Block-Structured Adaptive Mesh Refinement Algorithms and Software

Phillip Colella
Lawrence Berkeley National Laboratory

Local Refinement for Partial Differential Equations

Variety of problems that exhibit multiscale behavior, in the form of localized large gradients separated by large regions where the solution is smooth.

- Shocks and interfaces.
- Self-gravitating flows in astrophysics.
- Complex engineering geometries.
- Combustion.
- Magnetohydrodynamics: space weather, magnetic fusion.

In adaptive methods, one adjusts the computational effort locally to maintain a uniform level of accuracy throughout the problem domain.

Adaptive Mesh Refinement (AMR)

Modified equation analysis: finite difference solutions to partial differential equations behave like solutions to the original equations with a modified right-hand side.

For linear steady-state problems $LU = f$:

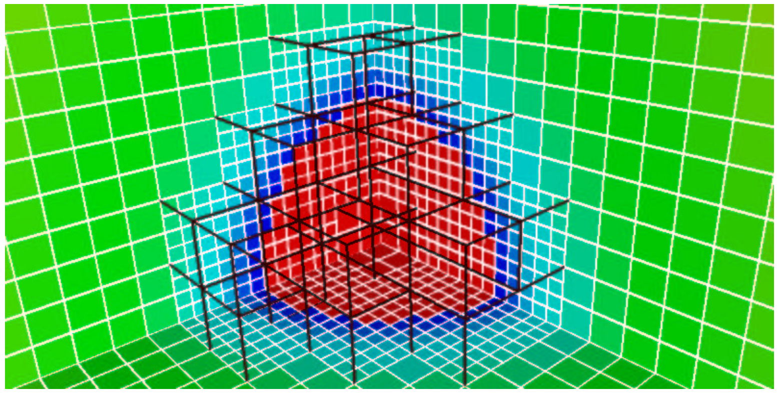
$$\epsilon = U^h - U \quad , \quad \epsilon \approx L^{-1}\tau$$

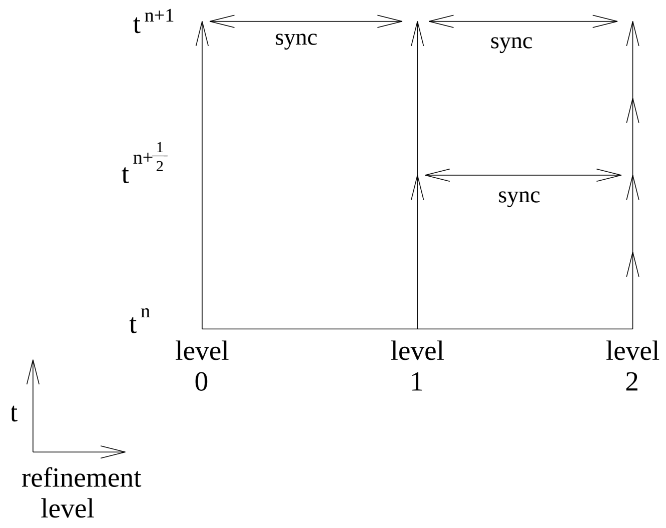
For nonlinear, time=dependent problems

$$\frac{\partial U}{\partial t} + L(U) = 0 \Rightarrow \frac{\partial U^h}{\partial t} + L(U^h) = \tau$$

In both cases, The truncation error $\tau = \tau(U) = (\Delta x)^p M(U)$, where M is a $(p+q)$-order differential operator.

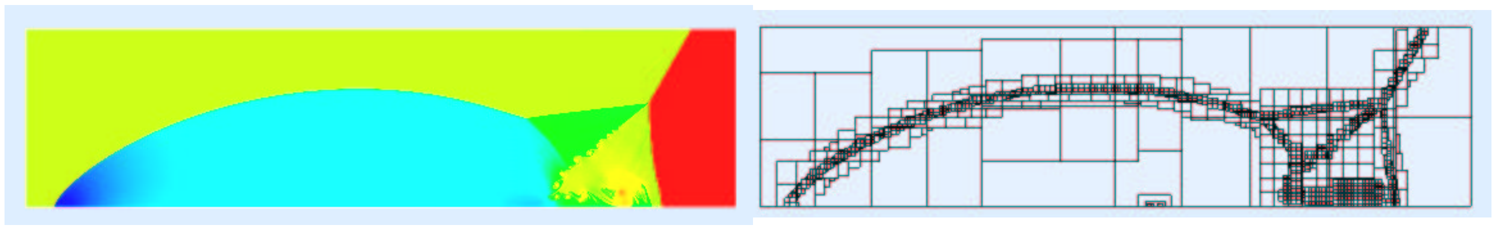
Block-Structured Local Refinement (Berger and Oliger, 1984)





Refined regions are organized into rectangular patches

Refinement performed in time as well as in space.



AMR Design Issues

Accuracy: effect of truncation error on solution error.

- How does one estimate the error?
- Failure of error cancellation can lead to large truncation errors at boundaries between refinement levels.

Stability: boundary conditions at refinement boundaries.

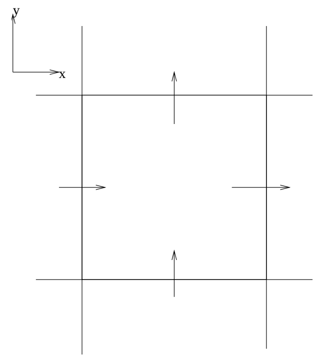
- Well-posedness of initial-boundary value problem.
- Refinement in time.
- Conservation and free-stream preservation.

The principal difficulty in designing block-structured AMR methods is determining the relationship between the numerical algorithms and the well-posedness of the free boundary-value problem for the underlying PDEs.

Conservation Form

Our spatial discretizations are based on conservative finite difference approximations.

$$\nabla \cdot \vec{F} \approx D(\vec{F}) \equiv \frac{1}{\Delta x} \sum_{s=1}^{d} \left(F^s_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^s} - F^s_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^s} \right)$$



Such methods satisfy a discrete form of the divergence theorem:

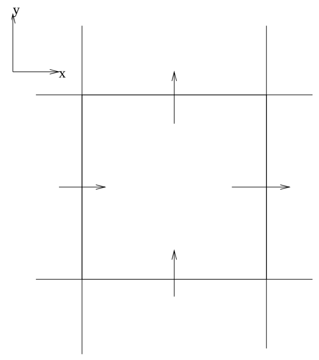
$$\sum_{\boldsymbol{j} \in \Omega} D(F)_{\boldsymbol{j}} = \frac{1}{\Delta x} \sum_{\boldsymbol{j}+\frac{1}{2}\boldsymbol{e}^s \in \partial\Omega} \pm F^s_{\boldsymbol{j}+\frac{1}{2}\boldsymbol{e}^s}$$

This class of discretizations essential for discontinuities, desirable for a large class of engineering applications.

AMR for Hyperbolic Conservation Laws (Berger and Colella, 1989)

We assume that the underlying uniform-grid method is an explicit conservative difference method.

$$U^{new} := U^{old} - \Delta t(D\vec{F}) \ , \ \vec{F} = \vec{F}(U^{old})$$



On a two-level AMR grid, we have U^c, U^f, and the update is performed in the following steps.

- Update solution on entire coarse grid: $U^c := U^c - \Delta t^c D^c \vec{F}^c$.

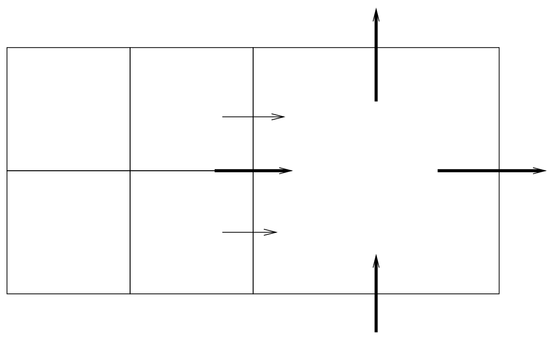
- Update solution on entire fine grid: $U^f := U^f - \Delta t^f D^f \vec{F}^f$ (n_{refine} times).

- Synchronize coarse and fine grid solutions.

Synchronization of Multilevel Solution

• Average coarse-grid solution onto fine grid.

• Correct coarse cells adjacent to fine grid to maintain conservation.



$$U^c := U^c + \frac{\Delta t^c}{h}\left(F^{c,s}_{\boldsymbol{i}^c - \frac{1}{2}\boldsymbol{e}} - \frac{1}{Z}\sum_{\boldsymbol{i}^f} F^{f,s}_{\boldsymbol{i}^f - \frac{1}{2}\boldsymbol{e}}\right)$$

Typically, need a generalization of GKS theory for free boundary problem to guarantee stability (Berger, 1985). Stability not a problem for upwind methods.

