LA-UR-18-28159
Approved for public release; distribution is unlimited.

Title: Adaptive Mesh Refinement in the Fastlane

Author(s): Dunning, Daniel Jeffrey
Marts, William Pepper

Intended for: Presentation to various computing groups (XCP, CCS)

Issued: 2018-08-27

Adaptive Mesh Refinement in the Fastlane

Daniel Dunning and William Marts

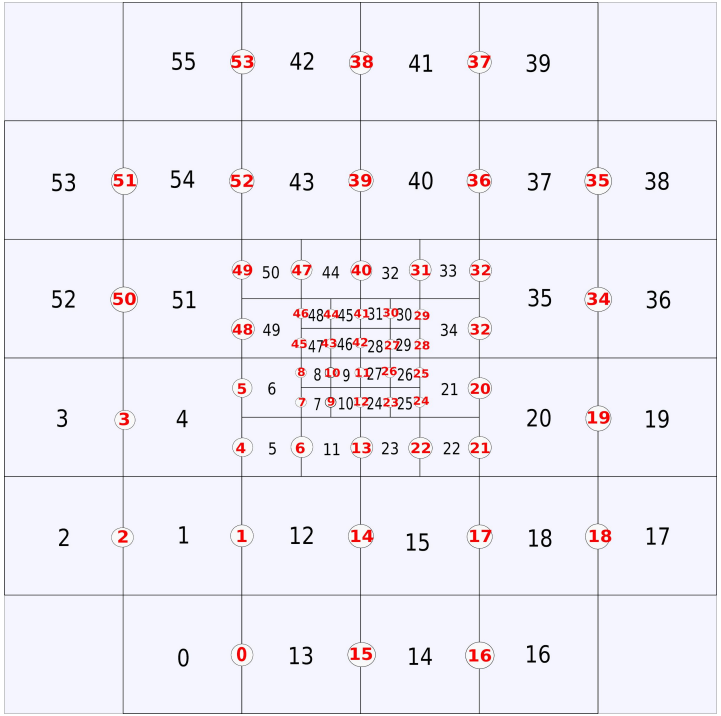
Eulerian Applications Group, Los Alamos National Laboratory
Computer Science Department, Texas Tech University
Computer Science Department, University of New Mexico

Introduction

- Adaptive Mesh Refinement (AMR) allows us to dramatically reduce memory usage by dynamically distributing resources to areas of physical significance.
  - An effective AMR implementation can reduce the memory consumption of fine grids by an order of magnitude.
  - Physical phenomena like shock waves can travel through large domains, but their behavior is determined at small scales.
- Our method is designed to allow the physics simulation to be adapted from a regular grid application with minimal change to the codebase.
  - At minimum it just needs to check a mask on whether it needs to compute a cell
  - An optional change from indexing the spatial position to a single dimension walk of the cell/faces.
- We are interested in providing a way for the entire computation to be run on the GPU.

Three-Level Original AMR Mesh

- Cell numbers in center of cell and face numbers in red in circles along faces.
- To transform our adaptive mesh into our pseudo–regular grid, we add phantom cells and faces to our lists.



Cell-based AMR

- More efficient (fewer cells)
- Easier load balancing
- Less mesh imprinting
  - Better curvilinear symmetry

- Data is unstructured
  - The mesh has structure, but the data can be in any order

Patch-based AMR

- Patches are a regular grid
  - Leverage research in stencil calculations
- More common
  - More research devoted (Berger and Leveque [1])

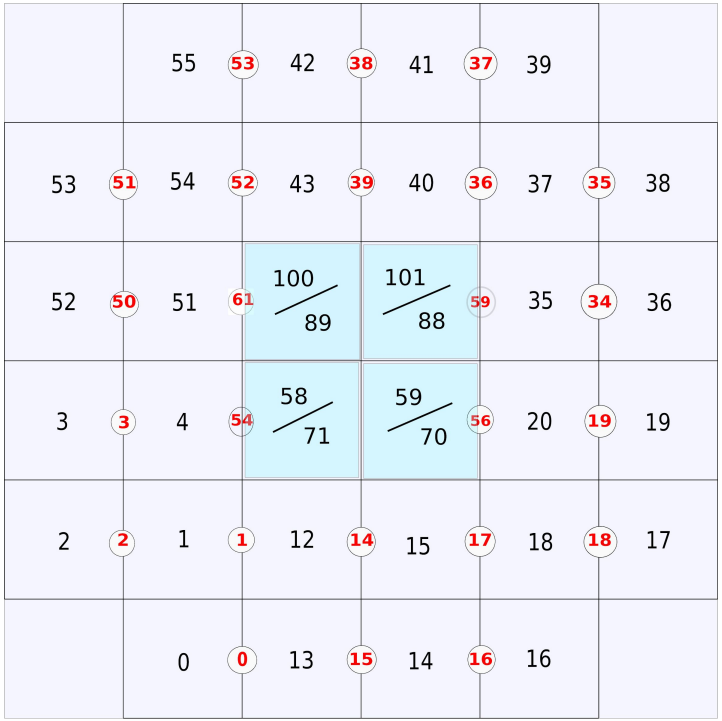
- Clustering of refinement is a non-local operation
  - Difficult to put on the GPU

