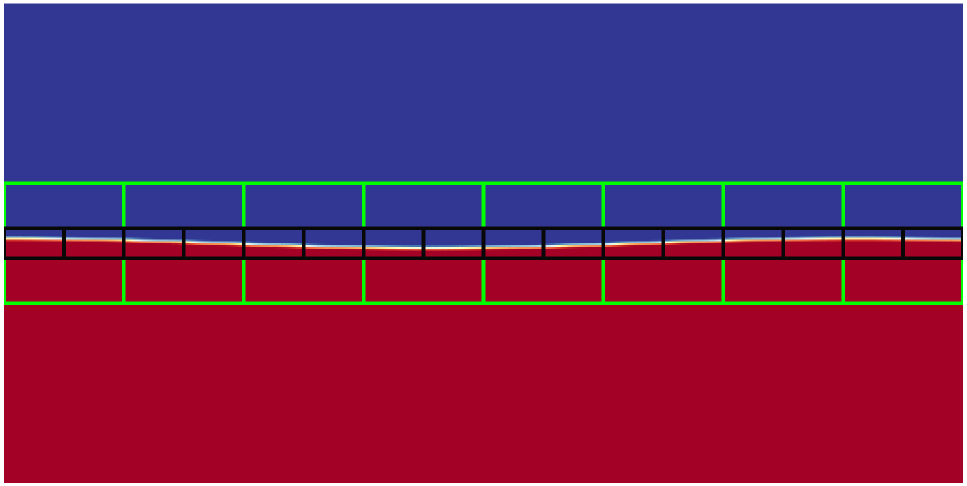
More About AMReX

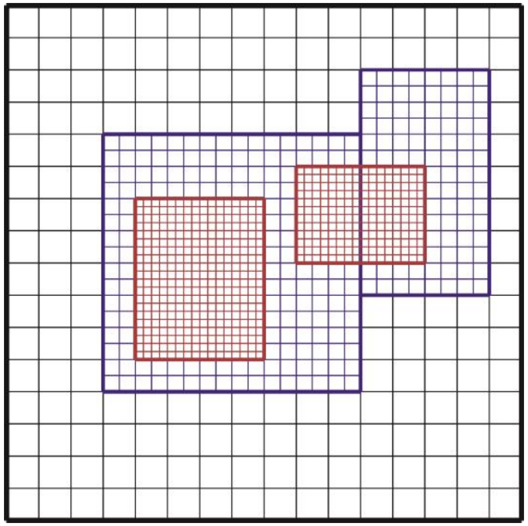
- Supports the development of block-structured AMR applications for current and next-generation architectures
  - Doesn't dictate anything about the physics, the discretization, or the numerics other than fundamentally uses block-structured mesh

- Provides support for
  - Explicit & implicit mesh operations
  - Multilevel synchronization operations
  - Particle and particle/mesh algorithms
  - Solution of parabolic and elliptic systems using geometric multigrid solvers
  - Embedded boundary (cut-cell) representation of geometry

What is Block-Structured AMR?

- In block-structured AMR, the solution is defined on a hierarchy of levels of resolution, each of which is composed of a union of logically rectangular grids/patches

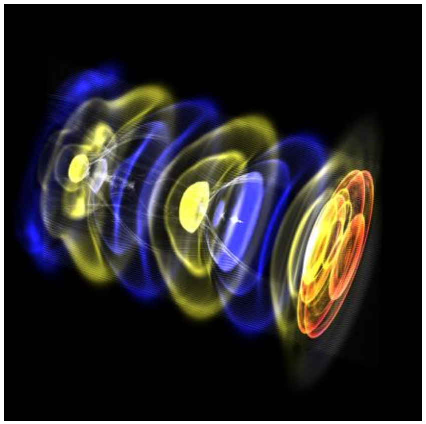




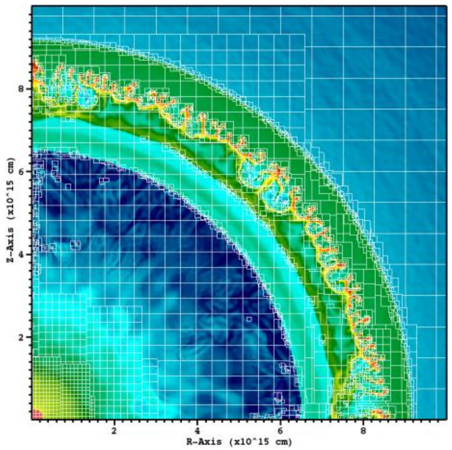
- Patches can change dynamically
- Oct-tree refinement with fixed size grids is special case
- More generally, patches need not be fixed size and do not have a unique parent-child relationship

