# Exascale-ready tools for implementing particle methods with AMReX

2023 NETL Workshop on Multiphase Flow Science

Andrew Myers (PI: John Bell)

August 1, 2023

## Acknowledgements

LBNL, CS Resarch:
- Weiqun Zhang
- Ann Almgren
- Jean Sexton
- Michele Rosso
- Vince Beckner
- Donald Wilcox
- Hengjie Wang
- Erik Palmer
- Houjun Tang

LBNL, ATAP:
- Maxence Thevenet
- Axel Heubl
- Remi Lehe
- Jean-Luc Vay

LBNL, NERSC:
- Kevin Gott
- Chris Daley

NVIDIA:
- Max Katz

NETL:
- Jordan Musser
- Roberto Porcu

LBNL, C3:
- Zarija Lukic

SBU:
- Chris Degrendele
- Alice Harpole
- Mike Zingale

Waterloo:
- Erik Schnetter

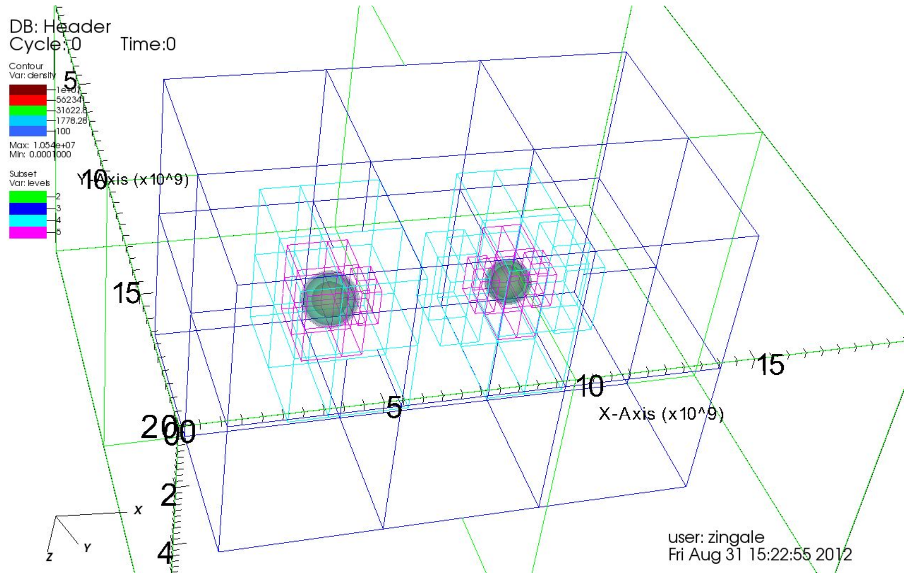
ORNL:
- Stuart Slattery

HPE:
- Steve Abbott

Modern Electron:
- Roelof Greenwald
- Phil Miller

## Overview of AMReX

Framework for building parallel, block-structured adaptive mesh refinement (AMR) applications.



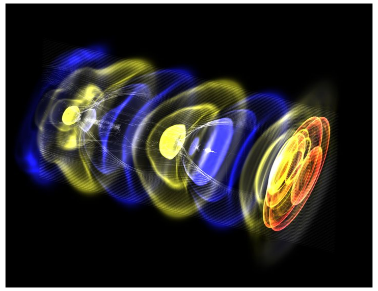
- Developed as part of ECP.
- Handles parallel communication, domain decomposition, load balancing.
- Support for cell, node, face, and edge-centered mesh data + particles
- Multilevel operations: flux registers, tagging, regridding
- Support for embedded boundaries via cut cell approach
- Geometric multigrid solvers for elliptic and parabolic systems

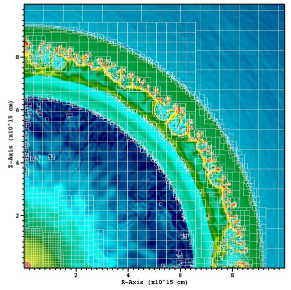
Github: https://github.com/AMReX-Codes/amrex
Docs: https://amrex-codes.github.io/amrex/

# AMReX Applications

**Astrophysics:**

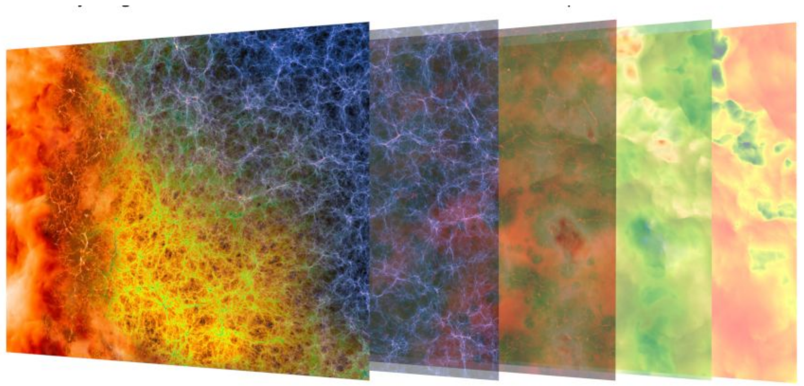
Castro (compressible)
MaestroEx (low-Mach)
SedonaEx (Monte Carlo radiation transport)
Emu (neutrino transport)
Quokka (radiation-hydrodynamics)





**Cosmology:**

Nyx



**Combustion:**

PeleC **(Compressible)**
PeleLM (Low Mach)

