# AMReX: Building a Block-Structured AMR Application (and More)

Presented to
**ATPESC 2020 Participants**

**Ann Almgren**

Deputy Director, AMReX ECP Co-Design Center
Senior Scientist and Group Lead, CCSE, LBNL

**Don Willcox**
Postdoctoral Researcher, LBNL

Date 08/04/2020

U.S. DEPARTMENT OF ENERGY | Office of Science

**ATPESC Numerical Software Track**

EXASCALE COMPUTING PROJECT

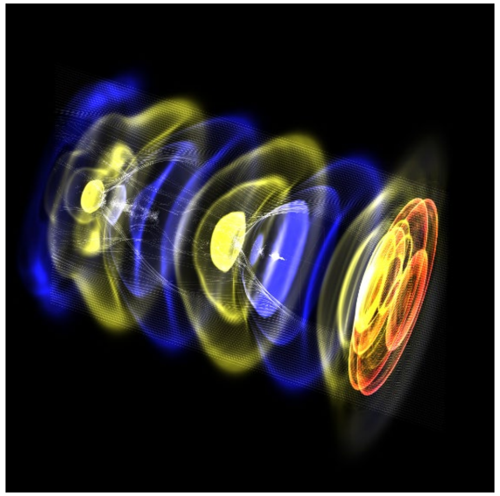
## Setting the Stage

Most of the problems we solve today are hard.

Characteristics of these problems are often that they couple multiple physical processes across a range of spatial and temporal scales.

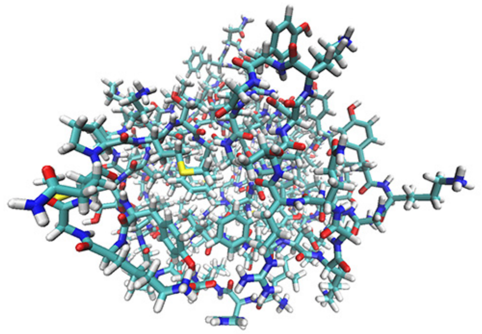
Gone are the days of simple physics, simple geometry, single algorithm, homogeneous architectures … ☹

So how do we build algorithms and software for hard multi-physics multi-scale multi-rate problems without starting over every time?



*WarpX project: Jean-Luc Vay, PI*

# Setting the Stage (p2)



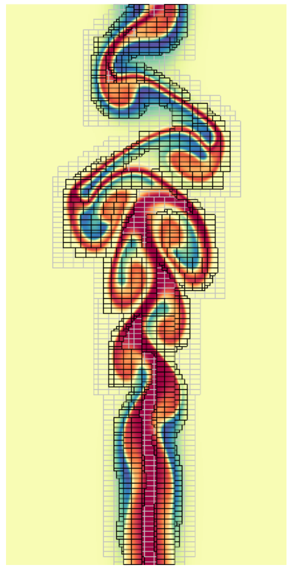
*TINKER: https://www.epcc.ed.ac.uk*

Not all simulations use a mesh

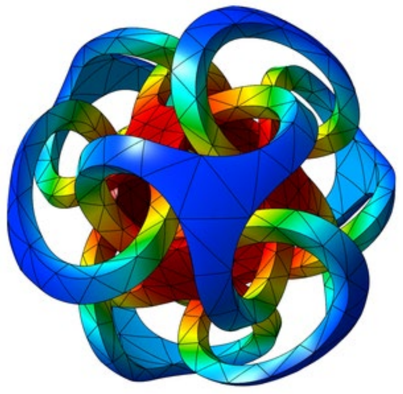
But for those that do, the choice is usually structured vs unstructured.

**Structured**:

- Easier to write discretizations
- Simple data access patterns
- Extra order of accuracy due to cancellation of error
- Easy to generate complex boundaries through cut cells but hard to maintain accuracy at boundaries



*AMReX: Emmanuel Motheau*

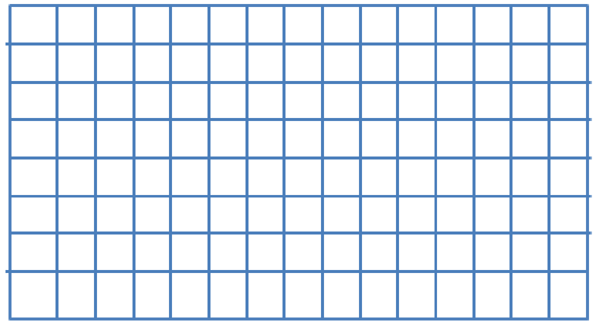


*https://ceed.exascaleproject.org/vis/*

**Unstructured**:

- Can fit the mesh to any geometry – much more generality
- No loss of accuracy at domain boundaries
- More "book-keeping" for connectivity information, etc
- Geometry generation becomes time-consuming

## Structured Grid Options

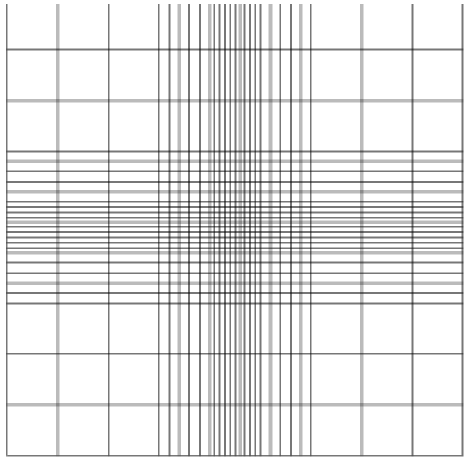


*https://commons.wikimedia.org/wiki*

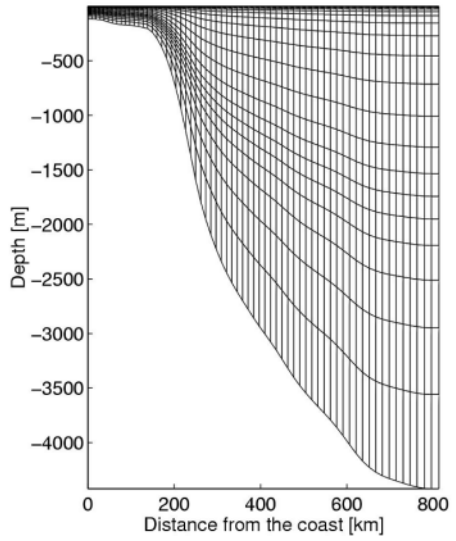
**Logically rectangular doesn't mean physically rectangular**

