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

Presented to
ATPESC 2019 Participants

Ann Almgren

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

Don Willcox
Postdoctoral Researcher, LBNL

Q Center, St. Charles, IL (USA)
Date 08/06/2019

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

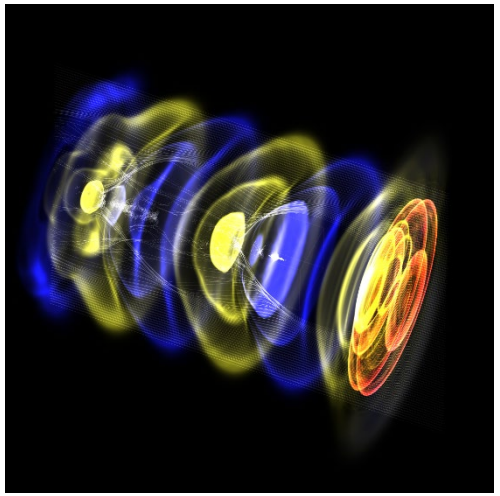
Setting the Stage

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 … ☹

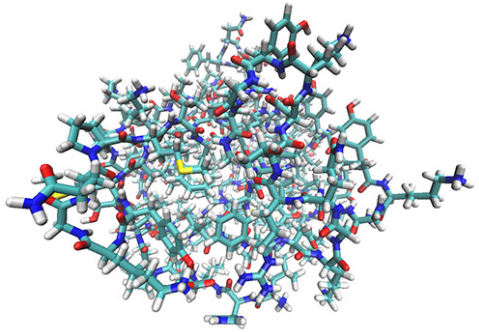
So how do we build algorithms and software for hard multi-physics multi-scale multi-rate problems without starting over every time?



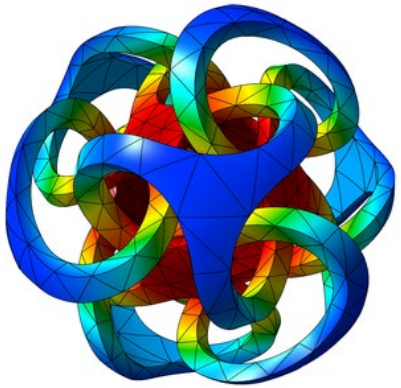
WarpX project: Jean-Luc Vay, PI

Setting the Stage (p2)

Not all simulations use a mesh

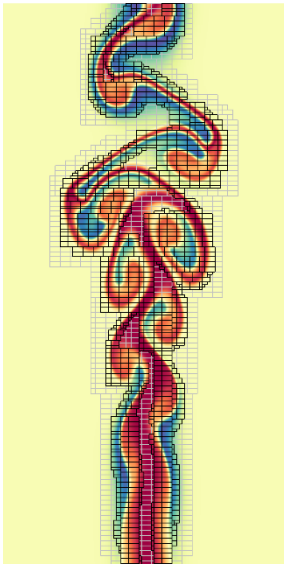


TINKER: https://www.epcc.ed.ac.uk



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

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



AMReX: Emmanuel Motheau

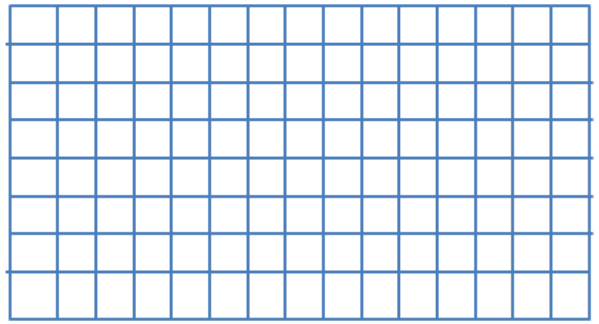
Structured:

- 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

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

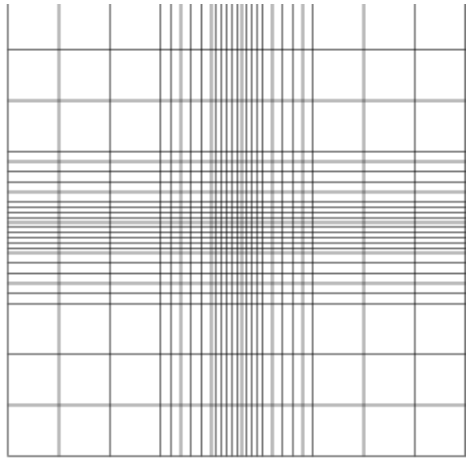


https://commons.wikimedia.org/wiki

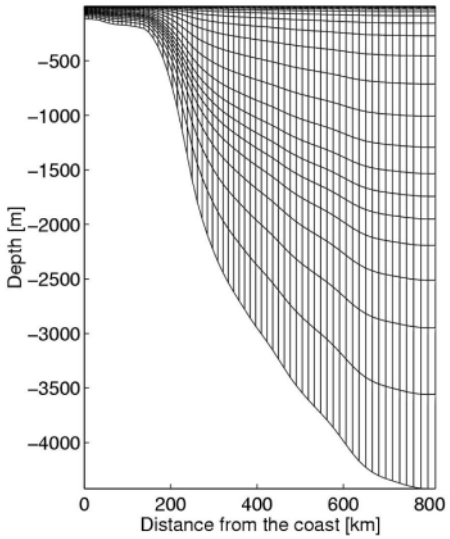
Logically rectangular doesn't mean physically rectangular

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

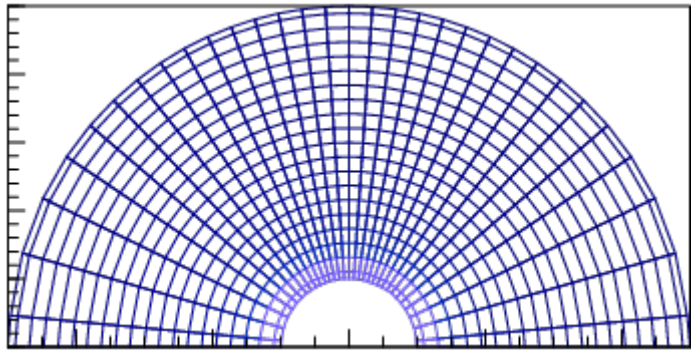
- Can fit (simple) non-rectangular boundaries while still having known connectivity
- Finer in certain regions (mesh refinement)
- Harder to maintain accuracy



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

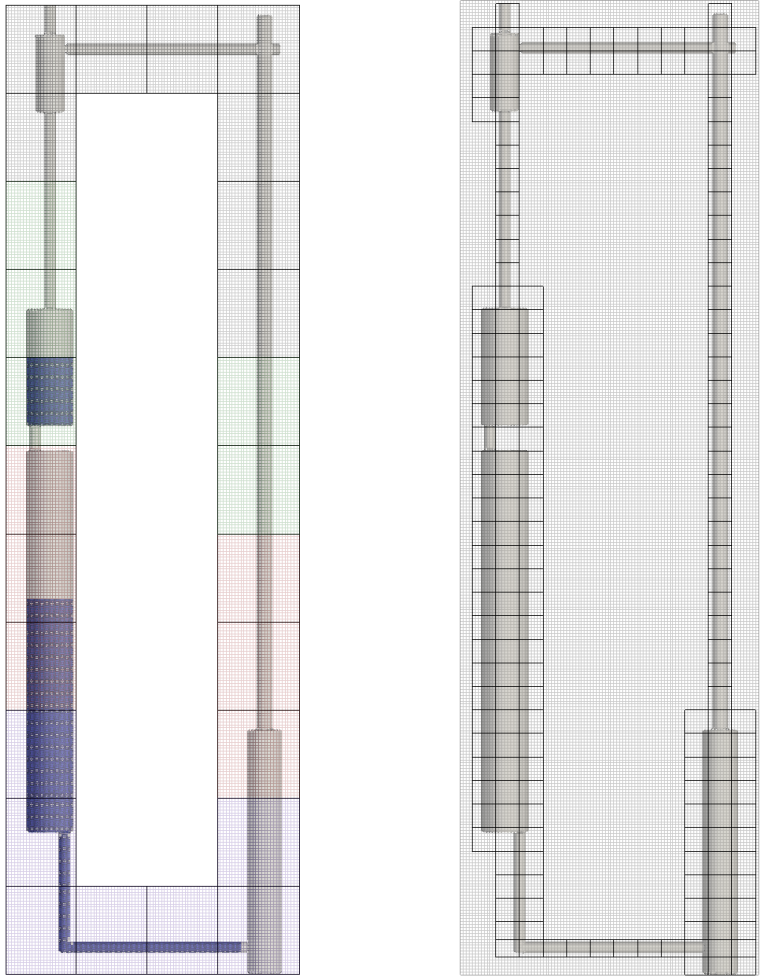


http://www.cfoo.co.za/simocean/modelsroms.php



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?