**Accelerator Modelling:**

WarpX
ImpactX
Hipace++

**Magnetically-confined fusion:**

GEMPIC

**Epidemiology:**

ExaEpi

**Incompressible Navier Stokes:**

IAMR
incflo

**Atmospheric science:**

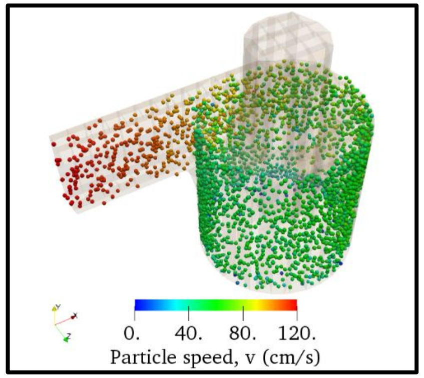
AMR-wind
ERF

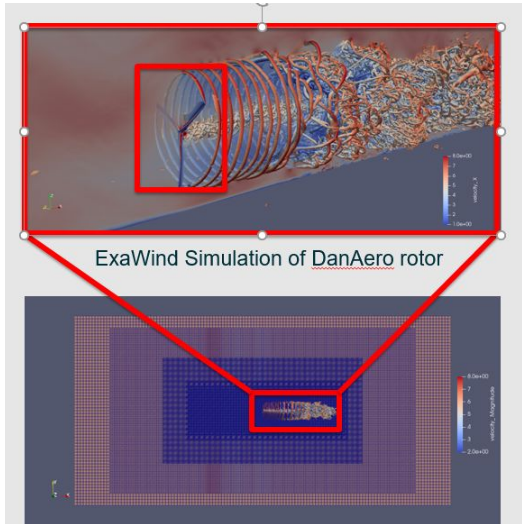
**Multi-phase Flow:**

MFIX-Exa

**Ocean modeling:**

ROMS-X



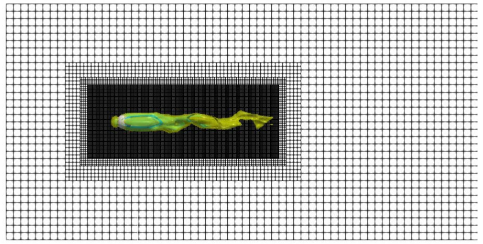


**Multiscale Modelling and Stochastic Systems:**

FHDeX

**Electromagnetics:**

ARTEMIS



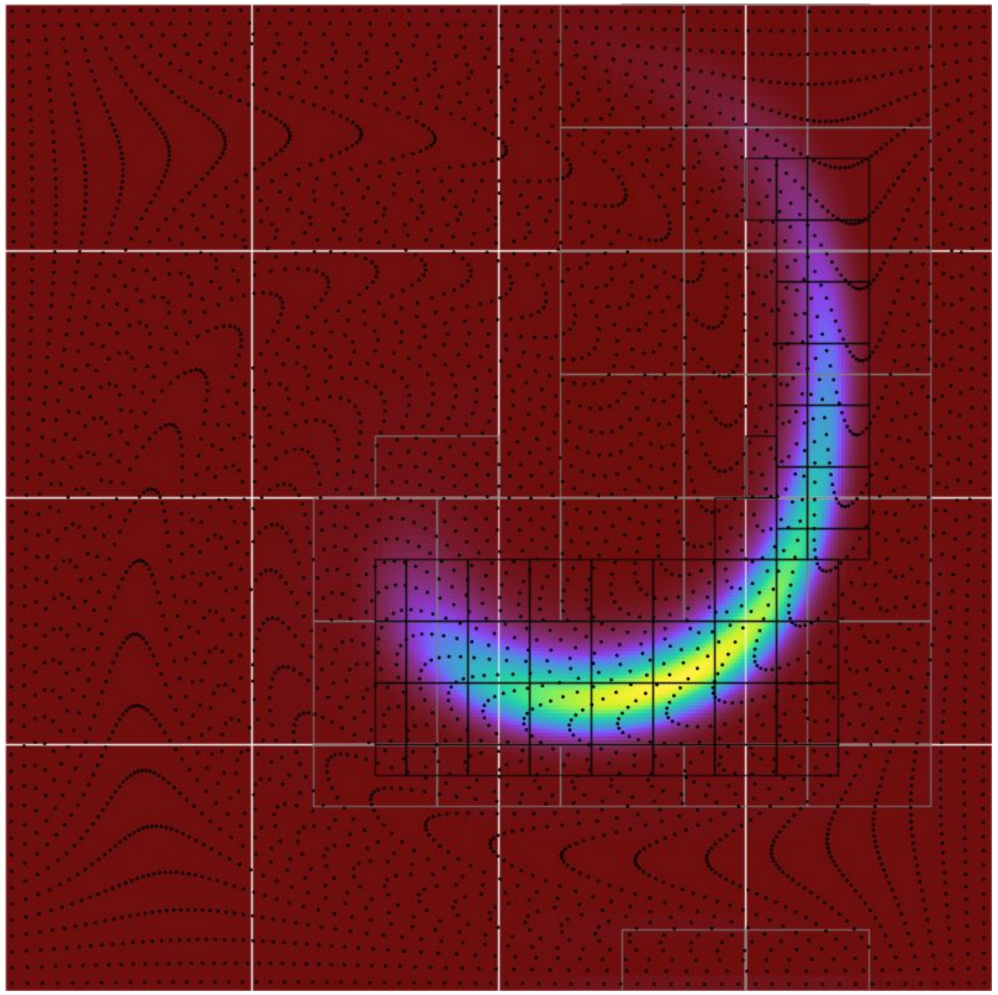
**Biological cell modelling:**

BoltzmanMFX
CCM

**Solid Mechanics:**

Alamo

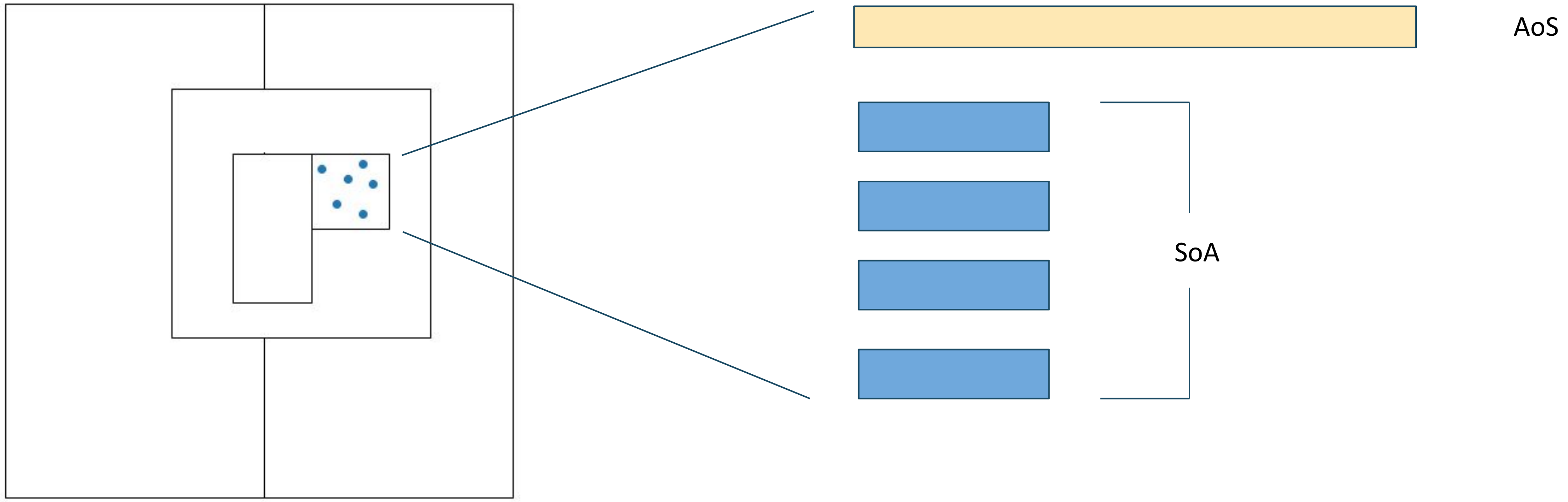
# Overview of Particles in AMReX



- AMReX provides tools for storing and iterating over **distributed particle data** that 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**.
- Tools for 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**.

## Flexible data layout

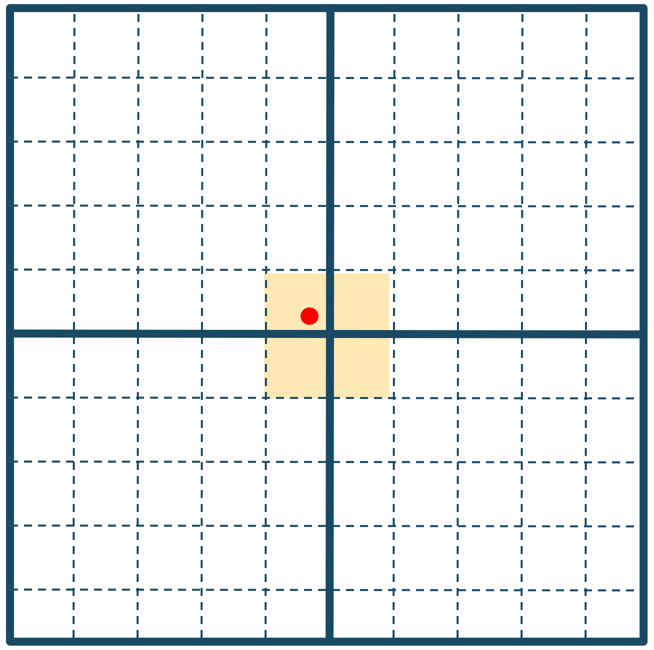
