Introduction to AMReX - a new framework for block-structured adaptive mesh refinement calculations

Andrew Myers
Lawrence Berkeley National Laboratory

Advanced Modeling and Simulation Seminar Series
NASA Ames Research Center
July 3, 2018
Outline

1. Overview of AMReX
2. Introduction to block-structured AMR
3. What tools does AMReX provide?
4. What can you build these tools?
5. Current development directions
6. Where to find more
Outline

1. Overview of AMReX
2. Introduction to block-structured AMR
3. What tools does AMReX provide?
4. What can you build these tools?
5. Current development directions
6. Where to find more
Overview of AMReX

- AMReX is an ECP-funded software framework to support the development of block-structured AMR applications for current and next-generation architectures
  
- Written in a mix of C++/Fortran

- Allows for a variety of algorithms, discretizations, and numerical approaches

- Supports a variety of programming models - MPI, OpenMP, Hybrid, MPI+MPI, and (increasingly) GPUs

- Provides the framework for many different application codes in combustion, astrophysics, accelerator
Outline

1. Overview of AMReX

2. Introduction to block-structured AMR

3. What tools does AMReX provide?

4. What can you build these tools?

5. Current development directions

6. Where to find more
Introduction to Block-Structured AMR

• Say you want to solve a system of time-dependent PDEs. If you had infinite compute power (and memory), you could discretize the equations on a uniform mesh and advance the solution in time with a fixed dt.
• Domain decomposition and parallelization could be done in a straightforward way

\[
U_{\text{new}}[i,j] = U_{\text{old}}[i,j] + \text{speed} \times dt
\]
Introduction to Block-Structured AMR

• Most problems are more complicated
• Multi-physics
• Multi-scale (time and space)

• Might have complex boundaries that make a regular domain impractical
• And, you don’t have unlimited resources
• Some form adaptivity is often needed to reduce memory footprint and compute cost
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 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

Maintains many of the nice features of uniform meshes:
- Connectivity is simple - uniform index space, even with AMR
- High-order methods
- Ease of vectorization, hierarchical parallelism
Example: RNS simulation of Hydrogen / Air Flame

Image courtesy of Emmanuel Motheau
Outline

1. Overview of AMReX
2. Introduction to block-structured AMR
3. What tools does AMReX provide?
4. What can you build these tools?
5. Current development directions
6. Where to find more
IntVect and Box

- Represent regions in an integer index space
  - IntVect represents a point
  - Box represents region covered by a patch
  - Coarsening / Refinement operators for both

- Box has notion of IndexType for representing nodal data:
BoxArray and DistributionMapping

- **BoxArray** stores a collection of boxes on a single level
  - All Boxes in a BoxArray share IndexType, methods for converting
  - methods for chopping, coarsening, refining
  - Internally uses std::shared_ptr to save memory
  - provide optimized functions for finding intersections

- **DistributionMapping** maintains a mapping between Box and MPI process.
  - Also has shared_ptr implementation
  - Several options for process assignment:
    - knapsack, space-filling curve, manual
Mesh data: FArrayBox, IArrayBox

- Store multi-dimensional arrays of mesh data associated with a single Box.
- Can be real, integer or other (implemented with templates)
- Data is stored in column-major order for ease of processing by Fortran subroutines
Distributed mesh data: MultiFab, IMultiFab

- Store arrays of mesh data associated with a BoxArray and DistributionMapping.
- Data is distributed.
- Provide parallel copy routines, ghost cell filling, all handled by the library.
- Knows how to handle all the different IndexTypes, overlapping BoxArrays, etc...
Iterating over MultiFabs

- MultiFabs can be operated on using add, divide, saxpy, etc..
- Also provide MFIter for looping over the FArrayBoxes in a MultiFab.
- Each proc loops only over the data it owns, details are hidden in application code

```cpp
for (MFIter mfi(mf); mfi.isValid(); ++mfi)
{
    const Box& box = mfi.validbox();
    FArrayBox& fab = mf[mfi];
    Real* a = fab.dataPtr();
    const Box& fbox = fab.box();
    f1(box.loVect(), box.hiVect(), a, fbox.loVect(), fbox.hiVect());
}
```
Logical Tiling and MFIter