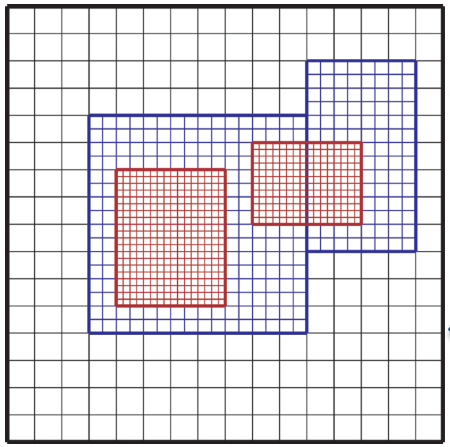
Can We Have the Best Of Both Worlds?

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



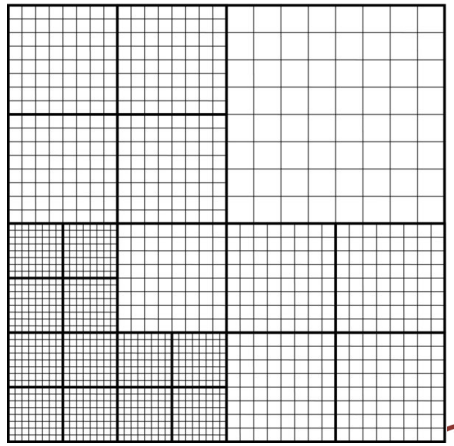
Grid sizes

May differ ← → Same

Child grid have unique parent?

