AMR (and AMReX)

Ann Almgren

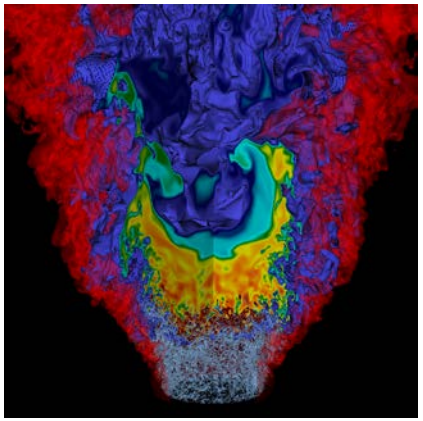
Lawrence Berkeley National Laboratory

April 24, 2018

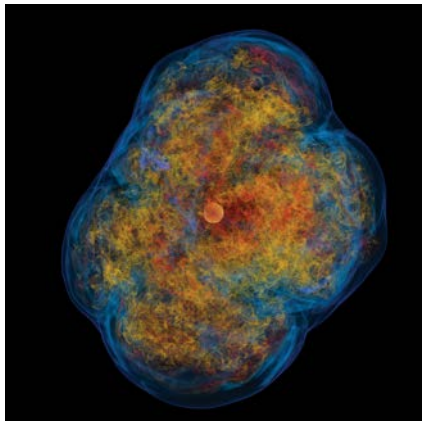
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

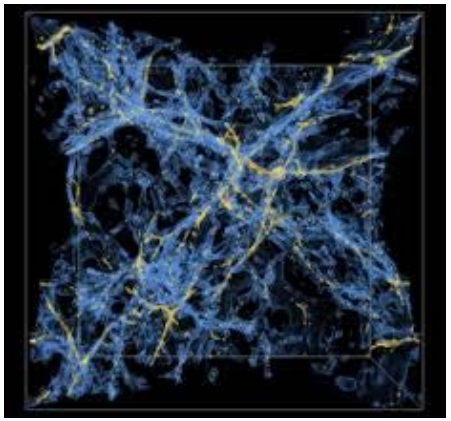
Block-structured AMR is in widely used in DOE applications



