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

Presented to
ATPESC 2022 Participants

Ann Almgren

Deputy Director, AMReX ECP Co-Design Center
Senior Scientist and Dept. Head, LBNL

Erik Palmer
Software Integration Engineer, NERSC, LBNL

Date 08/9/2022

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

Argonne NATIONAL LABORATORY · BERKELEY LAB · Sandia National Laboratories · Rensselaer · SMU

AMReX Hands-On Setup Instructions

- **1. Login into Theta:**

```
ssh -A <username>@theta.alcf.anl.gov
```

- **2. Create a local copy of the examples:**

```
cd ~
rsync -a /grand/ATPESC2022/EXAMPLES/track-5-numerical .
```

- **3. Log onto a GPU service node:**

```
ssh thetagpusn1 # or thetagpusn2
```

- **4. Request a session on a ThetaGPU node:**

```
qusb -I -q single-gpu -t 60 -n 1 -A ATPESC2022
```

- **5. Load the MPI module:**

```
module load openmpi/openmpi-4.1.4_ucx-1.12.1_gcc-9.4.0
```

- **6. Move to the AMReX directory:**

```
cd ~/track-5-numerical/amrex
```

- **7. Set AMReX environment variables and paths:**

```
source amrex_setup_env.sh
```

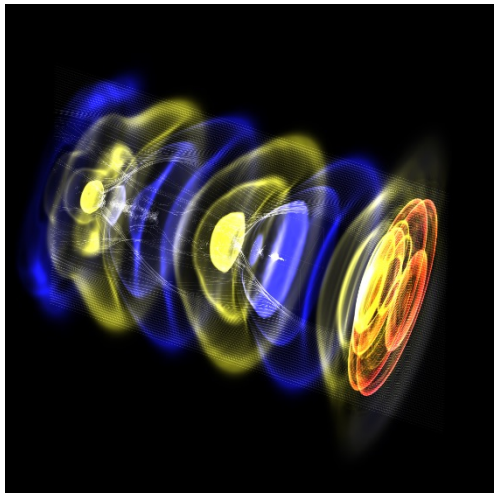
Setting the Stage

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.

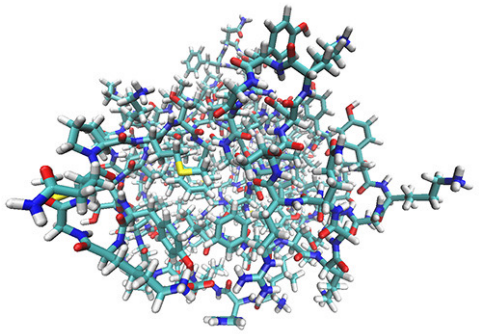
Gone are the days of simple physics, simple geometry, single algorithm, homogeneous architectures … ☹

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

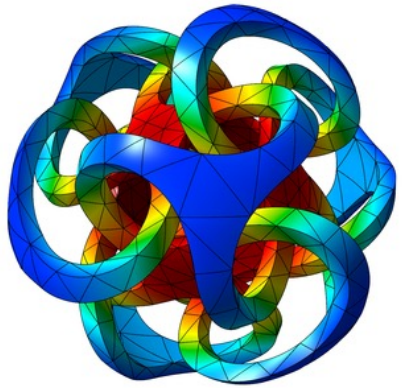


WarpX project: Jean-Luc Vay, PI

Setting the Stage (p2)



