The U.S. Department of Energy (DOE) Exascale Computing Project

ECP EXASCALE COMPUTING PROJECT

AMReX Tutorial

Andrew Myers

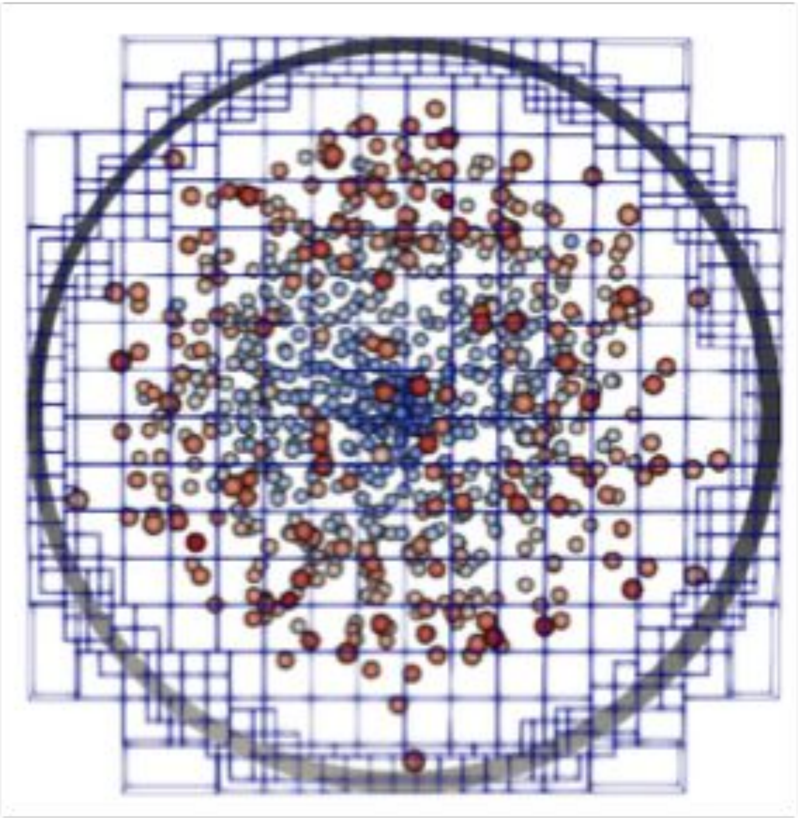
On behalf of the AMReX team – LBNL, NREL, ANL; PI: John Bell

WarpX / AMReX weekly meeting

June 7th, 2021

Outline

1. Overview (~20 min)
2. Q and A   (~10 min)
3. Hands-on tutorial (~ 1 hour)
   a. https://github.com/atmyers/ecp-tutorials
   b. Assumes some (but not much) familiarity with cmake, Jupyter notebooks
   c. Uses Gitpod (will need to sign in through Github or similar)



AMReX: Johannes Blaschke

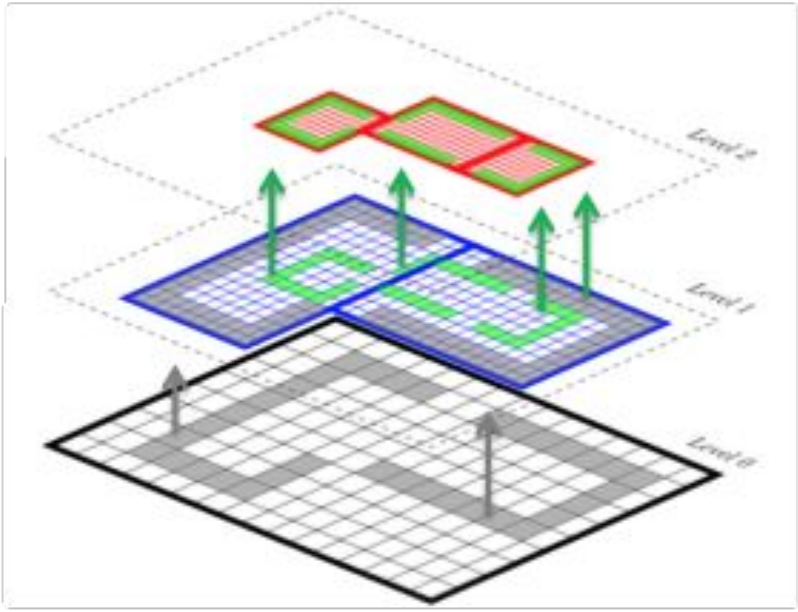
Particles, particle mesh, and level set mesh at the bottom of a cylinder in an MFiX-Exa simulation.