Structured with non-constant cells split pros and cons of structured vs unstructured:

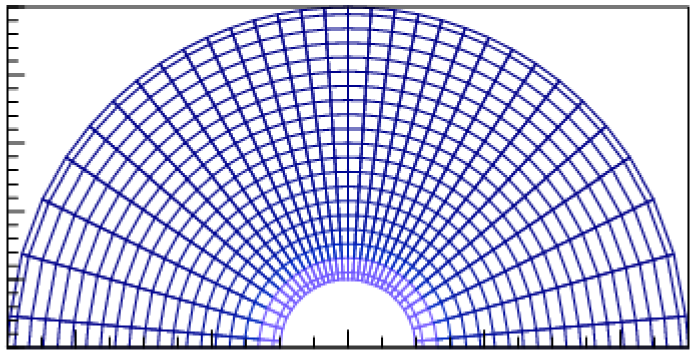
- Can fit (simple) non-rectangular boundaries while still having known connectivity
- Finer in certain regions (mesh refinement)
- Harder to maintain accuracy



*http://silas.psfc.mit.edu/22.15/lectures/chap4.xm l*

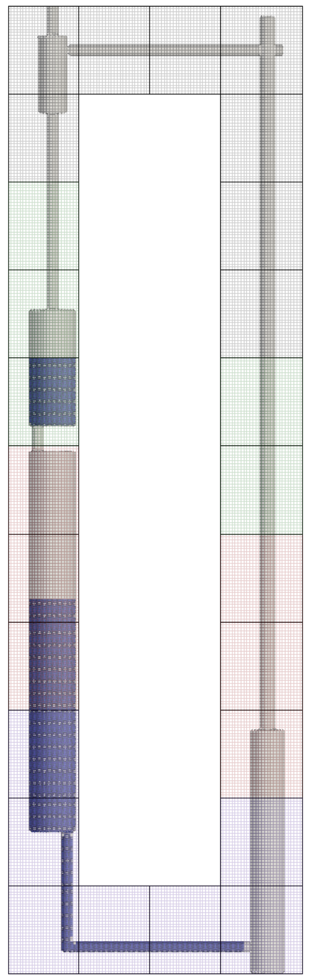


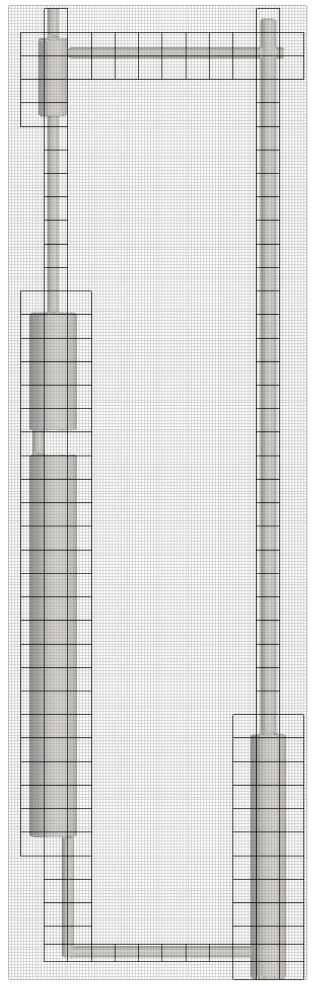
*http://www.cfoo.co.za/simocean/modelsroms.php*



*http://silas.psfc.mit.edu/22.15/lectures/chap4.xml*

## More Structured Grid Options





**Structured grid does not have to mean the entire domain is logically rectangular either.**

One can also “prune” the grids so as to not waste memory or MPI ranks – can still use rectangular cells in non-rectangular domain.

Grid pruning can save both memory and work.

## Why Is Uniform Cell Size Good?

Numerical Analysis 101:

$$\begin{aligned} \phi_{i+1} &= \phi(x_i) + \Delta x_r \phi_x + {}^1\!/\!_2(\Delta x_r)^2 \phi_{xx} + O(\Delta {x_r}^3) \\ \phi_{i-1} &= \phi(x_i) - \Delta x_l \phi_x + {}^1\!/\!_2(\Delta x_l)^2 \phi_{xx} + O(\Delta {x_l}^3) \end{aligned}$$

We often use a centered difference as an approximation for a gradient,

$$\frac{\phi_{i+1} - \phi_{i-1}}{\Delta x_l + \Delta x_r} = \phi_x + {}^1\!/\!_2 \boxed{(\Delta x_r - \Delta x_l)} \phi_{xx} + O(\Delta x^2)$$

Note we only get second-order accuracy if we use constant cell spacing.

Can we confine this error?

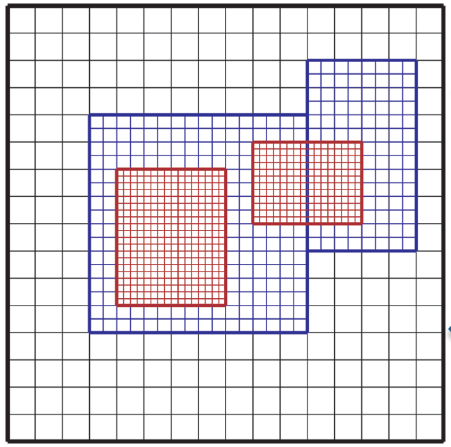
# Can We Have the Best Of Both Worlds?