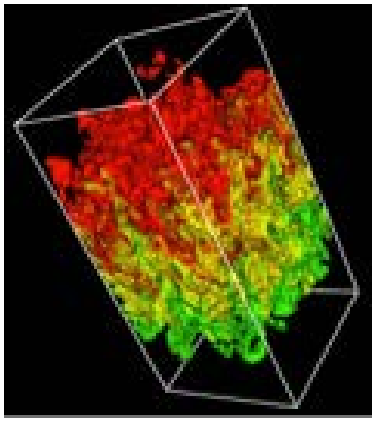
Combustion



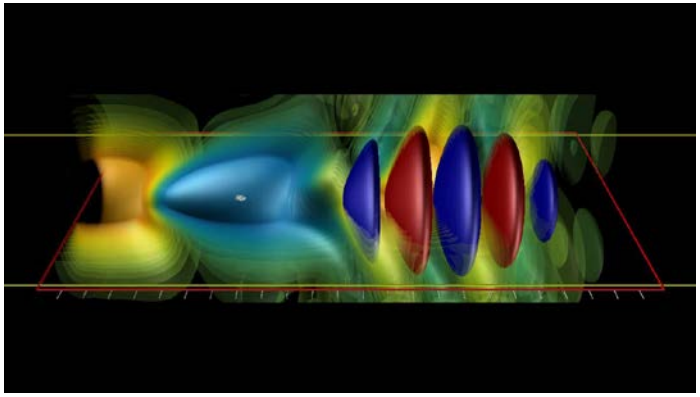
Astrophysics



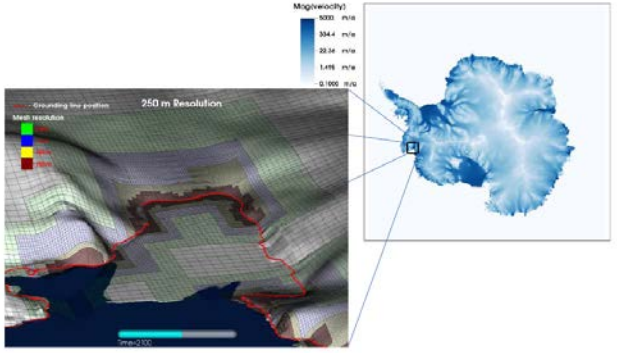
Cosmology



Fluid Instabilities

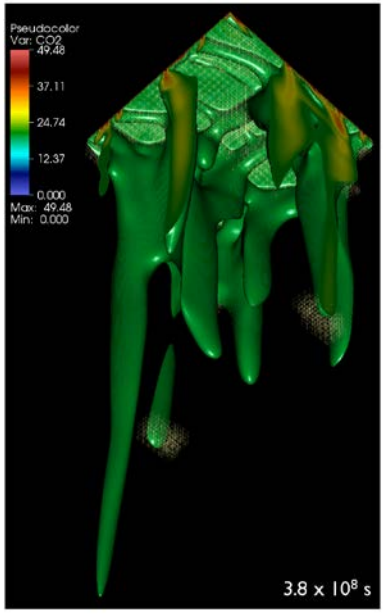


Accelerators



Ice sheets

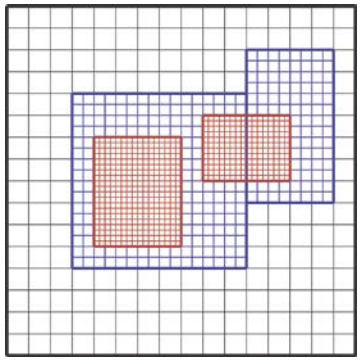
Subsurface Flow

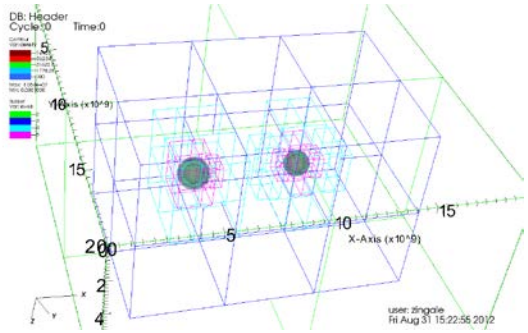


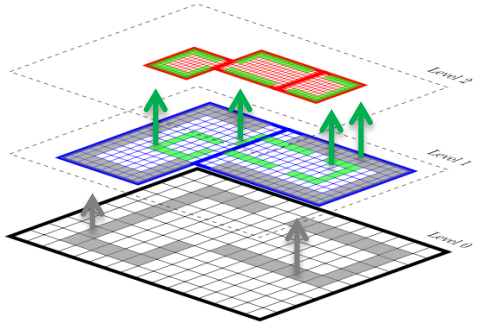
U.S. DEPARTMENT OF ENERGY | Office of Science

Block-Structured AMR Defines the Data Layout

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 change dynamically
- Oct-tree refinement with fixed size grids is special case
- More generally, patches may not be fixed size and may not have unique parent

