Block-Structured Adaptive Mesh Refinement

John Bell

jbbell@lbl.gov

Center for Computational Sciences and Engineering

Lawrence Berkeley National Laboratory, USA

http://seesar.lbl.gov/ccse/

Presented at: Adaptive Grid Approaches for Fluid Dynamics
Cambridge University
Cambridge, UK
December 14–15, 2004

Collaborators: A. Almgren, V. Beckner, M. Day, J. Grcar, M. Lijewski, C. Rendleman
M. Berger, P. Colella

Types of Refinement

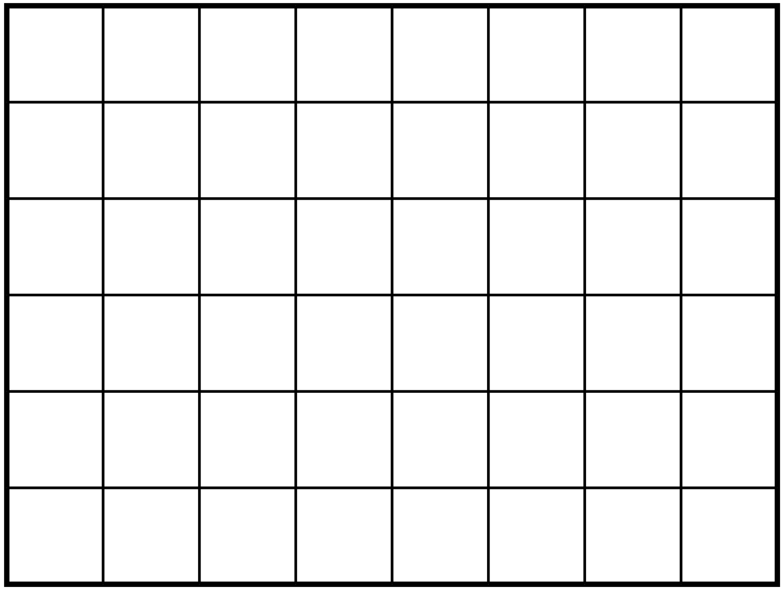
AMR approaches

- Unstructured

Types of Refinement

AMR approaches

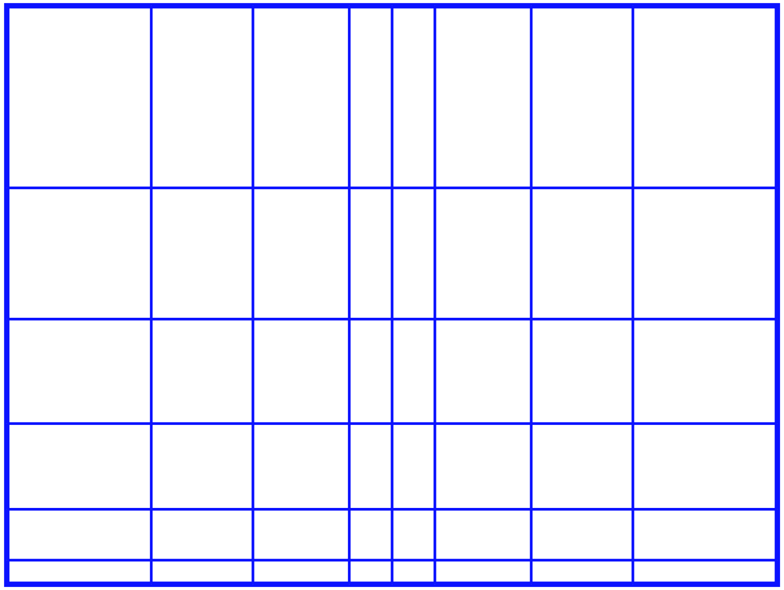
- Unstructured
- Mesh distortion



Types of Refinement

AMR approaches

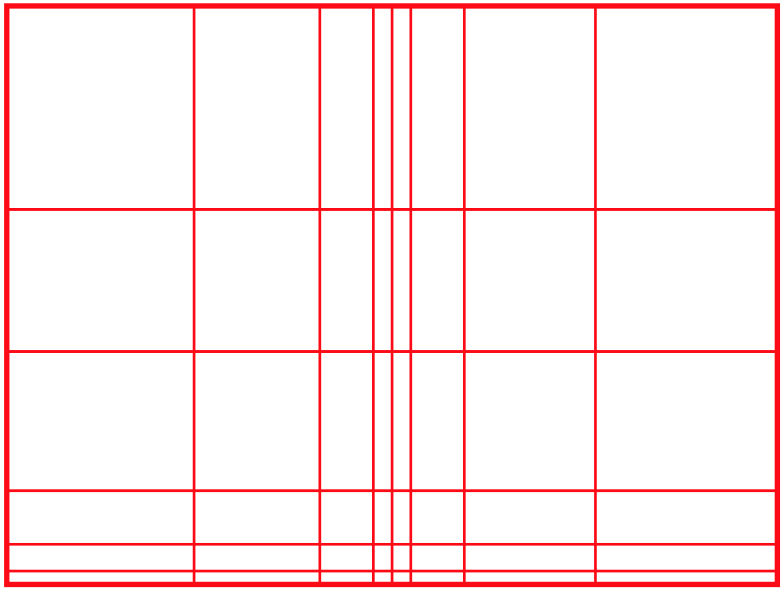
- Unstructured
- Mesh distortion



Types of Refinement

AMR approaches

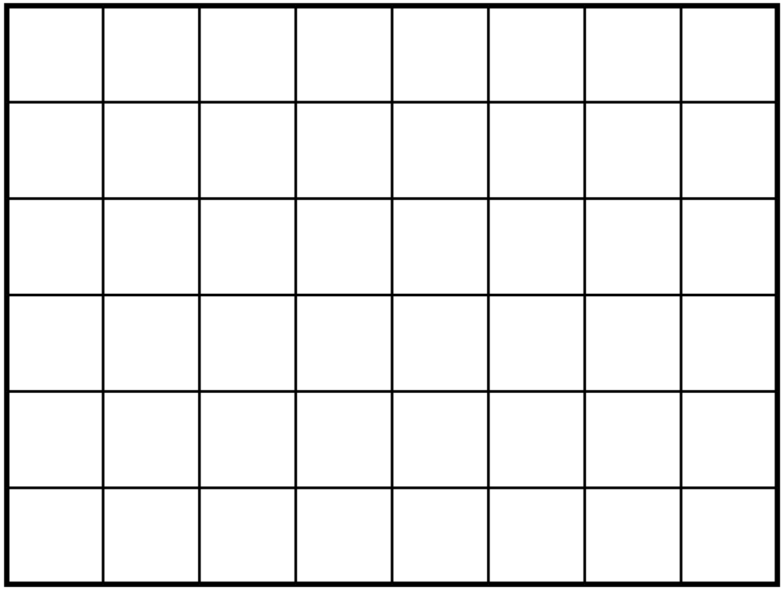
- Unstructured
- Mesh distortion



Types of Refinement

AMR approaches

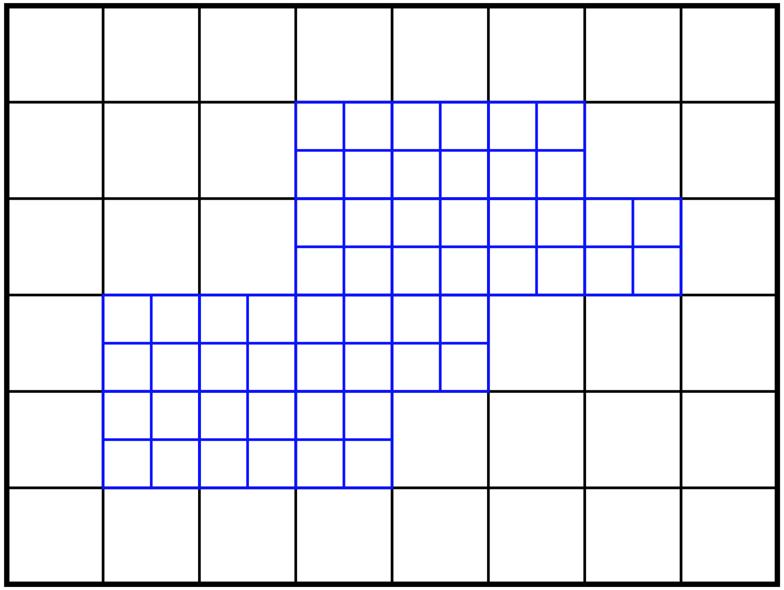
- Unstructured
- Mesh distortion
- Point-wise structured refinement



Types of Refinement

AMR approaches

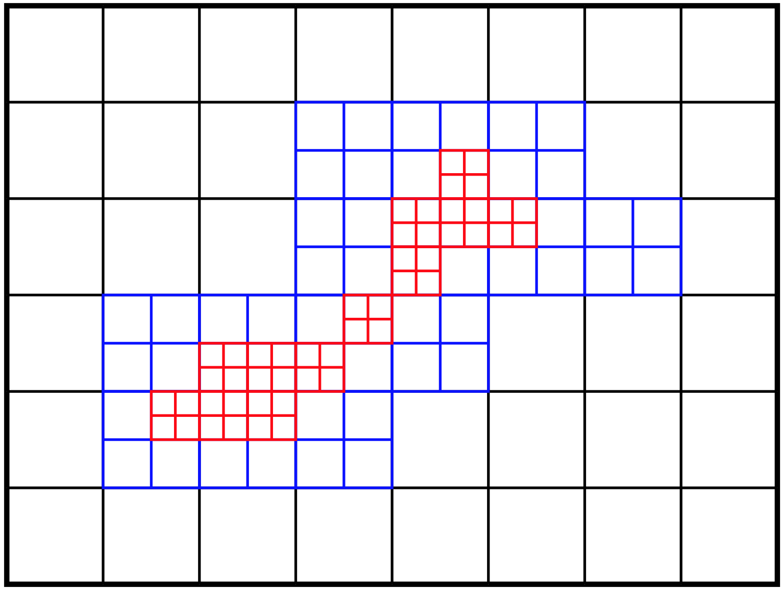
- Unstructured
- Mesh distortion
- Point-wise structured refinement