Distorting the mesh is not ideal, but we can't afford uniformly fine grid.

**Adaptive Mesh Refinement:**

- refines mesh in regions of interest
- allows local regularity – accuracy, ease of discretization, easy data access
- naturally allows hierarchical parallelism
- uses special discretizations only at coarse/fine interfaces (co-dimension 1)
- requires only a small fraction of the book-keeping cost of unstructured grids

Example:
AMReX



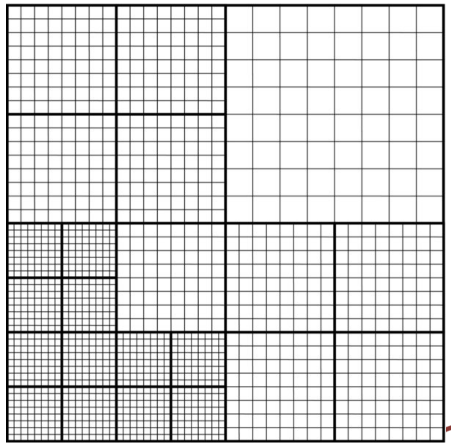
Grid sizes

May differ ← | → Same

Child grid have unique parent?

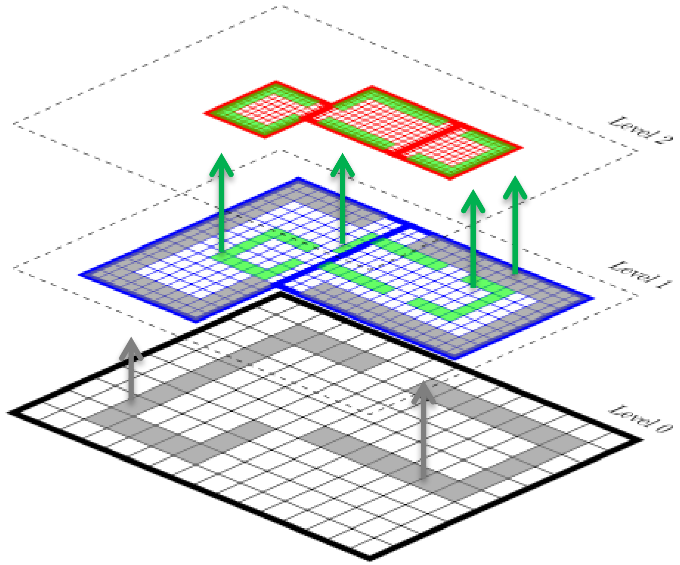
No ← | → Yes

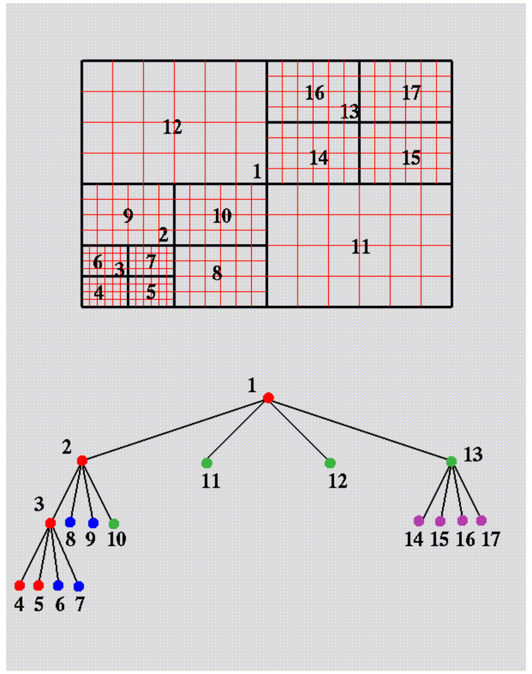
Example:
FLASH



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

# Patch-Based vs OctTree





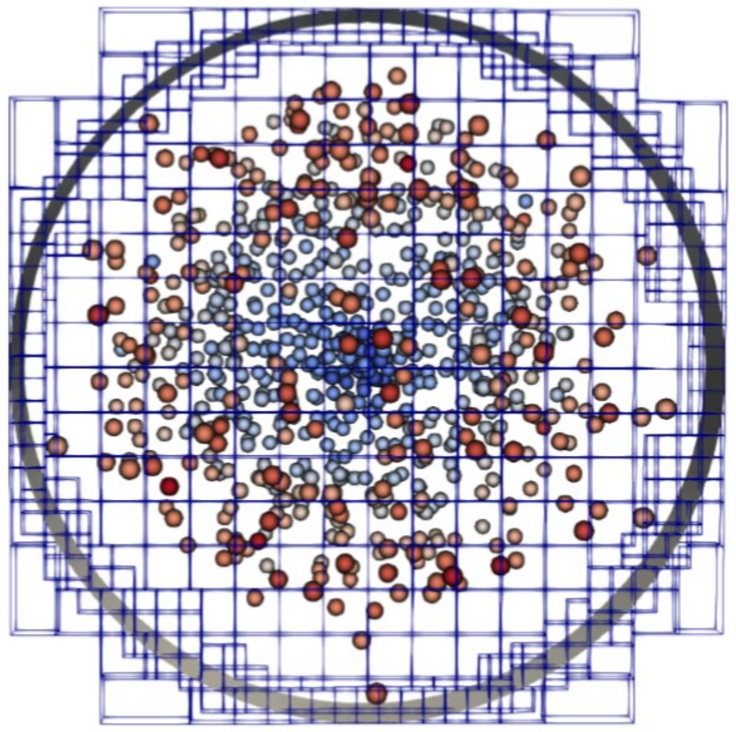
*http://cucis.ece.northwestern.edu/projects/DAMSEL/*

Both styles of block-structured AMR break the domain into logically rectangular grids/patches. Level-based AMR organizes the grids by levels; quadtree/octree organizes the grids as leaves on the tree.

