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.

Finite difference equations

Want to solve numerically

$$L\phi = \frac{d^2\phi}{dx^2} = f \ , \ 0 \leq x \leq 1 \ , \ \phi(0) = \phi(1) = 0$$

Using Taylor expansions, we know that

$$\frac{d^2\phi}{dx^2} = \frac{\phi((i+1)h) - 2\phi(ih) + \phi((i-1)h)}{h^2} + O(h^2)$$

So we define a discrete solution

$$\psi_i \approx \phi(ih) \ , \ i \in [0, ..., N] \ , \ h = \frac{1}{N}$$

And a discrete operator

$$L^h\psi = \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{h^2}$$

And solve the finite-dimensional linear system

$$(L^h\psi)_i = f_i \ , \ f_i = f(ih)$$
$$i \in [1, ..., N-1] \ , \ \psi_0 = \psi_N = 0$$

How do we know that this works ?

Lax Theorem

Consistency:

$$\phi_i = \phi(ih) \ , \ L^h(\phi)_i - f_i \equiv \tau_i = O(h)^2$$

Stability:

$$||(L^h)^{-1}|| \text{ is bounded independent of } \ h$$

The error is the difference between the computed solution and the exact solution:

$$\epsilon = \phi - \psi$$

It satisfies the relationship

$$L^h \epsilon = \tau$$

Stability + consistency implies convergence:

$$||\epsilon|| \leq ||(L^h)^{-1}\tau|| \leq C||\tau|| = O(h^2)$$

Stability

Stability is hard: $(L^h)^{-1}\eta$ must be bounded for any input η even if it doesn't come from evaluating a smooth function at grid points.

Typically, we can prove stability for only the simplest (linear) cases. In this case, L^h is a symmetric positive definite matrix and can be diagonalized using a discrete Fourier transform. Solutions to the homogeneous problem satisfy a maximum principle.

For hard problems, stability is investigated by a combination of techniques: rigorous analysis of simplified model problems, asymptotic analysis (physical reasoning), and careful numerical experimentation.

For problems derived from classical PDE, this methodology works because stability can be split into two pieces: long-wavelength problems (which are smooth, and for which therefore the well-posedness of the original operator is relevant) and short-wavelength problems (which are local in space).

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 $LU = f$:

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

For nonlinear, time-dependent problems

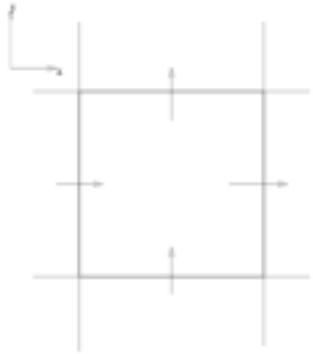
$$\frac{\partial U}{\partial t} + L(U) = 0 \Rightarrow \frac{\partial U^h}{\partial t} + L(U^h) = \tau$$

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

Conservation Form

Our spatial discretizations are based on conservative finite difference approximations.

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

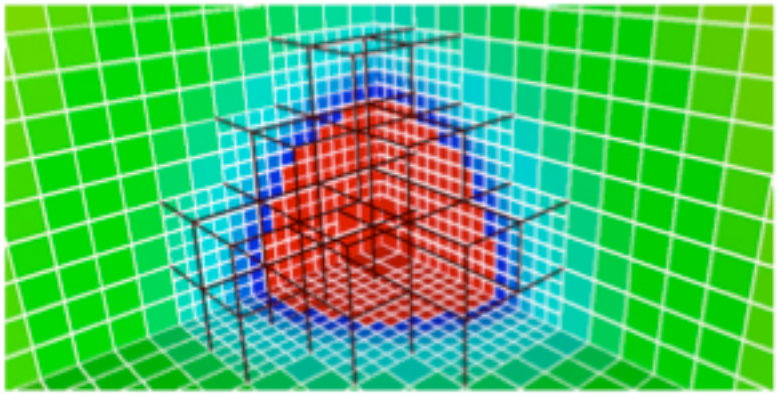


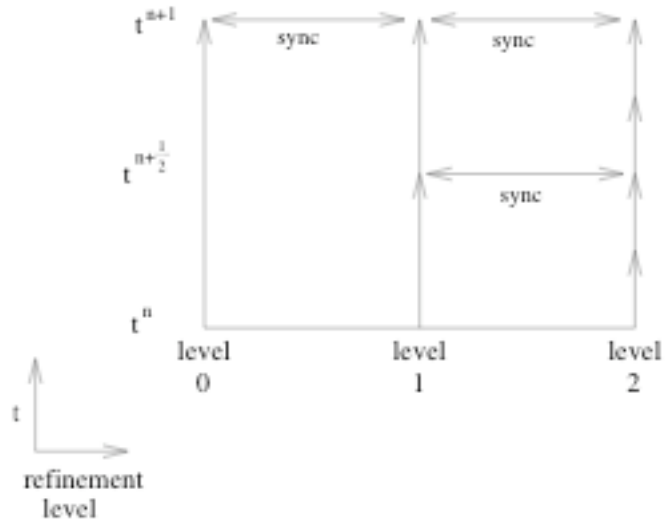
Such methods satisfy a discrete form of the divergence theorem:

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

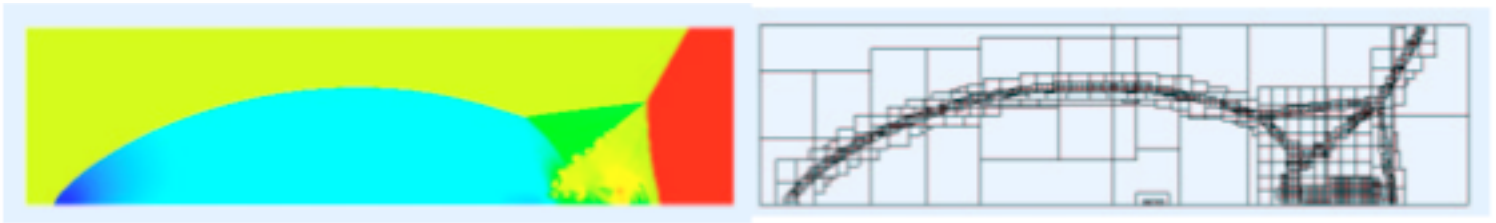
This class of discretizations is essential for discontinuities, and desirable for a large class of engineering applications.

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 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.

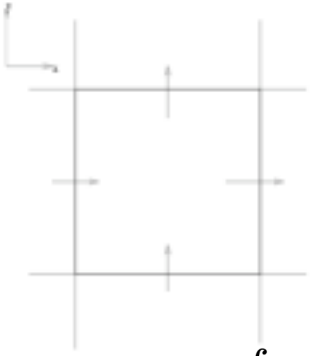
- Well-posedness of initial-boundary value problem.
- Refinement in time.
- Conservation and free-stream preservation

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.

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

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

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

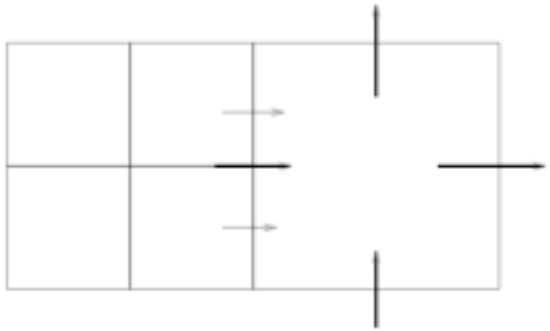


On a two-level AMR grid, we have U^f, U^c and the update is performed in the following steps.

- 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.



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

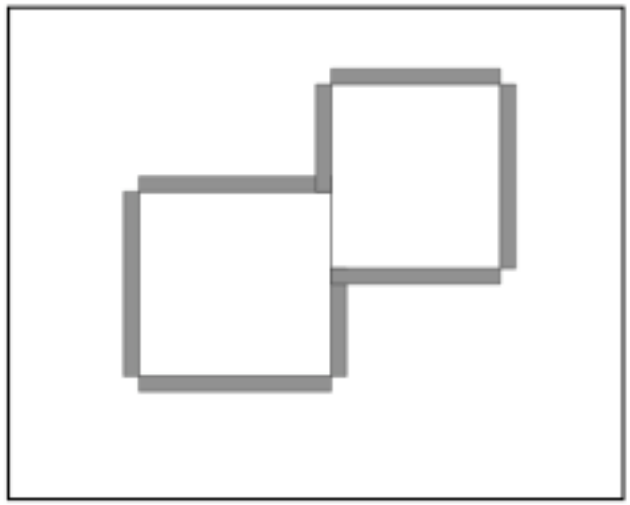
Typically, need a generalization of GKS theory for free boundary problem to guarantee stability (Berger, 1985). Stability not a problem for upwind methods.

Discretizing Elliptic PDE's

Naïve 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).