Types of Refinement

AMR approaches

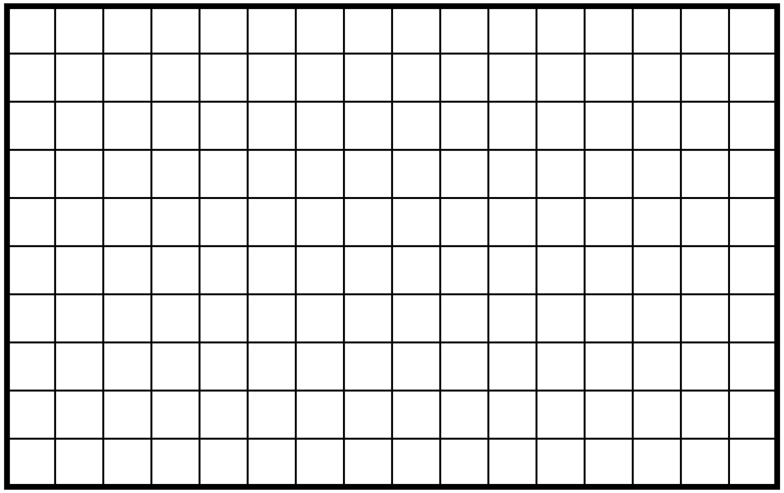
- Unstructured
- Mesh distortion
- Point-wise structured refinement



Types of Refinement

AMR approaches

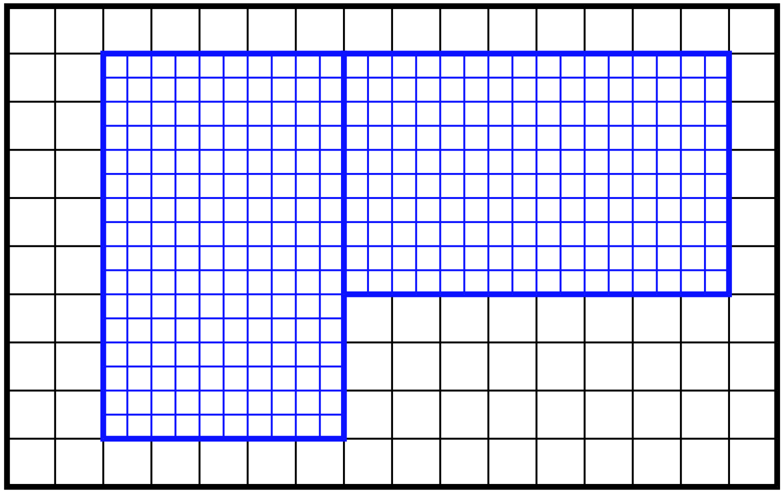
- Unstructured
- Mesh distortion
- Point-wise structured refinement
- Block structured



Types of Refinement

AMR approaches

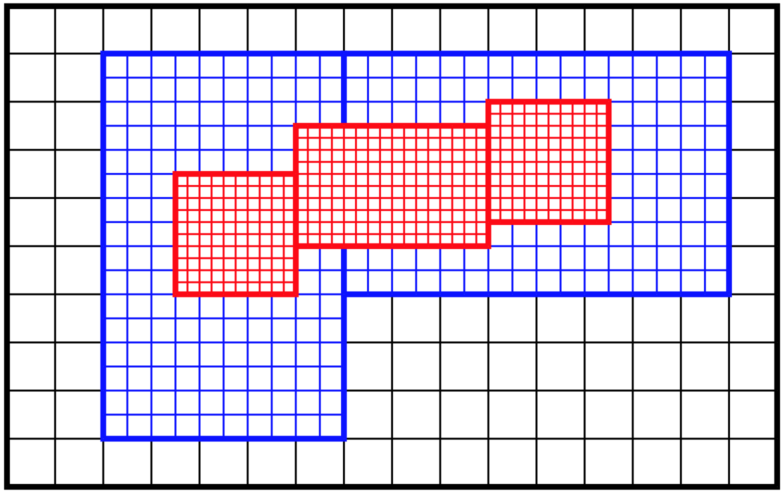
- Unstructured
- Mesh distortion
- Point-wise structured refinement
- Block structured



Types of Refinement

AMR approaches

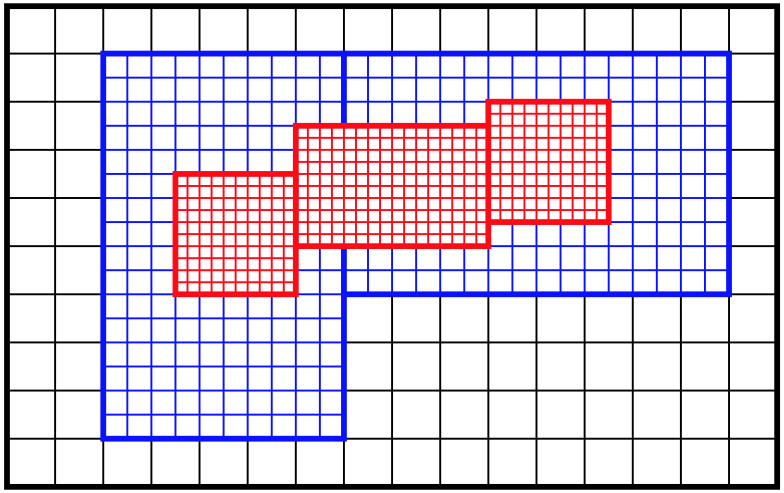
- Unstructured
- Mesh distortion
- Point-wise structured refinement
- Block structured



Types of Refinement

AMR approaches

- Unstructured
- Mesh distortion
- Point-wise structured refinement
- Block structured



The next four lectures focus on block-structured refinement for time-dependent problems

- Basic ideas
- Multi-physics applications
- Implementation issues

Overview of Lectures

Day 1:

1) AMR for Hyperbolic Conservation Laws ("Hello World")

- Preliminaries
- Key AMR ideas
- Software / Parallel Implementation

2) Extension to More General Systems

- Incompressible Navier-Stokes
  - Fractional Step Scheme
- 1-D AMR for “classical” PDE’s
  - hyperbolic
  - elliptic
  - parabolic
- Spatial accuracy considerations

Overview of Lectures (p.2)

Day 2:

3) IAMR and Extension to Multiphysics

- Incompressible AMR
- Software to support IAMR
- Multiphysics applications
  - LMC (Low Mach Number Combustion)
  - AMAR (Adaptive Mesh and Algorithm Refinement)

4) Geometry

- Embedded Boundary
- Software to support EB

AMR for Conservation Laws

Consider the 2-D hyperbolic conservation law

$$U_t + \mathbf{F}_x + \mathbf{G}_y = 0$$

where

$$\mathbf{F} = \mathbf{F}(U), \mathbf{G} = \mathbf{G}(U)$$

Basic discretization:

- Finite volume approach with cell-centered data
- Flux-based representation of conservation law
- Explicit in time update

$$\frac{U^{n+1} - U^n}{\Delta t} = \frac{\mathbf{F}^{n+\frac{1}{2}}_{i-1/2,j} - \mathbf{F}^{n+\frac{1}{2}}_{i+1/2,j}}{\Delta x} + \frac{\mathbf{G}^{n+\frac{1}{2}}_{i,j-1/2} - \mathbf{G}^{n+\frac{1}{2}}_{i,j+1/2}}{\Delta y}$$

Numerical fluxes computed from data at t^n in neighborhood of edge

Basic AMR Issues

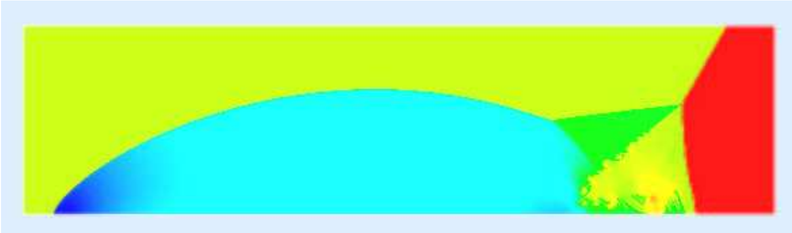
Basic design concepts for AMR

- Cover regions requiring high resolution with finer grids
- Use higher-order upwind methodology for regular grids to integrate PDE
- Refine in space and time
  - Maintain CFL across levels
  - Subcycle in time

Issues

- Generation of grid hierarchy
- How to integrate on hierarchy
  - Integration of data on a patch
  - Synchronization of levels

Shock Reflection



Original references:

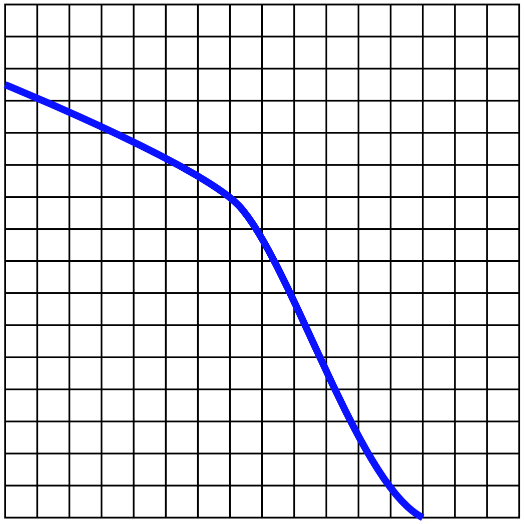
- 2-D: Berger and Colella, JCP 1989
- 3-D: Bell,Berger,Saltzman and Welcome, JCP 1994

Building the initial grid hierarchy

- Fill data at level 0
- Estimate where refinement is needed and buffer
- Group cells into patches according to a prescribed “grid efficiency” and refine $\Rightarrow B_1, ..., B_n$ (Berger and Rigoustos, 1991)
- Repeat for next level and adjust for proper nesting

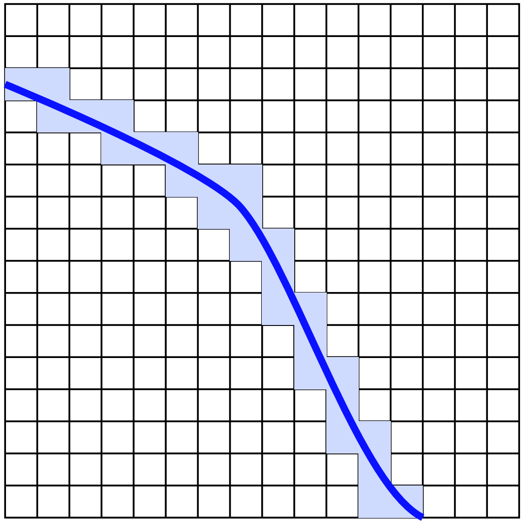
Building the initial grid hierarchy

- Fill data at level 0
- Estimate where refinement is needed and buffer
- Group cells into patches according to a prescribed “grid efficiency” and refine $\Rightarrow B_1, ..., B_n$ (Berger and Rigoustos, 1991)
- Repeat for next level and adjust for proper nesting



