

**An Incompressible Navier-Stokes with Particles Algorithm and Parallel
implementation**

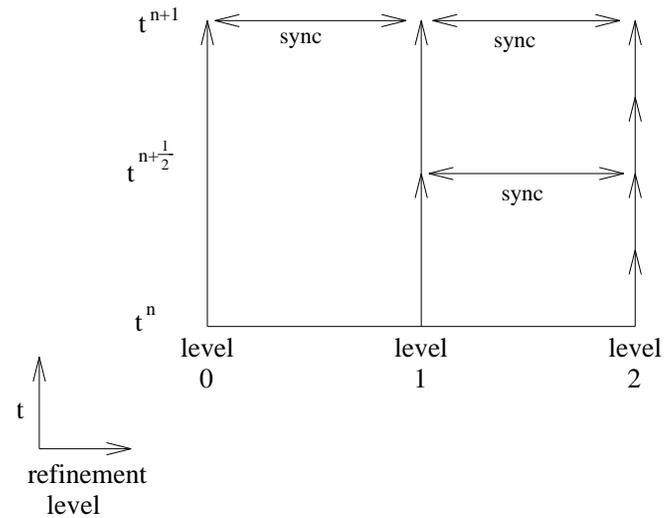
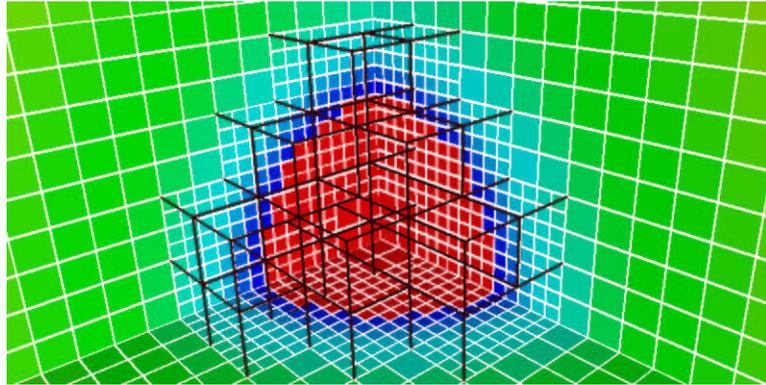
Dan Martin, Phil Colella, and Noel Keen
Applied Numerical Algorithms Group (ANAG)
Lawrence Berkeley National Laboratory

May 27, 2005

Outline:

- Adaptive Mesh Refinement
- Projection Methods for incompressible flow
- AMR Projection Method for 3d Incompressible Navier-Stokes
- AMR Projection Method for drag particles in incompressible flow
- The Chombo framework

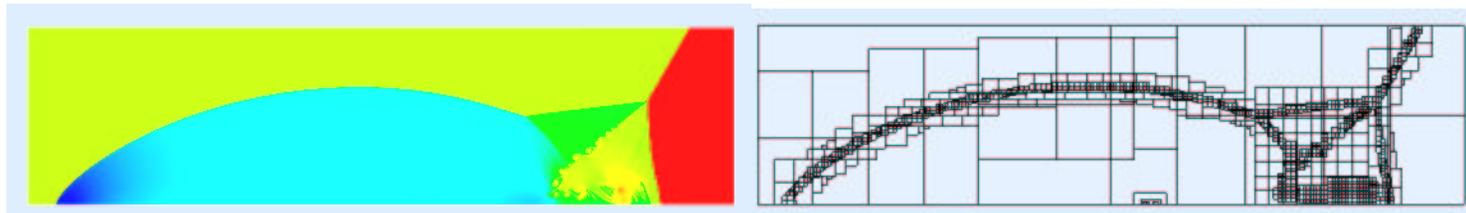
Block-Structured Local Refinement (Berger and Olinger, 1984)



Refined regions are organized into logically rectangular patches

Refinement performed in time as well as in space.

Paralellism by distributing patches on each refinement level among processors.



