UPC AMR Status Report

Michael Welcome

LBL - FTG

Outline

- Why Chombo AMR as a target application?
- Overview of the Chombo Framework
- Current status of UPC-AMR
- Preliminary performance data
- Ghost zone exchange analysis

Why AMR?

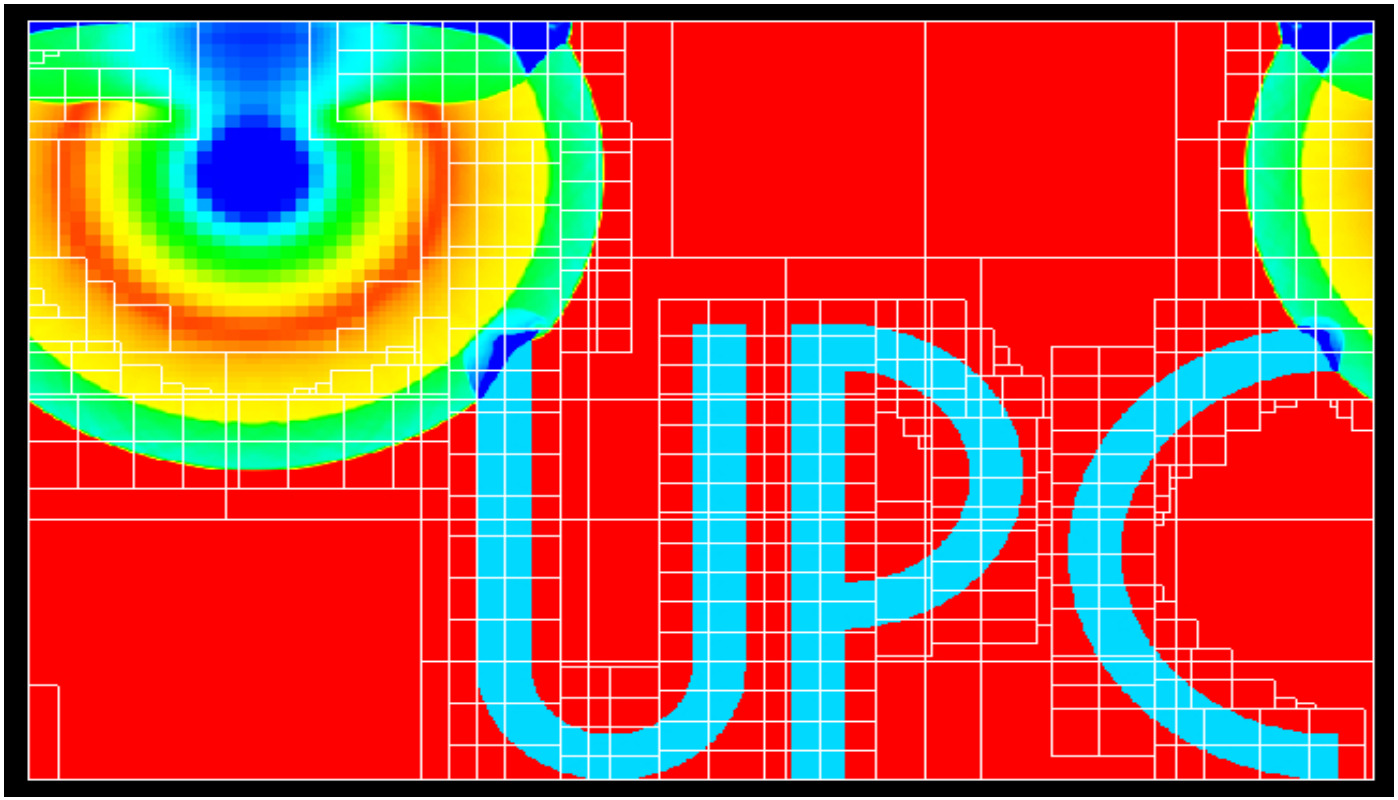
- Methodology of interest to DOE
  - Chombo, AmrLib, GrACE, Flash
  - Example of modern “efficient” algorithm
  - Compute intelligently: brains over brawn
- Large, complex application will stress compiler and runtime system
  - Mix of regular and irregular data structures
- But mostly… I know the algorithm, I worked on it for 10 years.
- Why Chombo AMR? – APDEC SciDac + local expertise

What is AMR?