- Particles on each rank are split up by levels and grid (with optional tiling)
- On a single grid, data is stored as a user-defined mix of Array-of-Structs (AoS) and Struct-of-Arrays (SoA)
- No longer any limitation of particle struct being present!



```
using AMRParticleData = std::vector<std::map<int, ParticleTile>>;
```

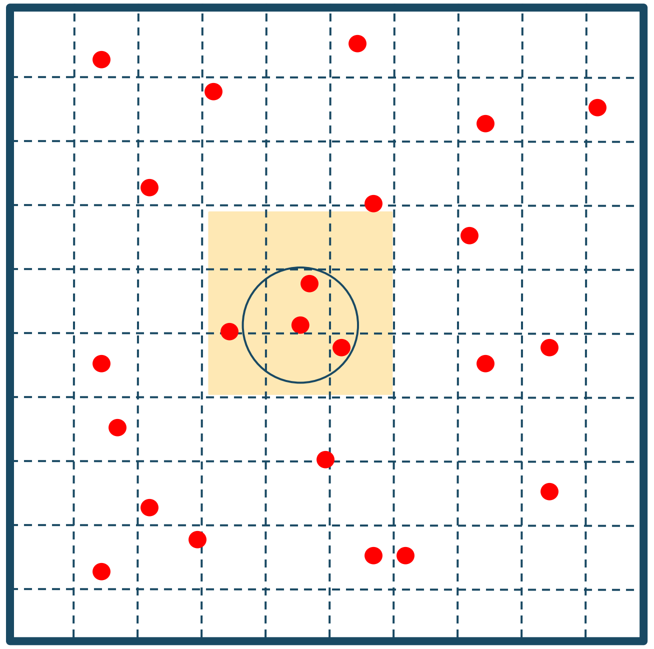
## Particle-mesh interaction

- AMReX provides a set of tools for handling particle-mesh interaction
- Simple interpolation kernels (NGP, CIC) for mesh-to-particle and particle-to-mesh interpolation are provided
- Lambda function interface for performing user-provided operations interpolations (e.g. high-order particle shapes, Gaussian kernels)
- Performs needed parallel communication “under-the-hood” (SumBoundary)
- Support for dual-grid
- Uses data duplication strategy for CPU threading, per-particle atomics for GPU threading.
- Shared memory approach for deposition can be a win for some kernels