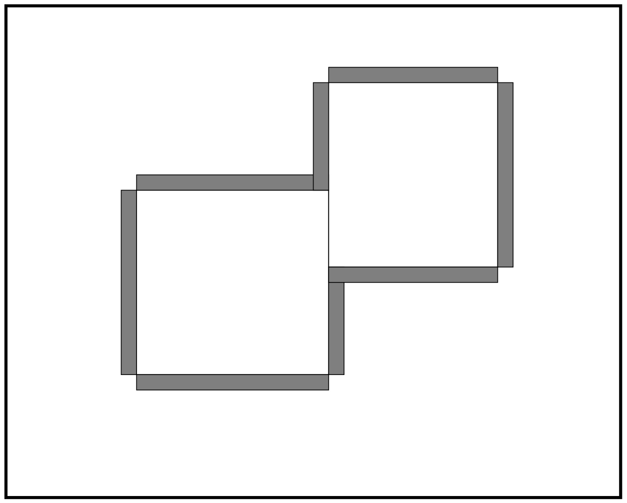
Discretizing Elliptic PDE's

Naive approach:

• Solve $\Delta\psi^c = g^c$ on coarse grid.

• Solve $\Delta\psi^f = g^f$ on fine grid, using coarse grid values to interpolate boundary conditions.

Such an algorithm yields coarse-grid solution accuracy on the fine grid (Bai and Brandt, Thompson and Ferziger).



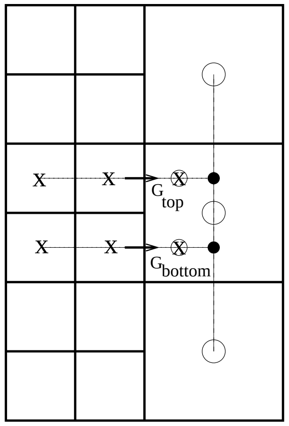
$\psi^c \approx \psi^{exact} + \Delta^{-1}\tau^c$. Using ψ^c to interpolate boundary conditions for fine calculation introduces coarse-grid error on fine grid.

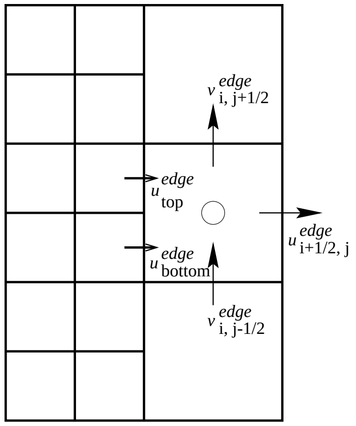
Solution: compute ψ^{comp}, the solution of the properly-posed problem on the composite grid.

$$\Delta^c \psi^{comp} = g^c \ \text{ on } \ \Omega^c - \mathcal{C}(\Omega^f)$$
$$\Delta^f \psi^{comp} = g^f \ \text{ on } \ \Omega^f$$

$$[\psi] = 0, \quad \left[\frac{\partial \psi}{\partial n}\right] = 0 \ \text{ on } \ \partial\Omega^{c/f}$$

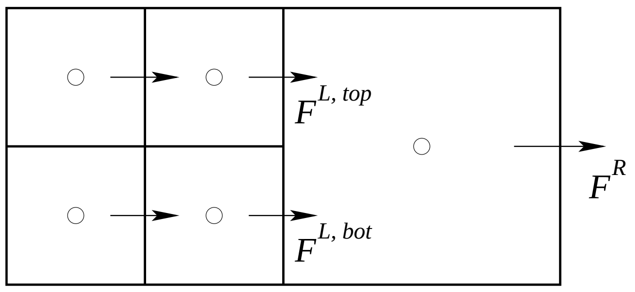
The Neumann matching conditions are flux matching conditions, and are discretized by computing a single-valued flux at the boundary. On the fine grid, one is still solving the same BVP as in the naive case, but with coarse grid values that have been modified to account for the Neumann matching conditions.





Modified equation: $\psi^{comp} = \psi^{exact} + \Delta^{-1}\tau^{comp}$, where τ is a *local* function of the solution derivatives.

Truncation Errors at Coarse-Fine Interfaces



$$\begin{aligned} D^*F =& \frac{1}{\Delta x}(F^R - \frac{1}{2}(F^{L,top} + F^{L,bot})) \\ =& \frac{1}{\Delta x}(F((i+\frac{1}{2})\Delta x, \overline{y}) - (F((i-\frac{1}{2})\Delta x, \overline{y}) + C(\Delta x)^2)) \\ =& \frac{\partial F}{\partial x} + O(\Delta x) \quad (\textbf{not } O(\Delta x^2)) \end{aligned}$$

For nonlinear time-dependent problems:

$$\begin{aligned} \frac{\partial U^{mod}}{\partial t} + \nabla \cdot F^{mod} \;=\; & \tau = O(\Delta x) \text{ at C/F boundary} \\ & = O(\Delta x^2) \text{ otherwise} \end{aligned}$$

For linear steady-state problems:

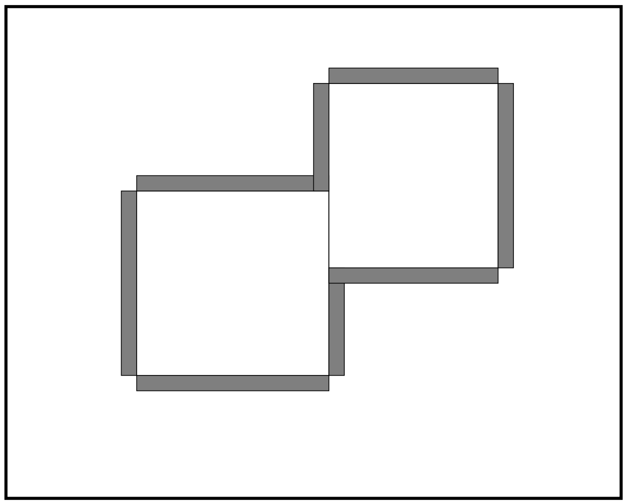
$$\psi^{computed} = \psi^{exact} + L^{-1}\tau$$

How does solution respond to singular forcing ?

Convergence Results for Poisson's equation ($\tau, \epsilon \sim h^P$)

D	Norm	Δx	$\|\|L(U_e) - \rho\|\|_h$	$\|\|L(U_e) - \rho\|\|_{2h}$	R	P
2	L_∞	1/32	1.57048e-02	2.80285e-02	1.78	0.84
2	L_∞	1/64	8.08953e-03	1.57048e-02	1.94	0.96
3	L_∞	1/16	2.72830e-02	5.60392e-02	2.05	1.04
3	L_∞	1/32	1.35965e-02	2.72830e-02	2.00	1.00
3	L_1	1/32	8.35122e-05	3.93200e-04	4.70	2.23

D	Norm	Δx	$\|\|U_h - U_e\|\|$	$\|\|U_{2h} - U_e\|\|$	R	P
2	L_∞	1/32	5.13610e-06	1.94903e-05	3.79	1.92
2	L_∞	1/64	1.28449e-06	5.13610e-06	3.99	2.00
3	L_∞	1/16	3.53146e-05	1.37142e-04	3.88	1.96
3	L_∞	1/32	8.88339e-06	3.53146e-05	3.97	1.99



$\psi^{comp} \approx \psi^{exact} + \Delta^{-1}\tau^{comp}$. $\tau = O(h)$ on a set with volume $O(h) \Rightarrow$ the contribution to the solution error is still $O(h^2)$.

Error Estimation and Grid Generation

To generate AMR grids, identify points needing refinement, and cover the parts of the grid so tagged with a union of rectangles.

Engineering Approach: use magnitude of some set of solution derivatives.

Mathematical Approach: compute Richardson estimate of truncation error.

• Truncation error is singular at refinement level boundaries $\rightarrow$ leads to spurious refinement (solution operator smooths effect of truncation error on the solution error).

• Ignoring truncation error at refinement level boundaries can lead to greater error in problems that have only a single scale.

• Possible approaches: weight truncation error at refinement level boundaries by surface-to-volume ratio; apply Richardson to fluxes; use estimates of higher derivatives corresponding to truncation error operator.

What does one do for multiphysics problems ?

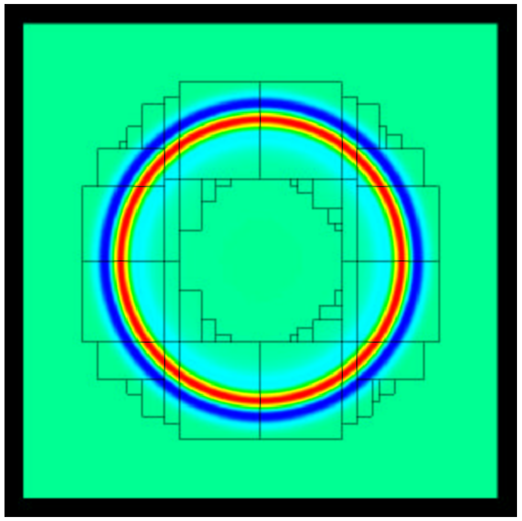
Wave Equation Solver

• Write second-order wave equation as first-order system in time.

$$\frac{\partial \varphi}{\partial t} = \pi$$
$$\frac{\partial \pi}{\partial t} = \Delta \varphi = \nabla \cdot \vec{F}$$

• Discretize Laplacian on AMR grid using RK4 in time, quadratic interpolation in space for coarse-fine boundary conditions.