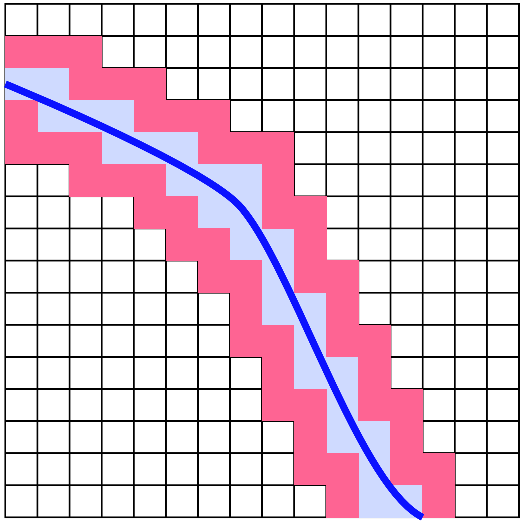
Building the initial grid hierarchy

- Fill data at level 0
- Estimate where refinement is needed and buffer
- Group cells into patches according to a prescribed “grid efficiency” and refine $\Rightarrow B_1, ..., B_n$ (Berger and Rigoustos, 1991)
- Repeat for next level and adjust for proper nesting



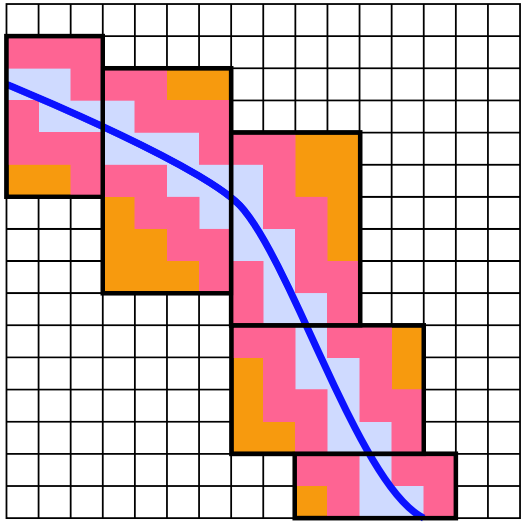
Building the initial grid hierarchy

- Fill data at level 0
- Estimate where refinement is needed and buffer
- Group cells into patches according to a prescribed "grid efficiency" and refine $\Rightarrow B_1, ..., B_n$ (Berger and Rigoustos, 1991)
- Repeat for next level and adjust for proper nesting



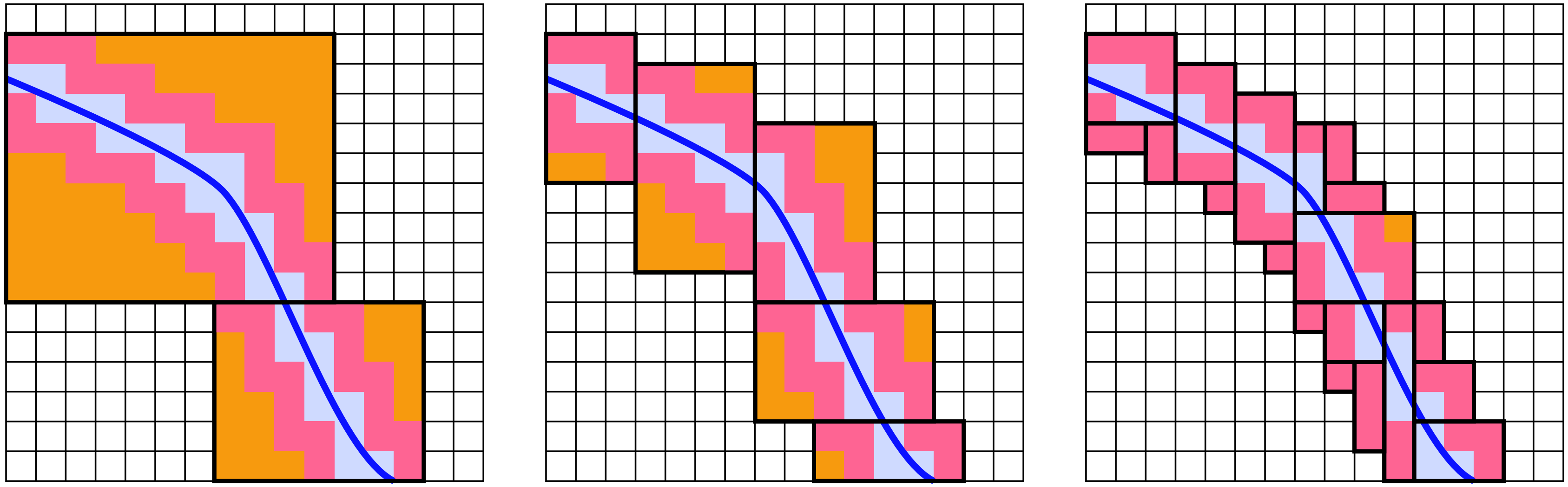
Building the initial grid hierarchy

- Fill data at level 0
- Estimate where refinement is needed and buffer
- Group cells into patches according to a prescribed "grid efficiency" and refine $\Rightarrow B_1, ..., B_n$ (Berger and Rigoustos, 1991)
- Repeat for next level and adjust for proper nesting



Building the initial grid hierarchy

- Fill data at level 0
- Estimate where refinement is needed and buffer
- Group cells into patches according to a prescribed “grid efficiency” and refine $\Rightarrow B_1, ..., B_n$ (Berger and Rigoustos, 1991)
- Repeat for next level and adjust for proper nesting



Efficiency = 0.5 | Efficiency = 0.7 | Efficiency = 0.9

Adaptive integration algorithm

Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.

Adaptive integration algorithm

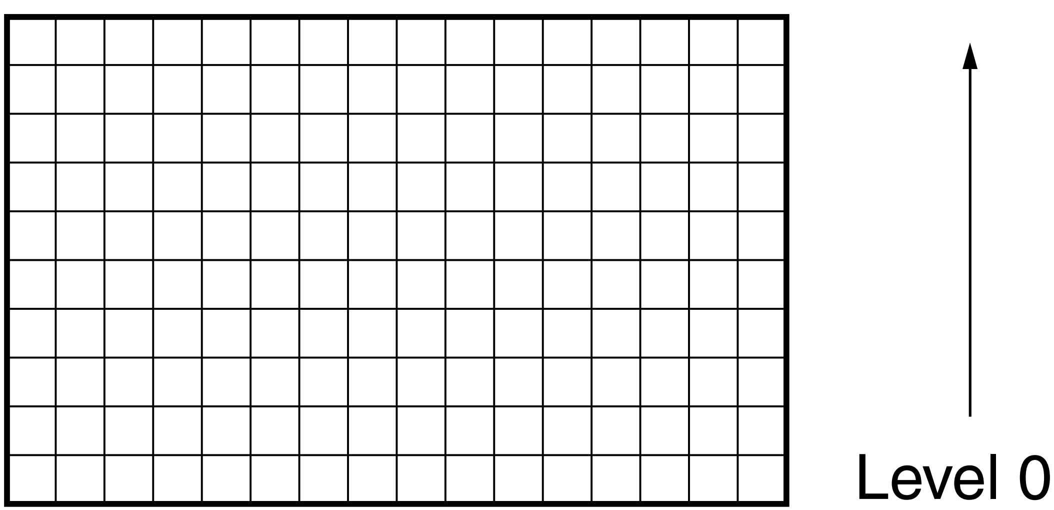
Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.



Adaptive integration algorithm

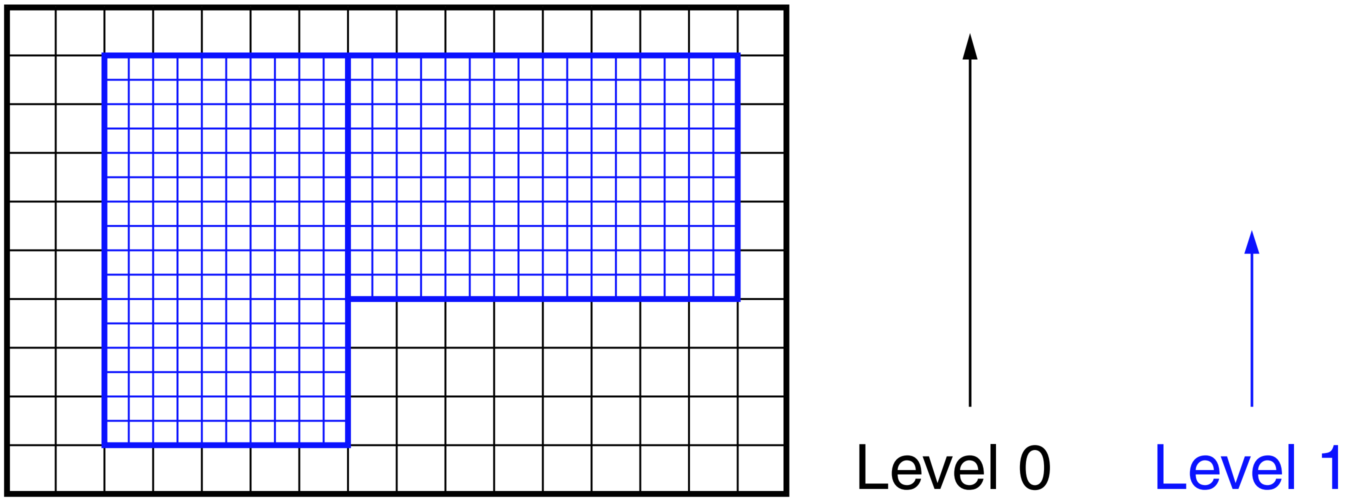
Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \to t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.