AMReX Widely Used in DOE Applications

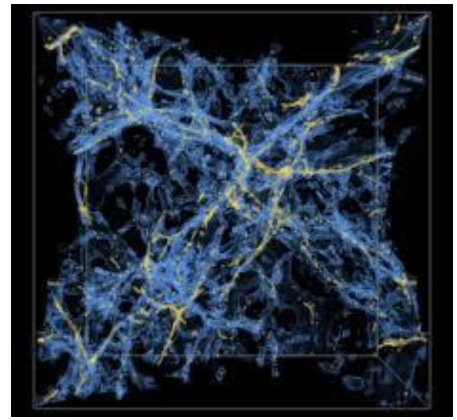
- Five ECP application projects that partner with AMReX:



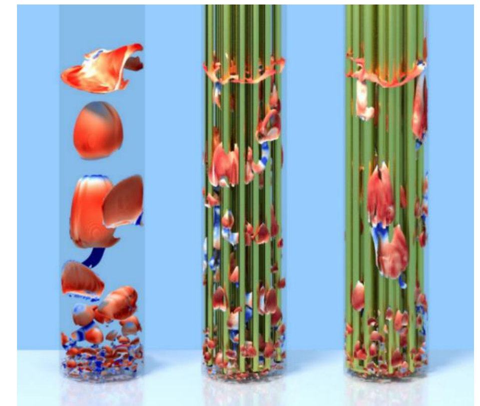
Accelerators



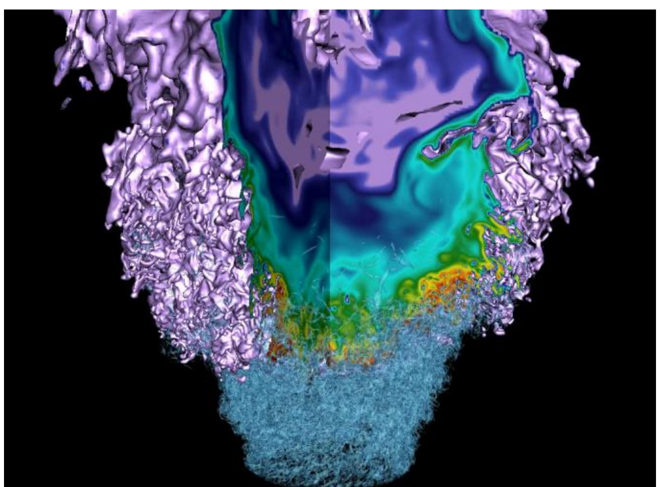
Astrophysics



Cosmology



Multiphase flow



Combustion

Other applications

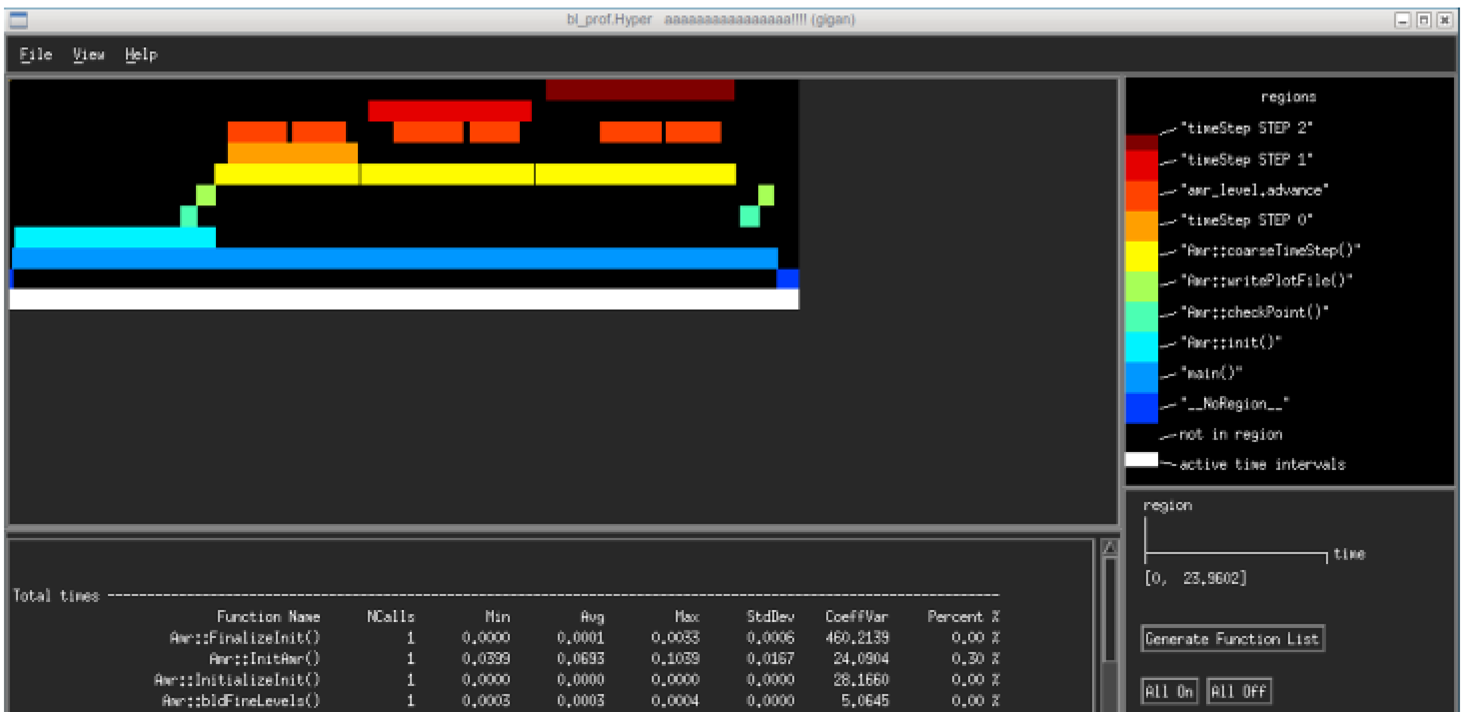
- Clouds / Atmospheric dynamics
- Fluctuating hydrodynamics (stochastic PDEs)
- Fluid-structure interaction
- Solid mechanics
- **Low Mach Number Astrophysics (MAESTRO)**

Additional Features

- Implemented in C++11 / Fortran90
- Open development model
  - Publicly available on Github; anybody can see the latest changes
  - Issues, pull requests encouraged (bug fixes, new features, documentation, etc...)
  - All branches public. Bleeding edge development branch, merged into master monthly.
  - Most ECP application codes and many other applications also publicly available.
- Extensive documentation
  - Sphinx, doxygen documentation hosted on Github pages, auto-generated with Travis.
  - Large number of tutorial codes to help you get started.
  - Github issues for user questions – prompt responses.

Additional Features

- Built in performance measurement tools
  - Simple summary characterization and/or highly detailed measurements
  - Measure both computation and communication
  - Ability to localize detailed measurements



Additional Features

- Interfacing with other Libraries
  - SUNDIALS ODE solvers
  - Hypre, HPGMG solvers
  - FFTW and other FFT libraries
  - In-situ and in-transit analytics - Sensei, ALPINE, Henson
- Visualization and I/O
  - In-house data format with efficient parallel I/O for both restart and plotfiles (has been much faster than HDF5 … although that is changing)
  - Visualization format supported by VisIt, Paraview, yt

Basic Data Structures

- Invect
  - Mesh point at a given level
- Box
  - Rectangular collection of mesh points at a level
- FArrayBox
  - Data defined on a box (double, integer, etc.)
  - Stored in column-major order (Fortran)
  - Optional contains space for boundary (ghost) data
- BoxArray
  - List of boxes at a level
- MultiFAB
  - List of FArrayBoxes associated with a BoxArray