# “AMR for One” does not have to mean “AMR for All”

For example, in the MFiX-Exa code, we define a level set that holds the distance to the nearest wall . The level set is only used by the particles to compute particle-wall collisions.

We refine the mesh on which the level set is defined in order to capture fine geometric features … but the particles and fluid are both defined on the coarser mesh only.



*AMReX: Johannes Blaschke*

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

# AMReX applications include …

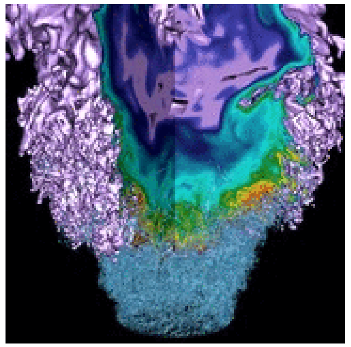
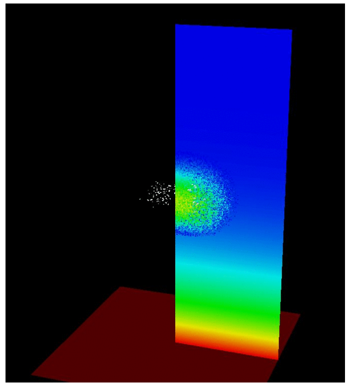
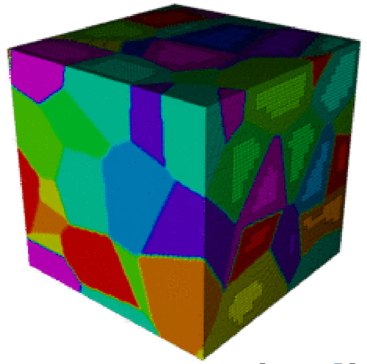
AMR has a long history in compressible astrophysics and other compressible phenomena.

Applications using AMReX include

- Low Mach number Combustion – heat release may look very different on coarse and fine levels
- Low Mach number astrophysics – 1-d background state plus perturbational solution
- Moist atmospheric modeling
- Solid mechanics, e.g. microstructure evolution
- Lattice Boltzmann, cellular automata ….

Flows with particles add complexity when particles and grids interact

Especially interesting ways to use AMR include AMAR – i.e. different physics / algorithms at different levels of refinement

## What about Time-Stepping?

AMR doesn't dictate the spatial or temporal discretization on a single patch, but we need to make sure the data at all levels gets to the same time.

The main question is:

**To subcycle or not to subcycle?**

Subcycling in time means taking multiple time steps on finer levels relative to coarser levels.

Non-subcycling:

- Same dt on every grid at every level
- Every operation can be done as a multi-level operation before proceeding to the next operation, e.g.  if solving advection-diffusion-reaction system, we can complete the advection step on all grids at all levels before computing diffusion

Subycling:

- dt / dx usually kept constant
- Requires separation of "level advance" from "synchronization operations"
- Can make algorithms substantially more complicated

## AMReX Hands-On Examples

Let’s do a few hands-on exercises that demonstrate AMReX capability:

“**AMR 101**”: AMR for scalar advection

- Multilevel mesh data – fluid velocity on faces and tracer on cell centers
- Dynamic AMR
- Two time-stepping options:
    - Subcycling in time with refluxing (to enforce conservation)
    - No subcycling in time

Instructions for how to access and run the code are on the web page:

**https://xsdk-project.github.io/MathPackagesTraining2020/lessons/amrex/**

Let’s all move to the slack channel to ask questions and share results!

## Ten minutes for hands-on exercises …

| Question | | |
|---|---|---|
| How many steps in subcycling vs not to reach t = 2? | | |
| Total time with subcycling vs not? | | |
| Was phi conserved? | | |
| How did run times compare with MPI? | | |
| Why could we just check conservation by summing at the coarsest level? | | |

EXASCALE COMPUTING PROJECT

# AMR 101:  Take-away

| Question | On my workstation | Take-away |
|---|---|---|
| 1. How many steps in subcycling vs not to reach t = 2? | 49 coarse / 392 fine (subcycling) vs 364 (no subcycling) | Additional factors – such as how often we compute dt – can change the answer |
| 2.  Total time with subcycling vs not? | Roughly 4:3 – subcycling faster | More steps at fine level but fewer total cells advanced -- which is faster may depend on how much work at each level |
| 3. Was phi conserved? | Yes with no-subcycling<br>Yes with subcyling but only if do_refluxing = 1 | Either approach can be conservative, but conservation is achieved differently |
| 4. How did run times compare with MPI? | Roughly 3x speedup with 4 MPI processes | Perfect scaling requires having enough work, making sure all parts of the code scale, etc |
| 5. Why could we just check conservation by summing at the coarsest level? | | Because we made sure to explicitly "average down" the fine solution onto the coarser meshes |

## Beyond Linear Advection

This tutorial wasn't actually very complicated, but hopefully it suggests that if you don't want to write a parallel GPU-ready AMR code from scratch, something like this might be a good starting point…

**Key things you need to know if you want to use AMR**:

- How to advance the data one patch at a time
- What the right matching conditions are at coarse/fine interfaces
  - This depends very much on the type of equation, e.g. hyperbolic vs elliptic, and what features of the solution are important (e.g., is conservation important?)
  - How you solve on the patch (e.g. with implicit vs explicit update) affects how you synchronize between levels