Adaptive integration algorithm

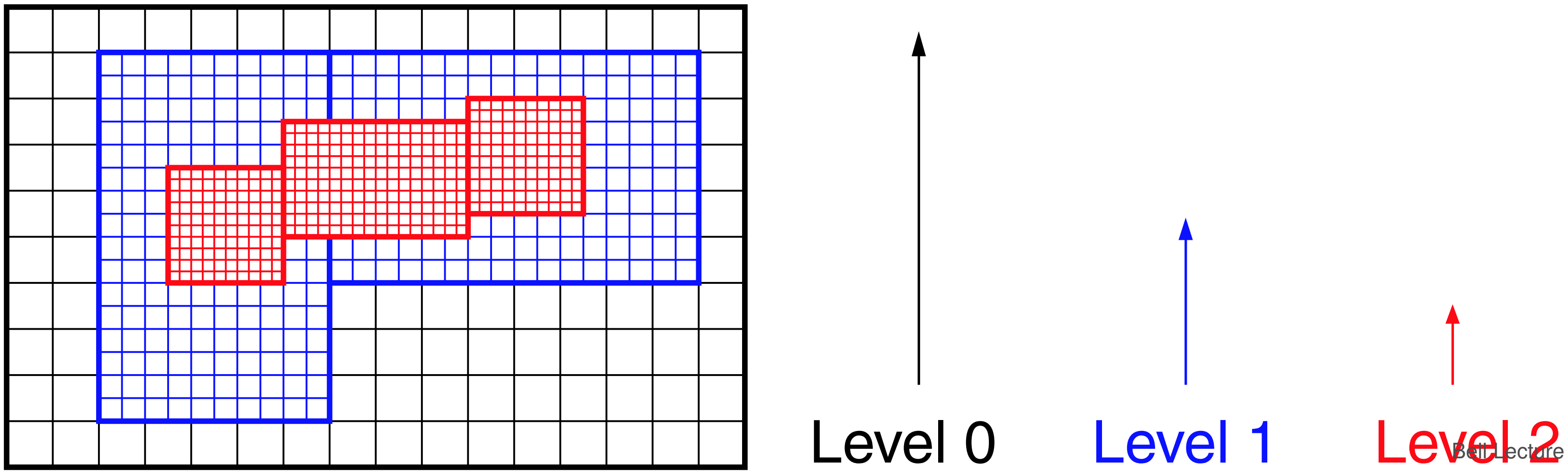
Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.



Adaptive integration algorithm

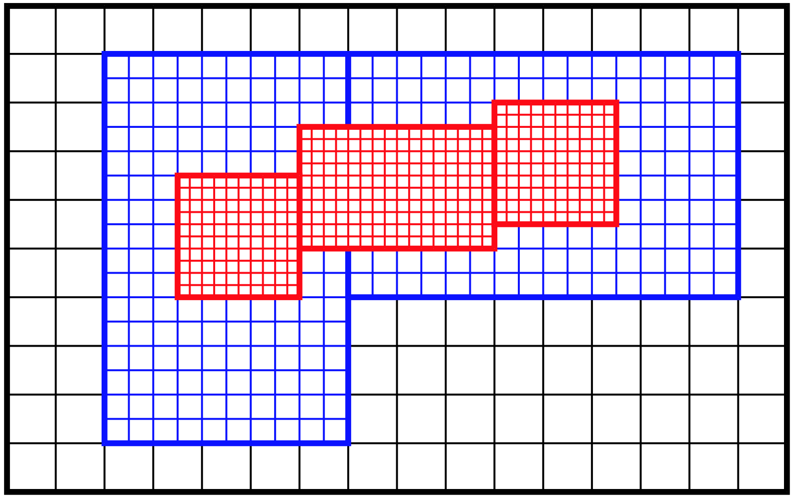
Consider two levels, coarse and fine, with refinement ratio r

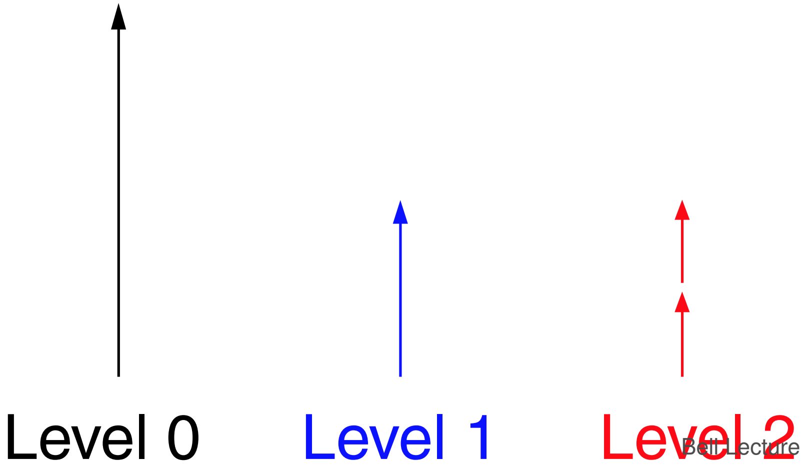
$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.





Adaptive integration algorithm

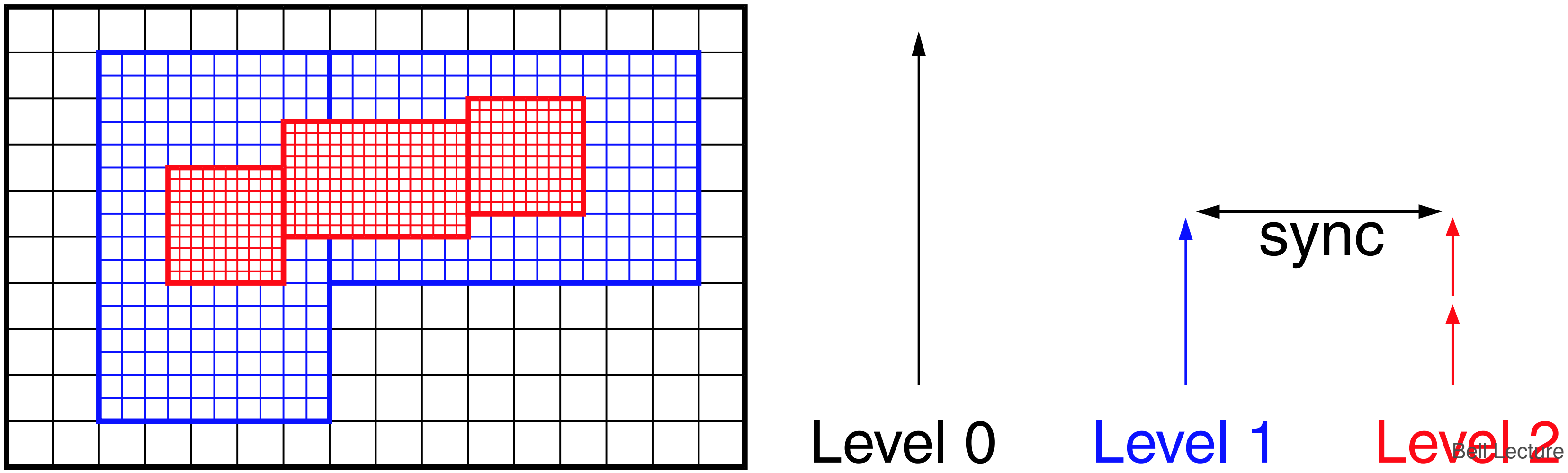
Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.



Adaptive integration algorithm

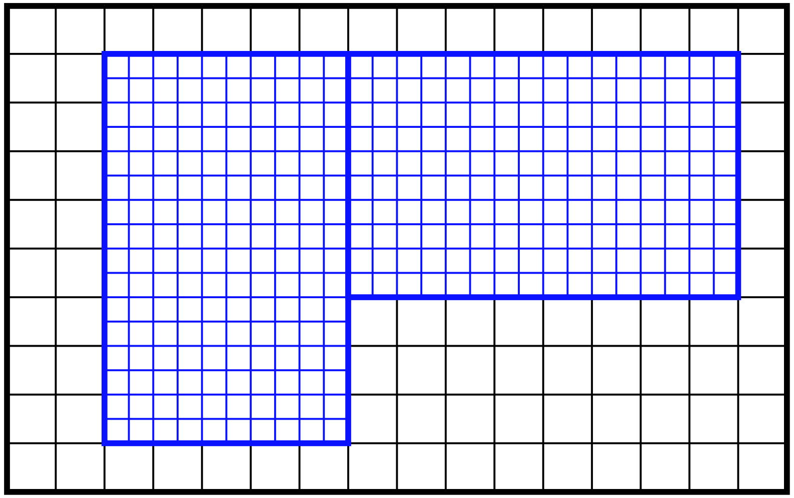
Consider two levels, coarse and fine, with refinement ratio r

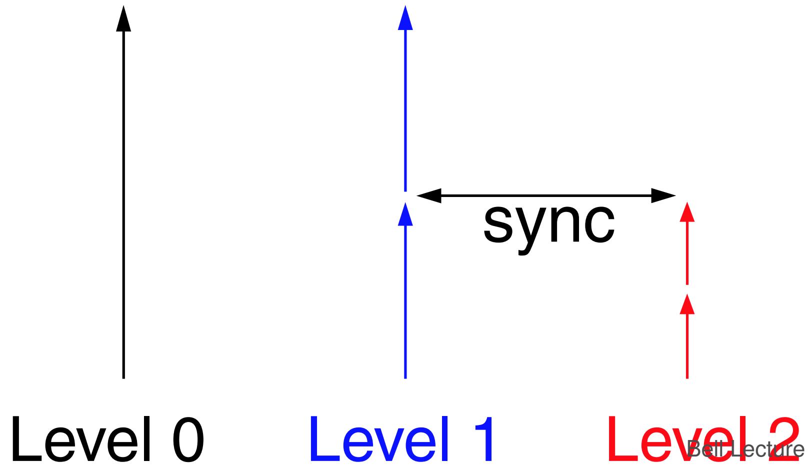
$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.





Adaptive integration algorithm

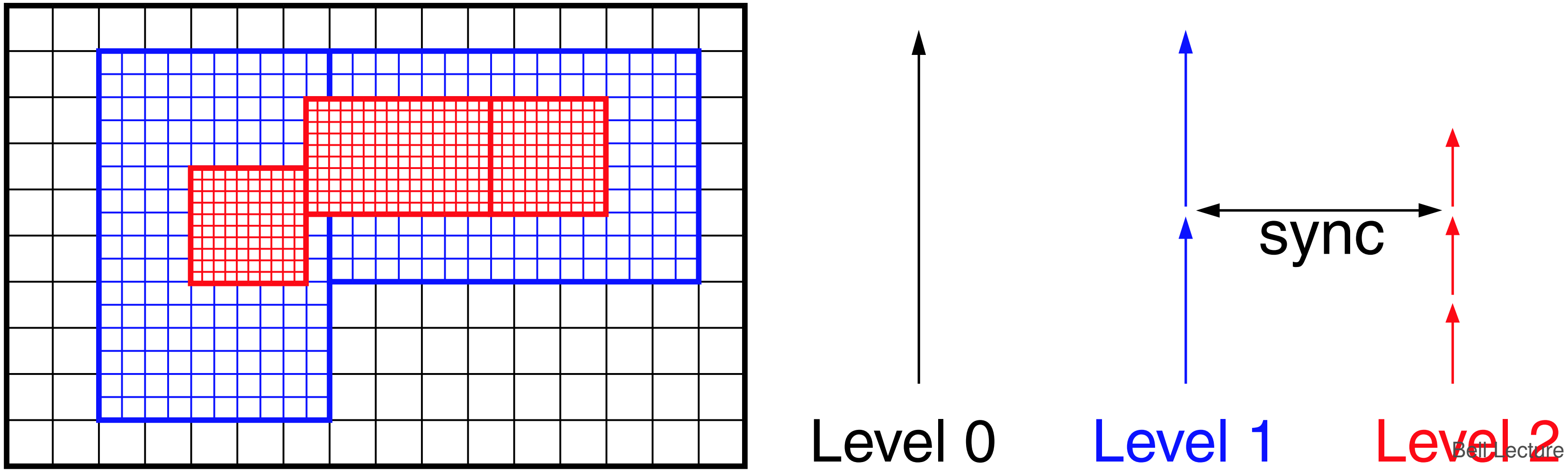
Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.