Phantom-Cell AMR: A Hybrid Approach

- The efficiency of cell-based
- Load balancing of cell-based
- Lack of mesh imprinting of cell-based
- Simple stencil calculations of patch-based
- Unstructured data complications are mitigated by constant time hash-table lookups.
- Local dynamic refinement makes GPU porting pragmatic.

Single-Level Representation

- Here we can see how the *phantom* cells provide the appearance of a regular grid for cells of a given refinement.
- A similar operation is done for every level of the mesh
- We use *phantom* to distinguish between these added cells and the ghost cells that are added for MPI communication halo and boundary cells.



The idea is to separate the physics and the mesh.

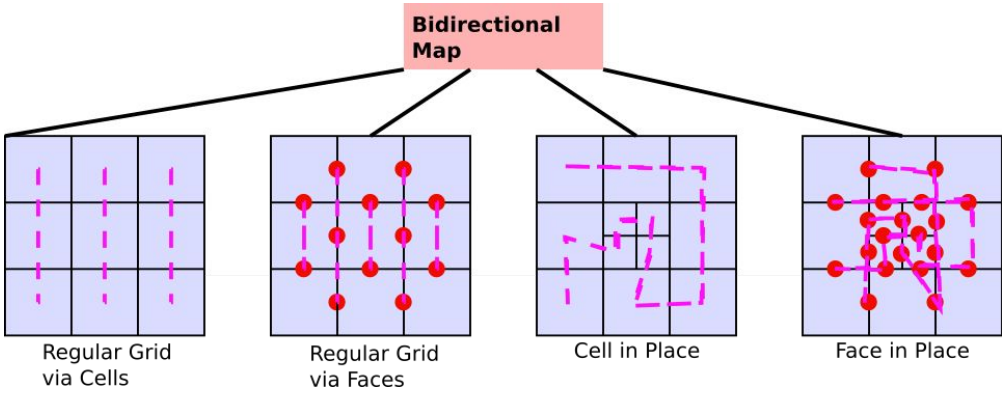
We provide the computational scientists the benefits of AMR with less complexity, fewer headaches, and fewer man-hours.

Traditional AMR is Invasive

- Otherwise simple actions become combinatorially complex.
  - Takes more time to write or modify code.
  - Branching conditionals are harder to debug.
- The physics developer should not have to fully understand the dynamically changing refined mesh.
  - What is the pattern of cells around this one?
  - How do stencils work?
  - What assumptions can I make?
- Optimization and parallelization is difficult.
  - How do I distribute this workload?

Methodology

- Space filling curve preserves locality.
- Bidirectional hash-maps ensure constant time neighbor lookups.
  - `right_neighbor = nrht[cellID];`
- Interpolated and flux-corrected values for phantom cells.

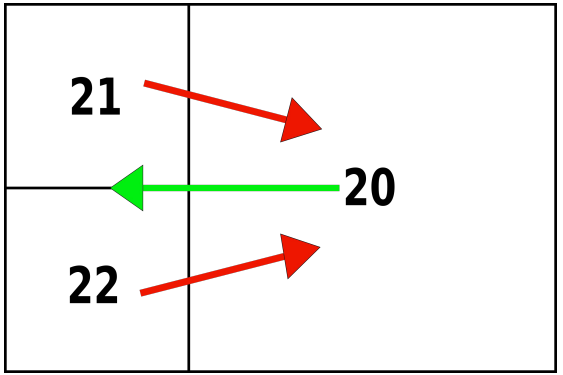


How Phantom Cells Work

- In our technique we add *phantom* cells along the boundaries of refinement levels.
- These *phantom* cells are interpolated from the values of the refined mesh to which they correspond, and are at the same refinement level as the neighbor they were created to assist.
- They allow the physics simulation to obtain data from the mesh without consideration of the scales of the cells in question. A cell can always ask for a neighbor, regardless of refinement.
    - If a cell's right neighbor is more refined, the mesh code will provide a phantom at the same refinement to perform the calculation. The programer does not need to add code to your physics simulation to check.
- Note that *phantom* cells can overlap spatially.
    - They are not replacing any cells. They are just a convenient means to provide interpolation seamlessly at boundaries of refinement.