What is AMReX?

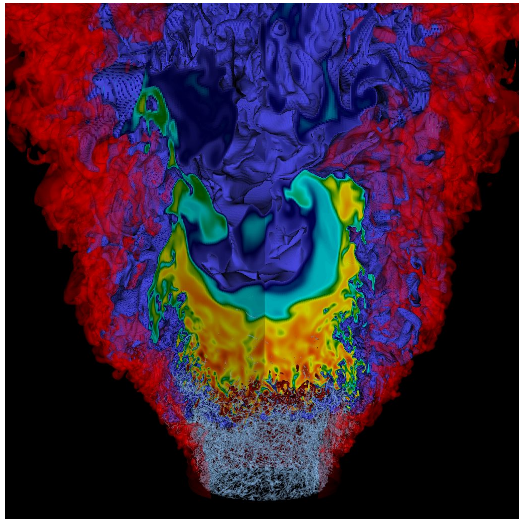
A software framework for Exascale-ready block-structured AMR applications

- Supports cell, face, edge, and node-centered mesh data and particles
- Native geometric multigrid solvers, embedded boundaries via cut-cells
- Tools for parallel communication, reductions, and load balancing
- Multi-level operations - tagging, regridding, interpolation / restriction
- Works on CPUs through OpenMP and GPUs through CUDA, HIP, or DPC++