- Well-known loop transformation technique that improves data locality
- Convert single loop into two nested loops - one over tiles, and one over the data elements within a tile.
- Logic is baked into the iterator

```cpp
// * true * turns on tiling
for (MFIter mfi(mf, true); mfi.isValid(); ++mfi) // Loop over tiles
{
    // tilebox() instead of validbox()
    const Box& box = mfi.tilebox();

    FArrayBox& fab = mf[mfi];
    Real* a = fab.dataPtr();
    const Box& abox = fab.box();

    f1(box.loVect(), box.hiVect(), a, abox.loVect(), abox.hiVect());
}
```
Logical Tiling and MFIter, continued

- Difference between valid and tile boxes for cell-centered Boxes (handles other types too)
- Tiling is purely logical - data layout in memory is unchanged
### Logical Tiling, single core performance

<table>
<thead>
<tr>
<th>Tile Size</th>
<th>GNU compiler</th>
<th>Intel compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time(s)</td>
<td>Speedup</td>
</tr>
<tr>
<td>128 × 4 × 4</td>
<td>8.5</td>
<td>3.4</td>
</tr>
<tr>
<td>128 × 8 × 8</td>
<td>9.0</td>
<td>3.2</td>
</tr>
<tr>
<td>128 × 16 × 16</td>
<td>9.6</td>
<td>3.0</td>
</tr>
<tr>
<td>128 × 32 × 32</td>
<td>23.7</td>
<td>1.2</td>
</tr>
<tr>
<td>128 × 64 × 64</td>
<td>24.4</td>
<td>1.2</td>
</tr>
<tr>
<td>no tiling</td>
<td>28.6</td>
<td>–</td>
</tr>
</tbody>
</table>

1 core of Edison 128^3 domain

Courtesy of Weiqun Zhang, Didem Unat and Tan Nguyen
Logical Tiling and OpenMP

• In addition to improving single-node performance, tiling provides a basis for hierarchical parallelism
• MFIter knows when its constructed in parallel region, details again hidden in application code
• Support for static and dynamic scheduling

```c++
#ifdef _OPENMP
#pragma omp parallel
#endif
for (MFIter mfi(mf,true); mfi.isValid(); ++mfi)
{
    const Box& box = mfi.tilebox();

    FArrayBox& fab = mf[mfi];
    Real* a = fab.dataPtr();
    const Box& abox = fab.box();

    f1(box.loVect(), box.hiVect(), a, abox.loVect(), abox.hiVect());
}
```
Logical Tiling and OpenMP

1 node of Edison (12 cores)  
1 node of Babbage (60 cores)

Courtesy of Weiqun Zhang, Didem Unat and Tan Nguyen
Multi-level Tools

- **Interpolation / Restriction**
  - Filling boundary conditions on fine levels from coarse level data
  - Representing fine solution on the coarse level

- **Flux Registers**
  - Used to store data on coarse / fine interfaces
  - Used to enforce conservation for
  - e.g. hyperbolic systems

- **Tagging / Regridding**
  - Accumulate sets of points
  - Generating BoxArrays that cover those points
Subcycling in time

- Sometimes, you might want to advance the levels with different time steps

- AMReX supports time-stepping approaches with or
Particles in AMReX

• Another core data type is the particle. In AMReX, particles live on and interact with an adaptive hierarchy of meshes.