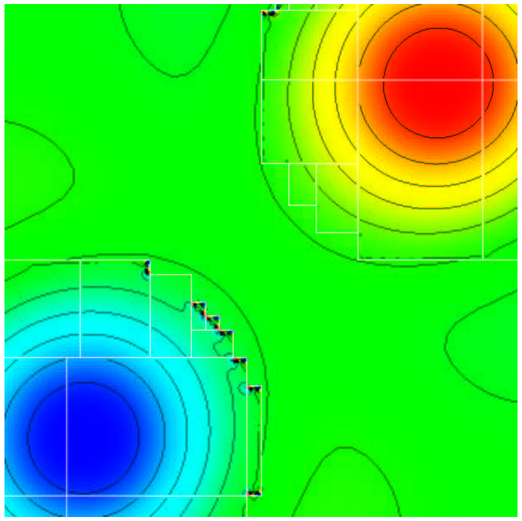
• Refinement in time: linear interpolation in time for coarse-fine boundary conditions, treat π as a conserved quantity for the purpose of refluxing.

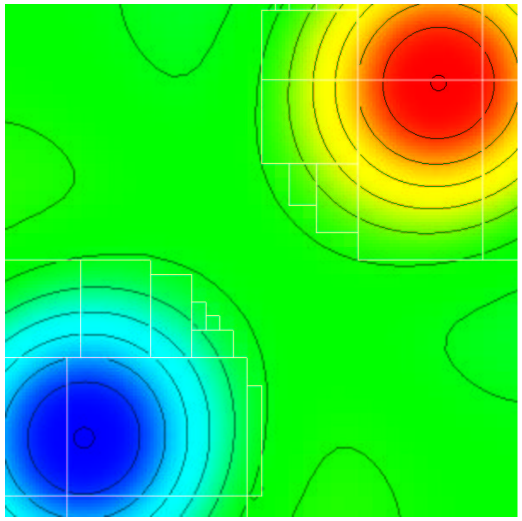


Time Discretization for Parabolic Problems

Example: time-dependent Landau-Ginzburg equation

$$\frac{\partial \phi}{\partial t} = \Delta \phi + f(\phi)$$





The marginal stability of the Crank-Nicolson method shown on the left suggests trying other more robust implicit Runge-Kutta methods that are second-order accurate and L_0 stable (Twizell, Gumel, and Arigu, 1996).

$$(I - r_1 \Delta)(I - r_2 \Delta) \phi^{n+1} = (I + a\Delta)\phi^n + \Delta t (I + r_4 \Delta) f^{n+\frac{1}{2}}$$

$$r_1 + r_2 + a = \Delta t \,, r_1 + r_2 + r_4 = \frac{\Delta t}{2} \,, r_1 + r_2 > \frac{\Delta t}{2}$$

Advection by an Incompressible Velocity Field

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0$$
$$\vec{u} = -k(\rho)\nabla\psi \quad , \quad \nabla \cdot \vec{u} = 0$$

AMR Hyperbolic algorithm:

• $\rho^c := \rho^c - \Delta t^c D(\rho\vec{u})$ on Ω^c;

• $\rho^f := \rho^f - \Delta t^f D(\rho\vec{u})$ on Ω^f (n_{refine} times);

• Synchronize coarse and fine solutions: average fine solution onto the coarse grid, and correct for conservation on coarse cells adjacent to fine grid.

What do we do for the elliptic PDE solution at the intermediate times ?
Freestream preservation: for the PDE, $\rho \equiv const. \Rightarrow \frac{\partial \rho}{\partial t} \equiv 0$. Can we preserve this at the discrete level ?

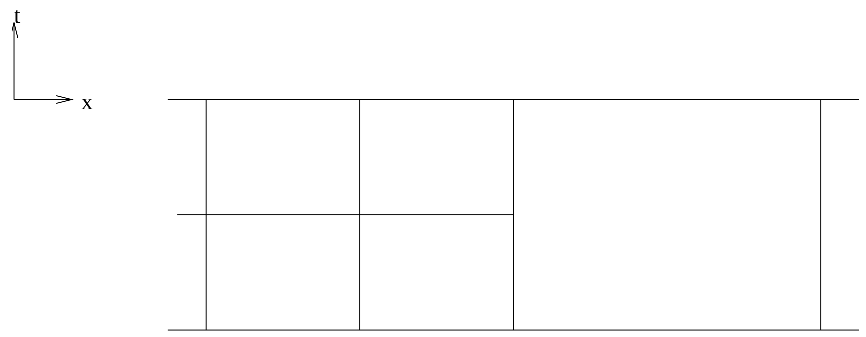
Boundary Conditions for Elliptic Solvers (Almgren et. al., 1998)

Idea: solve Poisson equation at a level using interpolated coarse boundary condition that includes lagged correction for the Neumann mismatch.

• At the beginning of the coarse time step, $\psi^c(t^c)$ is known. As part of the coarse time step, solve $-\nabla \cdot k(\rho)\nabla\psi^c$ on Ω^c. At that time, also solve the composite problem: $-\nabla \cdot k(\rho)\nabla\psi^{comp}$, and compute $\delta\psi = \psi^{comp} - \psi^c$.

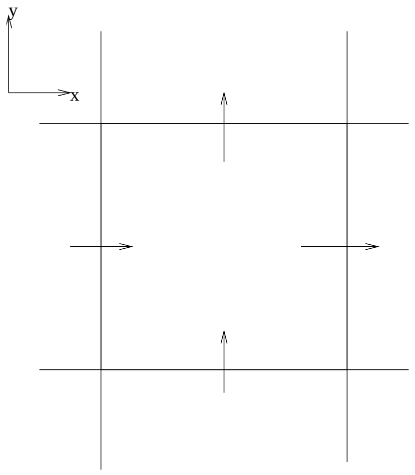
• At intermediate times t^{int}, solve

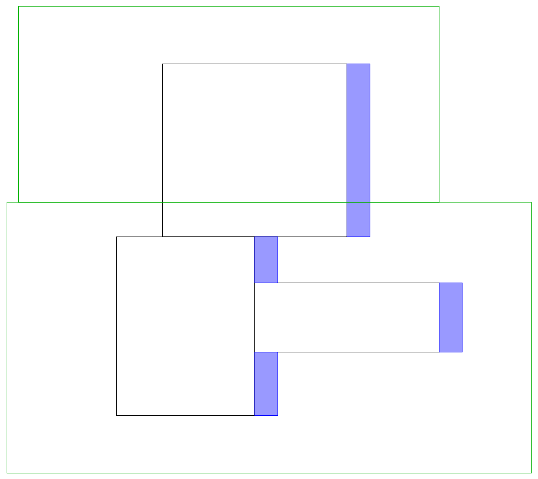
$$-\nabla \cdot k(\rho)\nabla\psi^f = 0 \text{ on } \Omega^f$$
$$\psi^f = I(\psi^c(t^{int}) + \delta\psi)$$



Freestream Preservation

$$\nabla \cdot (\rho \vec{u}) \approx D \cdot (\rho \vec{u})$$





Since $D(\vec{u}) = 0$, we want that $\rho = \rho_0 = const \Rightarrow D(\rho\vec{u}) = \rho_0 D(\vec{u}) = 0$. Away from coarse-fine boundaries, this condition can be guaranteed by our choice of discretization. At coarse-fine boundaries, this can fail due to refinement in time, with a coarse-fine correction that looks like

$$\rho_0 \left(u^{c,s}_{\boldsymbol{i}^c - \frac{1}{2}\boldsymbol{e}} - \frac{1}{Z} \sum_{\boldsymbol{i}^f} u^{f,s}_{\boldsymbol{i}^f - \frac{1}{2}\boldsymbol{e}} \right)$$

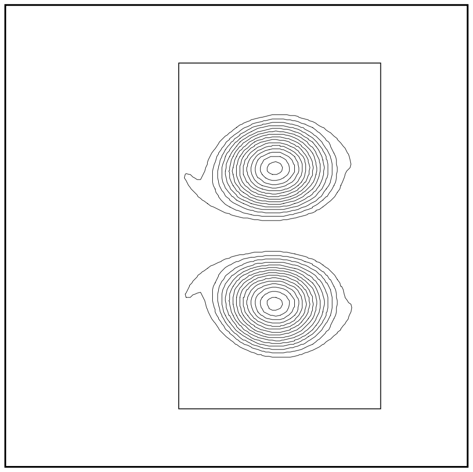
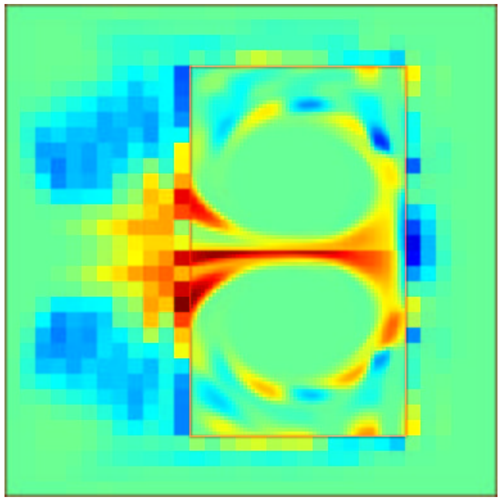
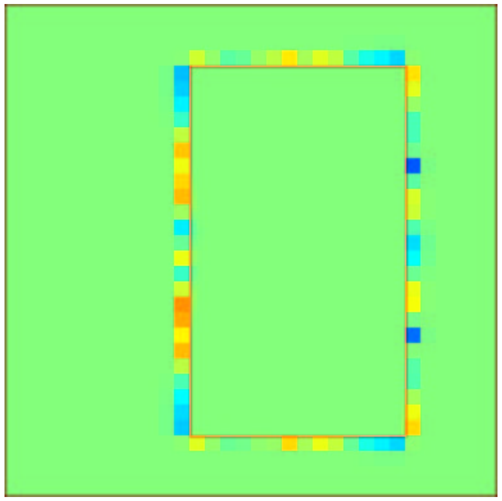
Volume Discrepancy Correction

We carry an additional conserved scalar that measures the integrated extent that we have compressed or expanded the fluid.