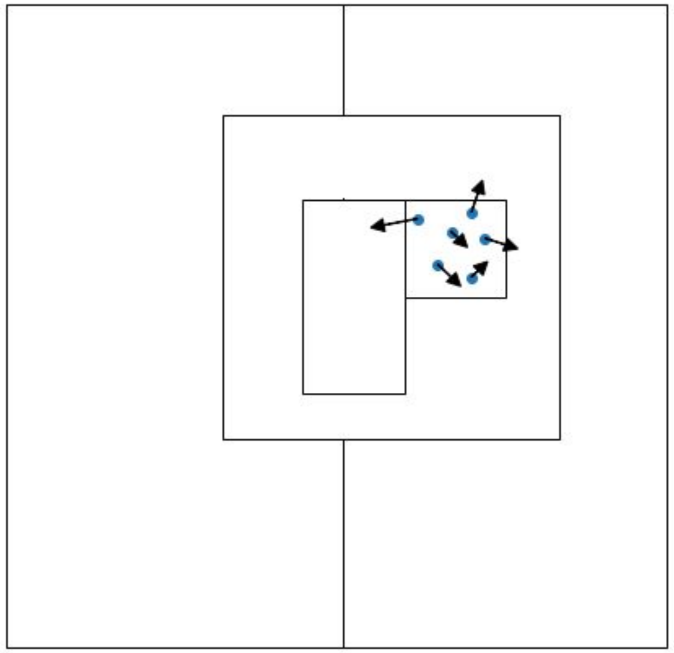
# Particle-particle interations

- AMReX includes tools for building and iterating over **neighbor lists** for DEM-style particle-particle interactions
- Pre-compute a list of potential interaction partners that can be reused for multiple time steps
- Uses a cell list to avoid direct N^2 search of all possible pairs
- Users can specify operation using lambda function.
- Support for half and full neighbor lists (tradeoff based on amount of contention / atomics performance)
- Roberto Porcu found the half-list usually performs best in MFiX-Exa on V100



## Communication: Redistribute

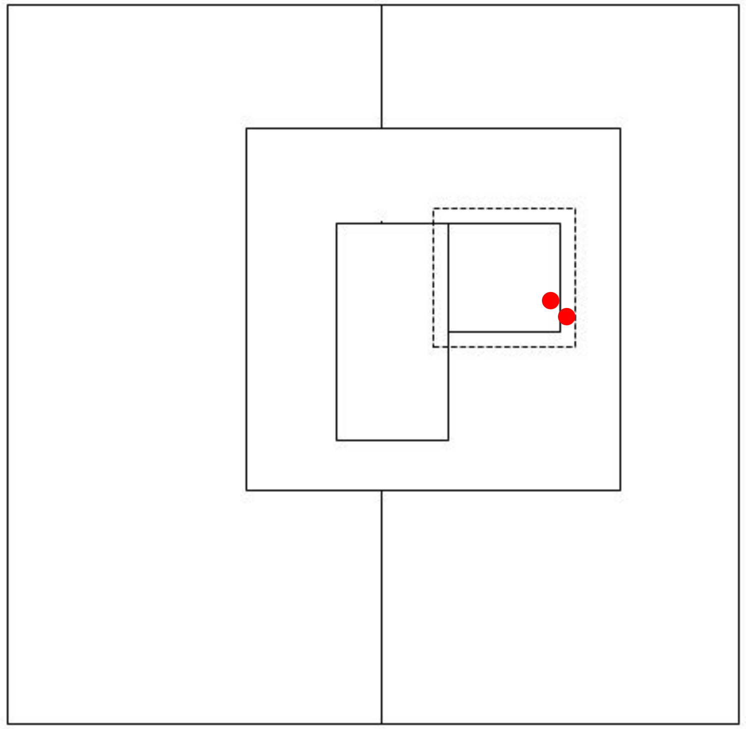
Moving particles back to the right level / grid / MPI rank once they move



- Harder problem, in some ways, than communicating mesh data, even with AMR. Data versus metadata.
- For GPU execution pathways, all kernels run on device. Can take advantage of GPU-aware MPI implementations, if available.
- Comes in both “local” (most codes, most of the time) and “global” (regridding, load balancing) flavors.
- Support for sub-cycling in time

# Communication: Halo exchange

The other main particle communication pattern
Often needed for particle-particle work