Constant-density Incompressible Navier-Stokes Equations

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} = -\nabla p + \nu \Delta \vec{u}$$

$$(\nabla \cdot \vec{u}) = 0$$

$$\frac{\partial s}{\partial t} + (\vec{u} \cdot \nabla) s = 0$$

Projection Methods

Enforce incompressibility constraint ($\nabla \cdot \vec{u} = 0$) by computing an intermediate update without regard to the divergence constraint:

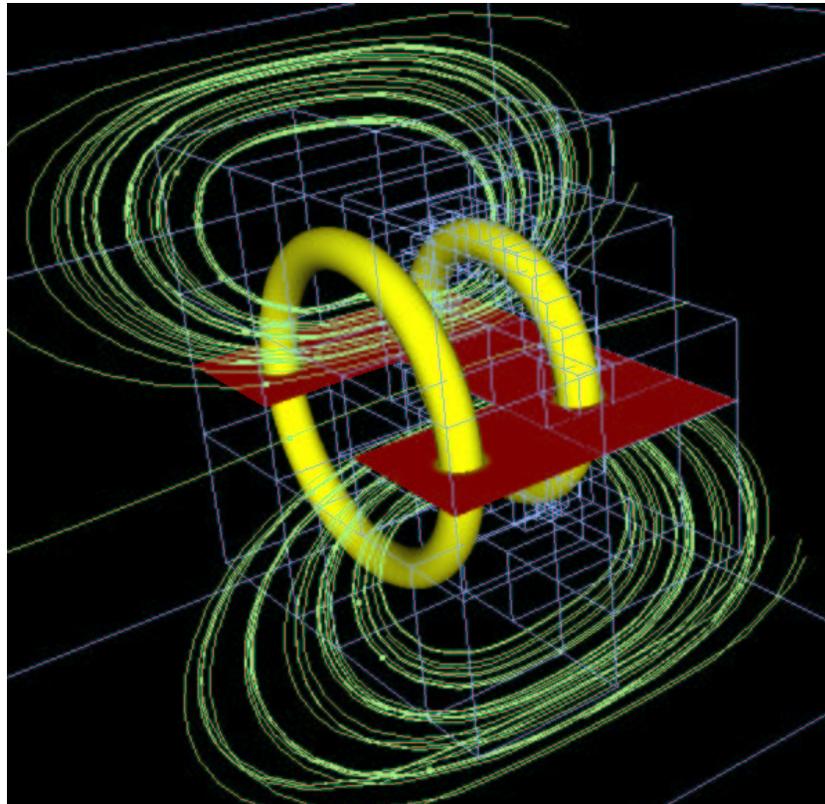
$$\vec{u}^* = \vec{u}^n - \Delta t [(\vec{u} \cdot \nabla) \vec{u}]^{n+\frac{1}{2}} + \nu \Delta \vec{u}^{n+\frac{1}{2}}$$

then “project” onto the space of divergence-free vectors:

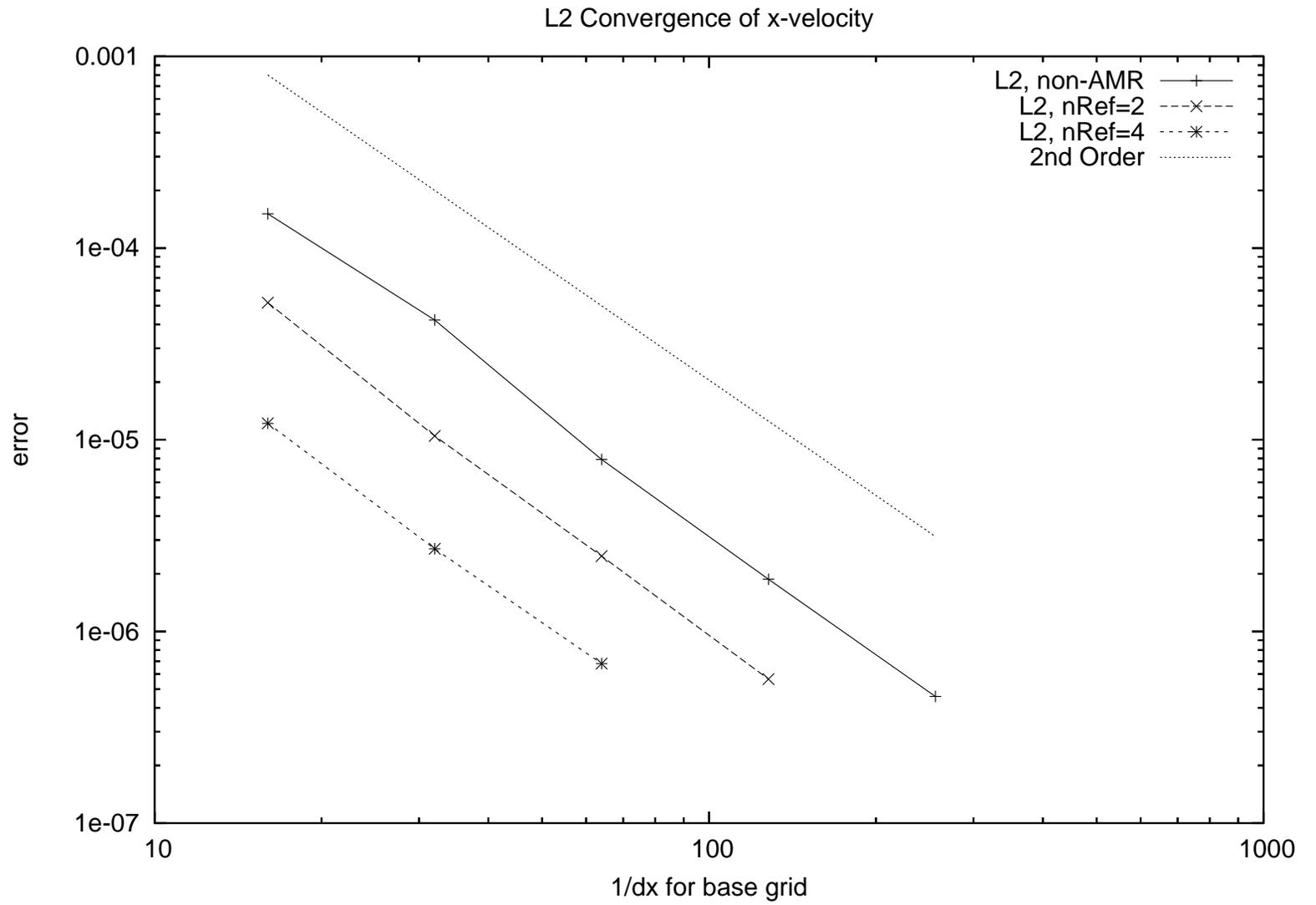
1. solve $\Delta p = \nabla \cdot \vec{u}^*$
2. correct \vec{u} : $\vec{u}^{n+1} = \vec{u}^* - \Delta t \nabla p$

Incompressible AMR Navier-Stokes (AMRINS)