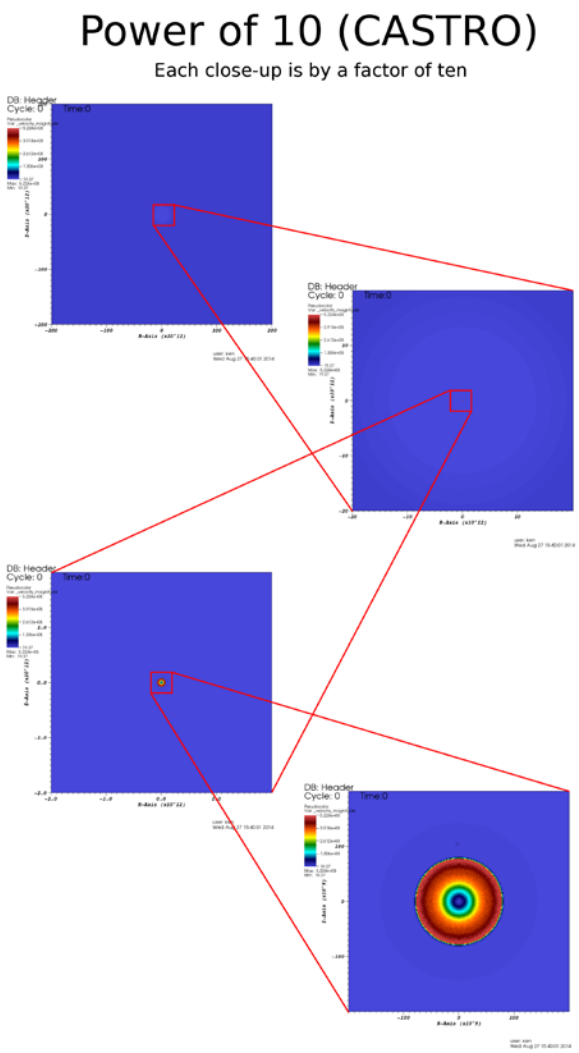
Data is in the form of

- mesh data
- Particle data
- Geometric (cut cell / EB) data

AMR Does Not Define the Discretizations

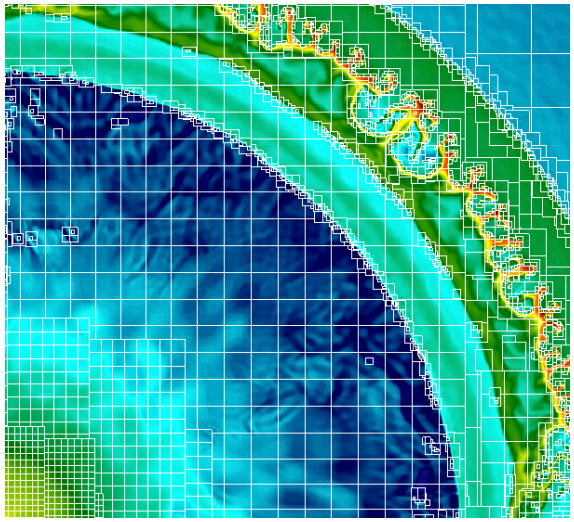
- Block-structured AMR does not define the spatial or temporal discretizations

- Time-stepping options including
  - Advancing all levels with a single time step
  - Subcycling in time (finer levels take multiple time steps for each coarser time step)
  - Optimal subcycling (subcycle between some but not all levels as determined by the time step constraints)
  - Multilevel iterative approaches such as MLSDC (multilevel spectral deferred corrections)



AMR provides opportunities on next generation architectures

- AMR provides a natural framework for reducing the memory footprint and computational cost of a structured grid simulation



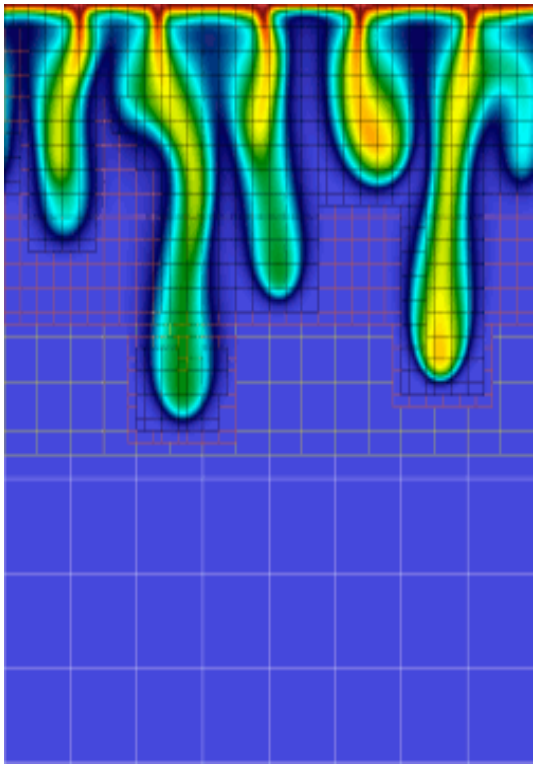
- The infrastructure to support block-structured AMR naturally supports hierarchical parallelism:
  - Coarse-grained dynamic load balancing due to decomposition into multiple grids at multiple levels
  - Fine-grained optimization opportunities due to regular patches of data

Key Issues for Next Generation AMR

1. Single-core and single-node performance
2. Load Balancing
3. Synchronicity / Scheduling / Overlapping
4. In Situ / In Transit Analytics & Visualization

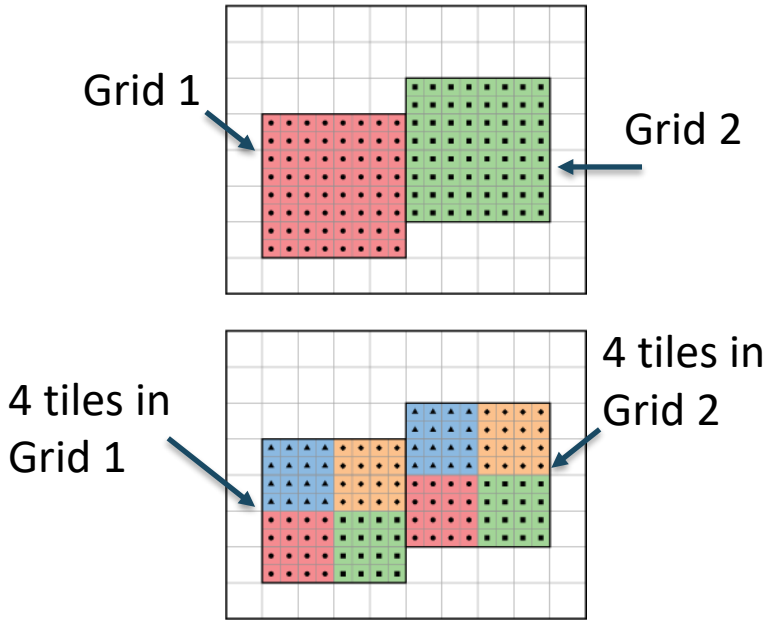
Single-Core Performance Still Matters!