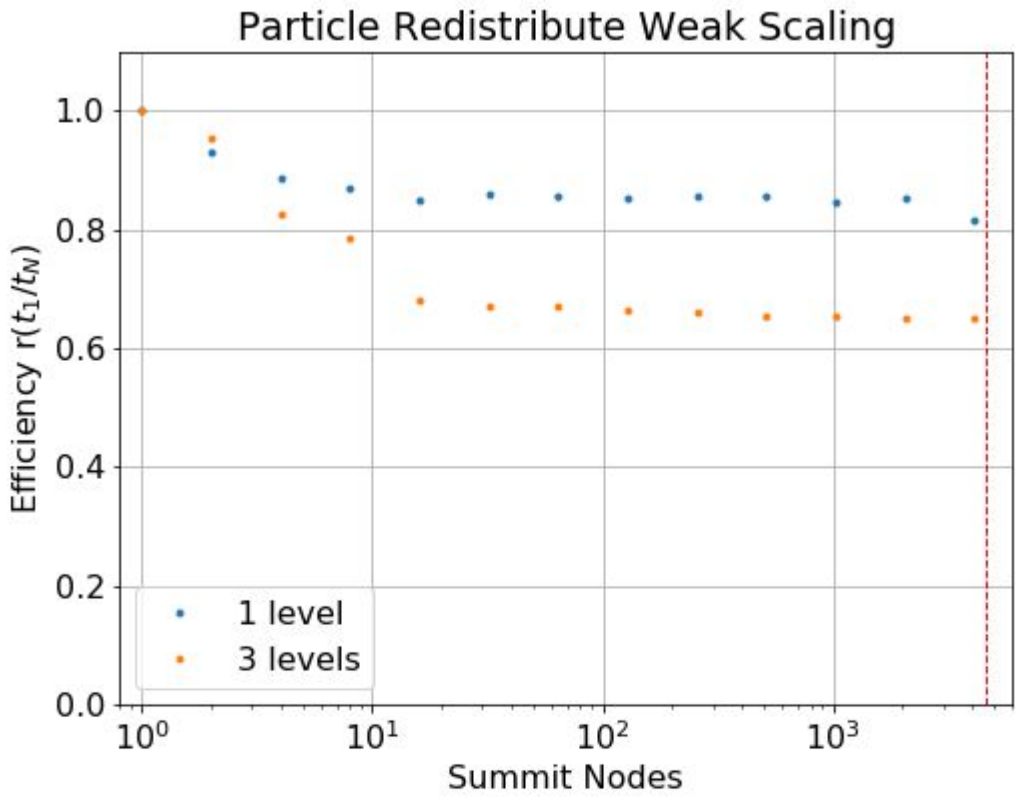


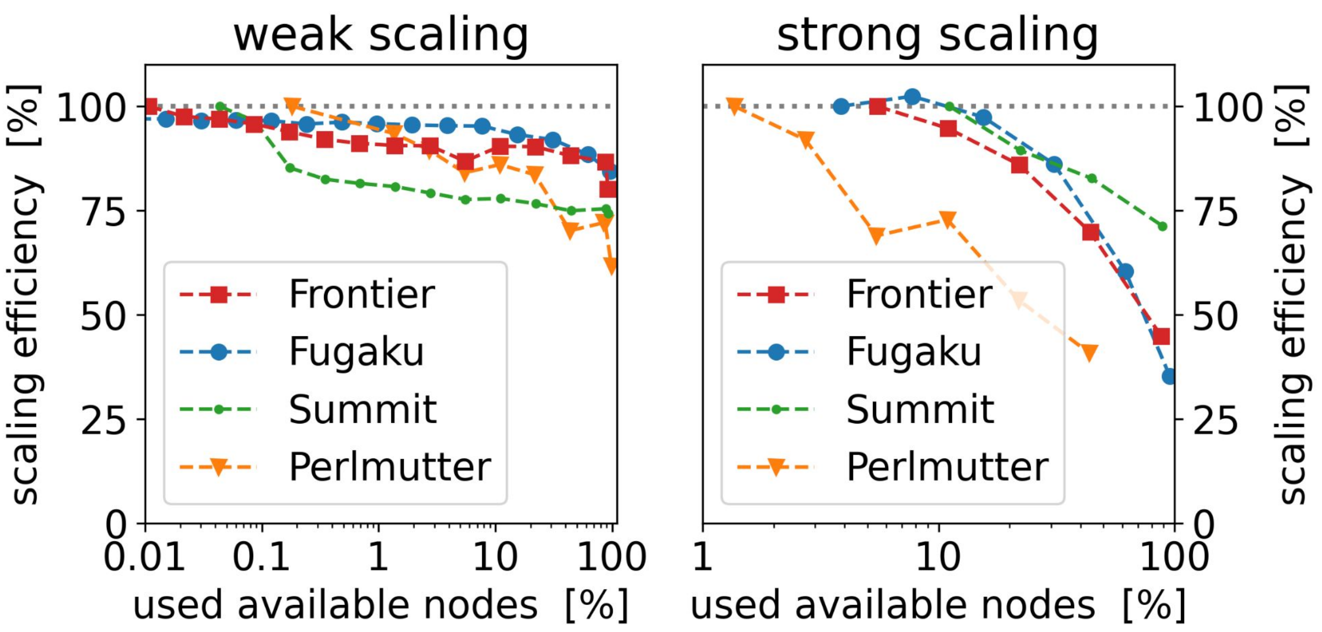
- Function for figuring out which levels and grids particles need to be ghosted to can also be written in terms of Box intersection