Adaptive integration algorithm

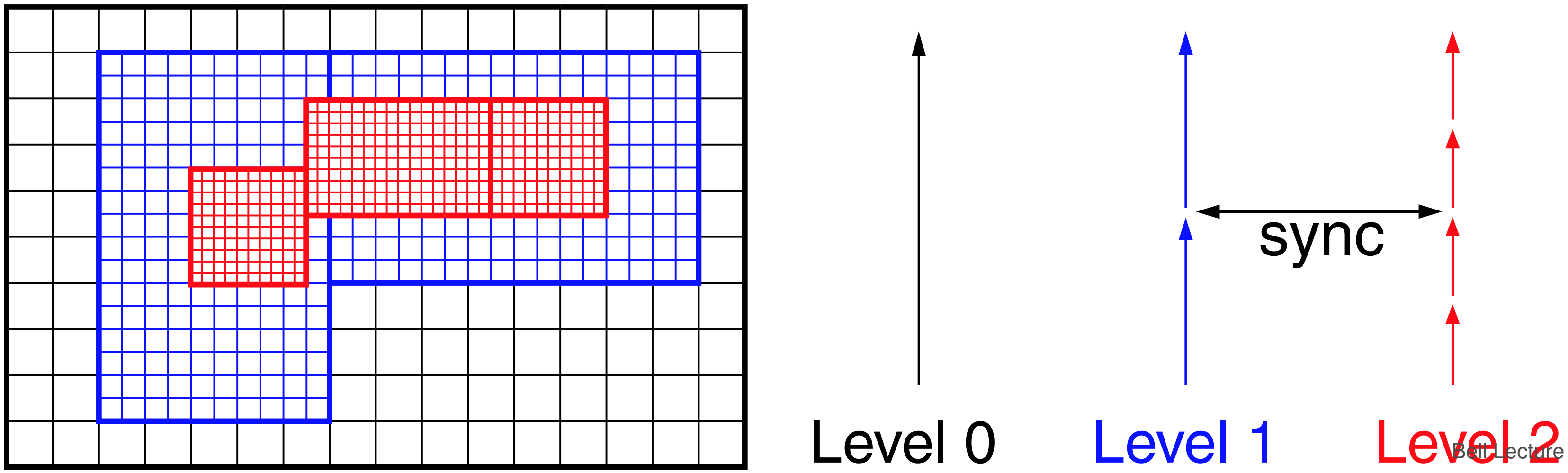
Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.



Adaptive integration algorithm

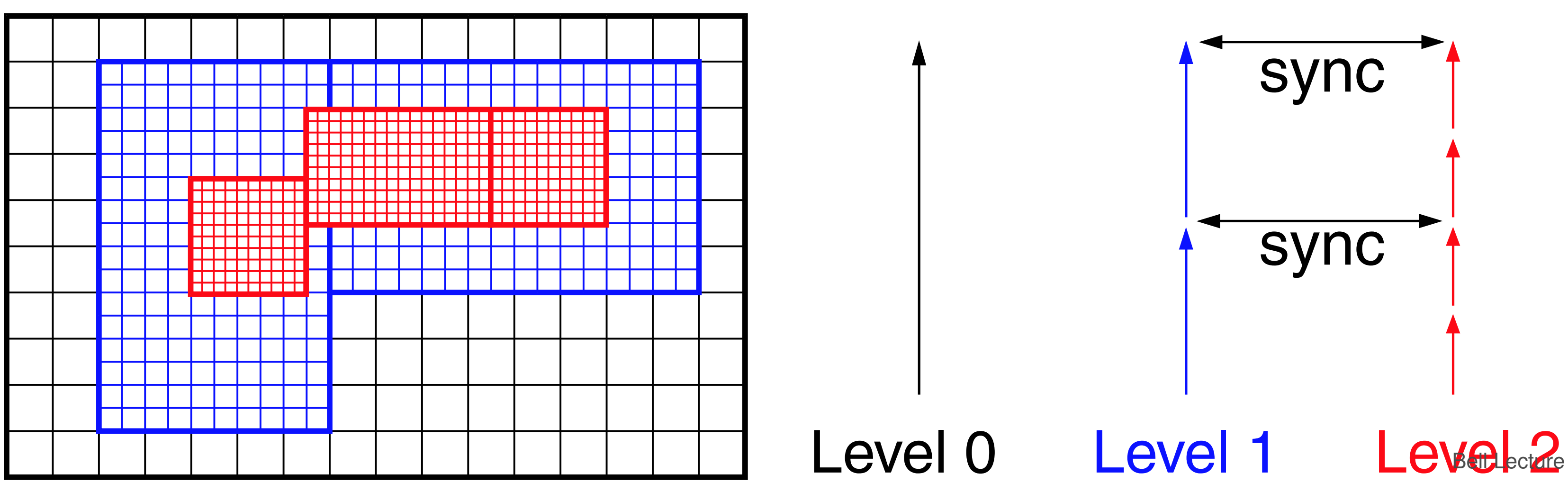
Consider two levels, coarse and fine, with refinement ratio r

$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.



Adaptive integration algorithm

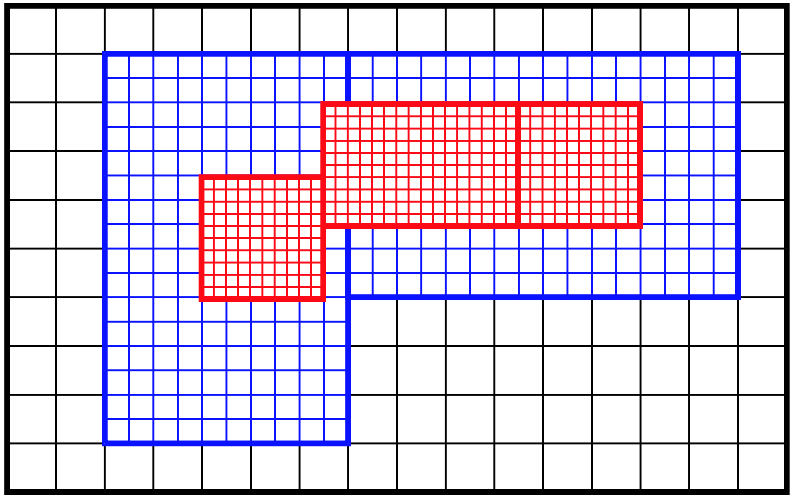
Consider two levels, coarse and fine, with refinement ratio r

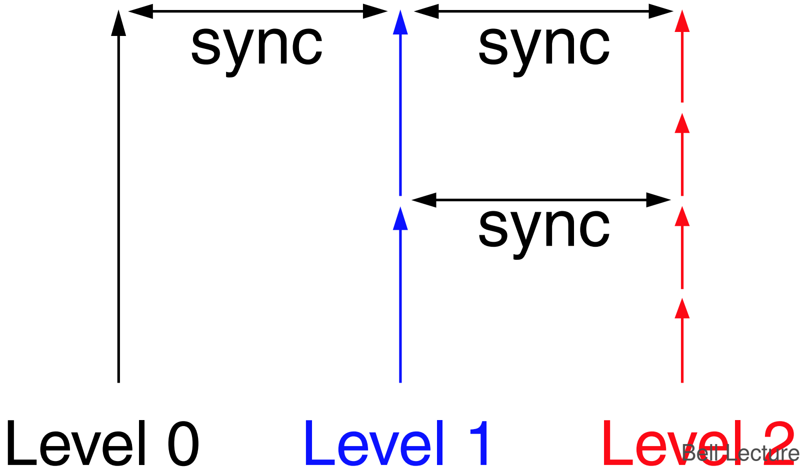
$$\Delta x_f = \Delta x_c / r \quad , \quad \Delta t_f = \Delta t_c / r,$$

To integrate

- Advance coarse grids in time $t_c \rightarrow t_c + \Delta t_c$
- Advance fine grids in time r times
- Synchronize coarse and fine data

Extend recursively to arbitrary number of refinement levels.





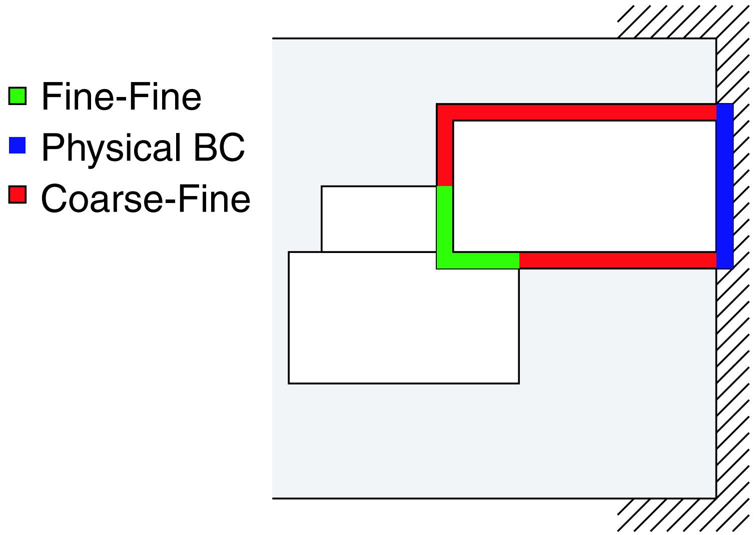
Integrating on grid patch

How do you integrate a patch of data at level ℓ?

Obtain boundary data needed to call integrator on uniform grid of data.