- AMR frameworks
  - provide the infrastructure to dynamically refine the computational domain in the numerical solution of Partial Differential Equations.
- AMR acts like a numerical microscope
  - Compute highly refined solution only where needed
  - Coarse grid solution elsewhere
- Ideal for problems of multiple scales
  - Combustion, turbulence, interface between particle methods and continuum mechanics

Example 2D Problem: Block Structured AMR

- Chombo AMR: Mach 4 blast wave hitting UPC logo
  - Periodic in X-direction, reflecting wall in Y-direction
  - Single level 0 grid patch
  - Large boxes show location of Level 1 grid patches (2x refinement)
  - Small boxes show location of Level 2 grid patches (4x refinement)



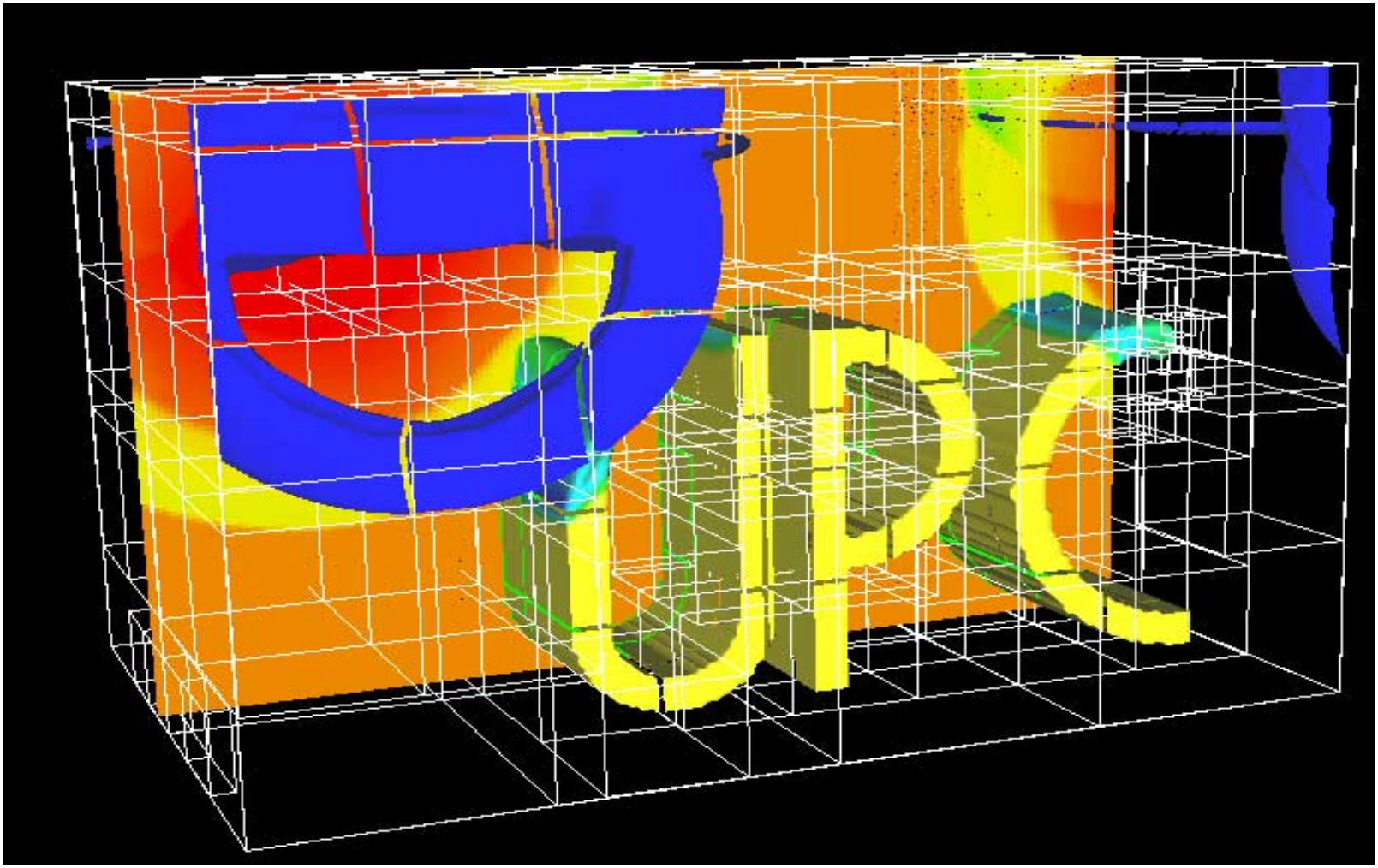
Chombo Overview

- Intent: make it easy for scientist to port grid-based code to parallel adaptive code
  - Basic unit of work is a grid patch (Nd fortran array)
  - Framework hides parallel implementation and gritty details of adaptive structure.
- C++ class library with calls to FORTRAN
  - Library: 62000 lines C++, 1100 lines FORTRAN
  - Examples: 70000 lines C++, 13000 lines FORTRAN
- Supports Hyperbolic, Parabolic & Elliptic solvers
  - Low Mach number combustion, MHF, Accelerator Physics, String theory