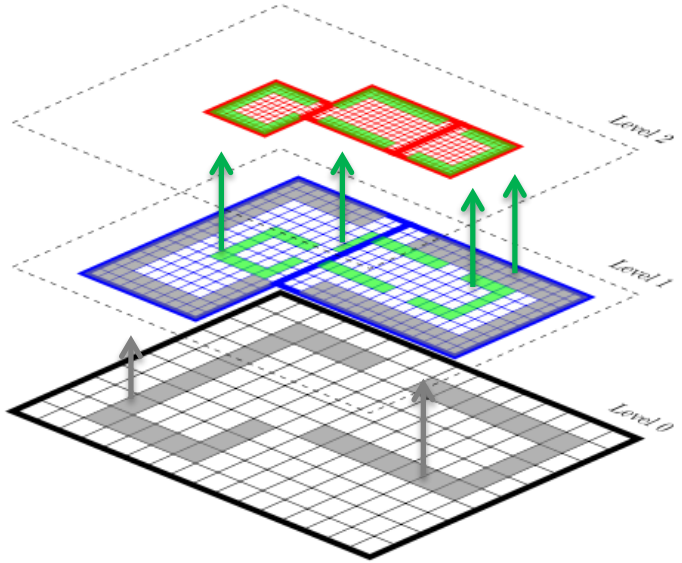
No ← → Yes

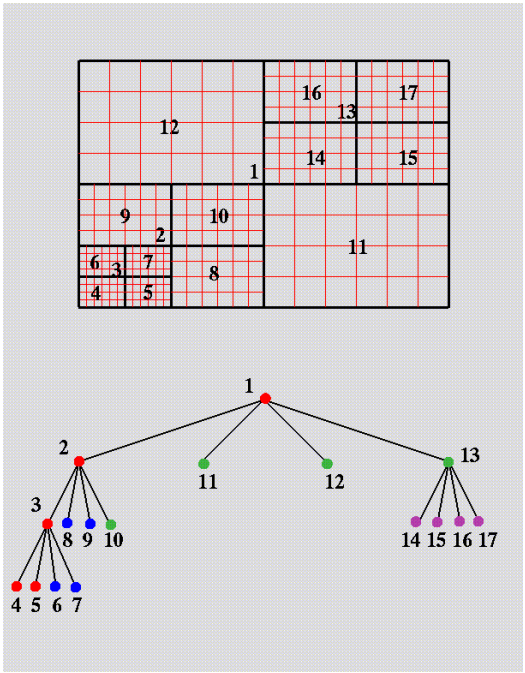
Example:
FLASH



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

Patch-Based vs OctTree





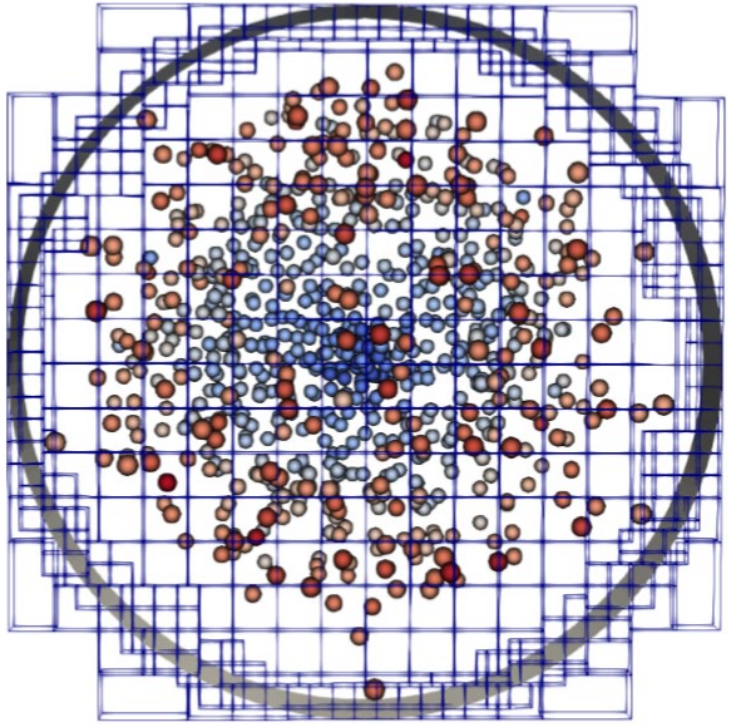
http://cucis.ece.northwestern.edu/projects/DAMSEL/

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.

“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.

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 applications include …

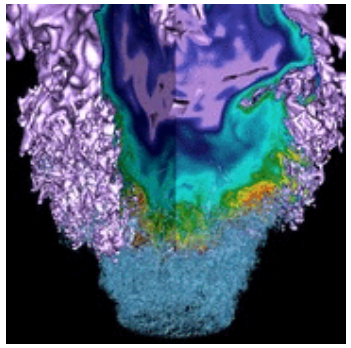
AMR has a long history in compressible astrophysics and other compressible phenomena.

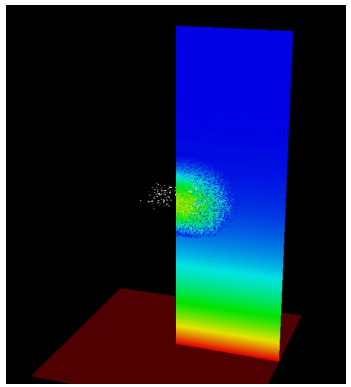
Extensions of AMR usage include

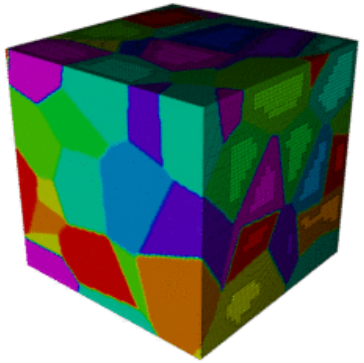
- Low Mach number Combustion – heat release may look very different on coarse and fine levels
- Low Mach number astrophysics – 1-d background state plus perturbational solution
- Moist atmospheric modeling
- Solid mechanics, e.g. microstructure evolution
- Lattice Boltzmann, cellular automata ….

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







Software Support for AMR

This may convince you that you don't want to write an AMR code from scratch!

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
- 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. AMR/GMG solvers?

AMReX:

Manages data containers for blocks of mesh and particle data at different (mesh) resolutions

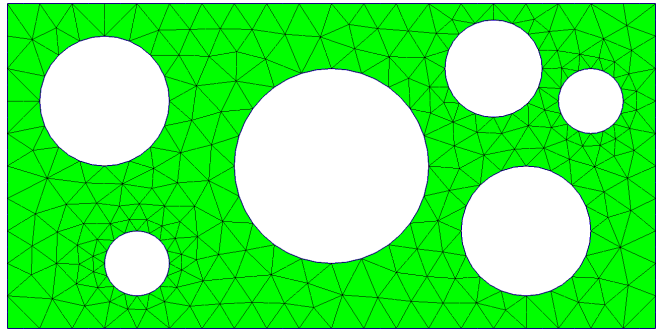
- mesh data on cell centers, faces and nodes
- particle data – multiple “types” with different numbers of attributes
- geometric data for solid obstacles/boundaries in the form of “cut cell” quantities (EB = embedded boundary representation)