- Assume explicit scheme with stencil width s_d

- Enlarge patch by s_d cells in each direction and fill with data using
  - Physical boundary conditions
  - Other patches at the same level
  - Coarse grid data

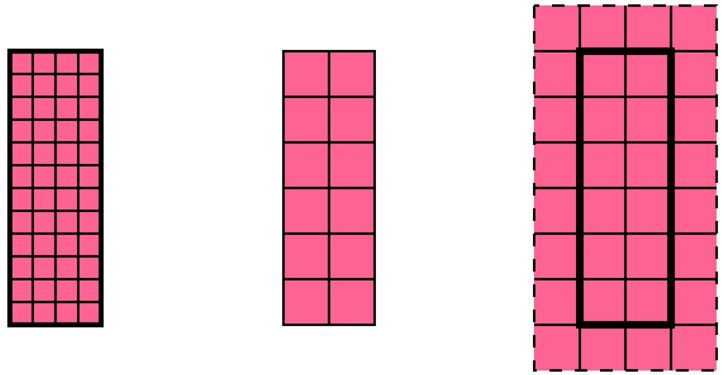


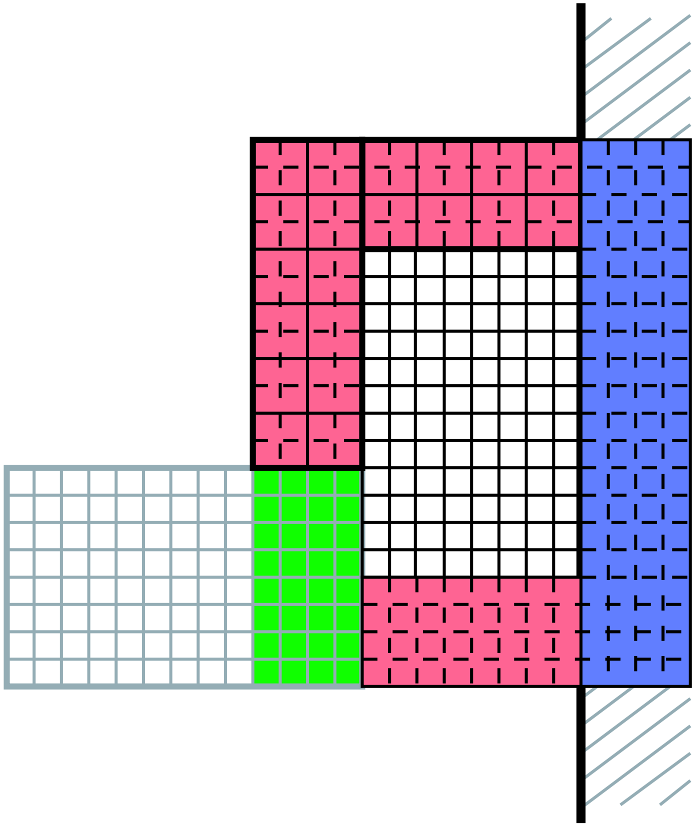
- Advance grid in time $t \to t + \Delta t$

FillPatch Operation

To fill fine grid "ghost cells" at $t+k\Delta t_f$, $k = 0, ..., r-1$, using coarse grid data

- Define coarse patch needed for interpolation





- Fill coarse patch at time t and $t + \Delta t_c$
- Time-interpolate data on coarse patch to time $t+k\Delta t_f$
- Interpolate coarse data to fine patch

Synchronization

After coarse grid time step and the subcycled advance of fine data, we have

- U^c at t_c^{n+1}
- U^f at t_c^{n+1}

However, U^c and U^f are not consistent

- Coarse data is not necessarily equal to the average of the fine grid data "over" it.
- Scheme violates conservation because of inconsistent fluxes at coarse-fine interface

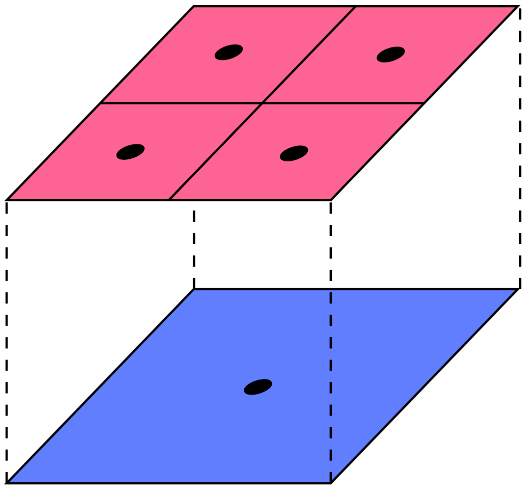
Synchronization

After coarse grid time step and the subcycled advance of fine data, we have

- U^c at t_c^{n+1}
- U^f at t_c^{n+1}

However, U^c and U^f are not consistent

- Coarse data is not necessarily equal to the average of the fine grid data "over" it.
- Scheme violates conservation because of inconsistent fluxes at coarse-fine interface



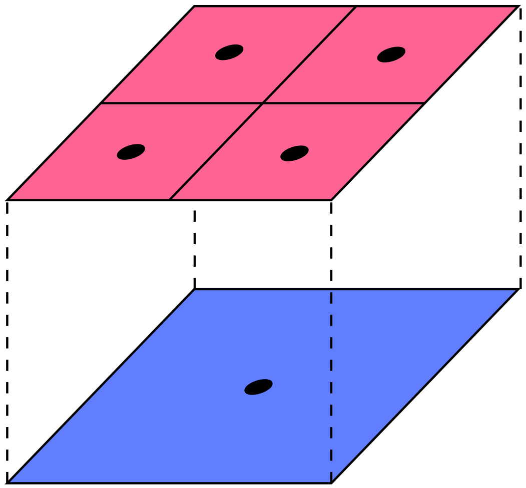
Synchronization

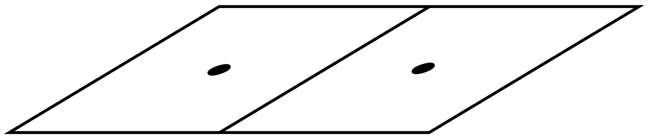
After coarse grid time step and the subcycled advance of fine data, we have

- U^c at t_c^{n+1}
- U^f at t_c^{n+1}

However, U^c and U^f are not consistent

- Coarse data is not necessarily equal to the average of the fine grid data "over" it.
- Scheme violates conservation because of inconsistent fluxes at coarse-fine interface





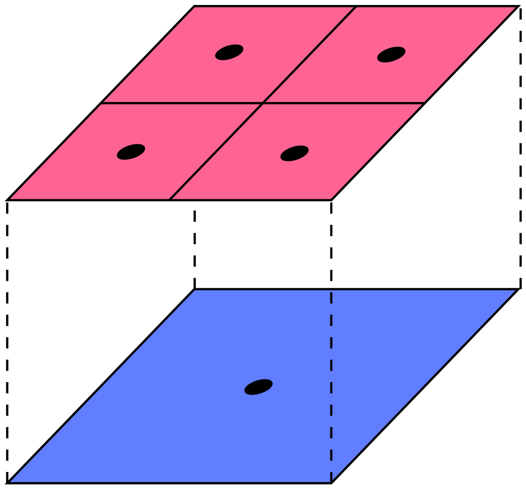
Synchronization

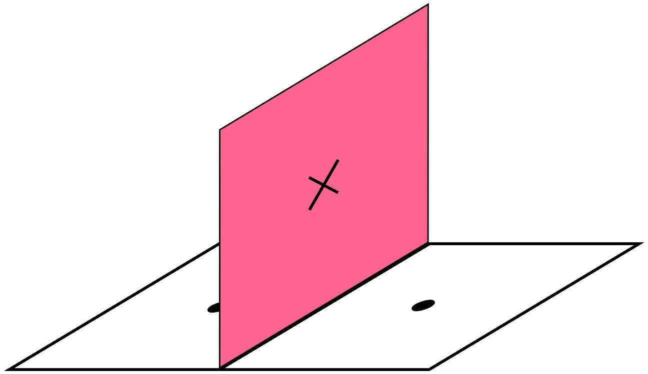
After coarse grid time step and the subcycled advance of fine data, we have

- U^c at t_c^{n+1}
- U^f at t_c^{n+1}

However, U^c and U^f are not consistent

- Coarse data is not necessarily equal to the average of the fine grid data "over" it.
- Scheme violates conservation because of inconsistent fluxes at coarse-fine interface





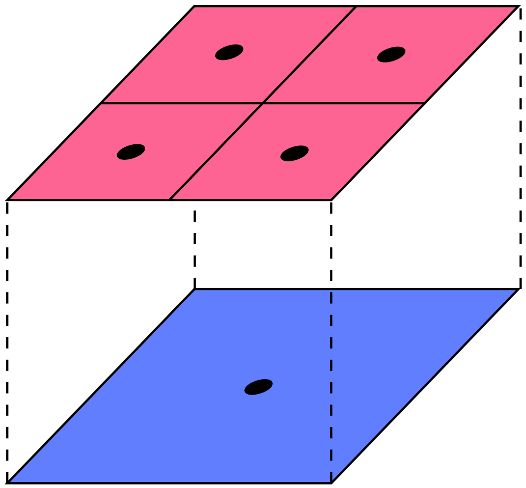
Synchronization

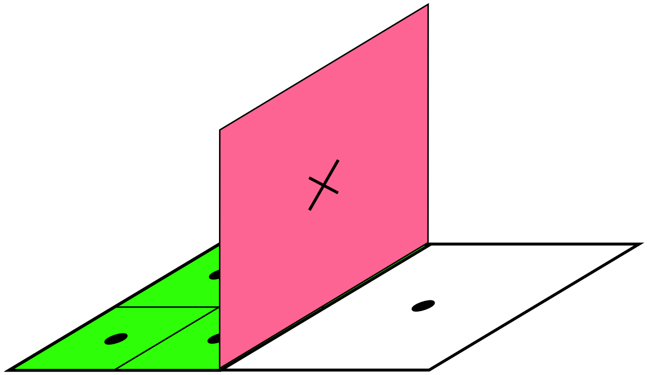
After coarse grid time step and the subcycled advance of fine data, we have

- U^c at t_c^{n+1}
- U^f at t_c^{n+1}

However, U^c and U^f are not consistent

- Coarse data is not necessarily equal to the average of the fine grid data “over” it.
- Scheme violates conservation because of inconsistent fluxes at coarse-fine interface





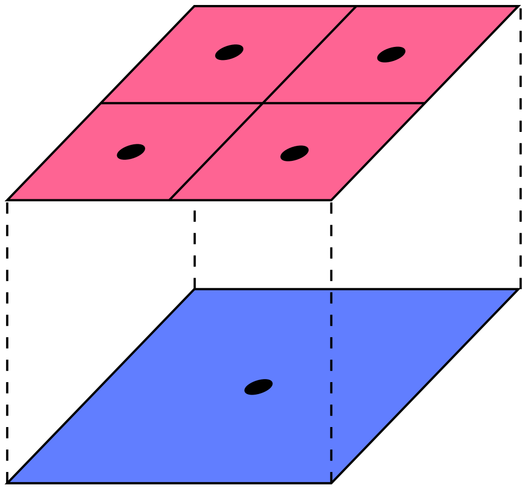
Synchronization

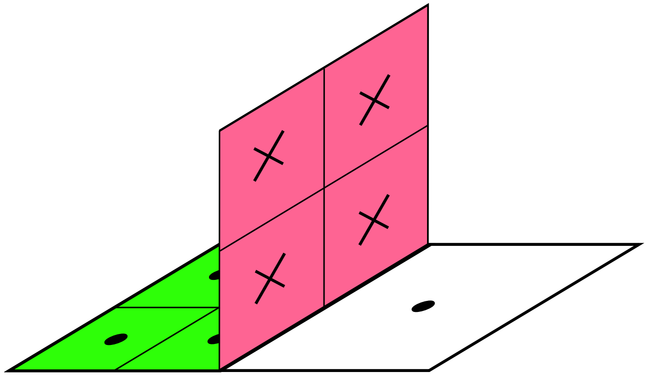
After coarse grid time step and the subcycled advance of fine data, we have