Chombo Overview (2)

- All processes have copy of all AMR metadata
- Data organized by refinement level: union of disjoint grid patches
- Each grid patch exists on (belongs to) one process
  - Load balance by distribution of grid patches
- Communication is done in MPI
- Parallel implementation buried in low-level Chombo library
  - Not (necessarily) visible at “user” level
- Communication via “Exchange” operation (copy-on-intersect) over union of grids at same refinement level.
  - Sender copies appropriate grid data to buffers, issues isend
  - Receiver issues irecv into buffer, then copies to target grid
- Communication between levels via restriction and prolongation operations plus Exchange.
- Some reductions/barriers sprinkled through code

Chombo example in 3D



UPC-AMR Overview

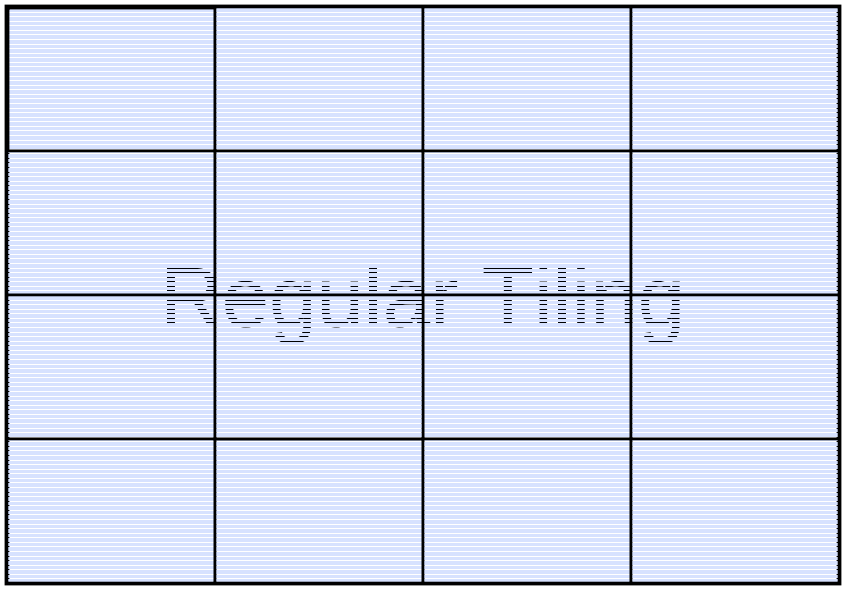
- Port subset of Chombo AMR to UPC
  - Basic AMR data structures (C++ to C/UPC)
    - IntVect, Box, ProbDomain, Godunov integrator, …
  - Calls to Chombo FORTRAN kernels
    - Identical numerics: Allows direct comparison
- Many Chombo C++ classes use templates, inheritance and the STL
  - Did not port directly
  - Re-wrote simplified versions in C
- Communication
  - Metadata in shared address space
    - All threads cache locally for fast access
  - Grid data in shared address space
    - UPC_memget for ghost data (more on this later)
  - Simple “min” reduction for timestep update