Core Parallelization Strategy

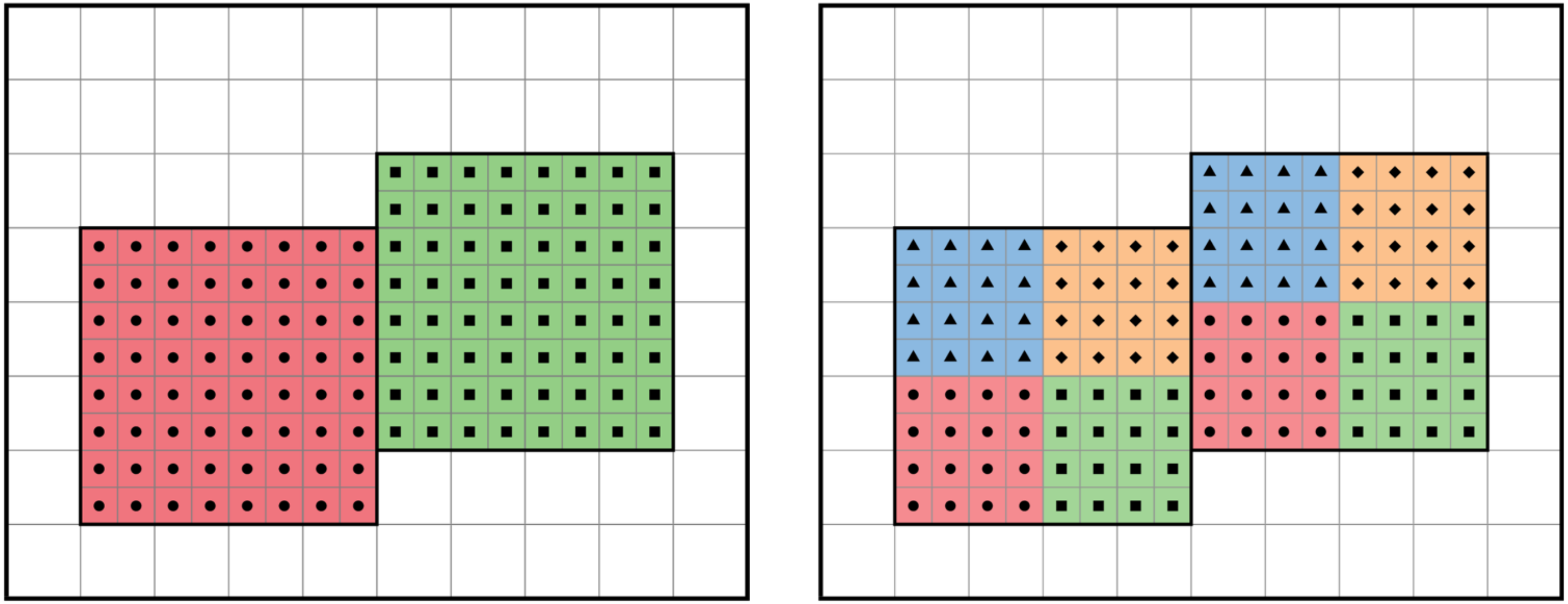
- DistributionMapping maintains an mapping between Boxes in a BoxArray and MPI rank
  - Several data distribution strategies such as knapsack and space-filling curve
  - Load-balancing based on work estimates
- Parallel operations defined on MultiFabs
  - MultiFabs can be operated on using add, divide, saxpy, etc..
  - Also provide MFIter for looping over the FArrayBoxes in a MultiFab.
    - Owner computes rule
    - Each proc loops only over the data it owns, details are hidden in application code

```cpp
MultiFab phi(boxArray, distributionMap, Ncomp, Nghost);

// loop over grids - owner computes
for ( MFIter mfi(phi); mfi.isValid(); ++mfi ) {
    const Box& bx = mfi.validbox(); // valid region

    work_on_phi(bx.loVect(),         // coordinates of valid region
                bx.hiVect(),
                phi[mfi].dataPtr(), // pointer to data
                phi[mfi].loVect(),  // coordinates of data (including ghost cells)
                phi[mfi].hiVect());
}
```

```fortran
subroutine work_on_phi(lo, hi, phi, philo, phihi) bind(C, name="work_on_phi")

  integer          , intent(in   ) :: lo(2), hi(2), philo(2), phihi(2)
  real(amrex_real), intent(inout) :: phi(philo(1):phihi(1),philo(2):phihi(2))

  integer i,j

  do j = lo(2), hi(2)
  do i = lo(1), hi(1)
     phi(i,j) = phi(i,j) + 1.d0
  end do
  end do

end subroutine work_on_phi
```