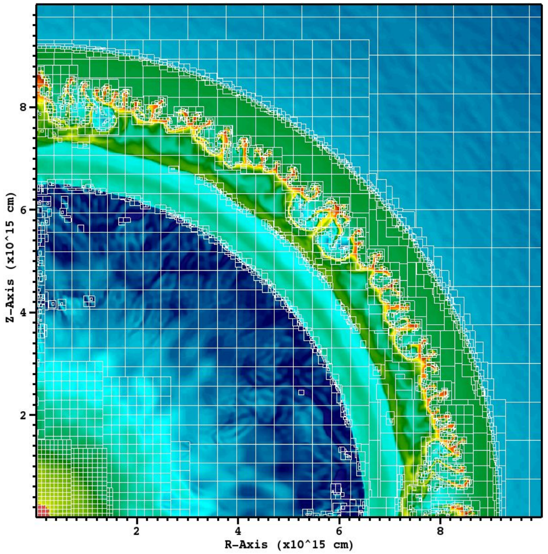
AMReX: Emmanuel Motheau



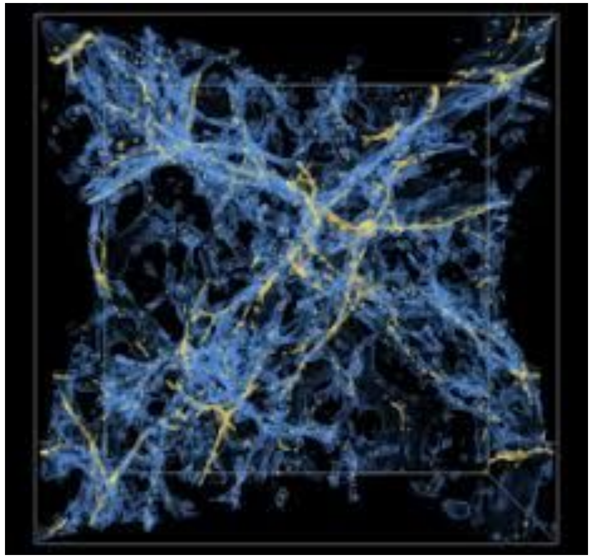
Seven ECP codes are built on AMReX



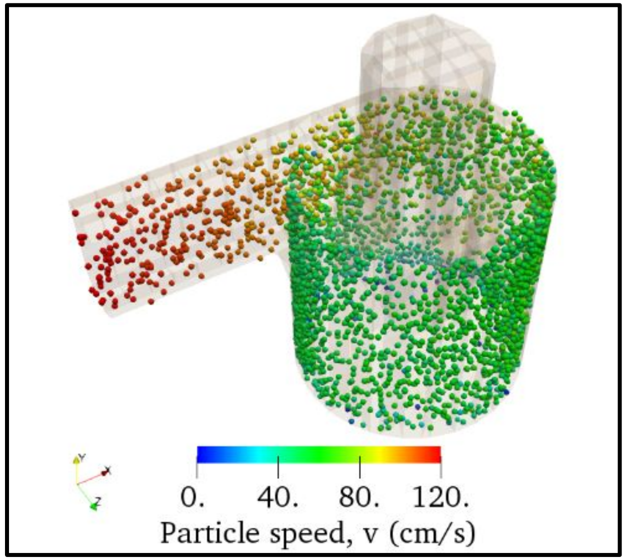
Combustion-PELE
(PeleC and PeleLM)



ExaStar (Castro)



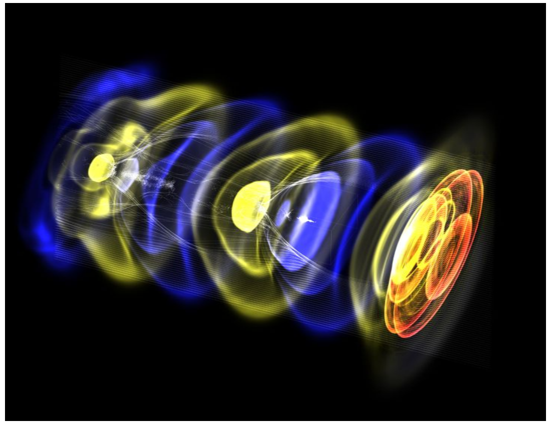
ExaSky (Nyx)



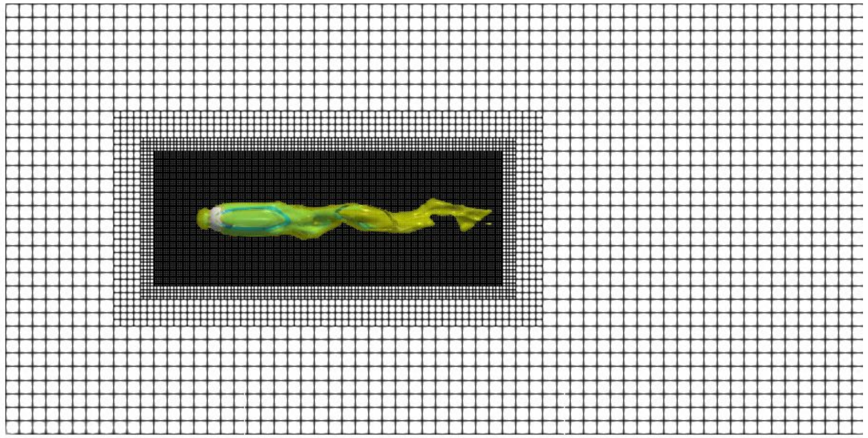
MFIX-Exa

Partial support for ExaAM
(TruchasPBF)

(and many other non-ECP apps…)



WarpX



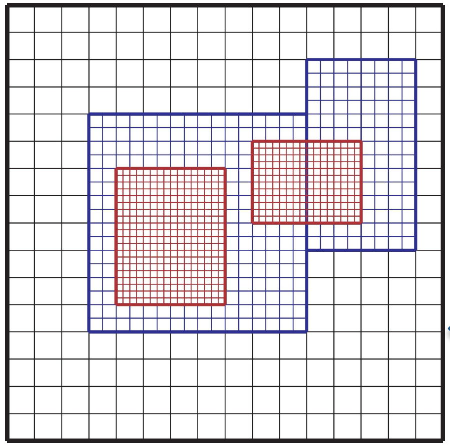
ExaWind (AMR-Wind)

EXASCALE COMPUTING PROJECT

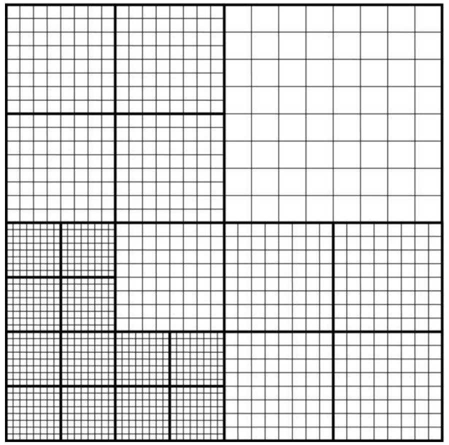
Adaptive Mesh Refinement

- Hierarchy of meshes that are refined in regions of interest.
- Preserves many nice features of regular mesh – accuracy, ease of discretization, regular data access
- Uses special discretizations only at coarse/fine interfaces (co-dimension 1)
- Naturally allows hierarchical parallelism
- Requires only a small fraction of the book-keeping cost of unstructured grids
- Block structured vs oct-tree (grid structure can be more general)

Example:
AMReX



Example:
FLASH



Grid sizes

May differ | Same

Child grid have unique parent?