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



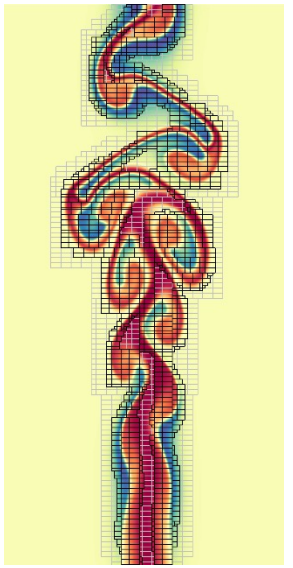
https://ceed.exascaleproject.org/vis/

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

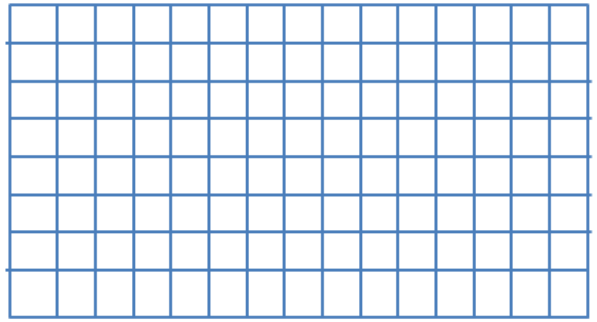


AMReX: Emmanuel Motheau

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

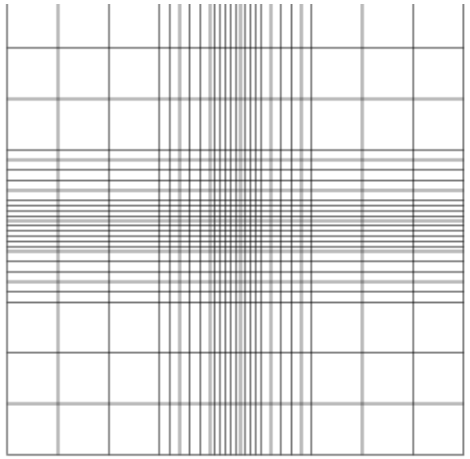


https://commons.wikimedia.org/wiki

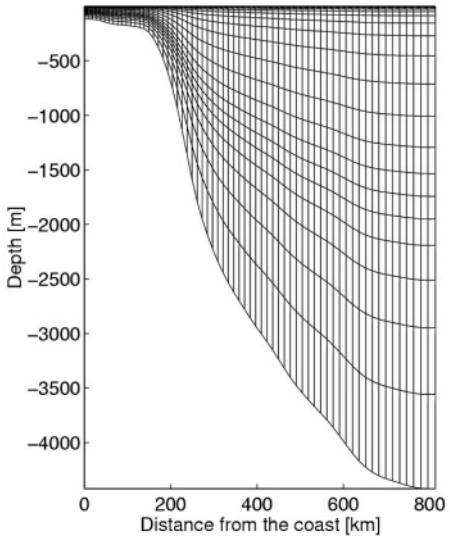
Logically rectangular doesn't mean physically rectangular

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

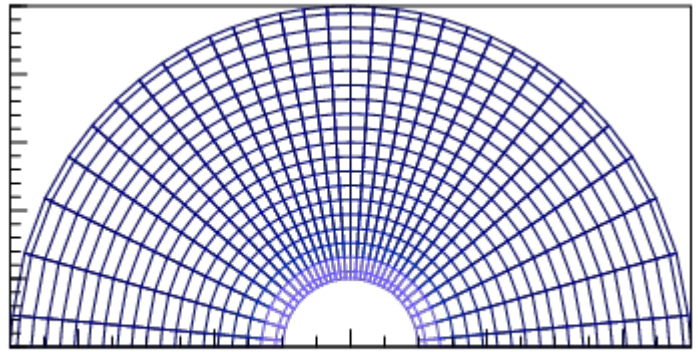
- Can fit (simple) non-rectangular boundaries while still having known connectivity
- Finer in certain regions (mesh refinement)
- Harder to maintain accuracy