- U^c at t_c^{n+1}
- U^f at t_c^{n+1}

However, U^c and U^f are not consistent

- Coarse data is not necessarily equal to the average of the fine grid data "over" it.
- Scheme violates conservation because of inconsistent fluxes at coarse-fine interface





Synchronization (p2)

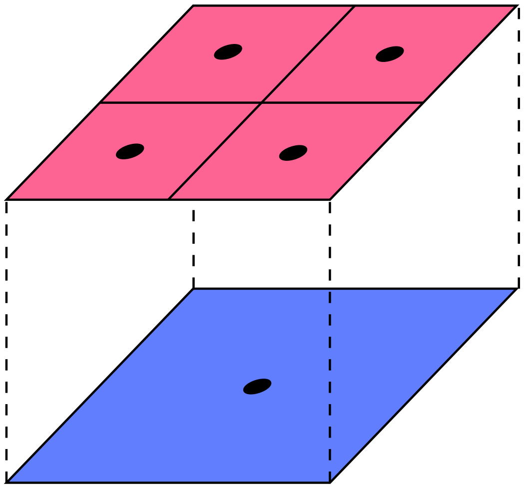
How do we address these problems with the solution?

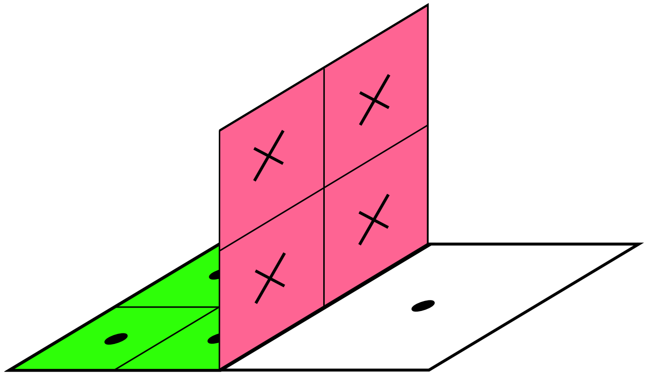
- Average down the fine grid data onto all underlying coarse cells

$$U^c = \frac{1}{r^d} \sum U^f$$

- Reflux at coarse-fine interfaces

$$\Delta x_c \Delta y_c U^c = \Delta x_c \Delta y_c U^c - \Delta t^c A^c \mathbf{F}^c + \sum \Delta t^f A^f \mathbf{F}^f$$





Regridding

Compute “error” at each cell : if “too big” then flag cell for refinement

- Richardson extrapolation
  - Coarsen data on a patch at t^{n-1} and advance by $2\Delta t$
  - Advance data at t^n by Δt and coarsen
  - Difference of these two solutions is proportional to error
- Functions of solution (e.g., vorticity)
- Geometric considerations

Compute refined patches as initially

Fill newly refined regions using conservative interpolation from coarse grid

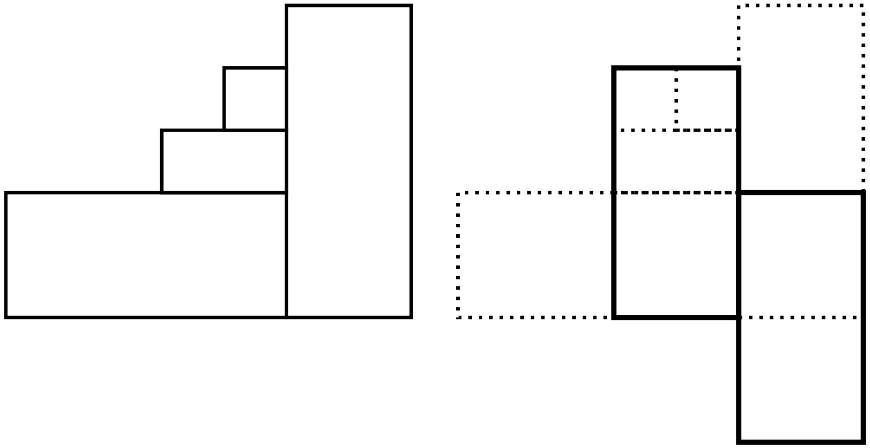
Regridding

Compute "error" at each cell : if "too big" then flag cell for refinement

- Richardson extrapolation
  - Coarsen data on a patch at t^{n-1} and advance by $2\Delta t$
  - Advance data at t^n by Δt and coarsen
  - Difference of these two solutions is proportional to error
- Functions of solution (e.g., vorticity)
- Geometric considerations

Compute refined patches as initially

Fill newly refined regions using conservative interpolation from coarse grid



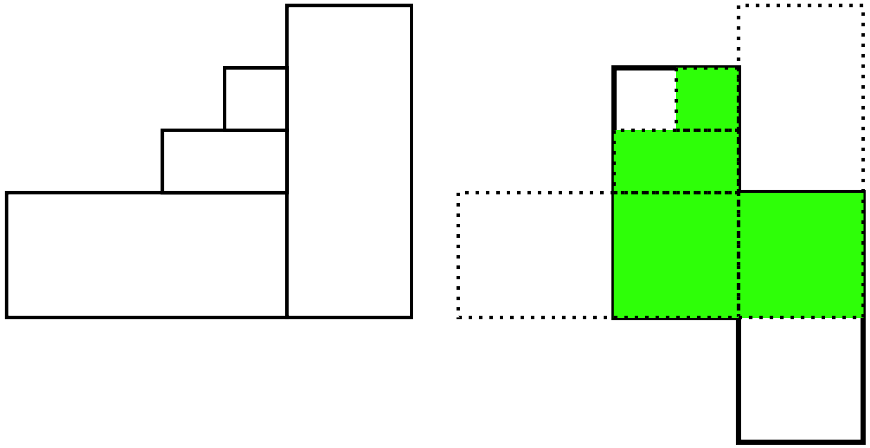
Regridding

Compute “error” at each cell : if “too big” then flag cell for refinement

- Richardson extrapolation
  - Coarsen data on a patch at t^{n-1} and advance by $2\Delta t$
  - Advance data at t^n by Δt and coarsen
  - Difference of these two solutions is proportional to error
- Functions of solution (e.g., vorticity)
- Geometric considerations

Compute refined patches as initially

Fill newly refined regions using conservative interpolation from coarse grid



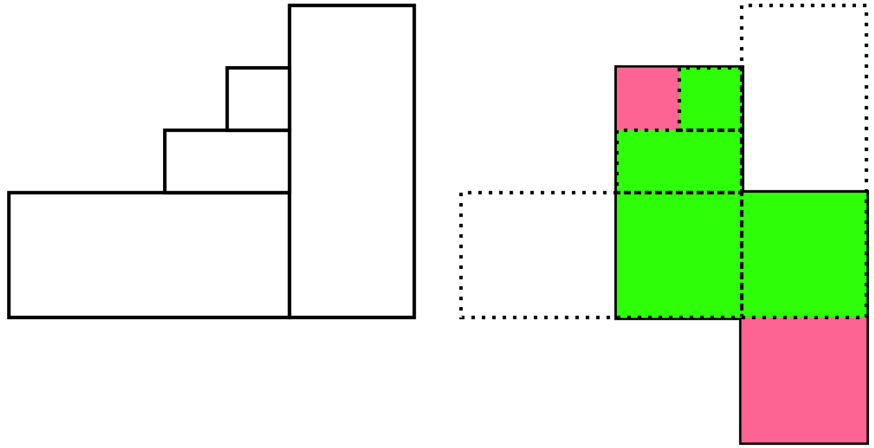
Regridding

Compute "error" at each cell : if "too big" then flag cell for refinement

- Richardson extrapolation
  - Coarsen data on a patch at t^{n-1} and advance by $2\Delta t$
  - Advance data at t^n by Δt and coarsen
  - Difference of these two solutions is proportional to error
- Functions of solution (e.g., vorticity)
- Geometric considerations

Compute refined patches as initially

Fill newly refined regions using conservative interpolation from coarse grid



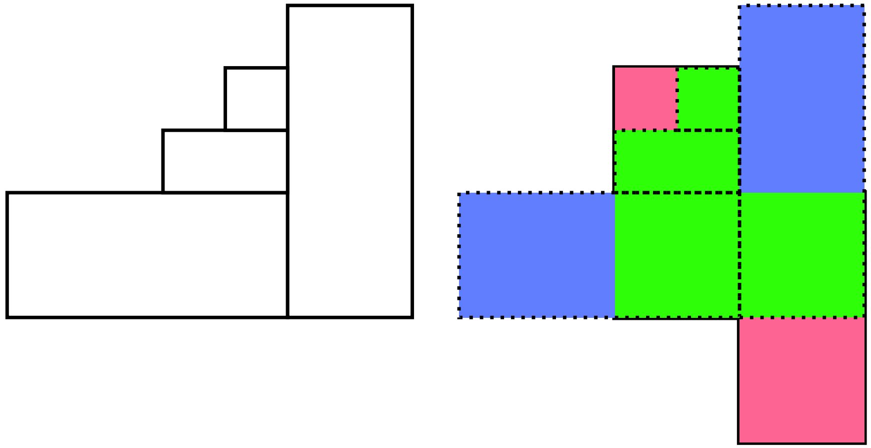
Regridding

Compute “error” at each cell : if “too big” then flag cell for refinement

- Richardson extrapolation
  - Coarsen data on a patch at t^{n-1} and advance by $2\Delta t$
  - Advance data at t^n by Δt and coarsen
  - Difference of these two solutions is proportional to error
- Functions of solution (e.g., vorticity)
- Geometric considerations

Compute refined patches as initially

Fill newly refined regions using conservative interpolation from coarse grid



Summary of Algorithm

Hyperbolic AMR

For $n = 1, \ldots, N_{final}$
  Advance$(0, t_0^n)$

Advance (ℓ, t)
  If (time to regrid) **then**
    Regrid(ℓ)