- Per-core performance still matters
- Memory access cost increasingly important
- Block-structured refinement provides natural framework for regular memory access
  - tiling
  - vectorization
  - autotuning
  - communication-avoiding algorithms



Logical Tiling Can Reduce Cost on a Single Core

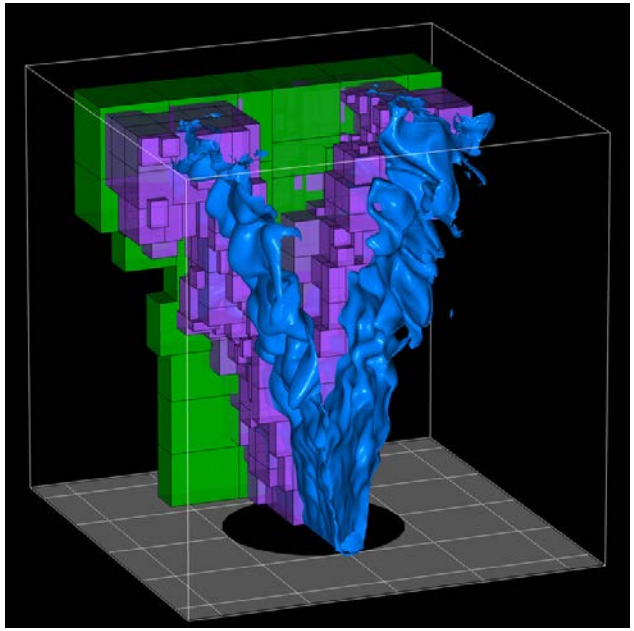
- With logical tiling, the data layout is unchanged but the unit of work is a tile rather than a grid
- In the example to the right, each grid is logically broken into 4 tiles, and each tile has 16 cells. There are 8 tiles in total.
- Can hide tiling in the iterator so is invisible to the application
- Leads to more efficient memory access



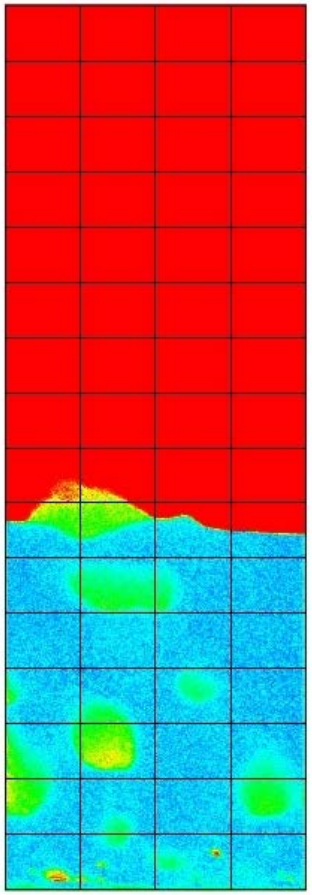
Logical Tiling Can Reduce Cost Across Cores

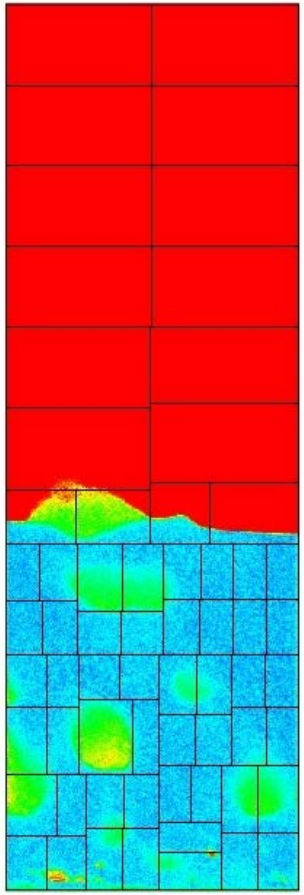
Distribute work effectively over all the cores effectively when Ngrids << Ncores