- Also runs fully on device

- Hengjie Wang recently implemented “culling” optimization

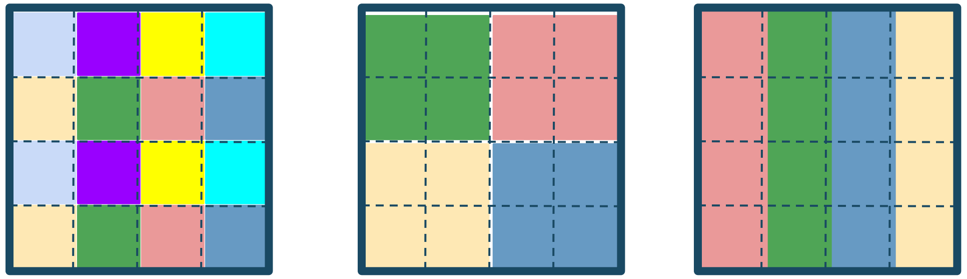
## Particle Communication Scaling





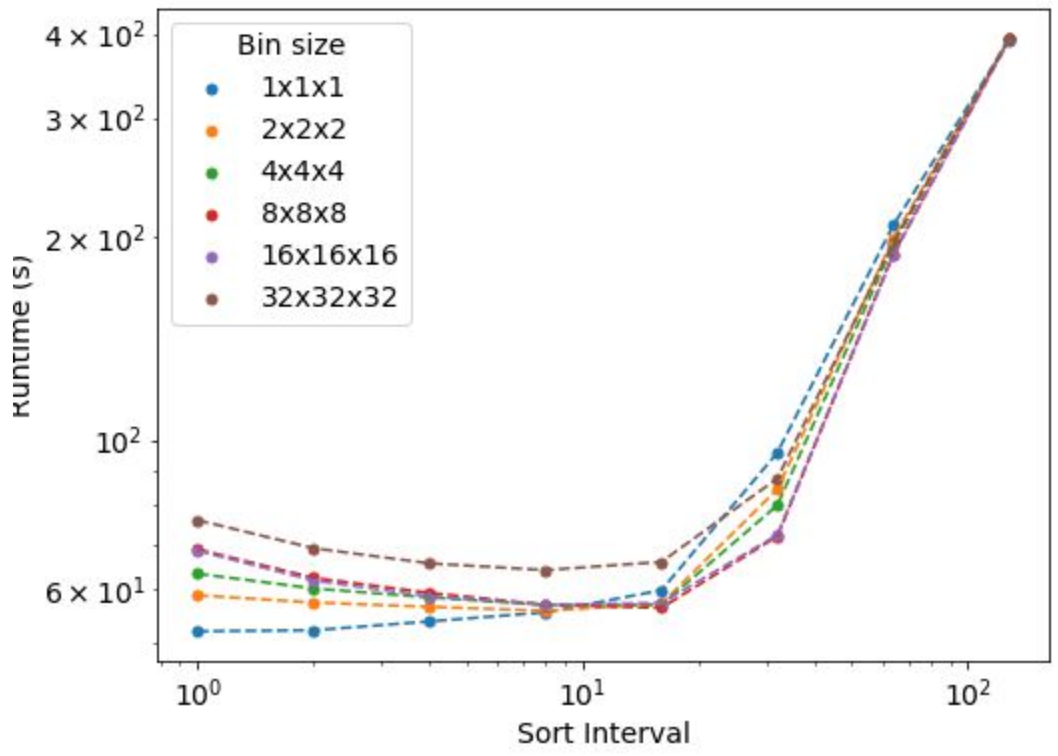
## Binning / Sorting / Partitioning particles

- AMReX also provides tools to sort, bin, and partition the particles on a box.
  - Binning = compute a permutation array that puts particles in some order
  - Sorting = actually reorder the particle data in memory
  - Partition = weaker than sort, just split into two groups.
- All implemented in terms of counting sort / prefix scan.
- Used internally (e.g. Redistribute, Neighbor Lists) or to implement application specific physics modules (binary collisions) or algorithmic options (gather / deposit buffers for mesh refinement)
- Can do cells, supercells, slices, or custom user functions of particle data