No | Yes

https://iopscience.iop.org/article/10.1088/0067-0049/186/2/457)

Time-Stepping:

AMReX doesn't dictate the temporal discretization on a single level -- typical algorithms include

- 2nd order Godunov methodology
- Method of Lines (MOL)
- Spectral deferred corrections (SDC)
- SUNDIALS time integrators (which understand AMReX single-level mesh data structures)

AMReX doesn't dictate how the time steps on each level relate to each other -- typical patterns are:

- Multilevel non-subcycling (same dt at all levels): "AmrCore"
  - individual operations typically performed across all levels before proceeding to next operation

- Multilevel subcycling (dt/dx constant across levels): "AmrLevel"
  - operations typically performed one level at a time
  - stubs available for standard sync operations, e.g. explicit refluxing
  - requires more complicated synchronization operations

- More general approaches -- including SDC across levels, "optimal subcycling", etc

Hierarchical Parallelism:

AMReX supports hierarchical parallelism through MPI+X (+X)

- Multi-core: “MPI over grids, OpenMP over tiles”
- Hybrid w/ GPUs: “MPI over grids, CUDA (or HIP or DPC++) on GPUs”

Written in **C++** (minimum requirement = C++14; can use features of C++17 if available)

ParallelFor routines w / lambda functions

Many GPU-specific optimizations - memory Arenas, fused kernel launches, etc...

Example - performance-portable looping over AMR data

```cpp
void example(Vector<MultiFab>& amr_data)
{
    int numLevels = amr_data.size();

    // loop over levels from Coarse to Fine
    for (int lev = 0; lev < numLevels; ++lev) {
        MultiFab& level_data = amr_data[lev];

        #ifdef _OPENMP
        #pragma omp parallel
        #endif
        // loop over local grids/tiles on this level using the MFIter
        for ( MFIter mfi(level_data, TilingIfNotGPU()); mfi.isValid(); ++mfi )
        {
            // the box holds the 3D index space for this grid/tile
            const Box& bx = mfi.tilebox();

            // the Array4 is a lightweight struct containing a pointer
            // to the local data array and an access operator ()
            const Array4<Real> array_data = level_data.array(mfi);

            // loop over the index space of this box (e.g. launch a GPU kernel)
            amrex::ParallelFor(bx,
            [=] AMREX_GPU_DEVICE (int i, int j, int k) noexcept
            {
                // access local data using spatial + component indexes
                array_data(i, j, k, 0) = 1.0;
            });
        }
    }
}
```

- The **ParallelFor** takes index space in a box and a C++ lambda function to call on each 3D index.
- The **Array4** contains a pointer and access operator(). Captured by value in the lambda.
  - Uses Fortran-like syntax
- AMReX **ParallelFor** launches kernel
- All we had to do was label our "work" lambda function as a GPU function

Mesh Data Structures

Support for multilevel **mesh** data on cell centers, faces, nodes …

- **IntVect**
  - Dimension length array for indexing.
- **Box**
  - Rectilinear region of index space (using IntVects)
- **BoxArray**
  - Union of Boxes at a given level
- **FArrayBox (FAB)**
  - Data defined on a box (double, integer, complex, etc.)
  - Stored in column-major order (Fortran)
- **MultiFab**
  - Collection of FArrayBoxes at a single level
  - Contains a Distribution Map defining the relationship across MPI Ranks.
  - Primary Data structure for AMReX mesh based work.