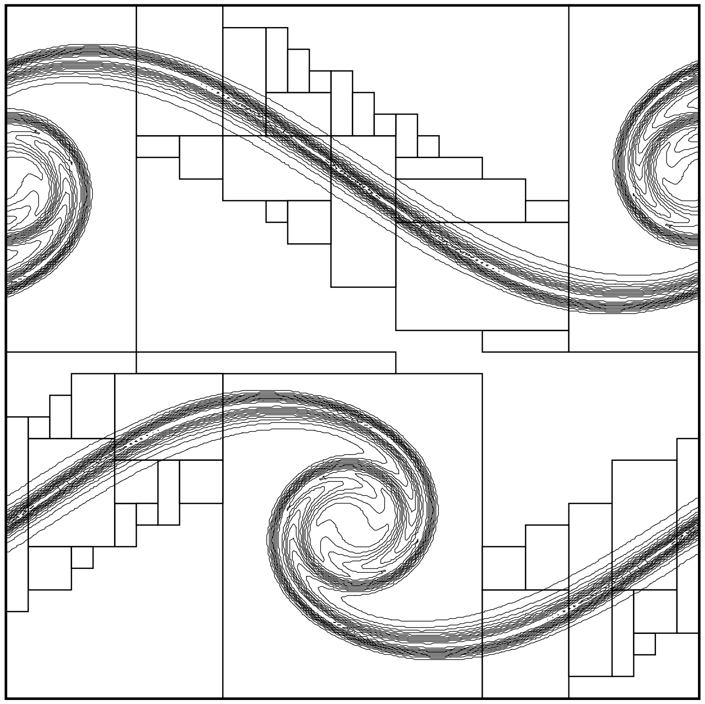
$$\frac{\partial \Lambda}{\partial t} + \nabla \cdot (\Lambda \vec{u}) = 0 \ \ , \ \ \Lambda(\boldsymbol{x}, 0) \equiv 1$$

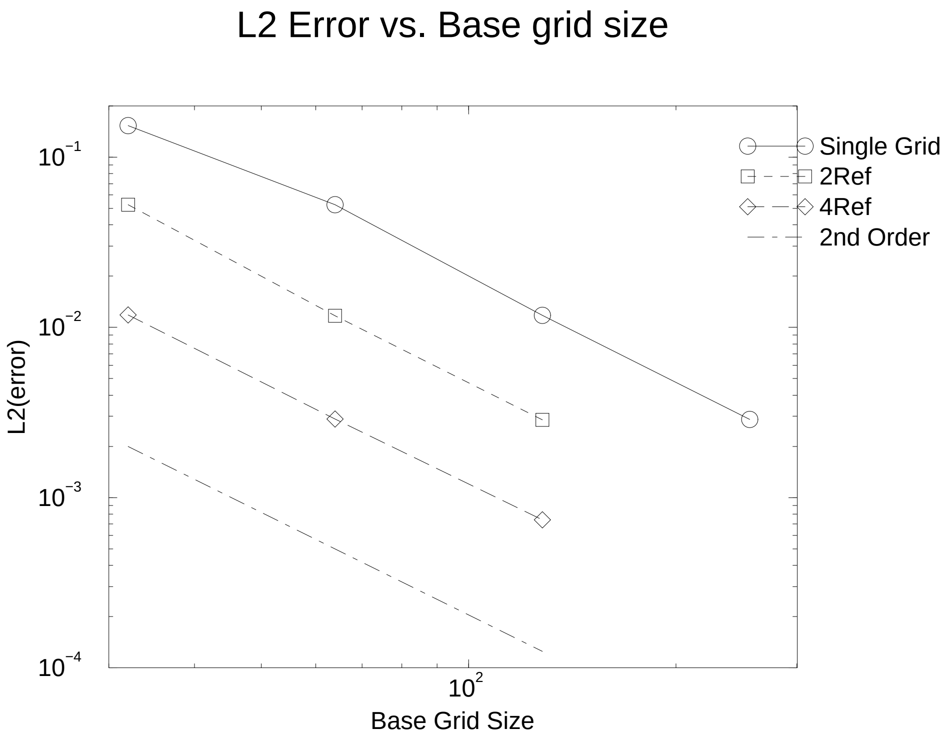
We then modify the right-hand side to the pressure equation to introduce local compressions / expansions to offset the failure of $\Lambda \equiv 1$

$$-\nabla \cdot k(\rho)\nabla\psi = \nabla \cdot \vec{u} = \eta(\Lambda - 1) \ \ , \ \ \eta = O\big(\frac{1}{\Delta t}\big)$$

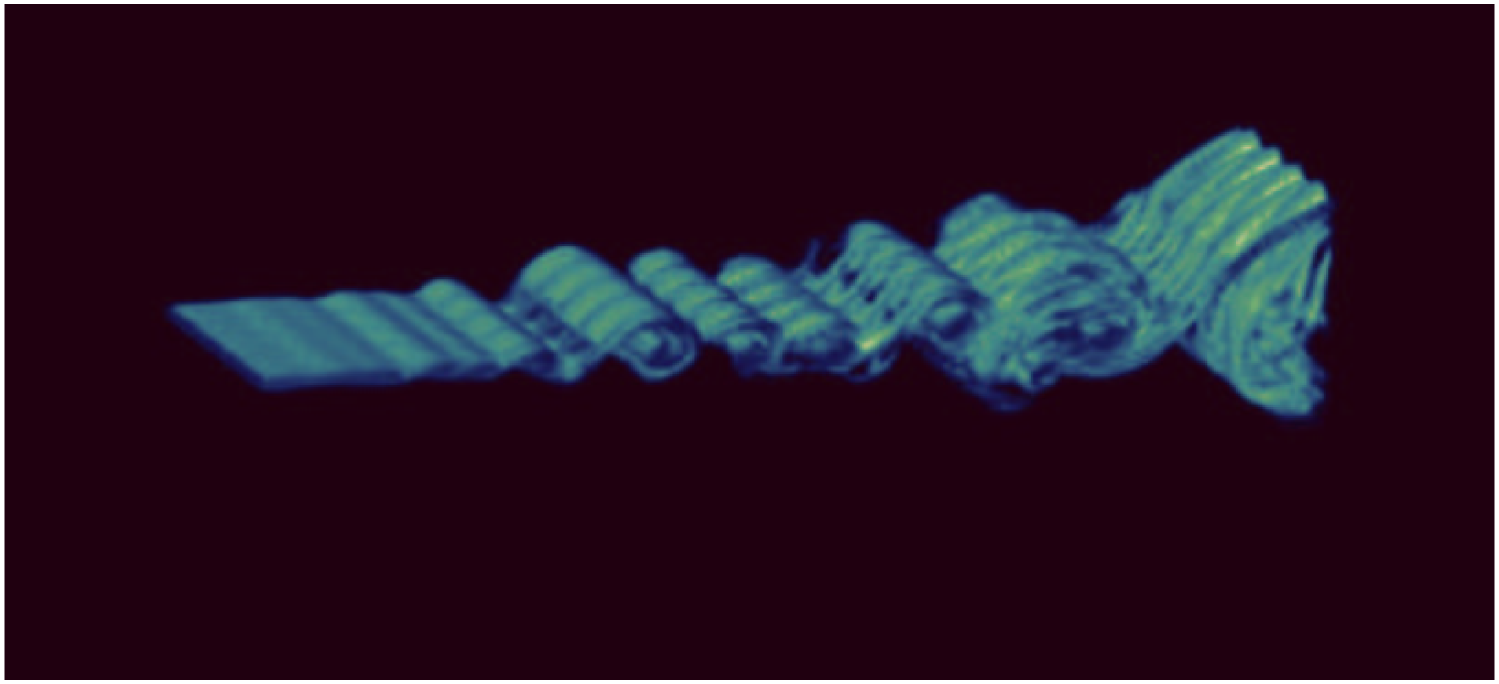




AMR Calculation of Inviscid Shear Layer (Martin and Colella, 2000)

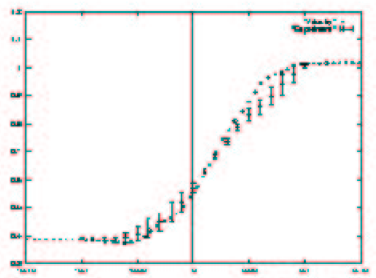




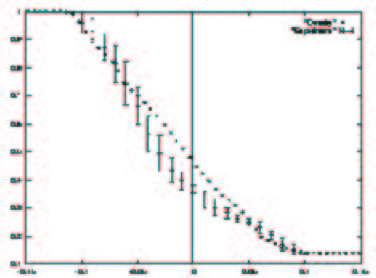
AMR Calculation of Brown-Roshko Shear Layer (Almgren, et. al., 1998)