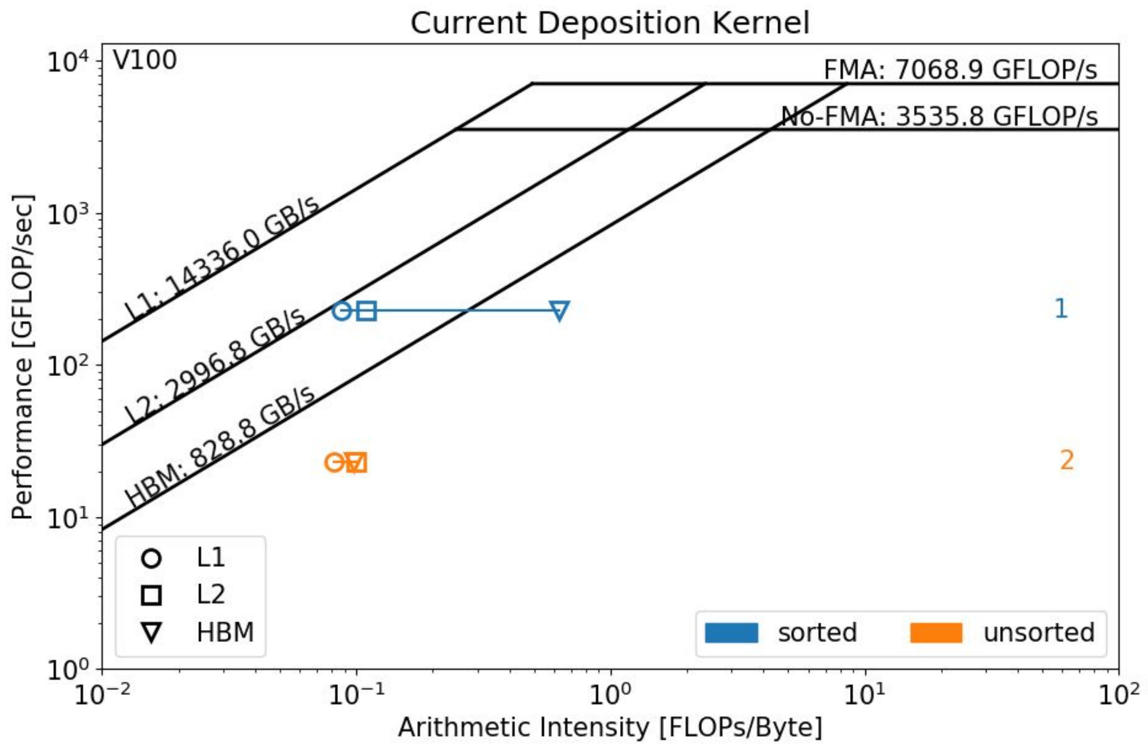


## Sorting example: particle-mesh performance

8x speed up between not sorting and optimal sorting

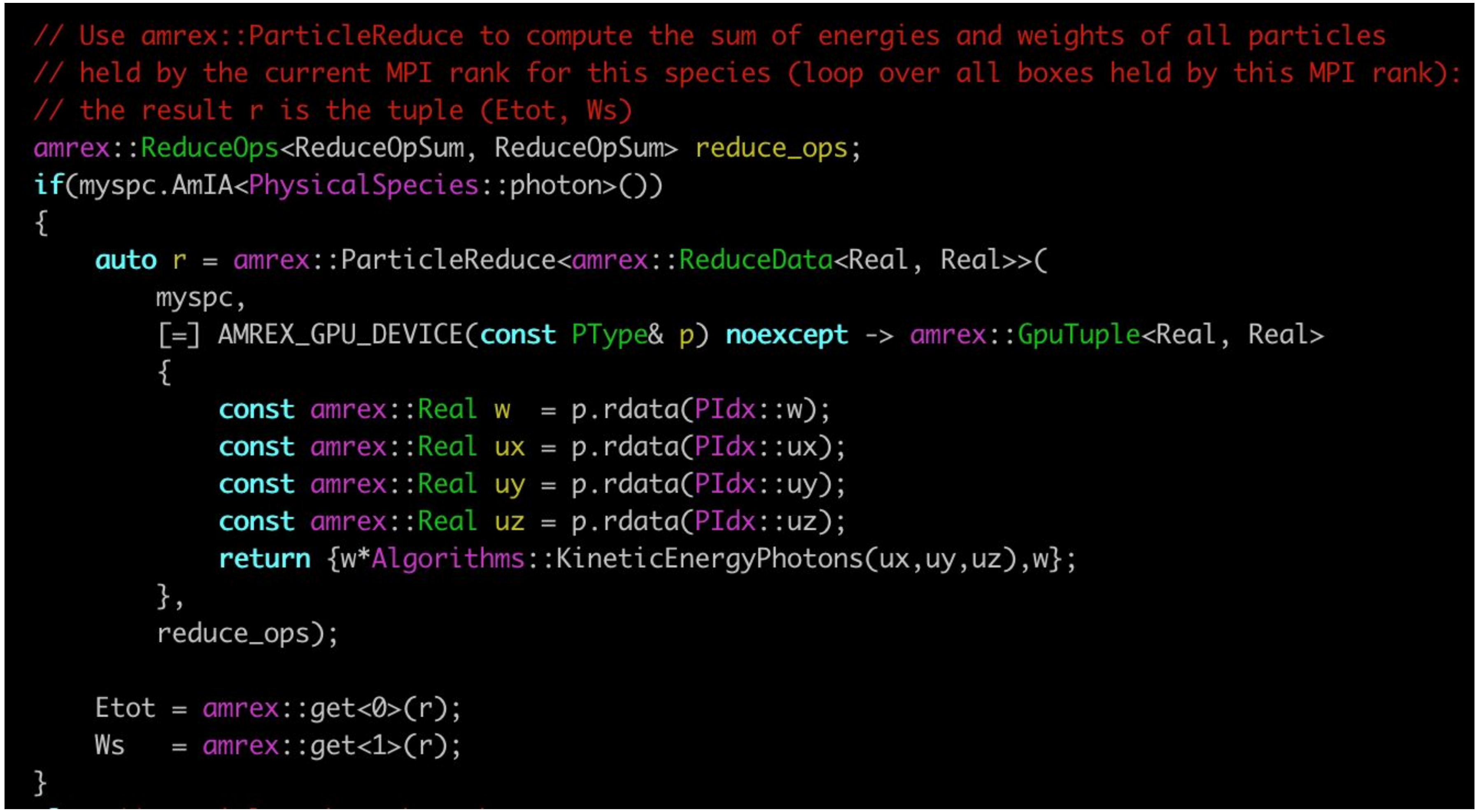


Roofline analysis confirms better L2 cache reuse in sorted version



# Reductions over particle data

- Often want to compute sum, max, min, etc… of some quantity over all particles
- Grid by grid, level by level, or over the entire container
- Implemented in terms of CUB / ROCprim
- User - provided lambda functions / functors
- Can fuse multiple reductions into one kernel using “tuple” reduce syntax