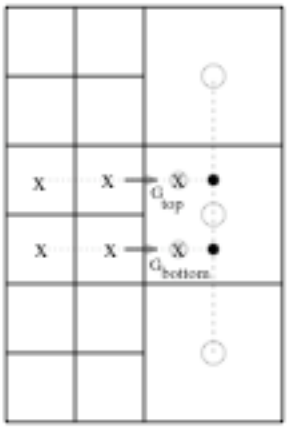


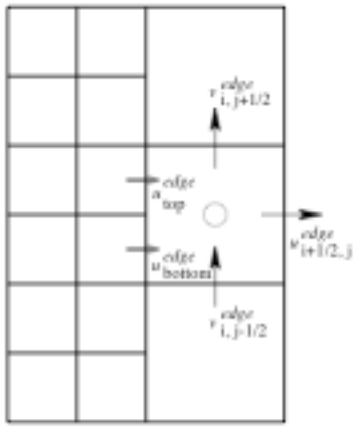
$\psi^c \approx \psi^{exact} + \Delta^{-1}\tau^c$. Using ψ^c to interpolate boundary conditions for fine calculation introduces coarse-grid error on fine grid.

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

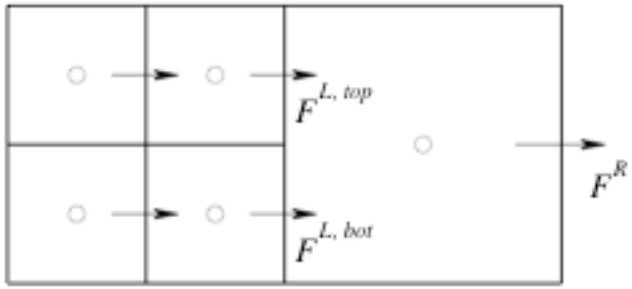
$$\Delta^c \psi^{comp} = g^c \text{ on } \quad \Omega^c - C(\Omega^f)$$
$$\Delta^f \psi^{comp} = g^f \text{ on } \Omega^f$$
$$[\psi] = 0, \quad [\frac{\partial \psi}{\partial n}] = 0 \text{ on } \partial\Omega^{c/f}$$

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 naïve case, but with coarse grid values that have been modified to account for the Neumann matching conditions.





Truncation Errors at Coarse-Fine Interfaces



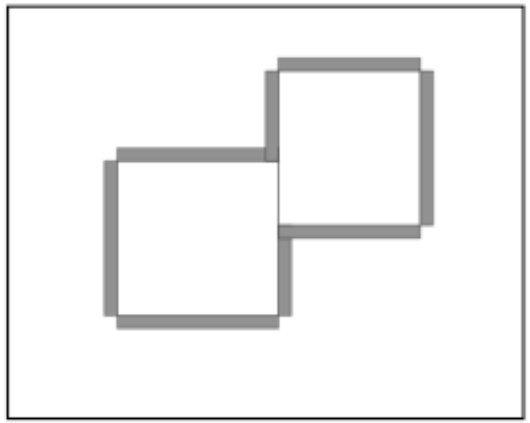
$$D^*F = \frac{1}{\Delta x}(F^R - \frac{1}{2}(F^{L,top} + F^{L,bot}))$$

$$= \frac{1}{\Delta x}(F((i+\frac{1}{2}\Delta x, \vec{y}) - (F((i-\frac{1}{2})\Delta x, \vec{y}) + C(\Delta x)^2))$$

$$= \frac{\partial F}{\partial x} + O(\Delta x) \text{ (not } O(\Delta x^2))$$

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

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



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

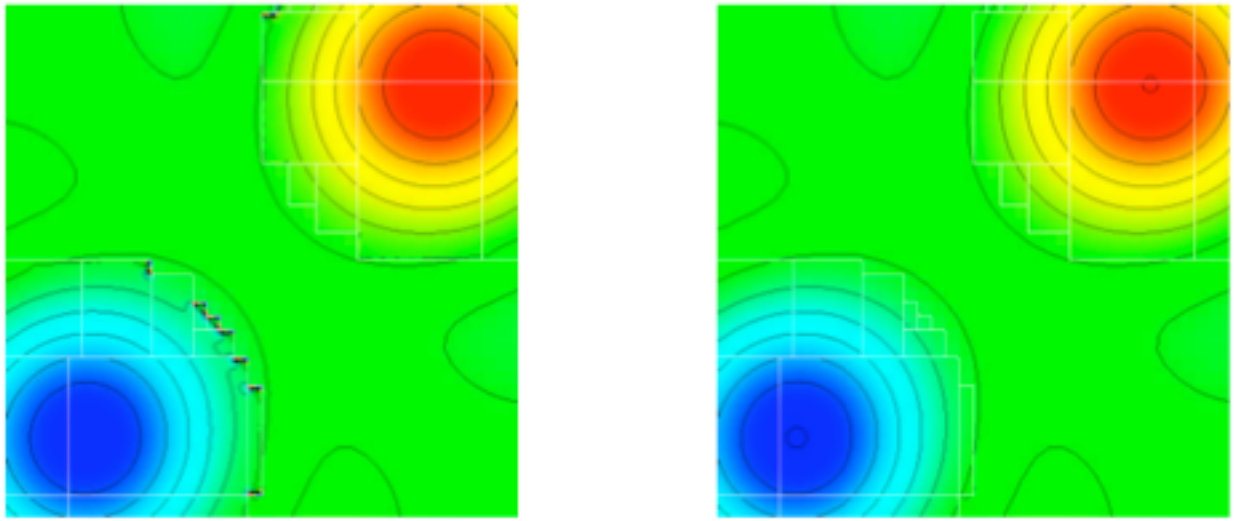
D	Norm	Δx	$\|\|L(U_e)-\rho\|\|_h$	$\|\|L(U_e)-\rho\|\|_{2h}$	R	P
2	L_∞	1/32	1.57048e-02	2.80285e-02	1.78	0.84
2	L_∞	1/64	8.08953e-03	1.57048e-02	1.94	0.96
3	L_∞	1/16	2.72830e-02	5.60392e-02	2.05	1.04
3	L_∞	1/32	1.35965e-02	2.72830e-02	2.00	1.00
3	L_1	1/32	8.35122e-05	3.93200e-04	4.70	2.23