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

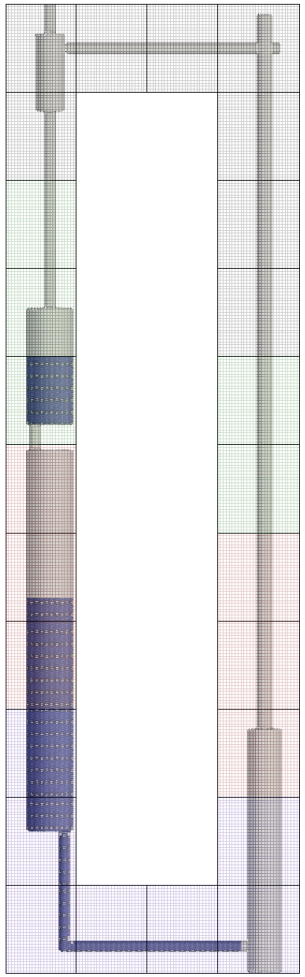


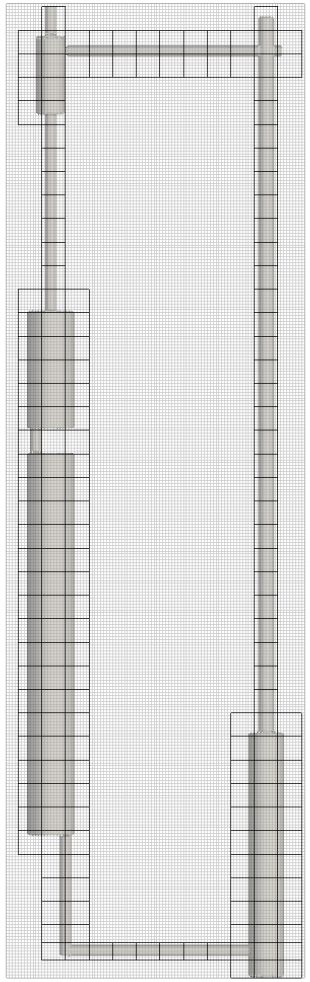
http://www.cfoo.co.za/simocean/modelsroms.php



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:

$$\phi_{i+1} = \phi(x_i) + \Delta x_r \phi_x + 1/2(\Delta x_r)^2 \phi_{xx} + O(\Delta x_r{}^3)$$

$$\phi_{i-1} = \phi(x_i) - \Delta x_l \phi_x + 1/2(\Delta x_l)^2 \phi_{xx} + O(\Delta x_l{}^3)$$

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

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

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

Can we confine this error?

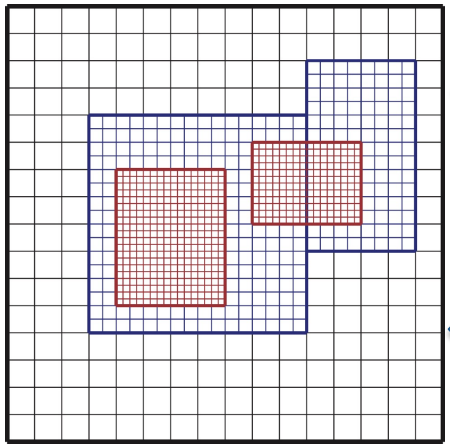
Can We Have the Best Of Both Worlds?

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



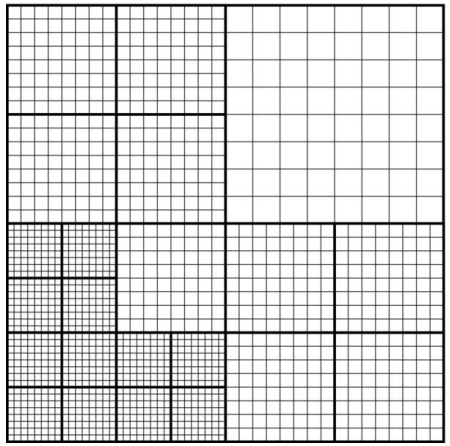
Grid sizes

May differ ← → Same

Child grid have unique parent?