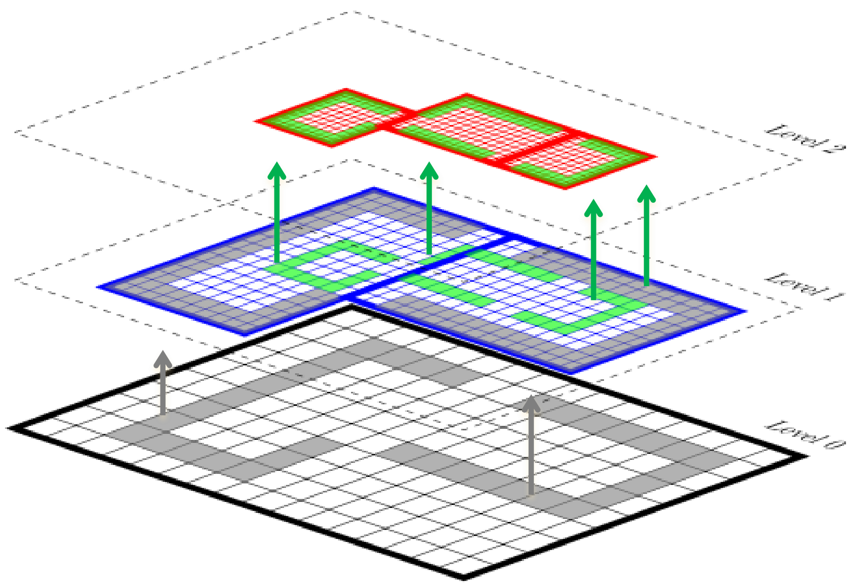
Core Parallelization Strategy

- Supports a variety of programming models - MPI, OpenMP, Hybrid, and (increasingly) GPUs (more on this later)
- OpenMP implemented via fortran loop directives or logical tiling
  - Tililng improves cache performance, even if using pure MPI



Multi-Level Tools

- All data structures are level aware
  - Well-defined mapping between levels
- Interpolation / Restriction
  - Filling boundary conditions on fine levels from coarse level data
  - Representing fine solution on the coarse level
- Flux Registers
  - Used to store data on coarse / fine interfaces
  - Used to enforce conservation for
  - e.g. hyperbolic systems
- Tagging / Regridding
  - Accumulate sets of points
  - Generating BoxArrays that cover those points