EXASCALE COMPUTING PROJECT

# Let’s model the dye a different way …

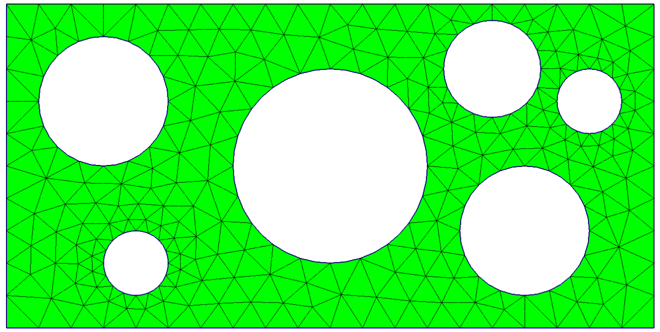
Let’s imagine instead that we model the dye as a collection of particles. And let’s make the flow more interesting by inserting an obstacle in the flow.

In addition to mesh data, AMReX supports

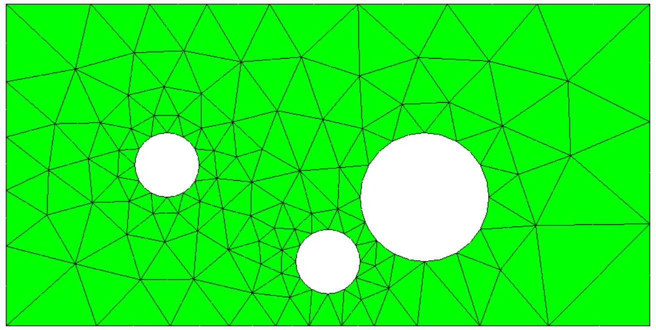
- particle data – multiple “types” with different numbers of attributes
- geometric data for solid obstacles/boundaries in the form of “cut cell” quantities (EB = embedded boundary representation)

Note that the cut cell / embedded boundary approach in a structured mesh is very different than an unstructured mesh:

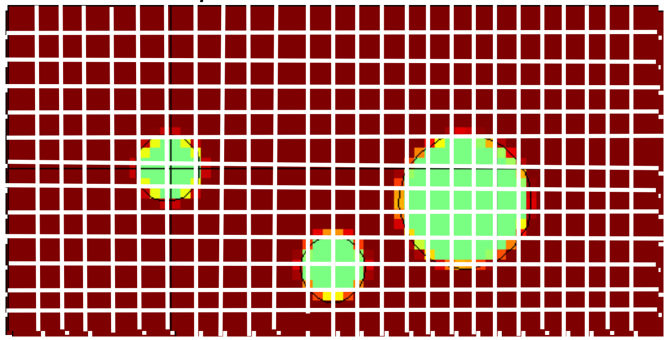
- In a structured mesh, “creating” the geometry means locally intersecting the object with the mesh
- In an unstructured mesh, “creating” the geometry means defining the entire mesh in a way that aligns with the object



*Unstructured meshes change with the geometry*



*Structured meshes don’t … but we need to compute new intersections*



## AMReX Hands-On Examples

Let’s do another hands-on exercise

“**AMR 102**”: Particles and Linear Solvers and EB, Oh My!

- Fluid velocity initialized on cell faces
- Obstacle placed in the flow using EB approach
- Velocity field “projected” to ensure divergence-free flow around the object
- Passive particles move with the velocity field, mimicking the presence of the dye

Instructions for how to access and run the code are on the web page:

**https://xsdk-project.github.io/MathPackagesTraining2020/lessons/amrex/**

Let’s all move to the slack channel to ask questions and share results!

# Ten minutes for hands-on exercise …

| Question | On My Workstation | Take-away |
|---|---|---|
| Did the particles make the same pattern as the tracer did? | | |
| Did it take a lot of time to generate the geometric information associated with the obstacle in the flow? | | |
| How does the the total run time of this method compare with our previous method (for single level)? | | |
| Was phi conserved with this approach? | | |

## AMR 102: Take-away

| Question | On My Workstation | Take-away |
|---|---|---|
| Did the particles make the same pattern as the tracer did? | Without the obstacle, yes.<br>With the obstacle, it was a different flow field | Both particles and mesh data can serve similar purposes… and we can move back and forth between them. |
| Did it take a lot of time to generate the geometric information associated with the obstacle in the flow? | Running the 3D calculation 216 steps to reach t=2, my total time was 33.8 seconds. Of that only 0.014s were spent creating the geometric information. | With structured grids, changing the geometry is much cheaper than with unstructured grids. |
| How does the the total run time of this method compare with our previous method (for single level)? | Amr101 was much faster!<br>(Be sure to set max_level = 0 in Amr101, and move the cylinder outside the domain in 102.) | Linear solvers are often the most expensive part of the simulation. |
| Was phi conserved with this approach? | Yes – because particles never lose their strength, and the deposition algorithm is conservative | |

EXASCALE COMPUTING PROJECT

## Software Support for AMR

There are a number of AMR software packages available –

They all

- Provide data containers for blocks of data at different resolutions
- manage the metadata – same-level and coarse-fine box intersections
- manage re-gridding (creation of new grids based on user-specified refinement criteria)

They differ on:

- what types of data they support – e.g. mesh data on cell-centers vs nodes, particles, …?
- what types of time-stepping they support (many are no-subcycling only)
- whether they support separate a "dual grid" approach
- what degree of parallelism do they support?  MPI only, MPI+X (what X?)
- what task iteration support – asynchronous, fork-join, kernel launching…?
- how flexible is the load balancing?
- what additional "native" features – e.g. AMG/GMG solvers?