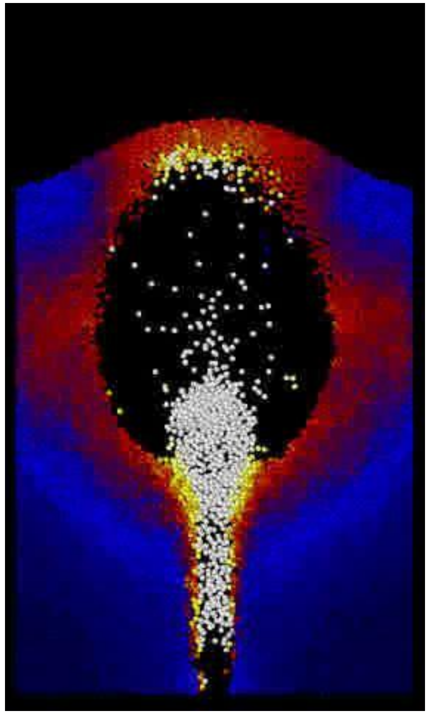
Simplest Structures

Most Complex Structures

Particle Data Structures

Multilevel **particle** data and iterators

- **Particle**
  - AoS and SoA data formats; integer or float or double
- **ParticleContainer**
  - Distributed container for particles defined on an AMR hierarchy (not necessarily the same one as the mesh data)
  - Assigned to levels, grids, and tiles based on their physical locations
- **ParIter**
  - Each rank iterates over the particles it owns



Single-Level Operations:

Parallel Copy

- The most general parallel communication routine for mesh data
- Copies between MultiFabs that can have different BoxArrays and DistributionMappings
- Can take general “copy” operator - copy or add

Ghost cell operations

- FillBoundary - fills ghost cells from corresponding valid cells
- SumBoundary - adds from ghosts to corresponding valid

Neighbor particles / lists

- Each grid can grab copies of particles from other grids within a certain distance
- Can precompute list of potential collision partners over next N steps

Particle-mesh deposition / interpolation

- General version that can take user-defined lambda function specifying the kernel

Multi-Level Operations:

Regridding
- Tagging, grid construction, data filling ...

Interpolation:
- Ghost cell filling and regridding

Restriction:
- Average fine onto coarse for synchronization

Flux registers:
- Available in AmrCore/AmrLevel based applications – stores fluxes at coarse-fine interfaces for easy refluxing

Virtual and Ghost Particle Construction
- For representing effect of particles at coarser / finer levels

Particle Redistribute
- Puts particles back on the right level / grid / tile after they have moved

Embedded Boundary (Cut Cell) Support:

Use a cut cell approach for complex geometries

- Special stencils at/near cut cells
- Regular stencils elsewhere
- Connectivity information stored in a single integer.

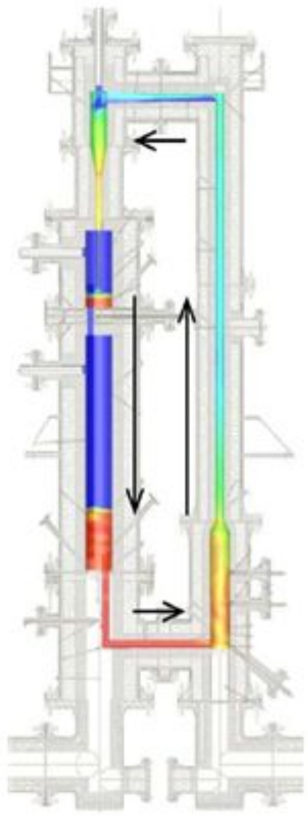
Grids/tiles can be queried as to whether they contain irregular cells

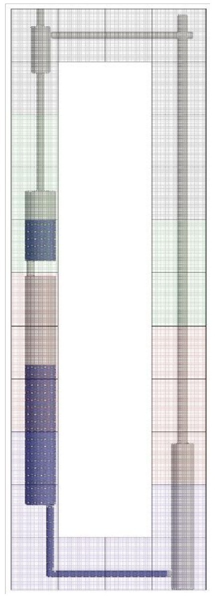
Support for EB-aware

- interpolation, e.g. from cell centers to face centroids
- restriction
- slopes
- linear solvers (cell-centered and nodal)

Option for mesh pruning (for memory savings)

Particles can see boundary through level set function





(L) Prototype of Chemical Looping Reactor (CLR) from MFIX-Exa project

(R) Grid generation using mesh pruning

EXASCALE COMPUTING PROJECT

Geometric Multigrid (GMG) Solvers:

Support for multi-AMR level GMG solvers

- cell vs nodal data
- variable coefficient Poisson or Helmholtz equation
- tensor solve for Navier-Stokes
- single- vs multi-component

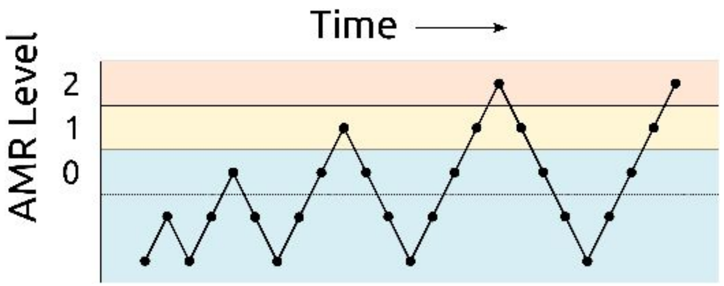
EB-aware options

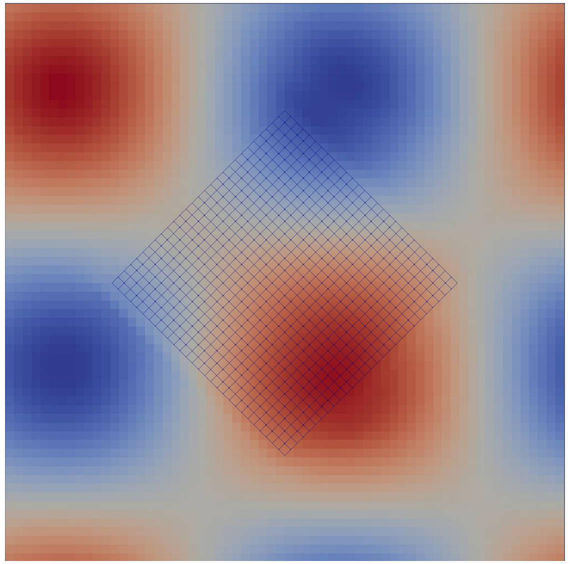
Option for masking to enable overset mesh algorithms

Native “bottom solver” is BiCG (or CG or both)

Interface to call hypre or PETSc as “bottom solver” --
which can be at any multigrid level, including the top

Options for agglomeration and consolidation on coarsening



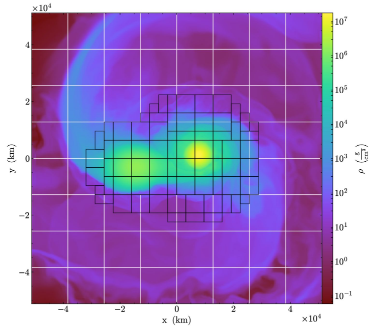


Over set mesh coupling

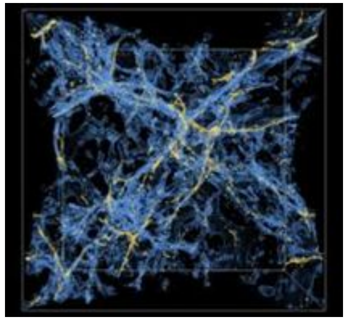
Native I/O

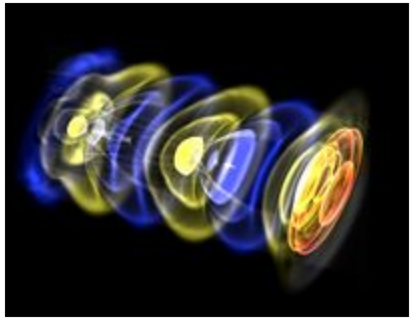
Native I/O format for plotfiles and checkpoint files

- multi-level mesh and particle data
- additional app-specific data may be included