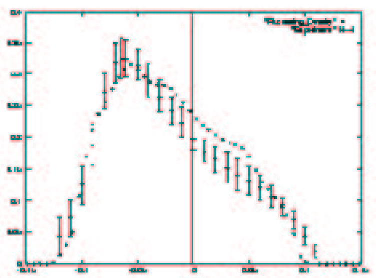
Comparison to Experimental Measurements



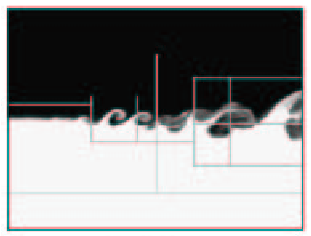
(a)



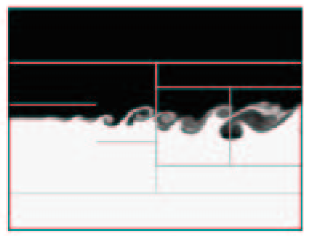
(b)



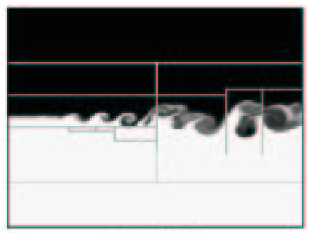
(c)



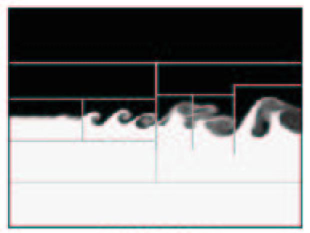
(a) t = 1585



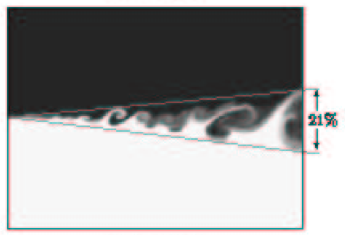
(b) t = 1630



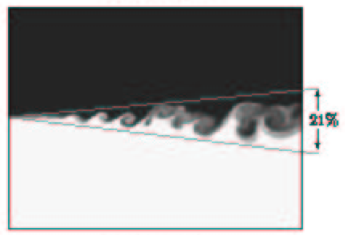
(c) t = 1673



(d) t = 1715



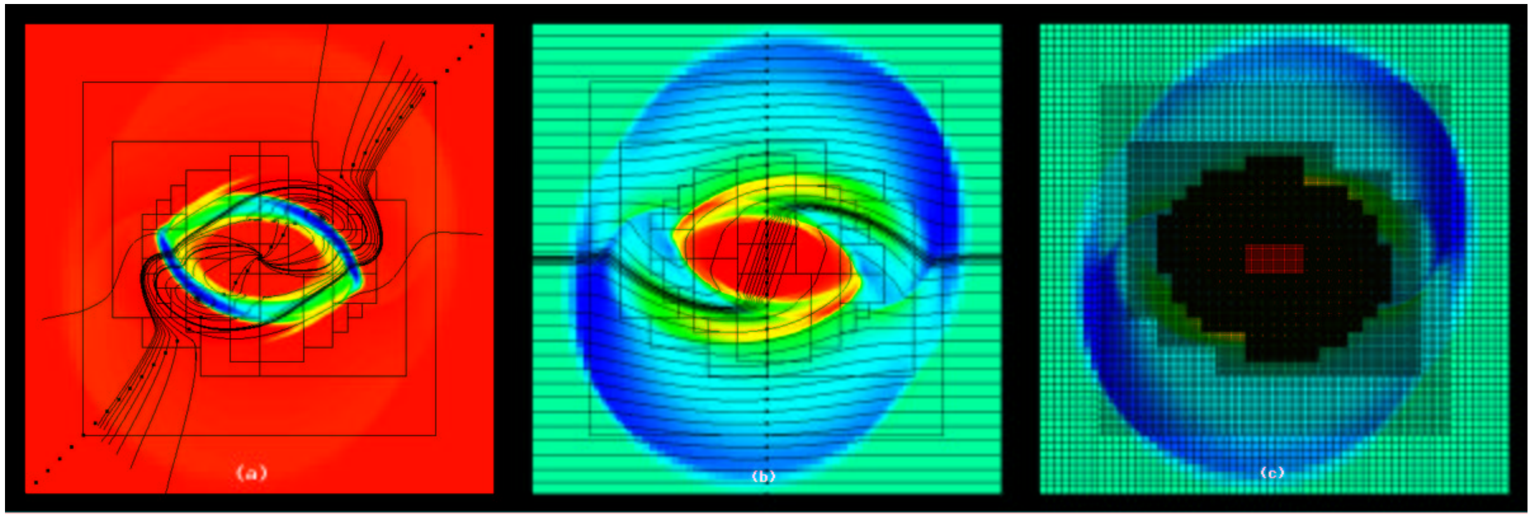
(e) t = 1585



(f) t = 1673

Magnetohydrodynamics (Samtaney, et. al., 2003)
Fluid representation: AMR for magnetohydrodynamics, based on semi-implicit methods.

- Explicit upwind discretizations for hyperbolic terms.
- Implicit discretizations for parabolic operators.
- Projection to enforce $\nabla \cdot \vec{B} = 0$ constraint.



AMR Stability and Accuracy Issues for Other Applications

• Low-Mach number combustion. Interaction of pressure constraint, inhomogeneous divergence constraint with refinement in time (Pember, et. al., 1998).

• S_n radiation, radiative heat transfer. Collection of steady-state, linear hyperbolic equations coupled by source terms. AMR / multigrid iteration scheme needs to respect upwinding. Radiation - matter coupling is stiff in optically thin regions, requires more implicit treatment of coarse-fine conservation (Jessee et. al., 1998; Howell et. al., 1999).

• Charged-fluid models of plasmas, semiconductor devices. Semi-implicit formulation in terms of classical PDE's leads to efficient and stable treatment of stiff dielectric relaxation time scale. Positivity-preservation for nonlinear elliptic systems (Colella, Dorr, Wake, 1999; Bettencourt, 1998).

• AMR for particle methods (PIC, PPPM). Relationship between the particle distribution and the refinement criterion (Almgren, Buttke, and Colella,1994).

• Mapped Grids. Coarse-fine stability for steady state problems. C^2 grid mappings lead to control volumes that depend on refinement level (Berger and Jameson, 1985; Bell, Colella, Tangenstein, Welcome, 1991; Dudek and Colella, 1999).

Chombo: a Software Framework for Block-Structured AMR

Requirement: to support a wide variety of applications that use block-structured AMR using a common software framework.

- Mixed-language model: C++ for higher-level data structures, Fortran for regular single-grid calculations.

- Reuseable components. Component design based on mapping of mathematical abstractions to classes.

- Build on public-domain standards: MPI, HDF5, VTK.

- Interoperability with other SciDAC ISIC tools: grid generation (TSTT), solvers (TOPS), performance analysis tools (PERC).

Previous work: BoxLib (LBNL/CCSE), KeLP (Baden, et. al., UCSD), FIDIL (Hilfinger and Colella).

Layered Design

- **Layer 1.** Data and operations on unions of boxes – set calculus, rectangular array library (with interface to Fortran), data on unions of rectangles, with SPMD parallelism implemented by distributing boxes over processors.

- **Layer 2.** Tools for managing interactions between different levels of refinement in an AMR calculation – interpolation, averaging operators, coarse-fine boundary conditions.

- **Layer 3.** Solver libraries – AMR-multigrid solvers, Berger-Oliger time-stepping.

- **Layer 4.** Complete parallel applications.

- **Utility layer.** Support, interoperability libraries – API for HDF5 I/O, visualization package implemented on top of VTK, C API's.

Examples of Layer 1 Classes (`BoxTools`)

• `IntVect` $\boldsymbol{i} \in \mathbb{Z}^d$. Can translate $\boldsymbol{i}_1 \pm \boldsymbol{i}_2$, coarsen $\frac{\boldsymbol{i}}{s}$, refine $\boldsymbol{i} * s$.

• `Box` $B \subset \mathbb{Z}^d$ is a rectangle: $B = [\boldsymbol{i}_{low}, \boldsymbol{i}_{high}]$. B can be translated, coarsened, refined. Supports different centerings (node-centered vs. cell-centered) in each coordinate direction.

• `IntVectSet` $\mathcal{I} \subset \mathbb{Z}^d$ is an arbitrary subset of $\mathbb{Z}^d$. $\mathcal{I}$ can be shifted, coarsened, refined. One can take unions and intersections, with other `IntVectSet`s and with `Box`es, and iterate over an `IntVectSet`.

• `FArrayBox A(Box B, int nComps)`: multidimensional arrays of Reals constructed with `B` specifying the range of indices in space, `nComp` the number of components. `Real* FArrayBox::dataPointer` returns pointer to the contiguous block of data that can be passed to Fortran.