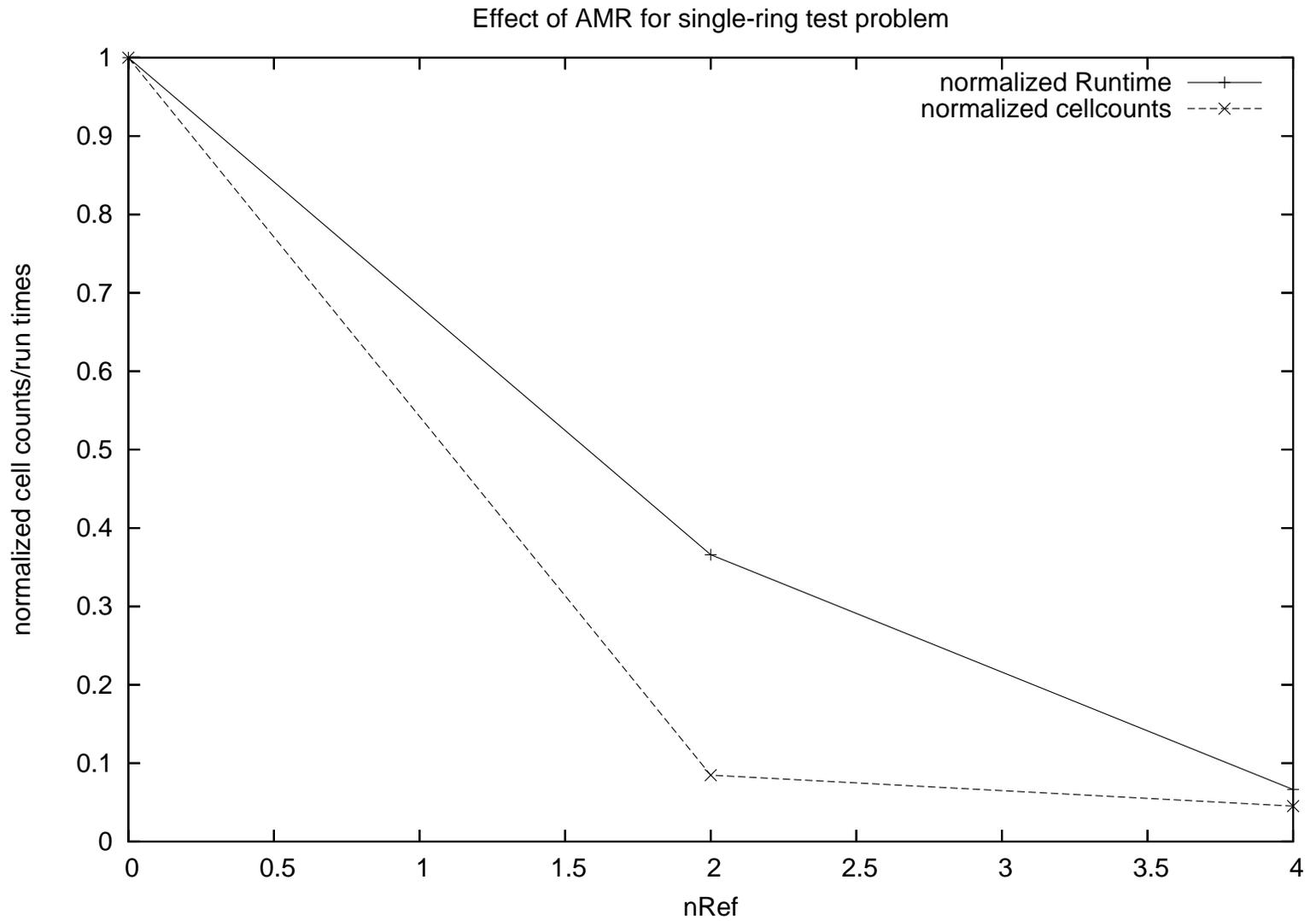
- Implements a projection method for incompressible viscous flow.
- Freestream preservation maintained approximately using an advection velocity correction computed using an auxiliary advected scalar.
- Viscous updates using L_0 -stable semi-implicit Runge-Kutta scheme
- level advance: 2 Poisson + 2*D Helmholtz solves
- synchronization: 2 Poisson + D Helmholtz multilevel AMR solves



Accuracy of AMRINS code:

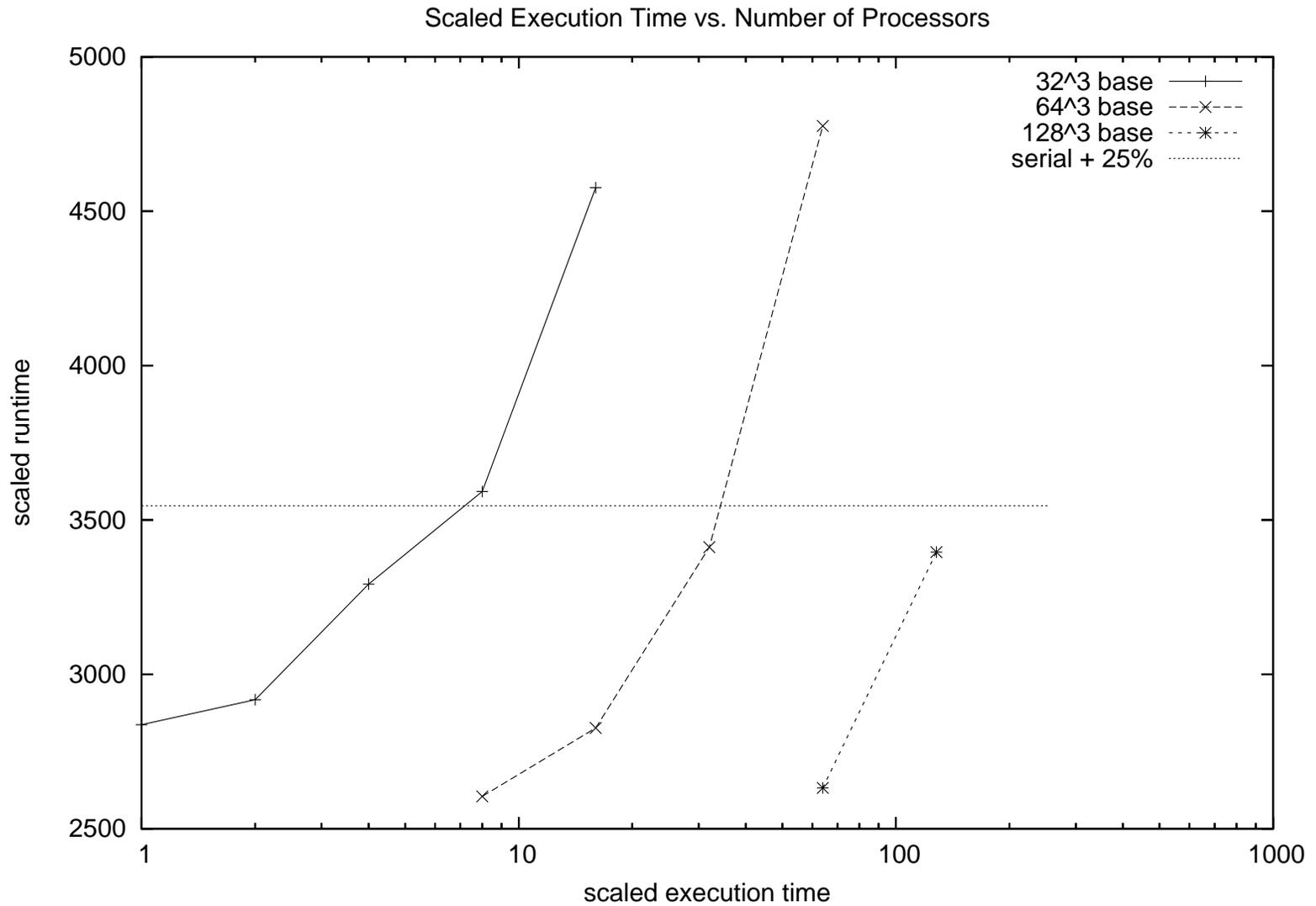


Serial Performance for single-vortex-ring test problem



Parallel performance

Define a scaled execution time as $\frac{n_{proc} \times runtime}{probSize}$



Prob size	Num Procs	Avg Memory MB	Min-Max mem MB	AMR Run secs
32x32x48	1	433	433-433	2837
32x32x48	2	240	239-242	1459
32x32x48	4	143	136-148	823
32x32x48	8	91	80-105	449
32x32x48	16	61	48-78	286
32x32x48	32	43	13-68	221
64x64x96	8	354	327-384	2605
64x64x96	16	209	180-230	1413
64x64x96	32	126	106-166	853
64x64x96	64	85	37-151	597
128x128x192	64	312	256-365	2632
128x128x192	128	197	158-268	1698

Table 1: Current parallel performance of AMRINS code for baseline vortex-ring problem

Base Problem Size	Num Procs	Large Problem Size	Large num processors	Scaled Efficiency
32x32x48	1	64x64x96	8	1.13
	2		16	1.03
	4		32	0.96
	8		64	0.75
32x32x48	1	128x128x192	64	1.07
	2		128	0.86
64x64x96	8	128x128x192	64	0.99
	16		128	0.83

Table 2: Scaled Efficiencies

Drag particles in an incompressible fluid

Particles and fluid interact through drag force, spread to mesh using numerical δ -function.