## In addition to what you've seen, AMReX supports:

- a "dual grid" approach – particles, e.g. can live on different grid layout than fluid does

- MPI + OpenMP on multicore; MPI + CUDA (HIP/DPC++) on GPUs
  - support using lambdas for kernel launching on CPU vs GPU
  - (Can also use OpenMP / OpenACC)
  - Kernels can be C++ or Fortran
  - Performance portability – e.g., set USE_CUDA = TRUE or FALSE at compile-time

- task iteration– asynchronous, fork-join, kernel launching…
  - new feature is support for fusing GPU kernel launches

- flexible load balancing

- "native" AMR/GMG solvers

- "native" **async** I/O along with support for HDF5 (WIP)
  - format supported by Visit, Paraview, yt

# You shouldn’t have to use just one package:

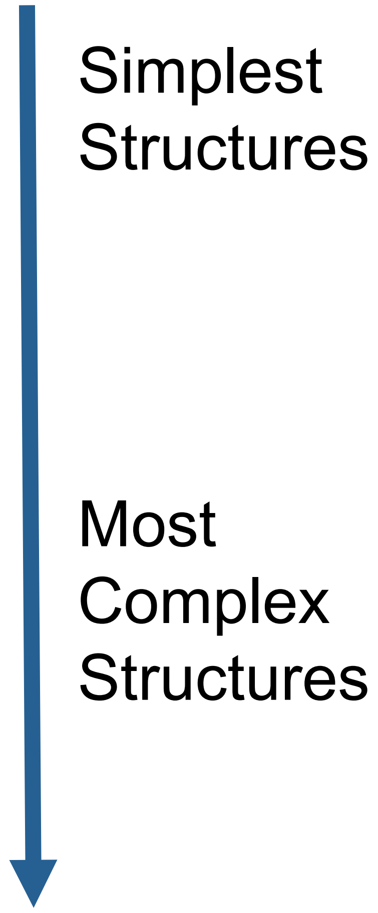
Suppose your linear systems are too “hard” for geometric multigrid?
- Call **hypre/petsc** – as a solver for the full equation, or as a “bottom solver” in the GMG hierarchy

Suppose you want to experiment with different time-stepping schemes?
- AMReX is interoperable with **SUNDIALS** (see Time Integration section) – SUNDIALS time integrators understand MultiFABs …

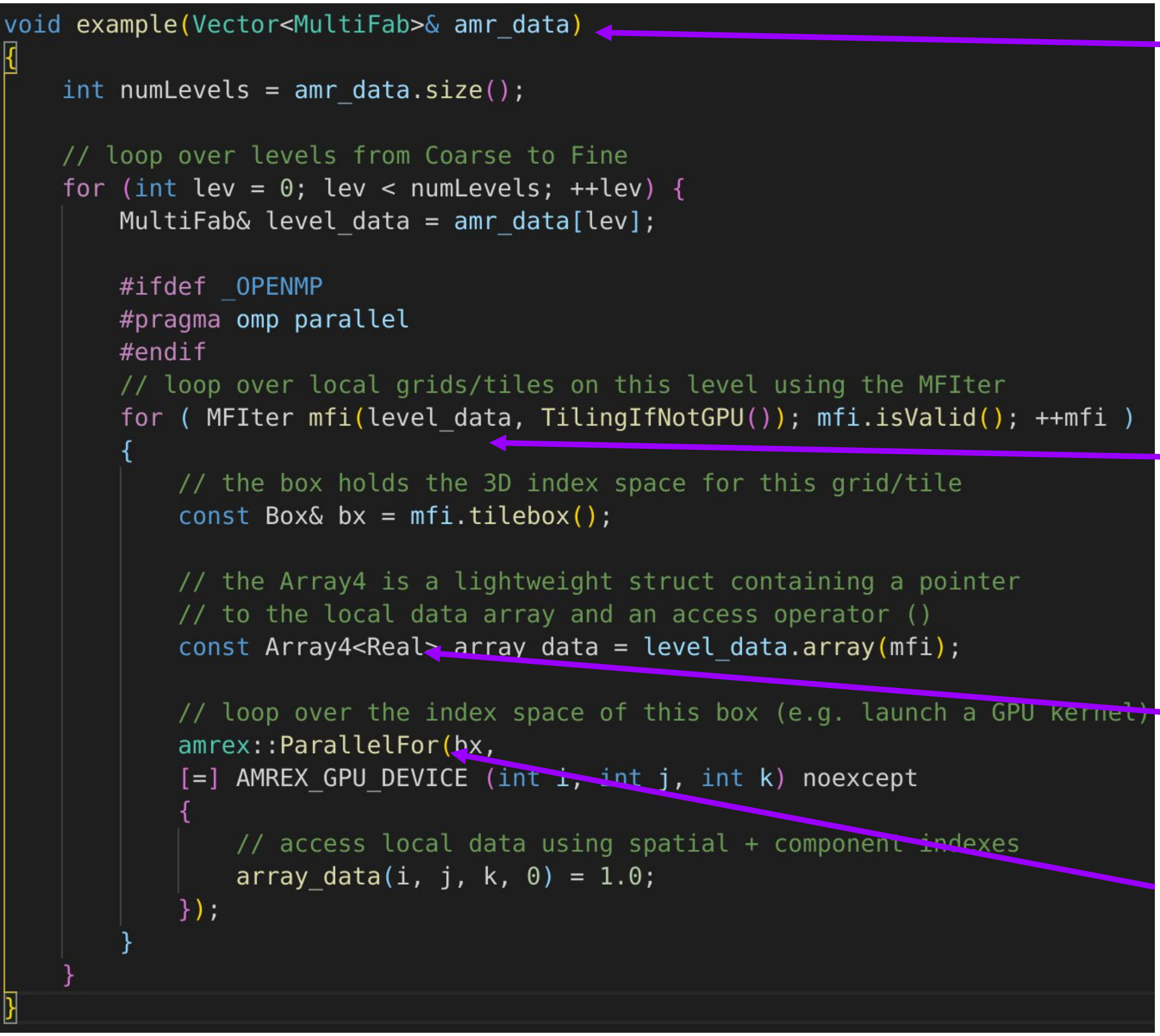
Suppose your equations are too painful to type out in stencil form?
- Use the new “**CodeGen**” python/sympy → AMReX translator to express – in ready-to-compile code – the initial data and right hand side for your time evolution equation

## AMReX Core Mesh Data Hierarchy

- **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.