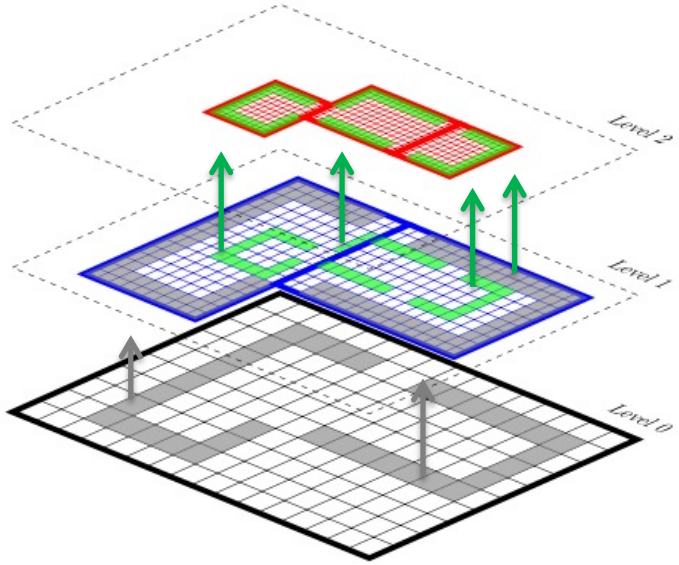
No ← → Yes

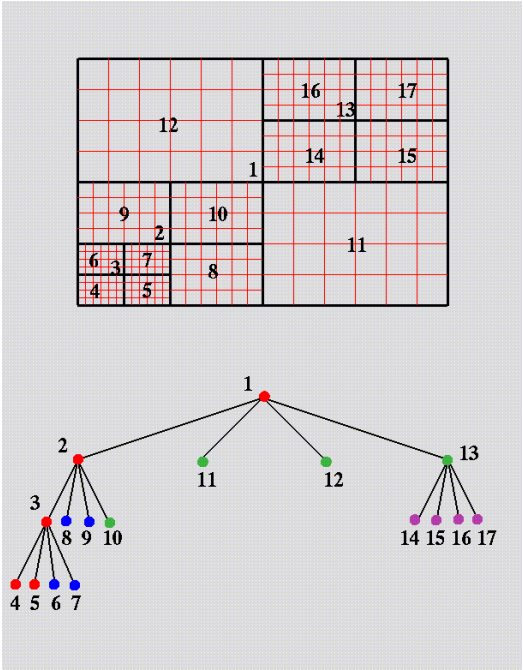
Example:
FLASH



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.

What is an “AMR algorithm”?

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

- How to advance the solution 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

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 capabilities

"**AMR 101**": AMR for scalar advection

- Multilevel mesh data – fluid velocity on faces and tracer on cell centers
- Dynamic AMR
- Strong scaling behavior
- Performance portability (CPU vs GPU)

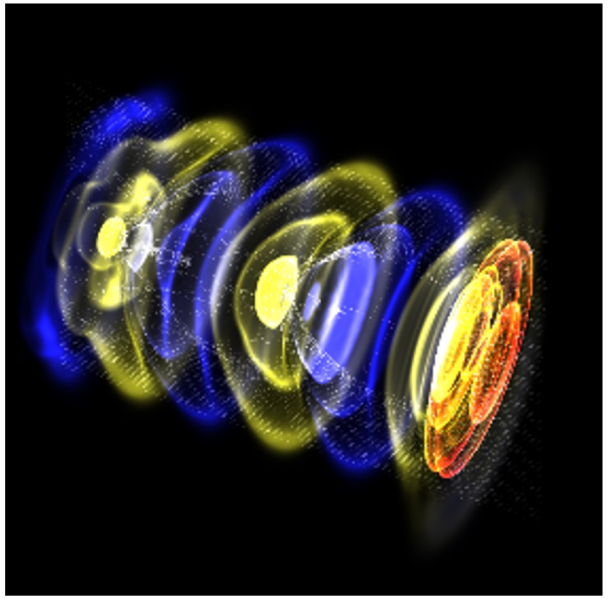
Instructions for how to access and run the code are on the web page:

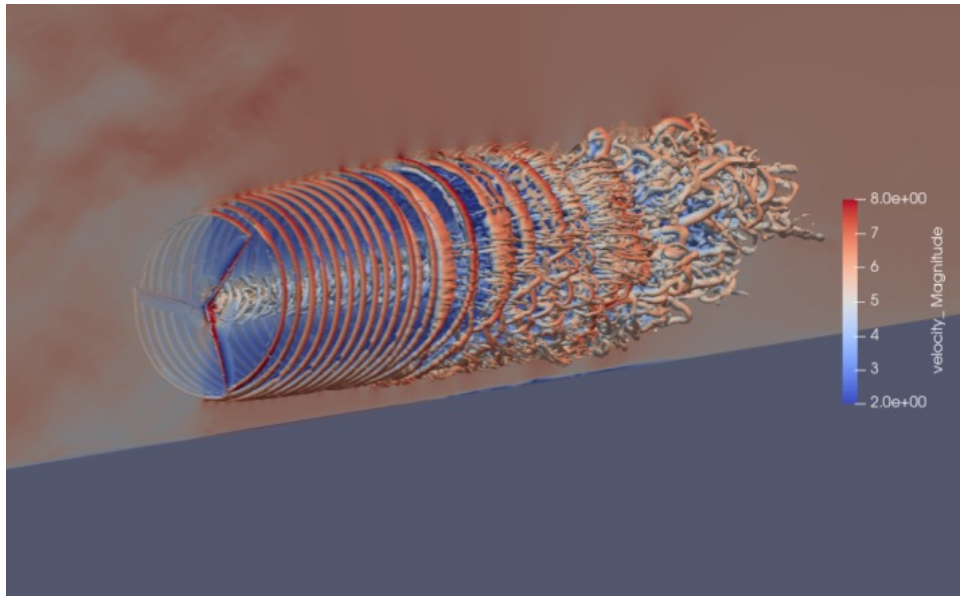
https://xsdk-project.github.io/MathPackagesTraining2022/lessons/amrex/

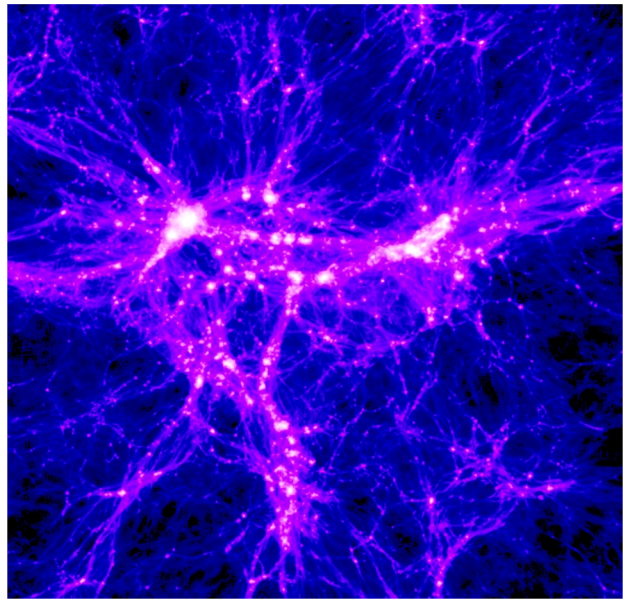
Let's all move to the slack channel to ask questions and share results!