Example: explicit heat equation solver on a single grid

```
// C++ code:

  Box domain(-IntVect:Unit,nx*IntVect:Unit);
  FArrayBox soln(grow(domain,1), 1);
  soln.setVal(1.0);

  for (int nstep = 0;nstep < 100; nstep++)
 {
  heatsub2d_(soln.dataPtr(0),
           &(soln.loVect()[0]), &(soln.hiVect()[0]),
           &(soln.loVect()[1]), &(soln.hiVect()[1]),
             region.loVect(), region.hiVect(),
             &dt, &dx, &nu);
 }
```

```
c Fortran code:
      subroutine heatsub2d(phi,nlphi0, nhphi0,nlphi1, nhphi1,
     &     nlreg, nhreg, dt, dx, nu)

      real*8 lphi(nlphi0:nhphi0,nlphi1:nhphi1)
      real*8  phi(nlphi0:nhphi0,nlphi1:nhphi1)
      real*8 dt,dx,nu
      integer nlreg(2),nhreg(2)

c Remaining declarations, setting of boundary conditions goes here.
...
       do j = nlreg(2), nhreg(2)
         do i = nlreg(1), nhreg(1)
            lapphi =
     &           (phi(i+1,j)+phi(i,j+1)
     &           +phi(i-1,j)+phi(i,j-1)
     &           -4.0d0*phi(i,j))/(dx*dx)

            lphi(i,j) = lapphi
         enddo
       enddo
```

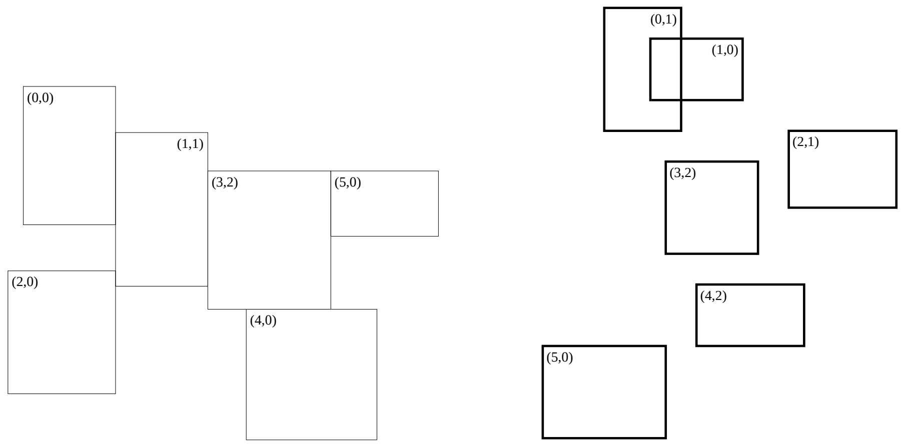
```
c Increment solution with rhs.

      do j = nlreg(2), nhreg(2)
         do i = nlreg(1), nhreg(1)
            phi(i,j) = phi(i,j) + nu*dt*lphi(i,j)
         enddo
      enddo

      return
      end
```

Distributed Data on Unions of Rectangles

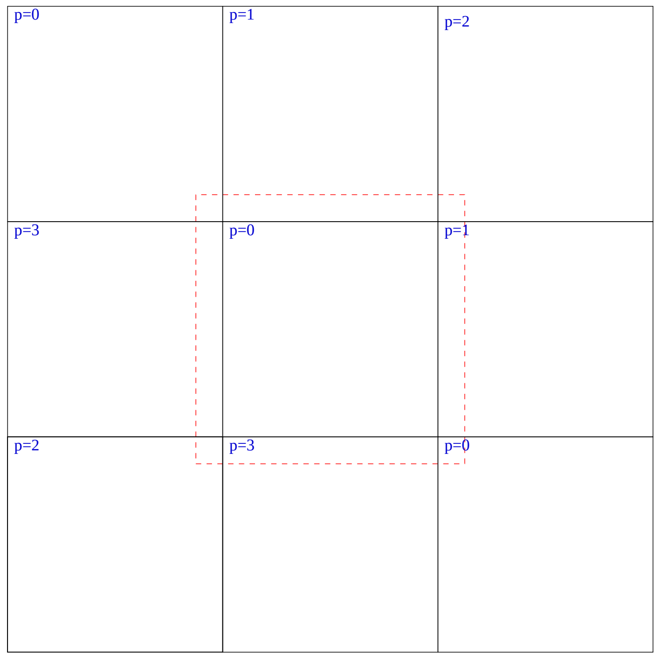
Provides a general mechanism for distributing data defined on unions of rectangles onto processors, and communications between processors.



• Metadata of which all processors have a copy: `BoxLayout` is a collection of `Box`es and processor assignments: $\{B_k, p_k\}_{k=1}^{nGrids}$. `DisjointBoxLayout:public BoxLayout` is a `BoxLayout` for which the `Box`es must be disjoint.

• `template <class T> LevelData<T>` and other container classes hold data distributed over multiple processors. For each `k = 1 ... nGrids`, an "array" of type `T` corresponding to the box B_k is allocated on processor p_k. Straightforward API's for copying, exchanging ghost cell data, iterating over the arrays on your processor in a SPMD manner.

Example: explicit heat equation solver, parallel case



Want to apply the same algorithm as before, except that the data for the domain is decomposed into pieces and distributed to processors.

• `LevelData<T>::exchange()`: obtains ghost cell data from valid regions on other patches.

• `DataIterator`: iterates over only the patches that are owned on the current processor.

```
// C++ code:
  Box domain;
  DisjointBoxLayout dbl;
// Break domain into blocks, and construct the DisjointBoxLayout.
  makeGrids(domain,dbl,nx);

  LevelData<FArrayBox> phi(dbl, 1, IntVect::TheUnitVector());

  for (int nstep = 0;nstep < 100;nstep++)
  {
...
// Apply one time step of explicit heat solver: fill ghost cell value
// and apply the operator to data on each of the Boxes owned by this
// processor.

  phi.exchange();
  DataIterator dit = dbl.dataIterator();

// Iterator iterates only over those boxes that are on this processor
  for (dit.reset();dit.ok();++dit)
  {
  FArrayBox& soln = phi[dit()];
  Box& region = dbl[dit()];
  heatsub2d_(soln.dataPtr(0),
           &(soln.loVect()[0]), &(soln.hiVect()[0]),
           &(soln.loVect()[1]), &(soln.hiVect()[1]),
             region.loVect(), region.hiVect(),
             domain.loVect(), domain.hiVect(),
             &dt, &dx, &nu);
  }
  }
```

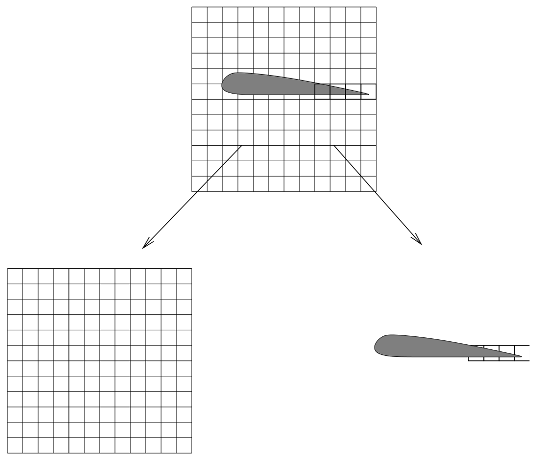
Software Reuse by Templating Dataholders

Classes can be parameterized by types, using the class template language feature in C++.

`BaseFAB<T>` is a multidimensional array whihc can be defined for for any type `T`. `FArrayBox:  public BaseFAB<Real>`

In `LevelData<T>`, `T` can be any type that ”looks like” a multidimensional array. Examples include:

- Ordinary multidimensional arrays, e.g. `LevelData<FArrayBox>`.
- A composite array type for supporting embedded boundary computations:



- Binsorted lists of particles, e.g. `BaseFab<List<ParticleType>>`

Layer 2: Coarse-Fine Interactions (`AMRTools`).
The operations that couple different levels of refinement are among the most difficult to implement AMR.

• Interpolating between levels (`FineInterp`).

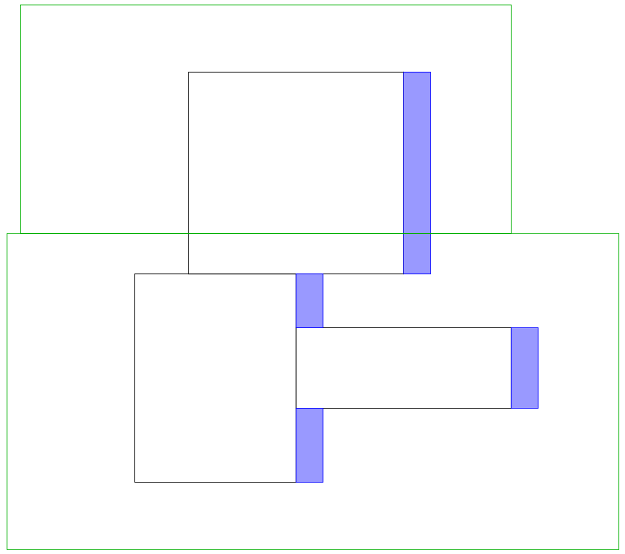
• Interpolation of boundary conditions (`PWLFillpatch, QuadCFInterp`).