- Logical tiling makes smaller units of work
- Can distribute tiles statically or dynamically depending on distribution of work per tile



Load Balancing is Critical for Performance





- Load balancing is critical even at a single level when you allow for non-uniform domain decomposiiton
- May want to distribute mesh and particle data on same grid structures or may want to use dual grid approach
- AMR makes load balancing harder because of irregular distribution of grids and time-dependent number and location of grids

Synchronicity Means Different Things

- Not clear that we really know what we need

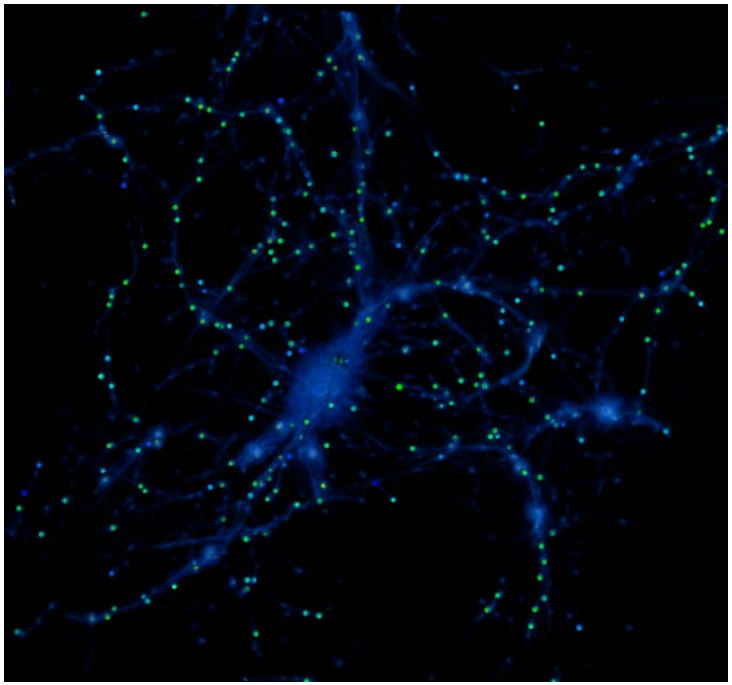
- Possible needs:
  - Low-level asynchrony: imagine operating on “interior” tiles while filling ghost cells of tiles touching boundaries –
    - invisible to the application
  - Medium-level asynchrony: imagine performing 4 multigrid solves (on different solution variables) at the same time in order to e.g., overlap computation of one with communication of the other –
    - visible to application but ok
  - High-level asynchrony – change ordering of high-level tasks
    - for an algorithm with many implicit operations this may be less effective – can’t have any one grid get too far ahead …
    - Potential memory bloat if can’t update solution in place
    - Needs to know a lot more about the algorithm!

We Still Need To Deal with the Data

- How to handle simulation output depends on whether we know what we're looking for, but solely post-processing is no longer an option

- AMR means
  - Less total data
  - More complicated data container (multiple grids at multiple levels)
- Tools need to understand the hierarchy

In transit halo finding in a Nyx simulation



What is AMReX?

- AMReX is an ECP-funded software framework to support the development of block-structured AMR applications for current and next-generation architectures

- Originally designed for solution of time-dependent PDEs but is not constrained to PDEs
  - 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

What is AMReX (p2)?

- Hierarchical parallelism: hybrid MPI + OpenMP with logical tiling to work efficiently on new multicore architectures
- Core functionality written in C++ with frequent use of Fortran kernels
- Highly efficient parallel I/O for both restart and plotfiles (has been much faster than HDF5 … we believe they're catching up)
- Visualization format supported by Visit, Paraview, yt, etc
- Tutorial examples, Users Guide, Doxygen documentation are all a work in progress right now

Particle Data Structures

- Particles are stored in ParticleContainers.
  - When we create the specific ParticleContainer we specify the amount, type and format of data carried by the particles

- Different types of particles can contain different amount of real and integer data

- Particle data can be in the form of SoA or AoS or both.

AMReX on Github

The AMReX repo is at https://www.github.com/AMReX-Codes/amrex

All branches are publicly available. The **development** branch is the most up-to-date. AMReX regression tests are run on the development branch nightly.

On the first work day of every month, the **development** branch is merged into **master** and it is tagged with the month/year.

We use an "agile" programming paradigm – the development branch is updated daily so that bug fixes and new features are available to users immediately.

Documentation of AMReX is very much a work in progress:

- Users Guide
- Doxygen / sphinx for reference
- Tutorials demonstrating functionality