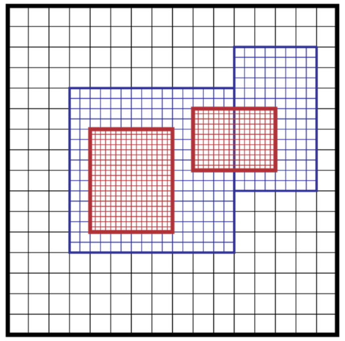


# How AMReX Loops over Mesh Data

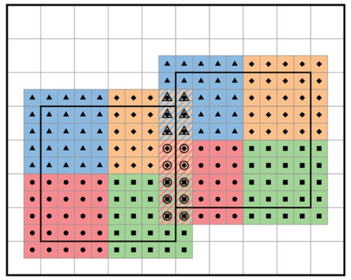


```
void example(Vector<MultiFab>& amr_data)
{
    int numLevels = amr_data.size();

    // 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;
            });
        }
    }
}
```

- MPI-distributed data at all levels stored as a vector of AMReX MultiFab objects

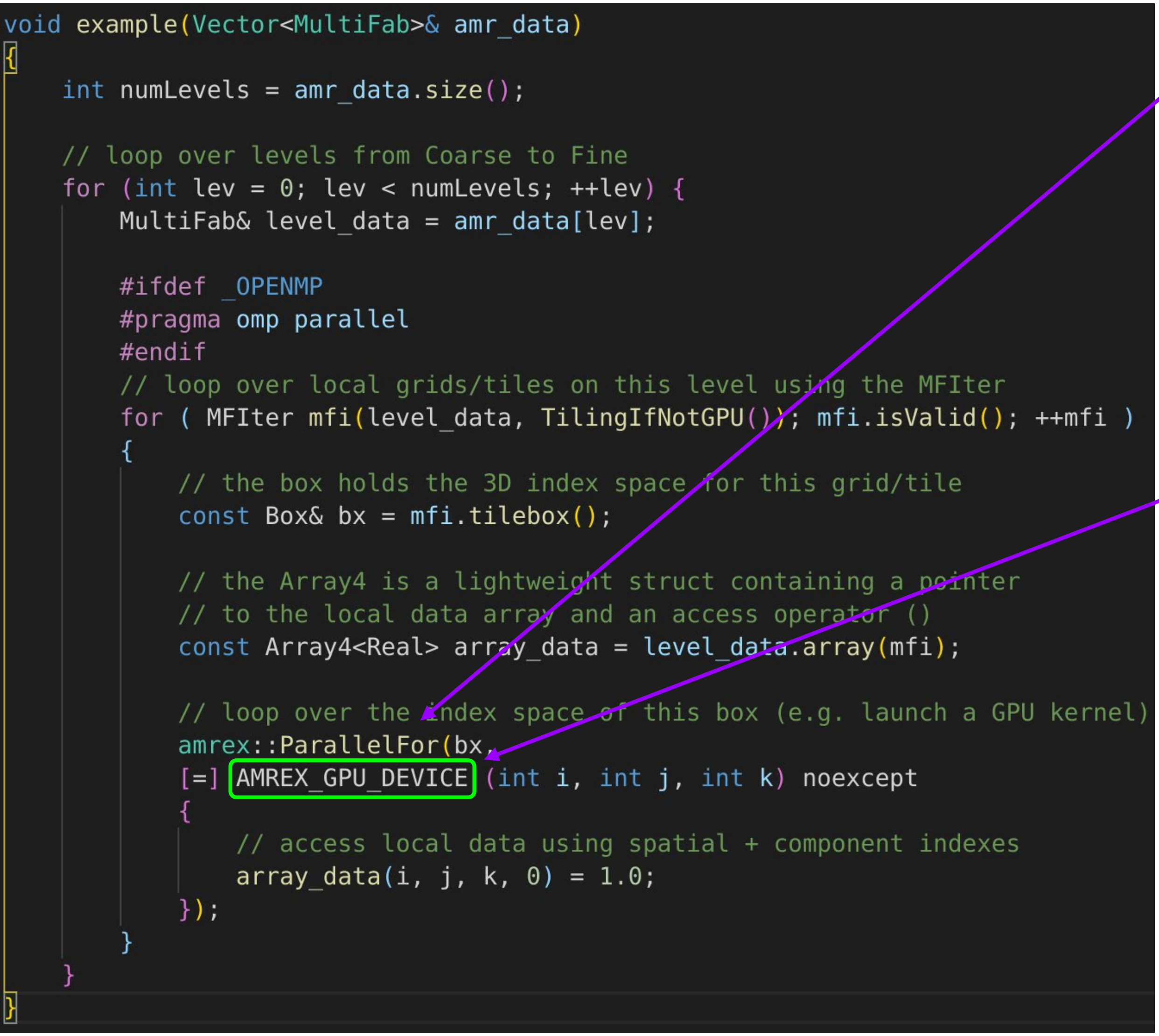


- Each **MultiFab** contains **pointers to local grid data** for one MPI rank + **grid distribution** metadata