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

Time Discretization for Parabolic Problems

Example: time-dependent Landau-Ginzburg equation

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

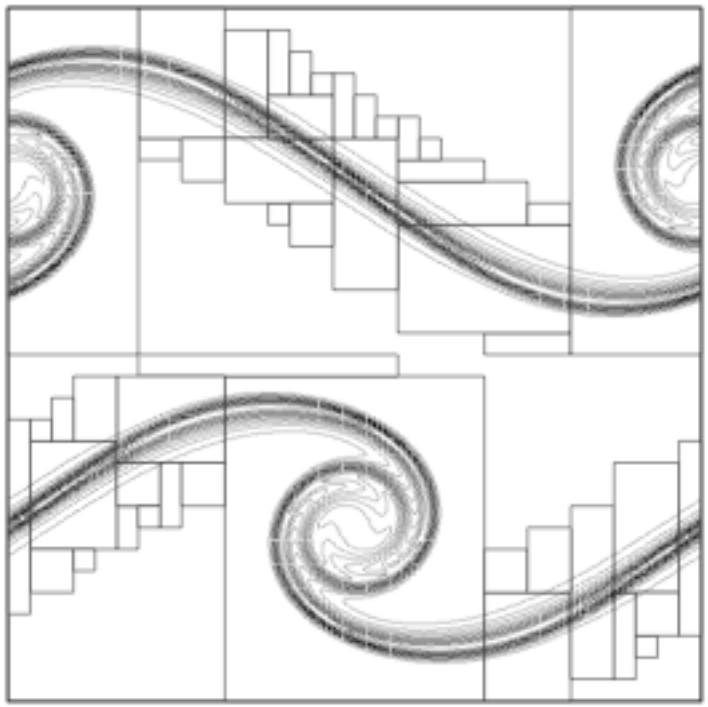


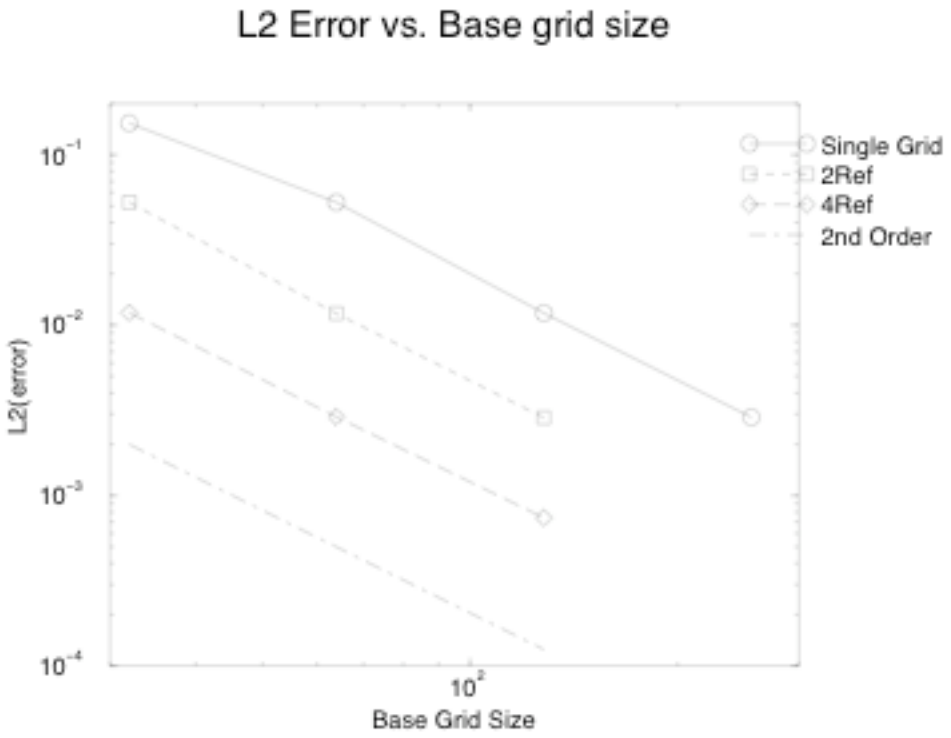
The marginal stability of the Crank-Nicolson method shown on the left suggests trying other more robust implicit Runge-Kutta methods that are second-order accurate and L_0 stable (Twizell, Gumel, and Arigu, 1996).