• Additional challenges:
  – Inherently irregular - amount of data varies
  – Connectivity is hard, e.g. finding neighbors.
  – Always changing, data structures adapt every time step or more

• Several different kinds of applications:
  – Passive tracers
  – Particle-in-cell (electro-magnetic, dark matter, drag)
  – Particle-particle, particle-wall collisions
Flexible data layout

- Quantity, type of particle data varies
- **Array-of-structs versus Struct-of-Arrays**

**Array-of-Structs**

![Diagram of Array-of-Structs]

**Struct-of-Arrays**

![Diagram of Struct-of-Arrays]
The ParticleContainer

- Particles are stored in a ParticleContainer, using stl containers. Particle type itself is handled using templates:

  ```cpp
  typedef ParticleContainer<1, 2, 2, 2> MyParticleContainer;
  ```

- Parallel Communication handled under the hood via Redistribute() routine.
- Flexible enough to support sub-cycling.
- “Ghost particles” for representing fine data on the coarse level.
- “Neighbor particles” for when you need to access particles on other MPI processes
The ParticleContainer

- AMRex provides several useful routines:
  - Advection on a MAC or cell-centered velocity grid
  - Single and multi-level PIC interpolation, deposition
  - Cell linked lists, neighbor list

- Particle struct is POD and compatible with Fortran via iso_c_binding:

  ```fortran
  use amrex_fort_module, only: amrex_particle_real
  use iso_c_binding, only: c_int

  type, bind(C) :: particle_t
    real(amrex_particle_real) :: pos(3)
    real(amrex_particle_real) :: mass
    integer(c_int) :: id
    integer(c_int) :: cpu
    integer(c_int) :: phase
    integer(c_int) :: state
  end type particle_t
  ```
The ParIter

- Particle data can be iterated over much like the mesh data
- Aware of tiling (not logical any more), OpenMP.

```cpp
const int lev = 0;
#ifdef _OPENMP
#pragma omp parallel
#endif
for (ParIter pti(*this, lev); pti.isValid(); ++pti)
{
    const auto np = pti.numParticles();

    auto& array_of_structs = pti.GetArrayOfStructs();

    auto& struct_of_arrays = pti.GetStructOfArrays();
    auto& ux = struct_of_arrays[PIdx::ux ];
    auto& uy = struct_of_arrays[PIdx::uy ];
    auto& uz = struct_of_arrays[PIdx::uz ];
    auto& ginv = struct_of_arrays[PIdx::ginv];

    set_gamma(np, ux.data(), uy.data(), uz.data(), ginv.data());

    push_position_boris(np, particles.data(),
                        ux.data(), uy.data(), uz.data(), ginv.data(), dt);
}
```
OpenMP scaling of particles on Cori KNL
Load Balancing with particles

- Particle data introduces an additional challenge to load balancing.
- If you have a significant amount of particle work, number of cells is not a good estimate any more.
- Can use work estimates based on number of cells plus number of particles, but doesn’t work for all applications (e.g. Monte Carlo).
- In some applications we use real time measurements to estimate the work distribution
- Dynamic scheduling of OpenMP threads can also help
Load Balancing with particles

Simulation run with WarpX

- 128x8
- 64x16
- 128x8 + DLB (knapsack)
- 64x16 + DLB (knapsack)
- 128x8 + DLB (SFC)
- 64x16 + DLB (SFC)

Time for 100 iterations vs. Number of iterations

Figure courtesy of Remi Lehe
Weak scaling with particles

b)

![Graph showing weak scaling run time for different configurations before and after optimization. The graph plots time (s) against the number of nodes (n_node).](image)

- Before optimization:
  - 32 ppc, 64^3 cells/node
  - 0 ppc, 256^3 cells/node

- After optimization:
  - 32 ppc, 64^3 cells/node
  - 0 ppc, 256^3 cells/node

---

Courtesy of Maxence Thévenet
Fluid riser (drag forces, particle-particle work)
Embedded Boundaries

- Use a cut cell approach to complex geometries.
- Still block-structured, but cells labelled covered, cut, or regular.
- Within an MFIter loop, ask whether the tile contains any cut cells.
- If not, treat in normally.
- If it does, pass in extra geometric, connectivity information.
- All data structures fully inter-operable with Fortran.
- Connectivity info for all 27 potential neighbors is stored in a single integer.
- Doesn’t sacrifice essential regularity far from domain boundaries.
- Much more work to do near boundaries, benefits from dynamic OpenMP scheduling.