Allows operation on data via “iterators” – performance portability between CPU-based and GPU-based operations (“Same kernel, different launch”)

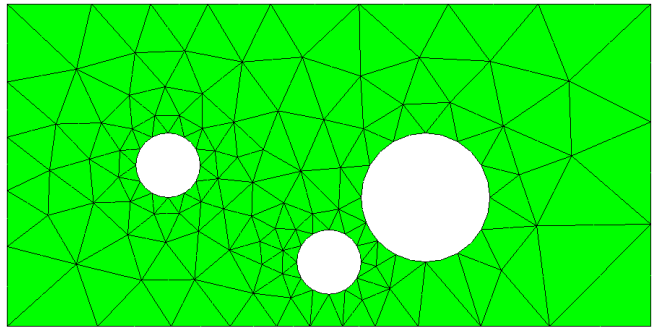
Handles all the metadata, aka “book-keeping”

- caches same-level and coarse-fine box intersections

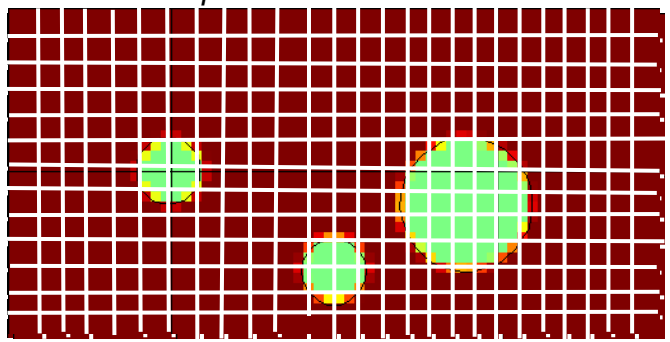
Manages re-gridding (= creation of new grids / deletion of old grids based on user-specified refinement criteria)



Unstructured meshes change with the geometry



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



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) 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 – set USE_CUDA = TRUE or FALSE at compile-time

- task iteration– asynchronous, fork-join, kernel launching…

- flexible load balancing

- “native” AMR/GMG solvers

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

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.

What does a loop look like: Fortran vs C++?

Fortran

```
do k = lo(3), hi(3)
    do j = lo(2), hi(2)
       do i = lo(1), hi(1)+1
          fluxx(i,j,k) += 1.0
       end do
    end do
 end do
```

C++

```
Array4<Real> const& fab = mf.array(mfi);
for (int k = lo.z; k <= hi.z; ++k) {
    for (int j = lo.y; j <= hi.y; ++j) {
        for (int i = lo.x; i <= hi.x; ++i) {
            fab(i,j,k) += 1.;  }}}
```