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…

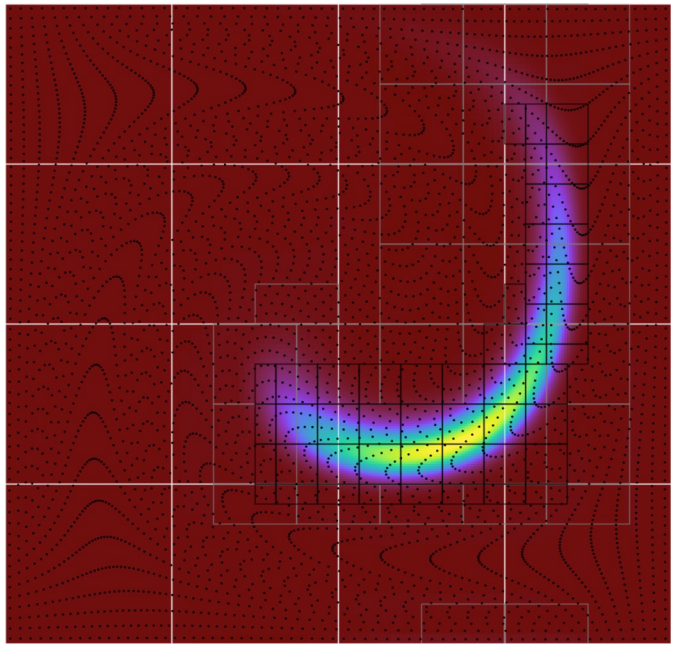






“Particles” in AMReX

- AMReX provides tools for storing and iterating over **distributed particle data** associated with an adaptive mesh refinement hierarchy.
- Flexible data layout allows users to choose between **AoS** and **SoA** (or a combination of the two) at compile-time.
- MPI communication routines for **redistribution** and **halo exchange**.



- Tools for building and iterating over **neighbor lists,** particle-mesh **deposition** and **interpolation** operations.
- Functions for **binning**, **sorting**, or **partitioning** particles, parallel **reductions**, **stream compaction** operations.
- All functionality works with **NVIDIA**, **AMD**, and **Intel** GPUs, and supports **OpenMP** for CPU-only execution.
- Particle methods implemented with AMReX have been scaled up to the full sizes of some of the biggest supercomputers in the world: **Summit**, **Frontier**, **Fugaku**, and **Perlmutter**.

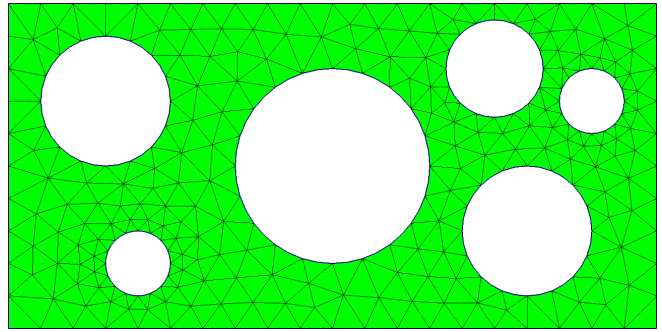
Let’s model the dye a different way …

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.

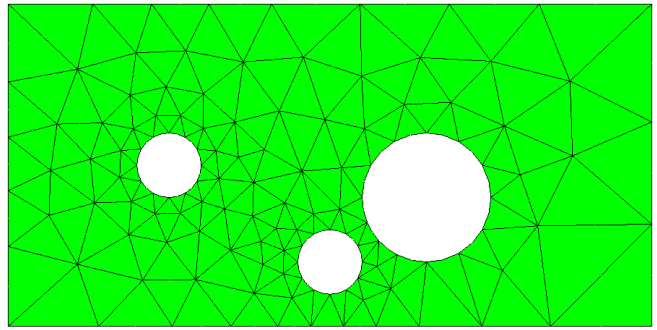
In addition to mesh and particle data, AMReX supports geometric data for solid obstacles/boundaries in the form of “cut cell” quantities (EB = embedded boundary representation)