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

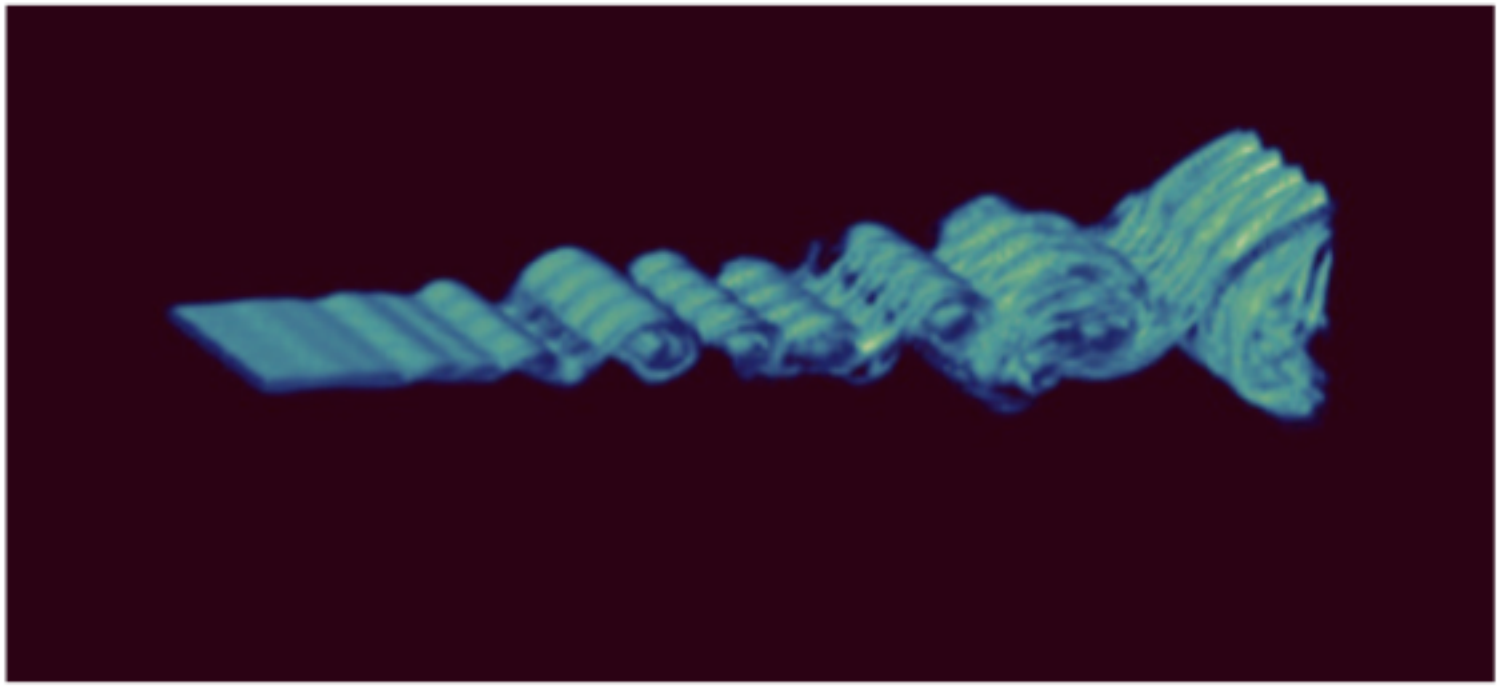
$$r_1 + r_2 + a = \Delta t, r_1 + r_2 + r_4 = \frac{\Delta t}{2}, r_1 + r_2 > \frac{\Delta t}{2}$$