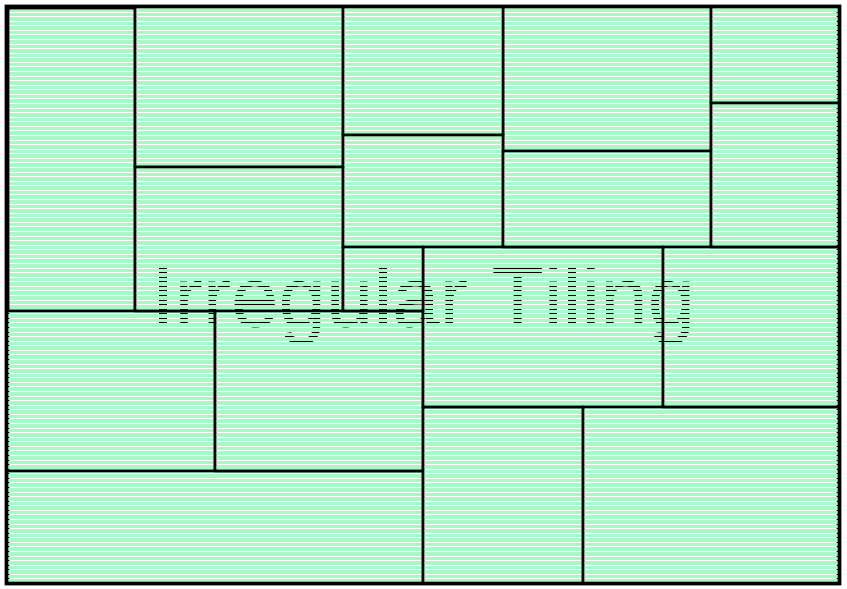
UPC-AMR: Current Status

- Supports a single level of grids tiling a rectangular domain
  - Currently no adaptivity, no refinement
  - Data structures, operations to support these do exist.
- Gas dynamics only (hyperbolic PDE)
  - No parabolic or elliptic solvers
- 12,000 lines of C/UPC, 1700 lines of FORTRAN
- Difficult test-case for our portable translator
  - Just got it working recently (last week)
- Ghost zone exchange
  - Upc_memget working
  - Non-blocking strided memget is coded, not tested

Domain tiling in UPC-AMR

- Problems run with simple, regular grid tiling
- Will work with any tiling
  - Needed for general adaptive framework
  - Implemented as list/array of grid patches
  - NOT implemented as block-cyclic array