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

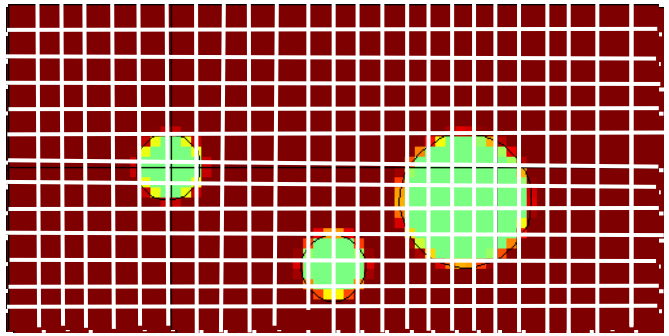
- In a structured mesh, “creating” the geometry means locally intersecting the object with the mesh
- In an unstructured mesh, “creating” the geometry means defining the entire mesh in a way that aligns with the object



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/MathPackagesTraining2022/lessons/amrex/

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

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?
- how many flavors of GPUs they can use?

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, kernel fusing…

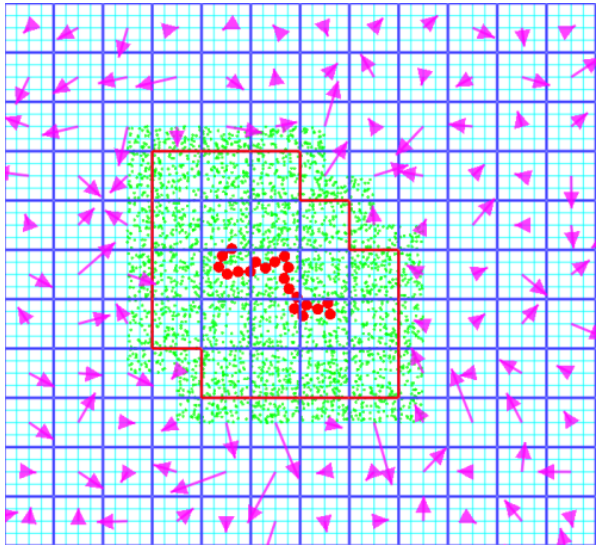
- flexible load balancing

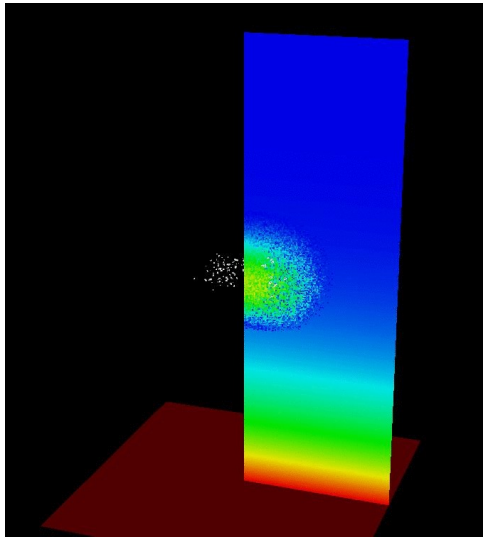
- "native" AMR/GMG solvers

- "native" async I/O along with support for HDF5 (some applications also use NetCDF)
  - format supported by Visit, Paraview, yt

Fun extensions include:

- **AMAR** – different algorithms at different levels (e.g. particle vs fluid representation)
- **Multiblock** – coupling same or different codes in different “blocks”, i.e. spatial regions





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

In summary …

Recall what we've seen in our examples…

- "**AMR 101**": AMR for scalar advection
  - Multilevel mesh data – fluid velocity defined on faces and scalar ("ink") defined on cell centers
  - Dynamic AMR
  - Strong scaling behavior and CPU vs GPU ("performance portability")

- "**AMR 102**" : use a Poisson solve to compute incompressible flow around an obstacles and 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/MathPackagesTraining2022/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
- the tutorials: https://amrex-codes.github.io/amrex/tutorials_html/
- some movies based on AMReX: https://amrex-codes.github.io/amrex/gallery.html

A final takeaway …