• Averaging down onto coarser grids (`CoarseAverage`).

• Managing conservation at coarse-fine boundaries (`LevelFluxRegister`).

These operations typically involve interprocessor communication and irregular computation.

Example: `class LevelFluxRegister`



$$U^c := U^c + \frac{\Delta t^c}{h}(F^{c,s}_{\boldsymbol{i}^c - \frac{1}{2}\boldsymbol{e}} - \frac{1}{Z}\sum_{\boldsymbol{i}^f} F^{f,s}_{\boldsymbol{i}^f - \frac{1}{2}\boldsymbol{e}})$$

The coarse and fine fluxes are computed at different times in the program, and on different processors. We rewrite the process in the following steps:

$$\delta F = 0$$

$$\delta F := \delta F - \Delta t^c F^c$$

$$\delta F := \delta F + \Delta t^f < F^f >$$

$$U^c := U^c + D_R(\delta F)$$

A `LevelFluxRegister` object encapsulates these operations:

- `LevelFluxRegister::setToZero()`

- `LevelFluxRegister::incrementCoarse`: given a flux in a direction for one of the patches at the coarse level, increment the flux register for that direction.

- `LevelFluxRegister::incrementFine`: given a flux in a direction for one of the patches at the fine level, increment the flux register with the average of that flux onto the coarser level for that direction.

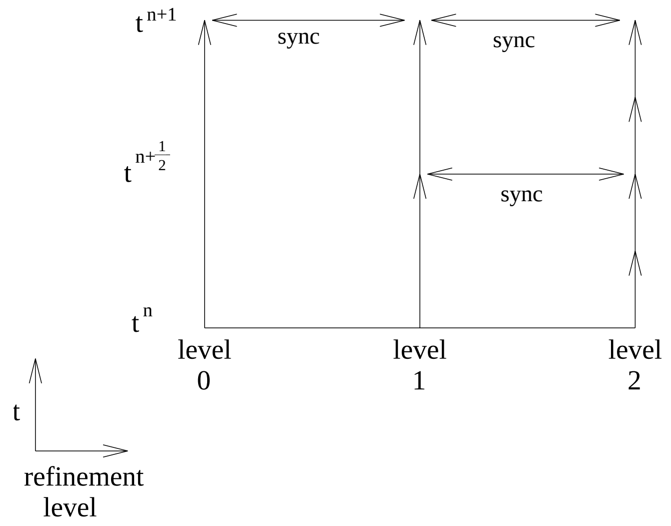
- `LevelFluxRegister::reflux`: given the data for the entire coarse level, increment the solution with the flux register data for all of the coordinate directions.

Layer 3: Reusing Control Structures Via Inheritance (`AMRTimeDependent`, `AMRElliptic`).
AMR has multilevel control structures are largely independent of the details of the operators and the data.

- Berger-Oliger refinement in time.
- Multigrid iteration on a union of rectangles.
- Multigrid iteration on an AMR hierarchy.

To separate the control structure from the details of the operations that are being controlled, we use C++ inheritance in the form of *interface classes*.

Example: `AMR / AMRLevel` **interface for Berger-Oliger timestepping**



We implement this control structure using a pair of classes.

`class AMR`: manages the Berger-Oliger time-stepping process.

`class AMRLevel`: collection of virtual functions called by an `AMR` object that perform the operations on the data at a level, e.g.:

- `virtual void AMRLevel::advance() = 0` advances the data at a level by one time step.
- `virtual void AMRLevel::postTimeStep() = 0` performs whatever sychronization operations required after all the finer levels have been updated.

`AMR` has as member data a collection of pointers to objects of type `AMRLevel`, one for each level of refinement:

```
Vector<AMRLevel*> m_amrlevels;
```

`AMR` calls the various member functions of `AMRLevel` as it advances the solution in time:

```
m_amrlevels[currentLevel]->advance();
```

The user implements a class derived from `AMRLevel` that contains all of the functions in `AMRLevel`:

```
class AMRLevelWaveEquation : public AMRLevel
// Defines functions in the interface, as well as data.
...
virtual void AMRLevelWaveEquation::advance()
{
// Advances the solution for one time step.
...
}
```

To use the `AMR` class for this particular application, `m_amrlevel[k]` will point to objects in the derived class, e.g.,

```
AMRLevelWaveEquation* amrLevelWavePtr = new AMRLevelWaveEquation(...);
m_amrlevel[k] = static_cast <AMRLevel*> (amrWavePtr);
```

AMR Utility Layer

• API for HDF5 I/O.

• Interoperability tools. We are developing a framework-neutral representation for pointers to AMR data, using opaque handles. This will allow us to wrap Chombo classes with a C interface and call them from other AMR applications.

• Chombo Fortran - a macro package for writing dimension-independent Fortran and managing the Fortran / C interface.

• `Parmparse` class from BoxLib for handling input files.

• Visualization and analysis tools (ChomboVis).

I/O Using HDF5

NSCA's HDF5 mimics the Unix file system.

- Disk file $\leftrightarrow$ "/".

- Group $\leftrightarrow$ subdirectory .

- Attribute, dataset $\leftrightarrow$ files. Attribute: small metadata that multiple processes in a SPMD program may write out redundantly. Dataset: large data, each processor writes only the data it owns.

Chombo API for HDF5

- Parallel neutral: can change processor layout when re-inputting output data.

- Dataset creation is expensive - one does not want to create one per rectangular grid. Instead, create one dataset for each `BoxLayoutData` or `LevelData`. Each grid's data is written into offsets from the origin of that dataset.

Load Balancing

For parallel performance, need to obtain approximately the same work load on each processor.

• Unequal-sized grids: knapsack algorithm provides good efficiencies provided the number of grids / processor ≥ 3 (Crutchfield, 1993). Disadvantage: does not preserve locality.

• Equal-sized grids can provide perfect load balancing if algorithm is reasonably homogeneous. Disadvantage: many small patches can lead to large amounts of redundant work.

Both methods obtain good scaling into 100's of nodes for hyperbolic problems. Alternative approach: space-filling curves using equal-sized grids, followed by agglomeration.

Spiral Design Approach to Software Development

Scientific software development is inherently high-risk: multiple experimental platforms, algorithmic uncertainties, performance requirements at the highest level. The Spiral Design Approach allows one to manage that risk, by allowing multiple passes at the software and providing a high degree of schedule visibilty.

Software components are developed in phases.

• Design and implement a basic framework for a given algorithm domain (EB, particles, etc.), implementing the tools required to develop a given class of applications.

• Implement one or more prototype applications as benchmarks.

• Use the benchmark codes as a basis for measuring performance and evaluating design space flexibility and robustness. Modify the framework as appropriate.

• The framework and applications are released, with user documentation, regression testing, and configuration for multiple platforms.

Components that have been released will be revisited as part of the design spiral for new capabilities being added, providing additional opportunities to improve the code.

Software Engineering Plan

• All software is open source:
`http://seesar.lbl.gov/anag/software.html`.

• Documentation: algorithm, software design documents; *Doc++ / Doxygen* manual generation; users' guides.

• Implementation discipline: CVS source code control, coding standards, TTPRO bug tracking system.

• Portability and robustness: flexible make-based build system, regression testing.

• Interoperability: C interfaces, opaque handles, permit interoperability across a variety of languages (C++, Fortran 77, Python, Fortran 90). Adaptors for large data items a serious issue, must be custom-designed for each application.

Current Research Areas

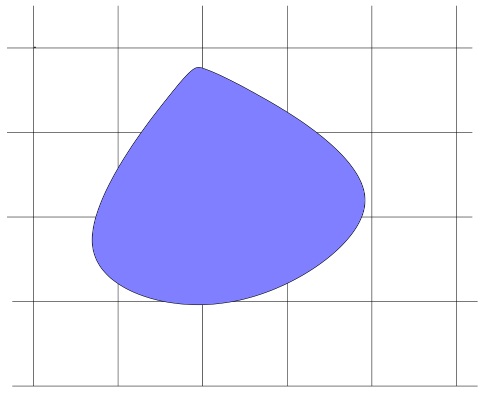
- Embedded boundary / cut-cell methods.
- Second-order systems: wave equations, General Relativity.
- Systems with gauge constraints: MHD, GR, solid mechanics written as a first-order system.
- Multiphase flows: particles, interfaces.

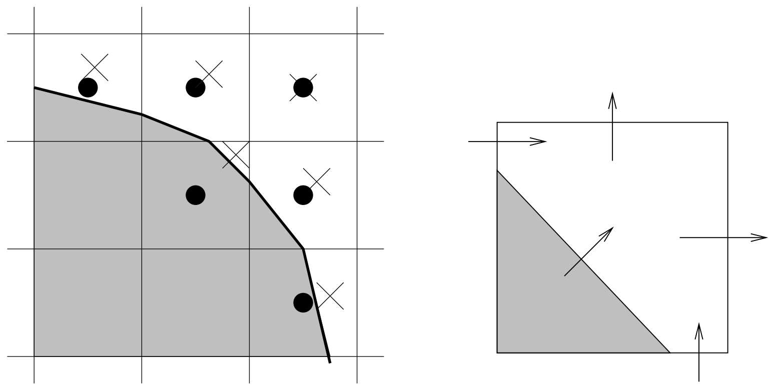
Open Problems

- Climate modeling: develop methods based on well-posed boundary-value formulation (i.e. non-hydrostatic), that are well-behaved for near-hydrostatic conditions.
- AMR for on high-order ($> 2^{nd}$ order) methods. Need to distinguish averages (cell, face), and point values; refinement in time.
- Analysis-based non-iterative domain decomposition methods for elliptic PDEs (Greengard and Lee, 1996; Balls and Colella, 2002).
- Hybrid methods: adaptive mesh and algorithm refinement (Garcia, Bell, Crutchfield, Alder, 1999).