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} = -\nabla p + \nu \Delta \vec{u} + \sum_{n=1}^{numParticles} \vec{f}_n \delta_\epsilon(\vec{x} - \vec{x}_n)$$

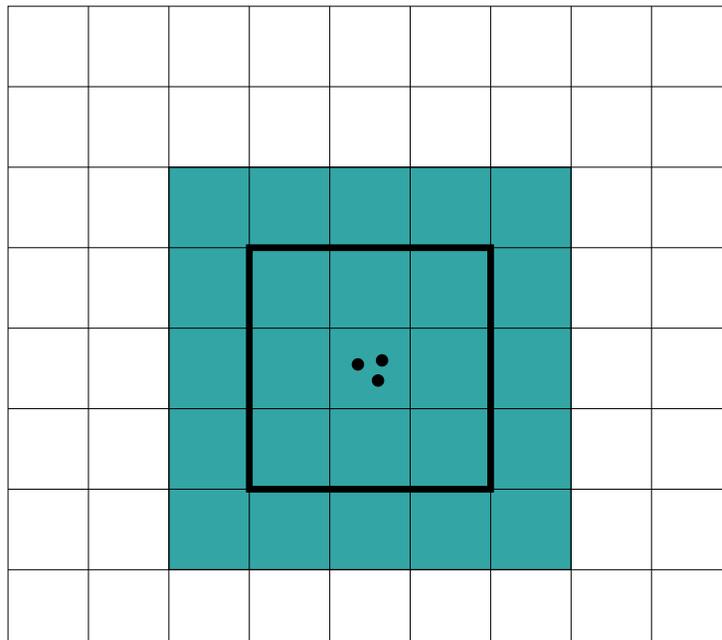
$$\vec{f}_n = k(\vec{u}_{fluid}(\vec{x}_n) - \vec{u}_n)$$

- Forcing on fluid tends to be singular (problem for the projection) – what we really want is the divergence-free **projection** of the drag force.
- Projection of drag force on fluid can be computed analytically –
$$\vec{F}_d = (I - grad(\Delta^{-1})div)\vec{F}$$

– more accurate than directly projecting the drag force due to the particles.
(Cortez and Minion)
Problem – does not have compact support.
- However, Laplacian(projected force) **does** have compact support.
- (Extends Cortez & Minion approach to 3D)

Spreading particle drag force onto mesh:

- Set $\vec{R}_h = 0$
- Over small region around particle, compute analytic divergence-free force $\vec{F}_d(\vec{x}) = \vec{f}\delta_\epsilon(\vec{x} - \vec{x}_i) - \text{grad}(\Delta^{-1})\text{div}(\vec{f}\delta_\epsilon(\vec{x} - \vec{x}_i))$. (local)
- Compute discrete Laplacian of \vec{F}_d : $\vec{R}_h = \vec{R}_h + L_h\vec{F}_D$. (local)
- Perform elliptic solve for projected force $L\vec{F} = R_h$ using infinite-domain boundary conditions. (physical boundary conditions at domain boundary will be enforced by the projection)
- use \vec{F} as forcing term for momentum update for fluid.