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

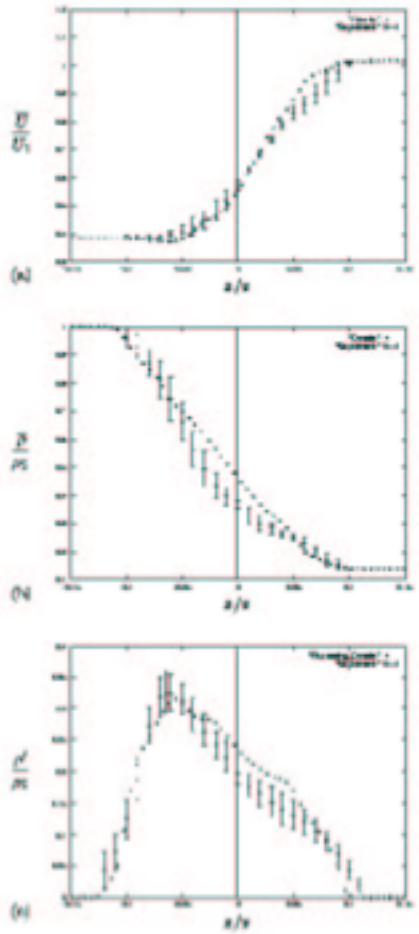


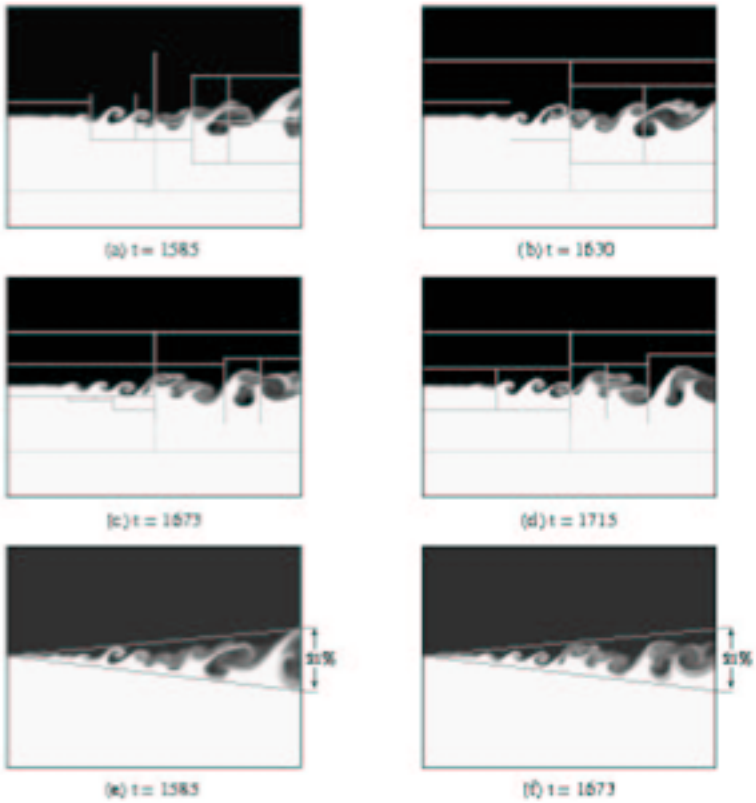


AMR Calculation of Brown-Roshko Shear Layer (Almgren, et. al., 1998)