Cartesian Grid Representation of Irregular Boundaries

Based on nodal-point representation (Shortley and Weller, 1938) or finite-volume representation (Noh, 1964).



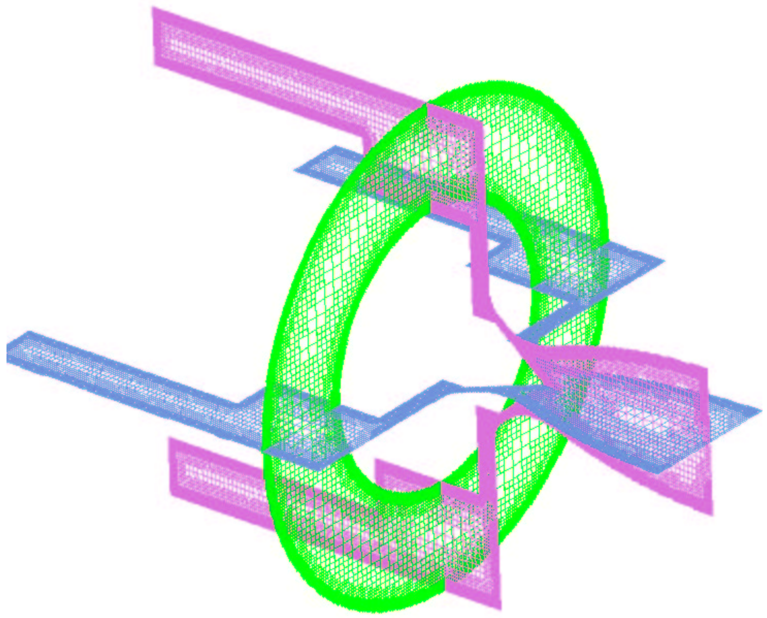
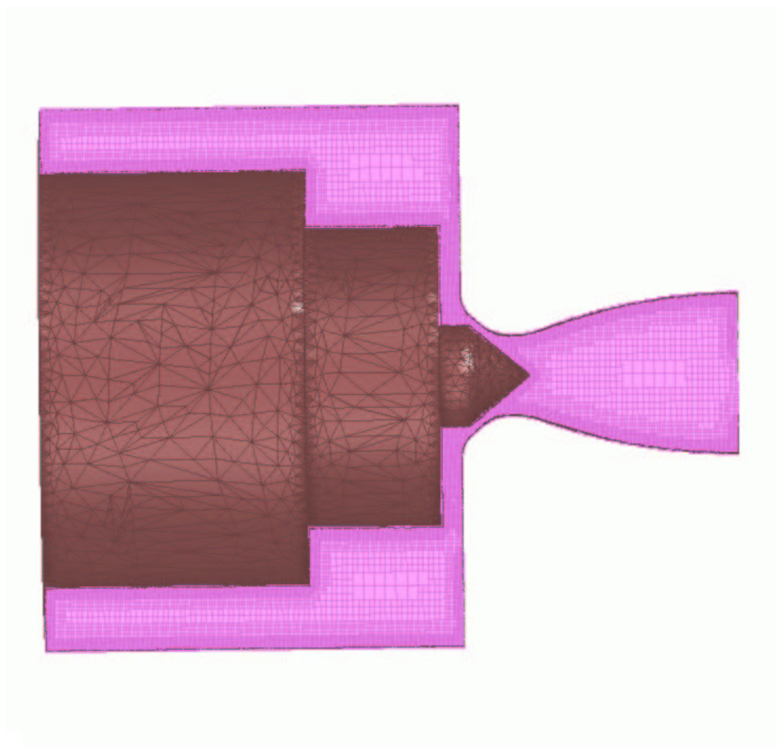
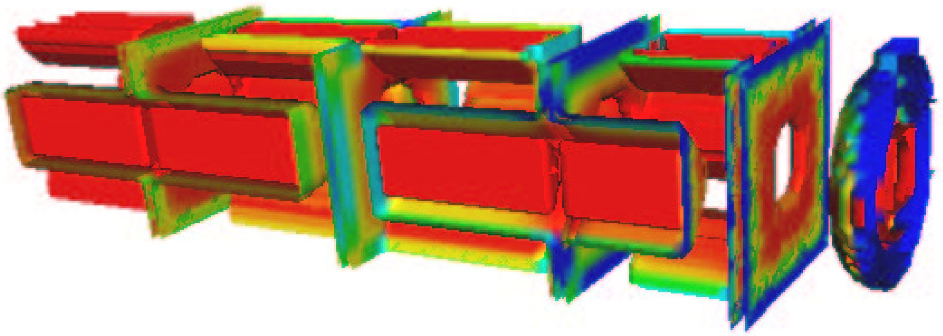
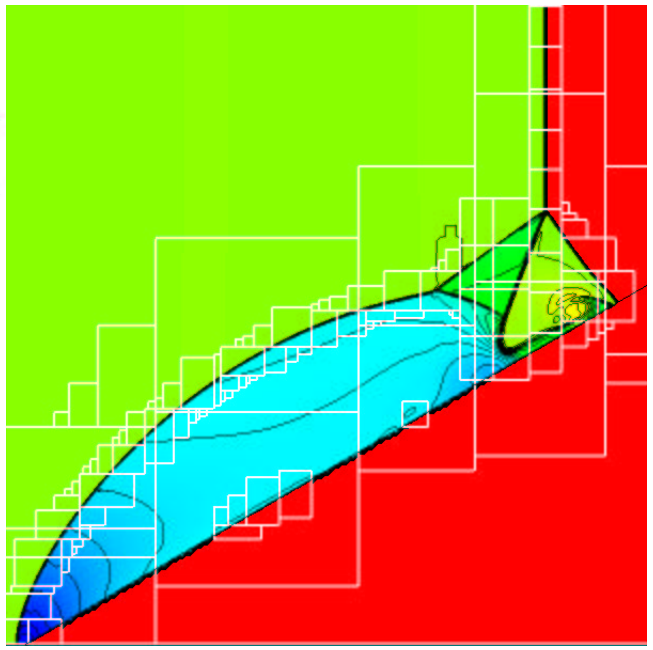


Advantages of underlying rectangular grid:

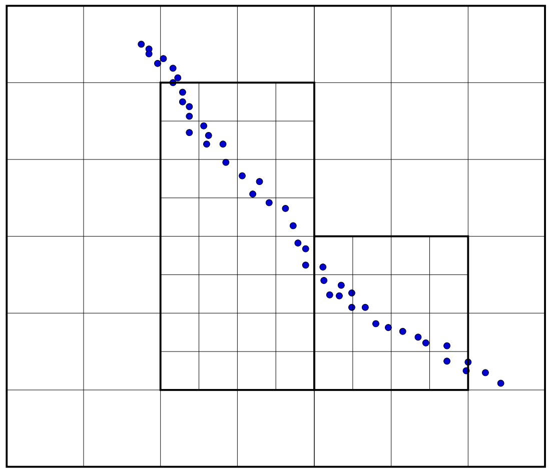
• Grid generation is tractable (Aftosmis, Berger, and Melton, 1998).

• Good discretization technology, e.g. well-understood consistency theory for finite differences, geometric multigrid for solvers.

• Straightforward coupling to structured AMR (Chern and Colella, 1987; Young et. al., 1990; Berger and Leveque, 1991).

Particles and AMR



Algorithmic Issues:

- Transfers of particles across coarse/fine interface boundaries
- Particle $\rightarrow$ grid and grid $\rightarrow$ particle transfers in the presence of refinement boundaries (modified stencils)
- preventing self-induced effects.

Acknowledgments

Funding:

• DOE HPCC, SciDAC Programs.

• NASA Earth and Space Sciences HPCC / Computational Technologies Program.

• DOE Applied Mathematical Sciences Program.

People:

• The Chombo development team: Dan Graves, Terry Ligocki, Noel Keen, Peter McCorquodale, Dan Martin, David Serafini, Peter Schwartz, Ted Sternberg, Brian van Straalen, (ANAG / LBNL); Ravi Samtaney (PPPL).

• Center for Computational Sciences and Engineering, LBNL.

• Collaborators: Robert Ryne, Alex Friedman, Dave Grote, Jean-Luc Vay, Julian Borrill, Adam Arkin (LBNL); Anders Petersson (LLNL); Steve Jardin (PPPL); Robert Crockett, Richard Klein, Chris McKee (UCB); Ben Keen (U. Michigan); Marsha Berger (NYU); Scott Baden, Greg Balls (UCSD).