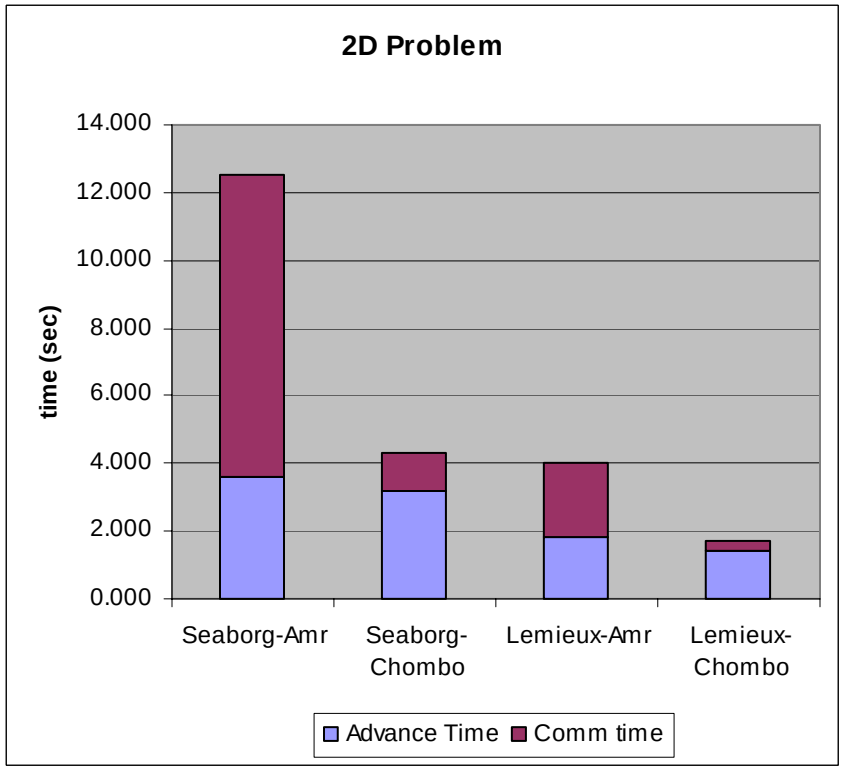


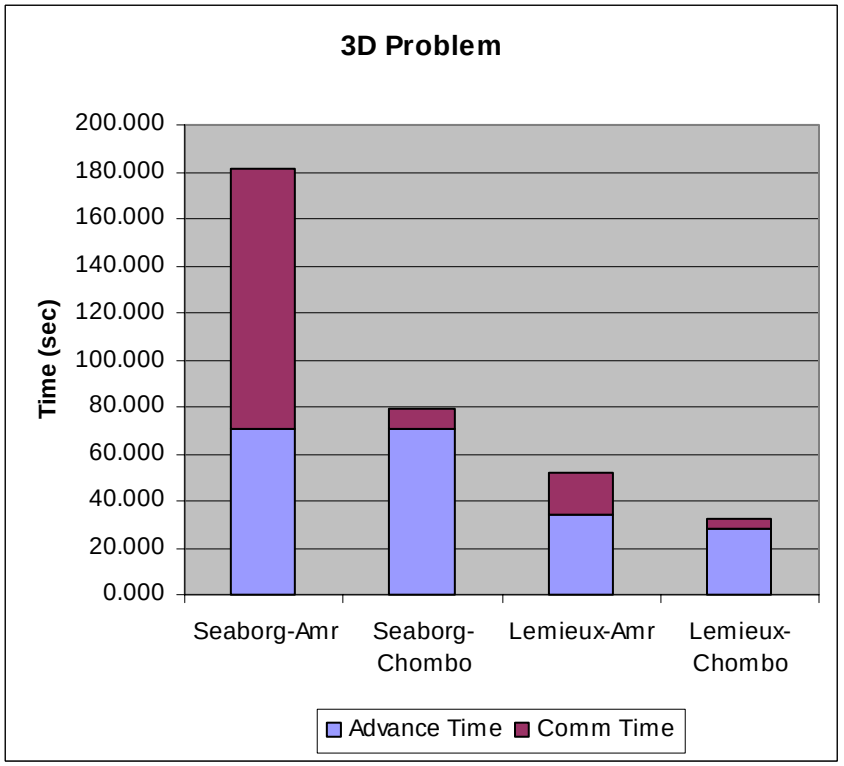


Simplified 1-level Algorithm

```
dt = compute_initial_timestep();
sim_time = 0;
While (sim_time < stop_time) {
   UPC_FORALL(mygrids) {
     Fill_ghost_region_with_EXCHANGE(G);
   }
   new_dt = infinity
   UPC_Forall(mygrids) {
     g_dt = advance_grid(G,dt);
     new_dt = MIN(new_dt,g_dt);
   }
   sim_time += dt;
   dt = UPC_REDUCE_MIN(new_dt);
}
```

Preliminary Performance Data Ouch!





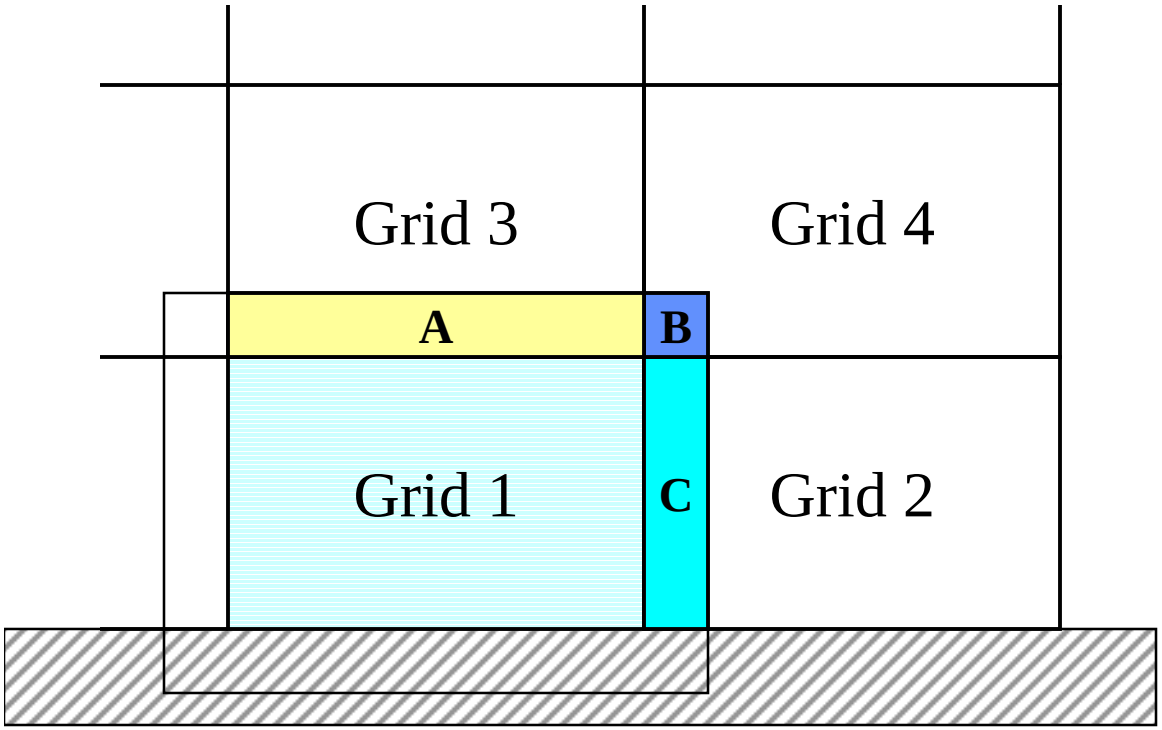
- 2D Problem:
  - 400x200 mesh, 32 grids, 100 steps,16 threads
- 3D Problem:
  - 192x128x128 mesh, 96 grids, 20 steps, 32 threads