  FillPatch$(\ell$,t)

  Integrate$(\ell, t, \Delta t_\ell)$

  If $(\ell < \ell_{finest})$ **then**
    For $i_{sub} = 1, \ldots, r_\ell$
      Advance$(\ell{+}1,\ t + (i_{sub} - 1)\Delta t_{\ell+1})$
    Average down$(\ell, t + \Delta t_\ell)$
    Reflux$(\ell, t + \Delta t_\ell)$
  End If

Regrid(ℓ): generate new grids at levels $\ell+1$ and higher

FillPatch$(\ell$,t): fill patch of data at level ℓ and time t

Integrate$(\ell, t, \Delta t)$: Advance data at level ℓ from t to $t + \Delta t$, averaging and storing fluxes at boundaries of level ℓ grids if $\ell > 0$ and level ℓ cells at boundary of $\ell + 1$

Average down(ℓ, t): average (in space) level $\ell+1$ data at time t to level ℓ

Reflux(ℓ, t): Add (time- and space-averaged) refluxing corrections to level ℓ cells at time t adjacent to level $\ell+1$ grids

Review of Data Operations

Single-level operations

- Fill a patch with data from same-level grids
- Fill data using physical boundary conditions
- Interpolate data in time
- Add corrections from stored fluxes at same resolution
- Integrate patch of data in time
- Find union of rectangles that contain a set of tagged points

Multi-level operations

- Map regions between different levels of refinement
- Interpolate : coarse $\rightarrow$ fine
- Average : fine $\rightarrow$ coarse
- Store fluxes from fine grid boundaries at coarse resolution

Software / Parallel Implementation

Available software frameworks for hyperbolic AMR

- BoxLib (LBNL)
- Chombo (LBNL)
- AMRClaw (UW)
- SAMRAI (LLNL)
- PARAMESH (NASA)
- GRACE (Rutgers)

Data Structures

- Support for Index space operations
- Real data stored in `FORTRAN`-compatible form

Parallelization Strategy

- Data distribution
- Dynamic load balancing

Data Structures

Index space

- Box : a rectangular region in index space
- BoxArray : a union of Boxes at a level

Real data at a level

- FAB: `FORTRAN`-compatible data on a single box
  - Data on a patch
- MultiFAB: `FORTRAN`-compatible data on a union of rectangles
  - Data at a level
- FluxRegister: `FORTRAN`-compatible data on the border of a union of rectangles
  - Data for reflux correction (at coarse level resolution)

Data Operations

Box calculus in C++:

- compute intersection of patch with union of boxes
- compute coarsening or refinement of patch coordinates
- identify coarse grids underlying a given patch

Real data operations happen in FORTRAN on individual rectangular regions, either grids in the BoxArray or temporary patches:

- time integration of a patch of data
- interpolation/averaging of a patch of data
- filling of flux registers

Parallel Data Distribution

AMR hierarchy represented by a BoxArray and MultiFAB at each level

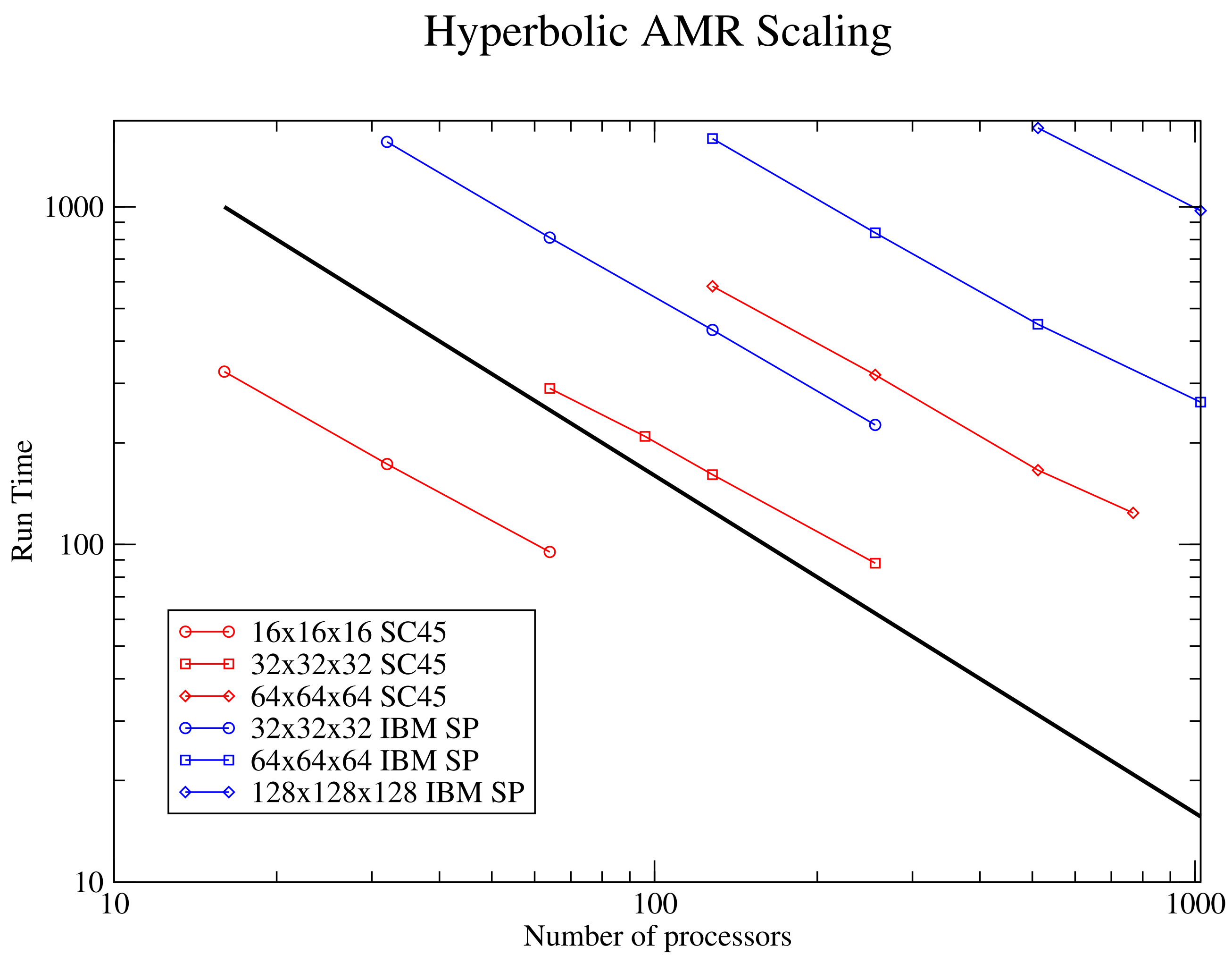
- Each processor contains the full BoxArray.
  - Simplifies data-communications: send-and-forget
- Data itself is distributed among processors; different resolutions are distributed independently, separately load-balanced.
- Owner computes rule on FAB data.
- Efficient implementation
  - Every MultiFAB with the same BoxArray has the same distribution
  - Each processor keeps list of its grids nearest neighbors and their processors
  - Each processor keeps list of coarse grids and their processors used to supply boundary conditions
  - Messages are ganged: no more than one message is ever exchanged between processors in an operation

Load Balance

AMR requires dynamic load-balancing.

- Point-wise uniform work load: work proportional to the grid volume.
- Dynamic programming approach based on the classic *Knapsack* problem.
- The LB problem as posed is NP, but a heuristic algorithm is provided that finds a good, if not optimal solution.
- Experience shows that good load-balance can be achieved with about three grids per processor.

Scaling Results for Hyperbolic Code

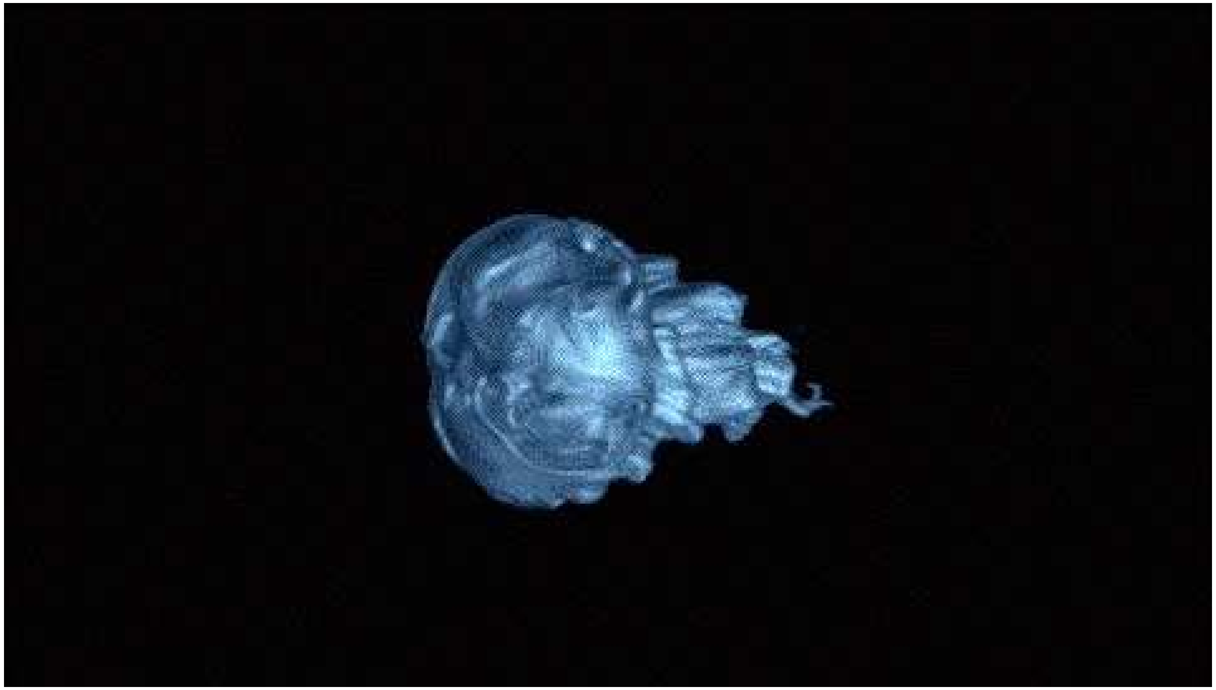


AMR using unsplit Godunov, 3 levels, factor of 4 refinement

Shock–bubble interaction

Mach 1.25 shock in air interacting with a helium bubble

- Domain $22.25\ cm\ \times 8.9\ cm\ \times 8.9\ cm$
- Base grid $80 \times 32 \times 32$
- 3 levels, $r_1 = 1$, $r_2 = 4$



Shock bubble