Particle Position and Velocity Updates

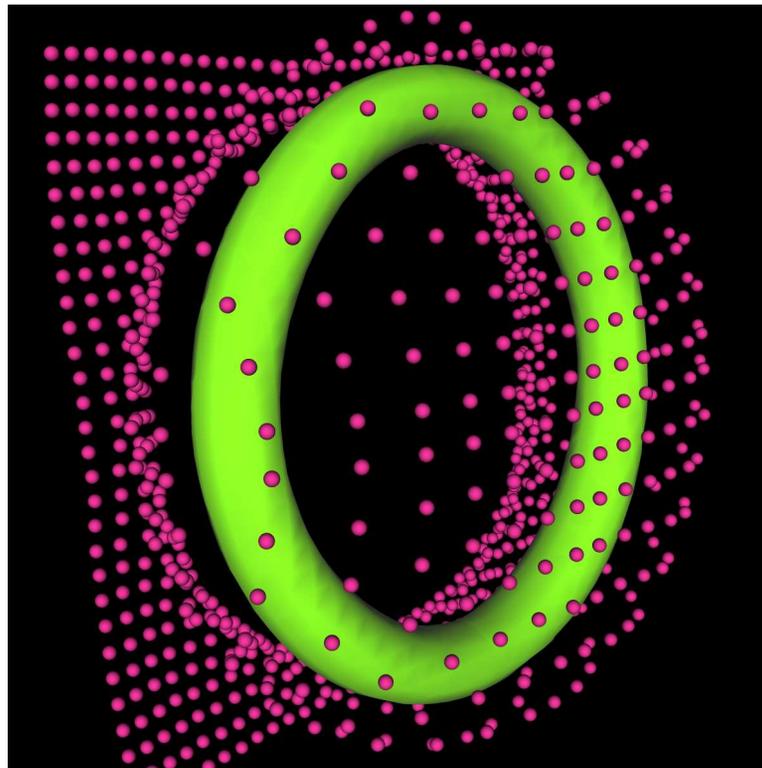
(predictor-corrector method):

(based on solution to equation of motion for particles)

- Interpolate (cell-centered) velocity field to particle positions
- predict velocity: $\vec{u}_i^* = (\vec{u}_i^n - (\vec{u}_{fluid} + \vec{g}/k))e^{-\Delta t * k/m} + (\vec{u}_{fluid} + \vec{g}/k)$
- $x_i^* = x_i^n + \Delta t \vec{u}_i^n$

AMR Particle Incompressible Navier-Stokes (PAMRNS) code

- No refinement in time.
- Separate processor distributions for particles and fluid



Prob size	Num Procs	Max Mem (MB)	AMR Run (sec)	Particle update (sec)
32x32x32	8	72.5	100.4	0.59
64x64x64	16	115.7	178.4	1.2
64x64x64	64	75.8	103.4	0.38
128x128x128	32	317.8	597.3	3.73
128x128x128	64	175.1	352.4	2.24
128x128x128	128	117.3	220.8	1.41

Table 3: Current parallel performance of AMRINS with particles code for vortex-ring problem with 32,768 particles

Base Problem Size	Num Procs	Large Problem Size	Large num processors	Scaled Efficiency
32x32x32	8	64x64x64	64	0.97
64x64x64	16	128x128x128	128	0.81

Table 4: Scaled Efficiencies

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.

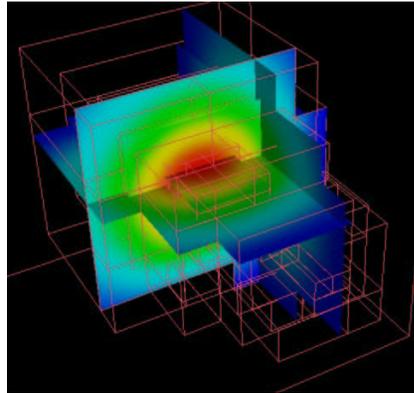
- Mixed-language model: C++ for higher-level data structures, Fortran for regular single-grid calculations.
- Reuseable components. Component design based on mapping of mathematical abstractions to classes.
- Build on public-domain standards: MPI, HDF5, VTK.
- Interoperability with other SciDAC ISIC 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.

ChomboVis Interactive Visualization and Analysis Tools



- “AMR-aware”
 - Block-structured representation of the data leads to efficiency.
 - Useful as a debugging tool (callable from debuggers (gdb))
- Visualization tools based on VTK, a open-source visualization library.
- Implementation in C++ and Python
 - GUI interface for interactive visualization
 - Command-line python interface to visualization and analysis tools, batch processing capability – goal is a full analysis tool.
- Interface to HDF5 I/O along with C API provides access to broad range of AMR users. (“Framework-neutral”)

Acknowledgements

DOE Applied Mathematical Sciences Program

DOE HPCC Program

DOE SciDAC Program

NASA Earth and Space Sciences Computational Technologies Program