Notes on Problem Runs

- Seaborg:
  - Ran with 8 processes/threads per node
  - AMR uses LAPI conduit
    - Each LAPI thread spawns 2 worker threads
    - 24 threads per node, 16 CPUs per node
  - Context switches in 3D problem:
    - AMR-UPC: > 50,000 context switches per LAPI task
    - Chombo: < 2,700 context switches per MPI task

Ghost Region Exchange

- Filling boundary or “ghost” zones for Grid 1 (4 zone wide)
  - A: from grid 3, B: from Grid 4, C: from Grid2, etc

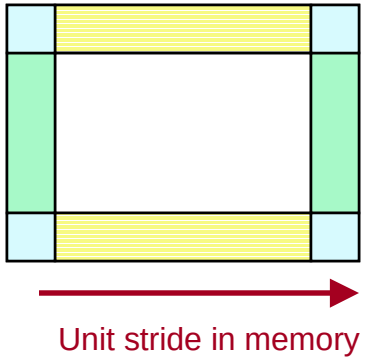


Options:

1. **Pointwise assignment through shared pointer**
2. **UPC_memget of contiguous segments**
3. **Use of proposed blocking or non-blocking strided memget operations**
4. **Pack to contiguous buffer, get, unpack**

Ghost region analysis
Regular tiling, 4 ghost zones

- Ghost regions in 2D: (32x32 grid, 4 components)
  - 8 regions: 4 faces, 4 corners
  - 4 4x4 corners (64 cells, 16 pencils len=4)
  - 2 4x32 faces (256 cells, 64 pencils len=4)
  - 2 32x4 faces (256 cells, 8 pencils len=32)
  - Total: 2304 cells, 352 pencils
- Ghost regions in 3D: (32x32x32 grid, 5 components)
  - 26 regions: 6 faces, 12 edges, 8 corners
  - 8 4x4x4 corners (512 cells, 128 pencils len=4)
  - 4 32x4x4 edges (2048 cells, 64 pencils len=32)
  - 4 4x32x4 edges (2048 cells, 512 pencils len=4)
  - 4 4x4x32 edges (2048 cells, 512 pencils len=4)
  - 2 4x32x32 faces (8192 cells, 2048 pencils len=4)
  - 2 32x4x32 faces (8192 cells, 256 pencils len=32)
  - 2 32x32x4 faces (8192 cells, 256 pencils len=32)
  - Total: 156169 cells, 18880 pencils



Exchange analysis

Number of Communication Operations in Exchange

Exchange Option:	2D	3D	Ease of Use
Pointwise Assignment	2304	156169	Very Easy
UPC_memget	352	18880	Easy
strided memget	8	26	Easy
Pack/Unpack	At most 8	At most 26	Hard

1. Pointwise assignment
   - **Very expensive** without good compiler optimization
2. UPC_memget (current implementation)
   - **Still expensive, especially in 3D**
3. (Proposed) strided memget
   - **Should work well especially with good HW and runtime system**
4. Pack/Unpack (used in Chombo)
   - **Minimal communication, but Harder to program**
     - **two sided, more coordination, extra metadata, two data copies**

Want: non-blocking strided memget/put

- Easy to program!
- Small number of communication calls per grid
- Non-blocking version allows for overlap of computation with communication:
  - Initiate non-blocking communication calls for all local grids.
  - Poll local grids, when all comm for a grid is complete, compute on it.
- Ideal when communication thread runs on dedicated hardware and can perform gather/scatter without host CPU support
  - Red Storm?

Gratuitous UPC-AMR picture

Conclusion

- Completed first stage of Chombo AMR port to UPC
  - Single level, gas dynamics
- Porting is slow
  - Not easy to translate C++ to C
  - Porting Chombo means re-writing Chombo
- Code exposed bugs in portable translator
  - Structure padding for different target architectures
- Initial performance numbers are disappointing, but understood
  - Optimized async strided memget operation needed.