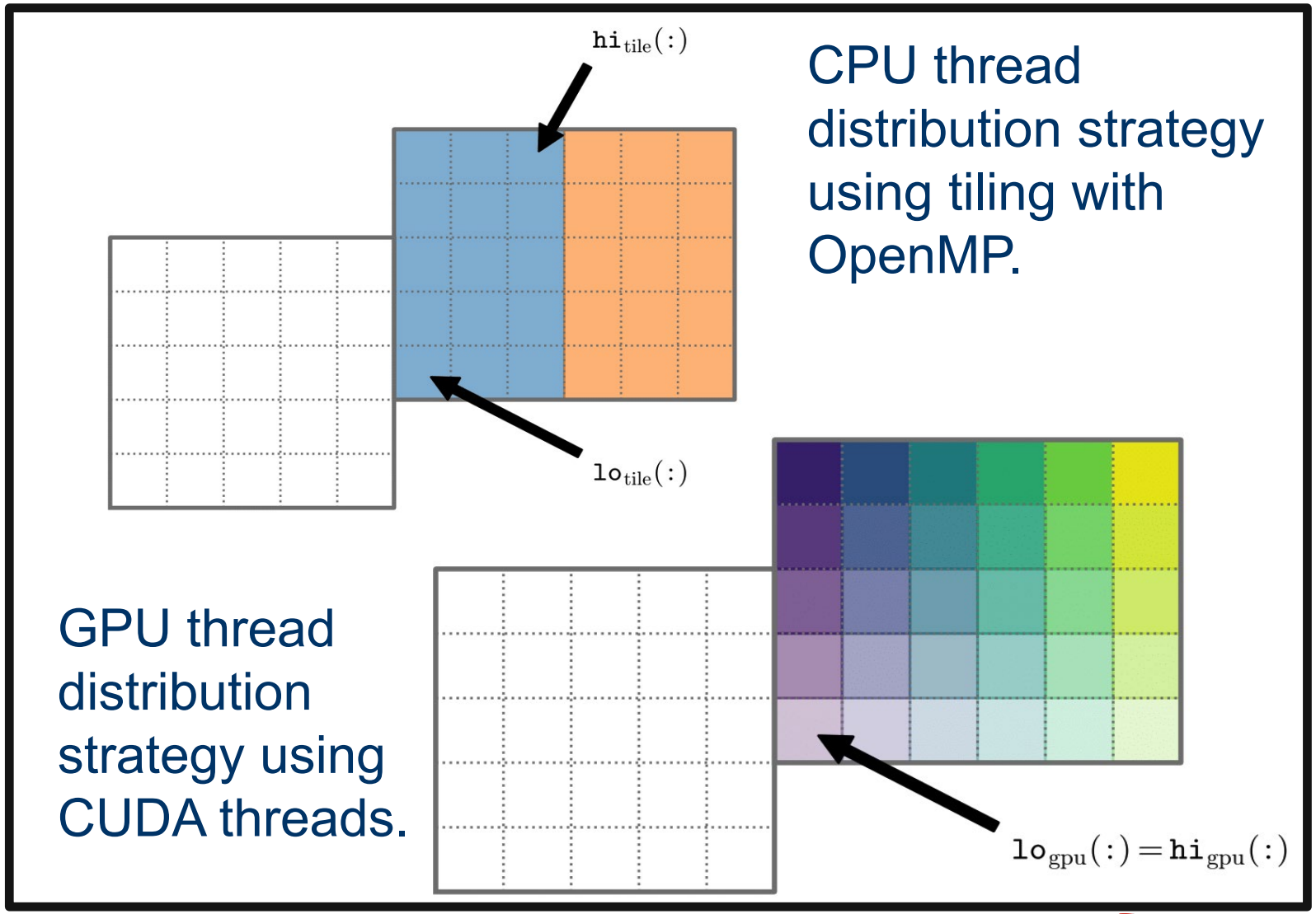
Array4<Real> holds:

- Pointer to the data of type **Real**
- Size of the object in 3D
- Striding information

and is fully accessible on the GPU.

CPU vs GPU Parallelism

- **OpenMP threads** across tiles on **local boxe**s (~10-100).
- GPU threads are on the order of local number of cells (~thousands).
- **GPU Parallelization** strategy is shifted to a finer-grained implementation **over cells**.



AMReX Loops over Mesh Data

```
phi_old.FillBoundary(geom.periodicity());
const Real* dx = geom.CellSize();
const Box& domain_bx = geom.Domain();
for ( MFIter mfi(phi_old); mfi.isValid(); ++mfi )
{
      const Box& xbx = mfi.nodaltilebox(0);
      Array4<Real> const& fluxx = flux[0].array(mfi);
      Array4<Real> const& phi = phi_old.array(mfi);

      amrex::ParallelFor(xbx,
      [=] AMREX_GPU_DEVICE (int i, int j, int k)
      {
          compute_flux_x(i,j,k,fluxx,phi,dxinv);
      });}

      // Additional launch for Y and Z fluxes.
}
```

Standard MPI Work:
Launches used to implement.

Loop over Boxes with local data.
CUDA streams incremented within.

Get Box and pointers to MultiFab data to work over.

Launch lambda function to perform over desired Box.

Device sync during MFIter destructor to guarantee data consistency.

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
  - Subcycling in time with refluxing (to enforce conservation)
  - Dynamic AMR

- "**Off to the races**" : use a Poisson solve to compute potential flow around 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

- "**AMReX-Pachinko**": let particles fall through an obstacle course, bouncing off the solid obstacles
  - Single-level mesh – EB obstacles only
  - Particle-obstacle and particle-wall collisions

https://xsdk-project.github.io/MathPackagesTraining/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 AMReX can be used in conjunction with SUNDIALS time stepping and TAO optimization packages.

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