- Loop over local data with the **M**ulti**F**ab **Iter**ator.
  For OpenMP, generate logical tiles for local grids.
- The **Array4** contains a pointer and access operator().
  The lambda captures it by value.
- The **ParallelFor** takes index space in a box and a C++ lambda function to call on each 3D index

# Built-In AMReX Features Enable Straightforward GPU Acceleration



```
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.
- AMReX memory arena uses **CUDA Unified Memory**
- AMReX **ParallelFor** launches CUDA kernel
- All we had to do was label our “work” lambda function as a GPU function!

# In summary …

Recall what we've seen in our examples…

- "**AMR 101**": AMR for scalar advection
  - Multilevel mesh data – fluid velocity on faces and tracer on cell centers
  - Subcycling in time (or not) with refluxing (or not)
  - Dynamic AMR
- "**AMR 102**" : use a Poisson solve to compute incompressible flow around an obstacles then advect the particles in that flow field
  - Single-level mesh data – fluid velocity on faces, EB obstacles defined by volume and area fractions
  - Linear solver (geometric multigrid)
  - Particle advection

We didn't have time for this one, but "**AMReX-Pachinko**" is an example of particles falling under gravity through an obstacle course, bouncing off the solid obstacles – here we see an example of particle-obstacle and particle-wall collisions

**https://xsdk-project.github.io/MathPackagesTraining2020/lessons/amrex/**

## Take Away Messages

- Different problems require different spatial discretizations and different data structures – the most common are
  - Structured mesh
  - Unstructured mesh
  - Particles (which can be combined with structured and/or unstructured meshes)

- Structured mesh doesn't equal "just" flow in a box

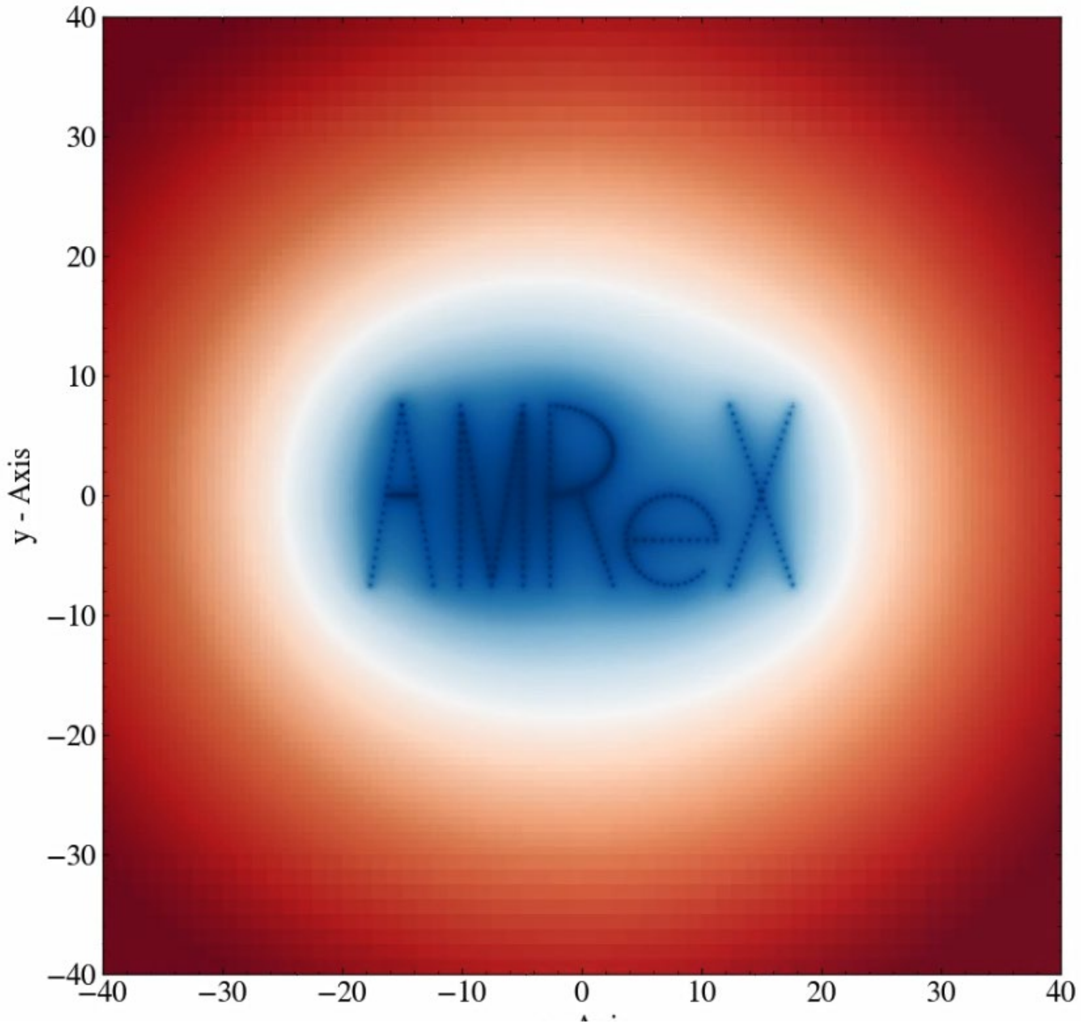
- There are quite a few AMR software packages – they have several commonalities and a large number of differences, both in what functionality they support and on what architectures they are performant

- Interoperability is important! See the next few sessions for how different packages can be used together.

If you're interested in learning more about AMREX:

- the software: https://www.github.com/AMReX-Codes/amrex
- the documentation: https://amrex-codes.github.io/amrex
- some movies based on AMReX: https://amrex-codes.github.io/amrex/gallery.html

## A final takeaway …


y - Axis
AMReX