Plasma Wakefield Accelerator
Nozzle
Embedded Boundaries with particles

• Level set approach used to compute whether particles collide with walls

Courtesy of Johannes Blaschke
Linear solvers

- AMReX provides native geometric Multigrid solvers for parabolic and elliptic systems.
- Cell-centered and node-centered data.
- Single level and multi-level AMR
- Box agglomeration to avoid coarsening limitations
- Current work - extension to EB, which makes the bottom solve much more complex
Visualization and IO

• In-house data format with efficient parallel I/O for both restart and plotfiles (has been much faster than HDF5 ... although that is changing)
• Visualization format supported by Visit, Paraview, yt

Image courtesy of Maxence Thévenet
Interfacing with other Libraries

- SUNDIALS ODE solvers
- Hypre, HPGMG solvers
- FFTW and other FFT libraries
- In-situ and in-transit analytics - Sensei, ALPINE, Henson
1. Overview of AMReX
2. Introduction to block-structured AMR
3. What tools does AMReX provide?
4. What can you build these tools?
5. Current development directions
6. Where to find more
Example Applications

Combustion

Astrophysics

Multiphase Flow

Particle Accelerators

Cosmology

FLASH
<table>
<thead>
<tr>
<th>Application Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Particles</strong></td>
</tr>
<tr>
<td>Combustion</td>
</tr>
<tr>
<td>Multiphase</td>
</tr>
<tr>
<td>Cosmology</td>
</tr>
<tr>
<td>Astrophysics</td>
</tr>
<tr>
<td>Accelerators</td>
</tr>
</tbody>
</table>

*Note: The letter 'X' indicates a requirement.*
Outline

1. Overview of AMReX
2. Introduction to block-structured AMR
3. What tools does AMReX provide?
4. What can you build these tools?
5. Current development directions
6. Where to find more
New architectures and programming models

• Much current work focuses on porting AMReX to GPUs
• Cuda’s Unified Memory for data motion
• Kernels offloaded through a variety of strategies
  – CUDA C/Fortran
  – OpenACC
  – OpenMP

• NVIDIA’s thrust library for sorting and searching (particles)
• Mini-App versions of Castro hydro (StarLord) and WarpX (Electromagnetic PIC) exist
• Approaches to parallelism other than MPI+OpenMP:
Outline

1. Overview of AMReX
2. Introduction to block-structured AMR
3. What tools does AMReX provide?
4. What can you build these tools?
5. Current development directions
6. Where to find more
Open source development model

- AMReX and many application codes available on Github:
  https://github.com/AMReX-Codes/amrex

- All branches public. Bleeding edge development branch, merged into master monthly.
- Sphinx, doxygen documentation hosted on Github pages, auto-generated with Travis
- Tutorials live in main code repository
- Issues, pull requests welcome (bug fixes, new features, documentation, etc...)
# Nightly regression testing

## Nyx regression tests

<table>
<thead>
<tr>
<th>Date</th>
<th>AMR-density</th>
<th>DR_restart</th>
<th>DoubleRarefaction</th>
<th>LyA</th>
<th>MiniSB</th>
<th>MiniSB-ref</th>
<th>Part-2line</th>
<th>Part-2line_restart</th>
<th>Part-mass_nosub</th>
<th>Part-mass_sub</th>
<th>SantaBarbara</th>
<th>Sedov</th>
<th>Sod</th>
<th>StrongShockTube</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-06-29</td>
<td>:</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
</tr>
<tr>
<td>2018-06-28-001</td>
<td>:</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
</tr>
<tr>
<td>2018-06-28</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
</tr>
<tr>
<td>2018-06-27</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
<td>:)</td>
</tr>
</tbody>
</table>
Questions?