Mass Conservation And Interpolation

- Uses technique from Berger-Leveque that takes the finer cell flux and applies it to the coarser cell.
- Various interpolations of the AMR mesh are feasible to create the *phantom cells.*



Finer cells 21 and 22 will pass their positive x Flux values (red) to their coarse neighbor, cell 20, who will use the average of those fluxes as its negative x Flux value.

Regular Grid

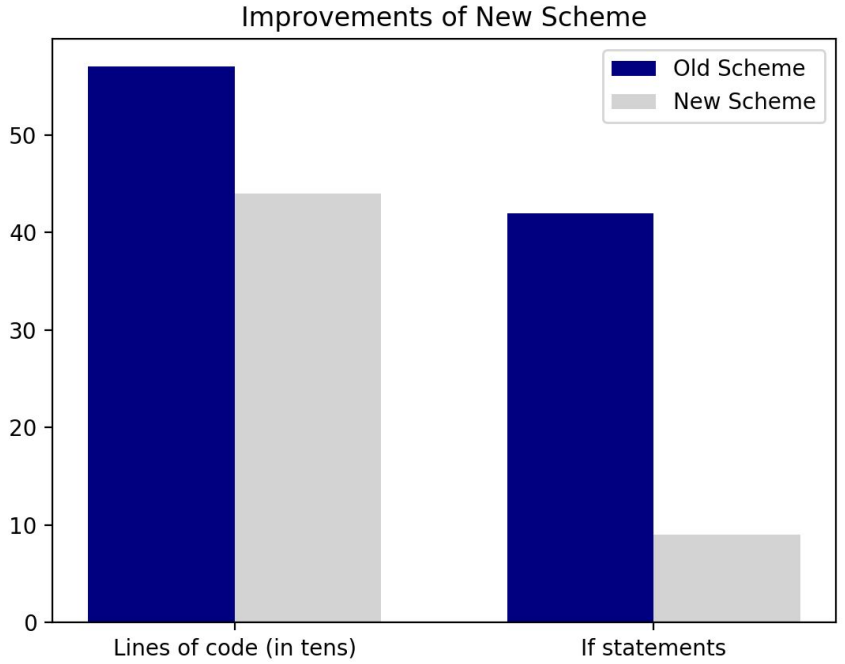
- Generate phantom cells at the boundaries of each refinement level.
- Separate the grid into pieces based on refinement level.
- Traverse the grid as usual using regular spatial coordinate indexing (i and j).
  - `right_neighbor = mesh[i+1][j];`
- Phantom cells are incorporated in each level of regular grid as a real cell and its values are used for calculation, but this does not affect the mask which points to the original cells

In Place Using Bidirectional Maps

- The mesh structure is not altered from the original multi-level array, the data structure is not separated by refinement level.
- The underlying mesh does not change, but neighbor hashmaps are added to point to phantom cells (where necessary).
  - `right_neighbor = nrht[cellID];`
  - Bidirectional hash-maps ensure constant time neighbor lookups.
- Each cell can access their neighbors as if they are part of a regular grid, but the physics code is still iterating through an AMR grid with differing levels of refinement.

Fewer Lines, Less Divergence

- Our technique allows developers to write simpler code that is analogous to working on a regular grid.
- It reduces the man hours needed by separating interpolation and complex indexing from the physics code.
  - 15% line reduction
  - 78% conditional reduction



Vastly Easier to Implement

- Developers can design code as if it was running a regular grid.
  - The mesh code handles the complexity of querying neighbors of varying refinements.
- Existing projects can leverage the localized refinement of AMR.
  - We dynamically devote resources.
  - The user doesn't need to predict the areas of significance.
- A simplified road-map to exascale computation.

Shifts in design

- Originally planned to only produce a single regular grid mesh type
  - Realized there could be three other methods to be manufactured
  - In-place methods would not need to be copied to a regular grid mesh
- Originally planned to use an interpolation of state values of the cells
  - This proved difficult to conserve mass
  - We turned to using techniques outlined by Berger and Leveque [1]

Conclusion

- Computational scientists can fully utilize Adaptive Mesh Refinement with less complexity in their code and fewer man-hours.
- AMR mesh code is decoupled from the physics code.
- Optimizations are easily ported from project to project.

Future Work

- Complete all four mesh methods
- Implement for all hardware and parallel architectures

There is interest in doing this work as part of dissertation research.

References

1. Marsha J Berger and Randall J LeVeque. **Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems**. *SIAM Journal on Numerical Analysis*, 25(6):2298-2316, 1998.

Acknowledgements:

Robert Robey Eulerian Applications Group, Los Alamos National Laboratory

Patrick Bridges Center for Advanced Research Computing, University of New Mexico