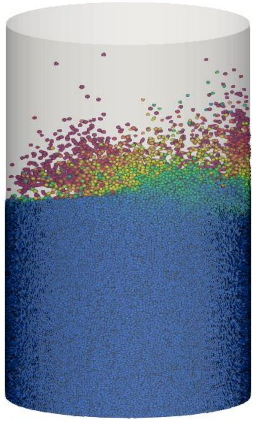
User-specified number of output streams





Plotfile format supported by Paraview, Visit, yt

HDF5 output format also supported



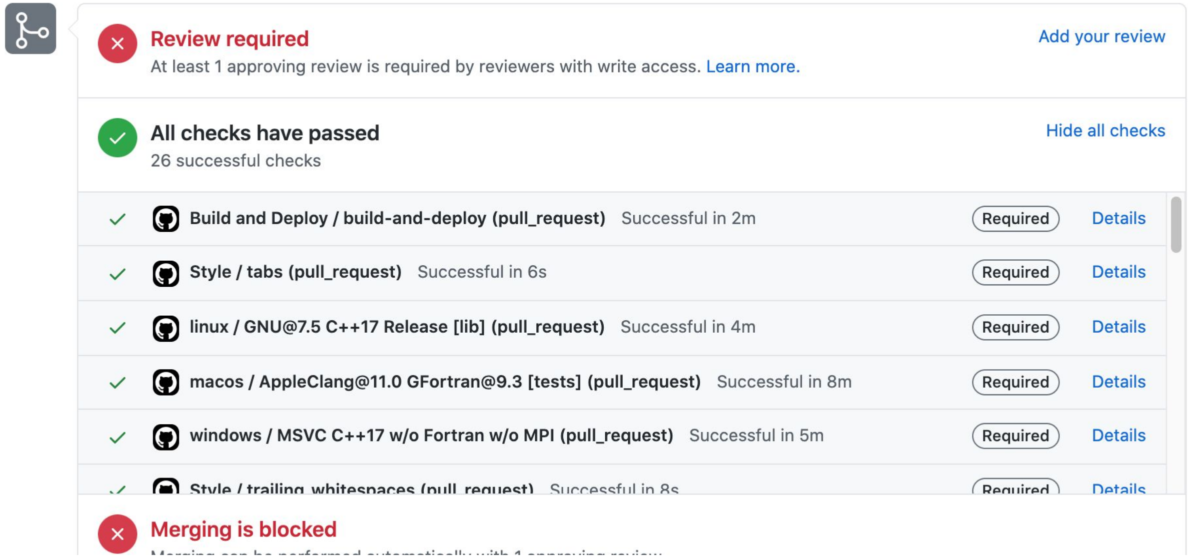
Async capability developed for CPU/GPU systems

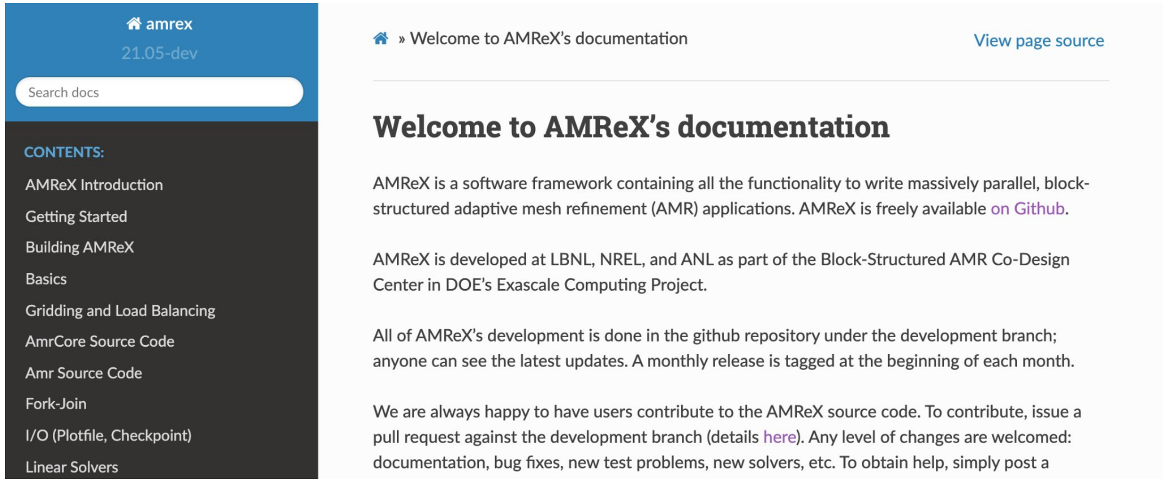
Open Source Development

Code is open-source, available on Github:
https://github.com/AMReX-Codes/amrex

Online documentation:
https://amrex-codes.github.io/amrex/docs_html/

Suite of tests run nightly and on every PR:





Live Tutorial

https://gitpod.io/#https://github.com/atmyers/ecp-tutorials