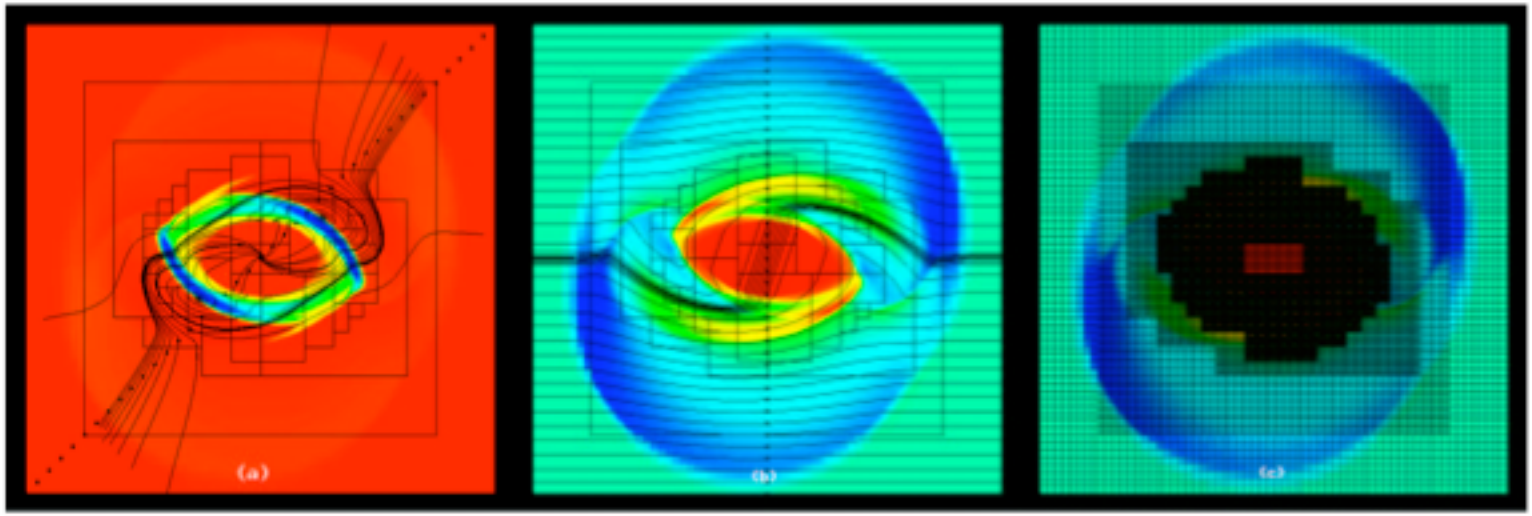
Comparison to Experimental Measurements





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, inhomogenous 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.
- Reusable components: Component design based on mapping of mathematical abstractions to classes.
- Build on public domain standards: MPI, HDF5, VTK.

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` $i \in \mathbb{Z}^d$. Can translate $\boldsymbol{i}_1 \pm \boldsymbol{i}_2$, coarsen $\boldsymbol{i} / s$, refine $\boldsymbol{i} \times s$.

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

- `FArrayBox A(Box B, int nComps)`: multidimensional arrays of doubles or floats constructed with `B` specifying the range of indices in space, `nComp` the number of components. `Real* FArrayBox::dataPtr` returns the 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
```

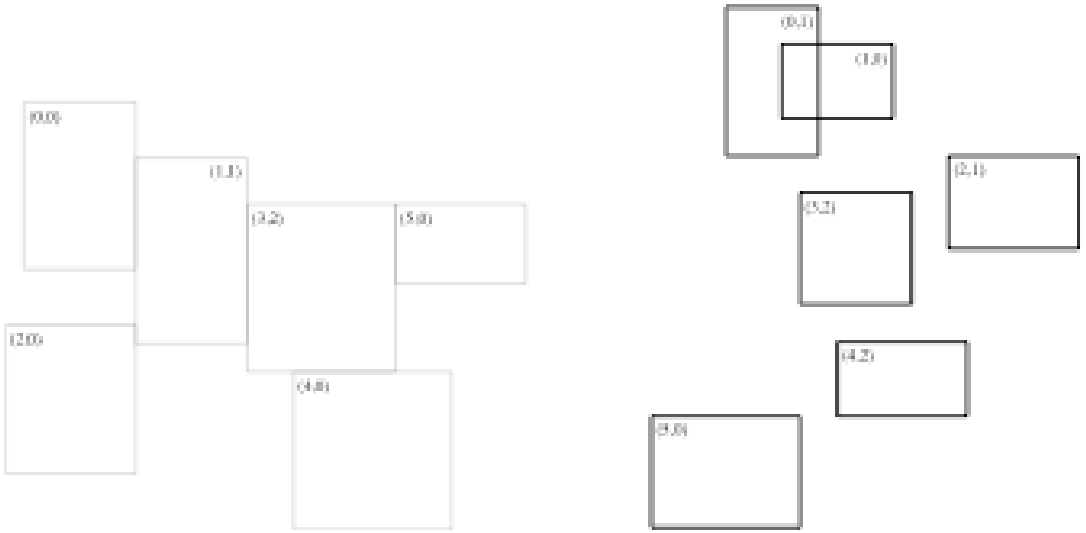
```
c Increment solution with rhs.