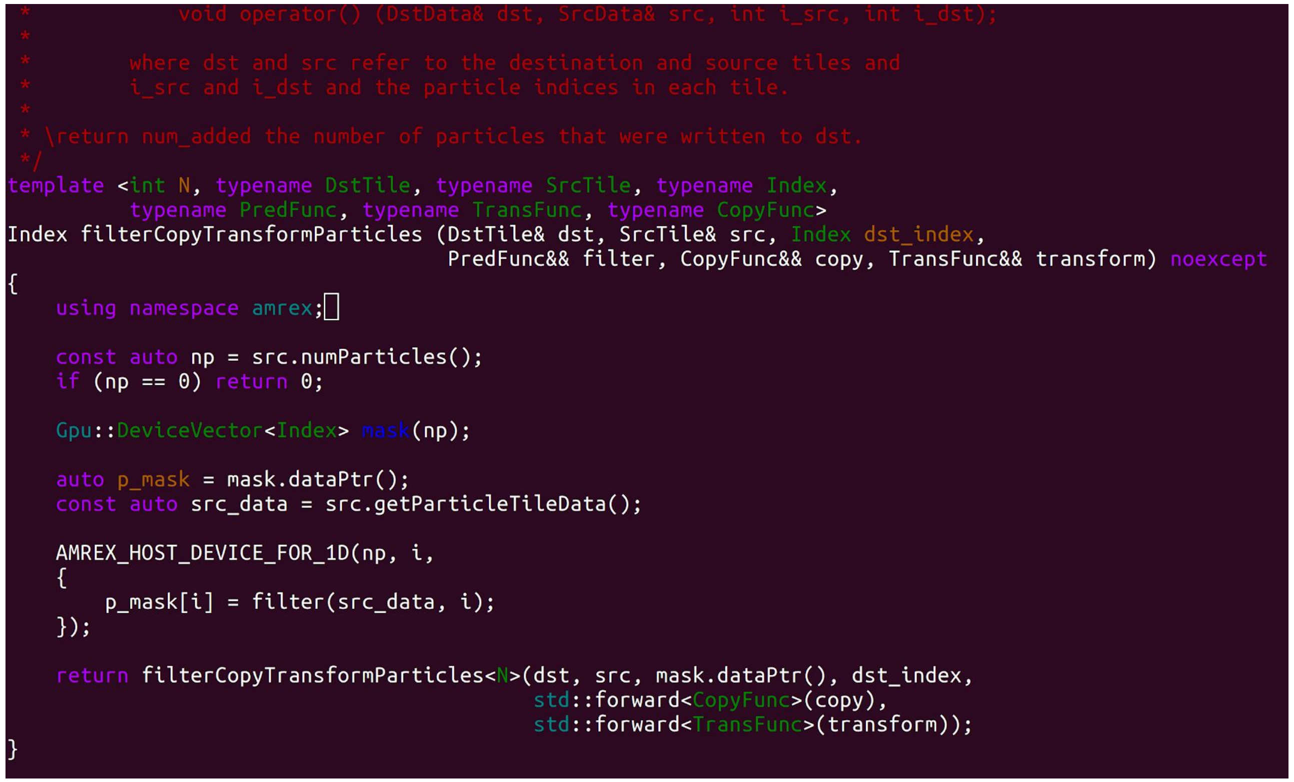
```
// Use amrex::ParticleReduce to compute the sum of energies and weights of all particles
// held by the current MPI rank for this species (loop over all boxes held by this MPI rank):
// the result r is the tuple (Etot, Ws)
amrex::ReduceOps<ReduceOpSum, ReduceOpSum> reduce_ops;
if(myspc.AmIA<PhysicalSpecies::photon>())
{
    auto r = amrex::ParticleReduce<amrex::ReduceData<Real, Real>>(
        myspc,
        [=] AMREX_GPU_DEVICE(const PType& p) noexcept -> amrex::GpuTuple<Real, Real>
        {
            const amrex::Real w  = p.rdata(PIdx::w);
            const amrex::Real ux = p.rdata(PIdx::ux);
            const amrex::Real uy = p.rdata(PIdx::uy);
            const amrex::Real uz = p.rdata(PIdx::uz);
            return {w*Algorithms::KineticEnergyPhotons(ux,uy,uz),w};
        },
        reduce_ops);

    Etot = amrex::get<0>(r);
    Ws   = amrex::get<1>(r);
}
```

# Filter / transform operations

- AMReX provides a variety of “stream compaction” functions for filtering / copying / transforming particle data.

- Processes where one type of particle conditionally creates one or more particles, possibly of different type:

  - Ionization: $i^{+}_{lev} \rightarrow i^{+}_{lev+1} + e^{-}$
  - Particle Splitting: $p \rightarrow N_{p}$ (4, 6, 8, etc…)
  - Pair Production: $h\nu \rightarrow e^{+} + e^{-}$ (opposite j)
  - Quantum Synchrotron Radiation: $e^{-} \rightarrow e^{-} + h\nu$
  - Can be triggered by mesh data or by interaction with EB walls



```
 *             void operator() (DstData& dst, SrcData& src, int i_src, int i_dst);
 *
 *         where dst and src refer to the destination and source tiles and
 *         i_src and i_dst and the particle indices in each tile.
 *
 * \return num_added the number of particles that were written to dst.
 */
template <int N, typename DstTile, typename SrcTile, typename Index,
          typename PredFunc, typename TransFunc, typename CopyFunc>
Index filterCopyTransformParticles (DstTile& dst, SrcTile& src, Index dst_index,
                                    PredFunc&& filter, CopyFunc&& copy, TransFunc&& transform) noexcept
{
    using namespace amrex;

    const auto np = src.numParticles();
    if (np == 0) return 0;

    Gpu::DeviceVector<Index> mask(np);

    auto p_mask = mask.dataPtr();
    const auto src_data = src.getParticleTileData();

    AMREX_HOST_DEVICE_FOR_1D(np, i,
    {
        p_mask[i] = filter(src_data, i);
    });

    return filterCopyTransformParticles<N>(dst, src, mask.dataPtr(), dst_index,
                                           std::forward<CopyFunc>(copy),
                                           std::forward<TransFunc>(transform));
}
```

AMReX

Thank you!

U.S. DEPARTMENT OF ENERGY | Office of Science

ECP EXASCALE COMPUTING PROJECT