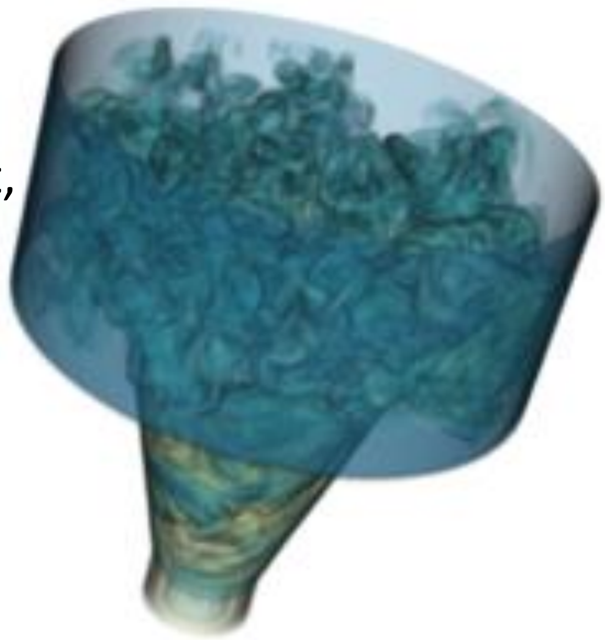


Linear Solvers

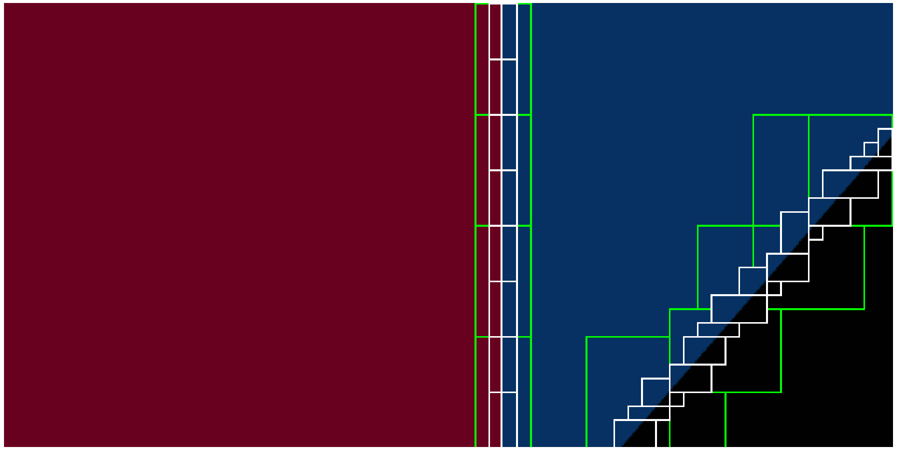
- AMReX provides native geometric Multigrid solvers for parabolic and elliptic systems
  - Currently also building a Stokes solver
- Cell-centered and nodal solvers
- Single-level and multi-level "composite" solvers
- Box agglomeration to avoid coarsening limitations
- Consolidation strategy to reduce ranks at coarser level
- Current work - extension to EB
  - Makes the bottom solve much more complex
  - Exploring transition to algebraic multigrid

Embedded Boundaries – Walls in Outer Space?

- Use a cut cell approach to complex geometries.
- Still block-structured, but cells labelled covered, cut, or regular
- Within an MFIter loop, ask whether the tile contains any cut cells.
- If not, treat in normally.
- If it does, pass in extra geometric, connectivity information.
- All data structures fully inter-operable with Fortran
- Doesn't sacrifice essential regularity far from domain boundaries.
- Much more work to do near boundaries, benefits from dynamic OpenMP scheduling



Prototype injector for gas turbine



Shock reflection test

Particles in AMReX

- Another core data type is the particle. In AMReX, particles live on and interact with an adaptive hierarchy of meshes.
- Additional challenges:
  - Inherently irregular - amount of data varies
  - Connectivity is hard, e.g. finding neighbors.
  - Always changing, data structures adapt every time step or more
- Several different kinds of applications:
  - Passive tracers
  - Particle-particle, particle-wall collisions
  - Particle-in-cell (electro-magnetic, dark matter, drag)

Nyx Code Cosmology – Dark Matter Particles

Particle-Wall Collisions (for Astrophysics?)

New Programming Models and Architectures

- Programming models
  - Fork-join model for coarse-grained asynchronous execution
    - Interface using C++11 lambda’s
  - Asynchronous iterators for fine-grained asynchronous execution
    - Tasked graph derived from AMR metadata
    - Runtime scheduling support
  - Investigating use of PGAS communication layer
- Much current work focuses on porting AMReX to GPUs
  - Cuda’s Unified Memory for data motion
  - Kernels offloaded through a variety of strategies
    - CUDA C/Fortran
    - OpenACC
    - OpenMP
  - NVIDIA’s thrust library for sorting and searching (particles)
  - Mini-App versions of Castro hydro (StarLord) and WarpX (Electromagnetic PIC) exist

3 Ways That Applications Use AMReX

- Core (library)
  - Application owns main
  - Support for single-level structured grid methods and particles
  - AMReX provides data containers and iterators for distributed data, as well as ghost cell exchange and communication
  - "User never types MPI…"
- AMRCore (library)
  - Application owns main
  - Support for block-structured AMR
    - Inter-level operations – Interpolation, restriction, refluxing (level synchronization)
    - AMR time step controlled by application
  - Application must understand how to specify multilevel algorithm
- AMRLevel (framework)
  - AMReX owns main
  - AMReX controls time step (particularly useful to support subcycling)
  - Stubs provided for time advance at single level as well as synchronization operations

END