      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
```

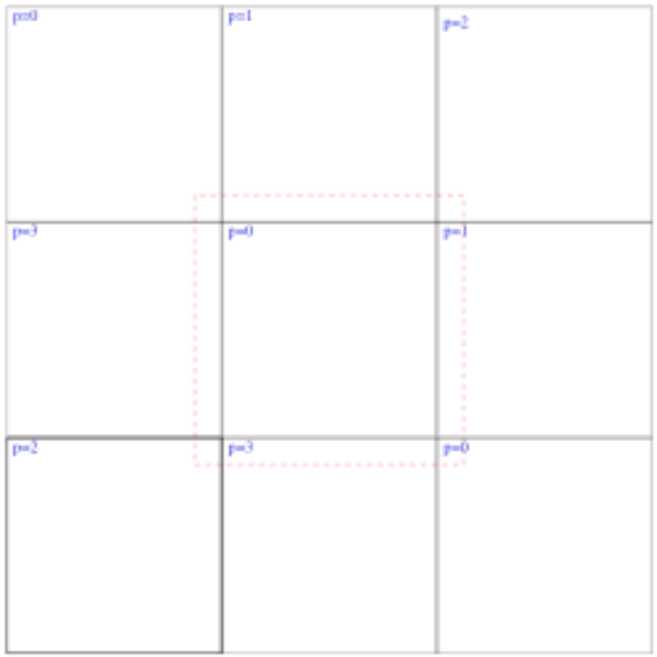
Distributed Data on Unions of Rectangles

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



- Metadata of which all processors have a copy: `BoxLayout` is a collection of `Boxes` and processor assignments: $\{B_k, p_k\}_{k=1}^{nGrids}$. `DisjointBoxLayout:public BoxLayout` is a `BoxLayout` for which the `Boxes` 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 located 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 catches 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

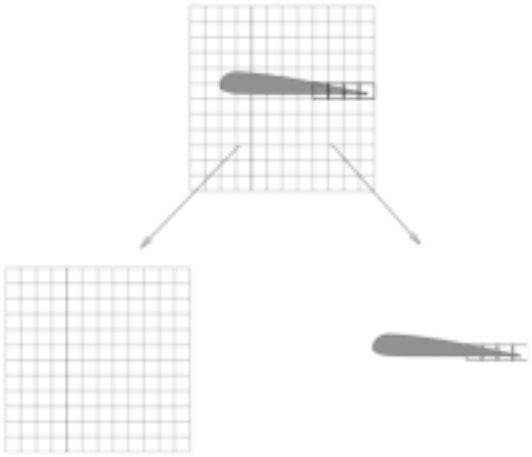
Classes can be parameterized by types, using the class template language feature in C++.

`BaseFAB<T>` is a multidimensional array which can be defined 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:



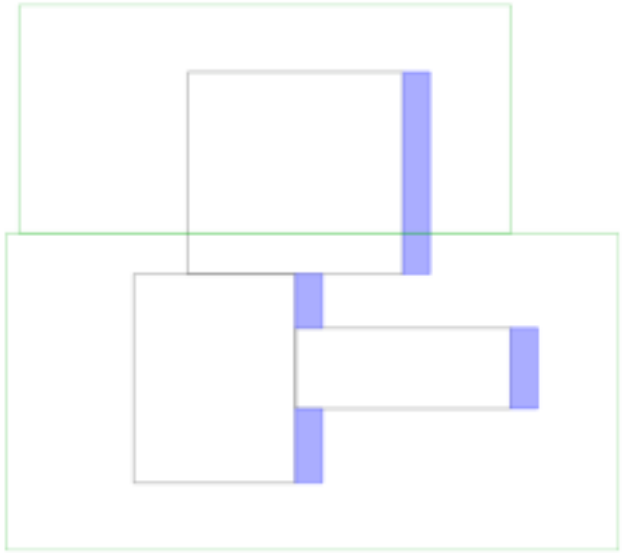
- Binsorted lists of particles, e.g. `BaseFAB<List<ParticleType>>`

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`



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

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

$$\delta F = 0$$
$$\delta F := \delta F - \Delta t^c F^c$$
$$\delta F := \delta F + \Delta t^f < F^f >$$
$$U^c := U^c + D_R(\delta F)$$

A `LevelFluxRegister` object encapsulates these operations:

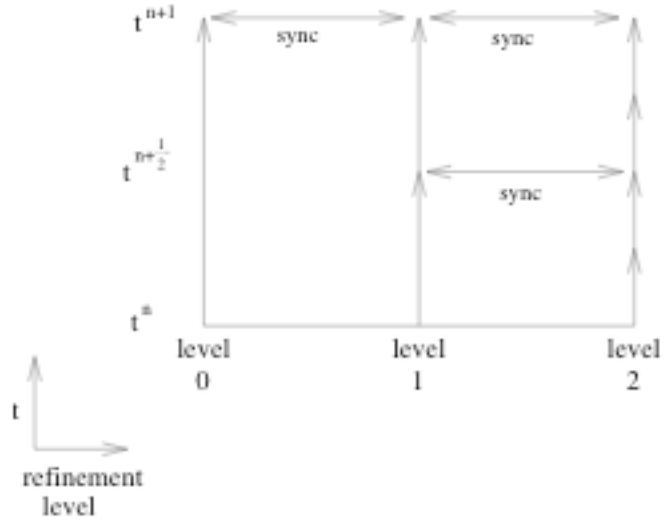
- `LevelFluxRegister::setToZero()`
- `LevelFluxRegister::incrementCoarse`: given a flux in a direction for one of the patches at the coarse level, increment the flux register for that direction.
- `LevelFluxRegister::incrementFine`: given a flux in a direction for one of the patches at the fine level, 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 that 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 synchronization operations required after all the finer levels have been updated.

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 ↔ "/"
- Group ↔ subdirectory.
- Attribute, dataset ↔ 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 homogenous. Disadvantage: many small patches can lead to large amounts of redundant work.

Both methods obtain good scaling into 1000's of processors for hyperbolic problems.

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 visibility.

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.

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 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.