

---

---

# OVERTURE: An object-Oriented Framework built on top of PADRE

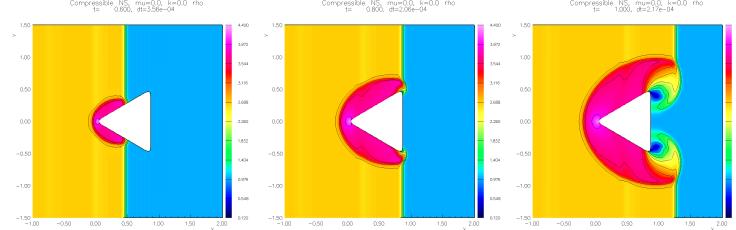
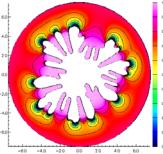
---

---

Dan Quinlan

Center for Applied Scientific Computing  
Lawrence Livermore National Laboratory

# The *Overture* team at CASC



**David Brown**

*finite volume  
methods, adaptive  
mesh refinement*



**Bill Henshaw**

*grid generation,  
CFD, combustion, multigrid,  
Overture Framework*



**Dan Quinlan**

*P++, PADRE, ROSE,  
AMR++*



**Kyle Chand**

*hybrid grids*



**Brian Miller**

*level set methods,  
parallel methods*



**Danny Thorne**

*cache-based optimization,  
ROSE*



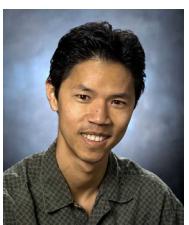
**Petri Fast**

*Hele-Shaw flows;  
fluid-elastic  
interactions*



**Anders Petersson**

*grid generation, CFD*



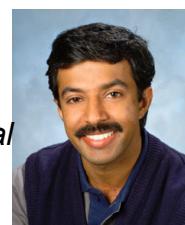
**Brian Gunney**

*ROSE, PADRE*



**Rick Pember**

*Computational  
Combustion*



**Bobby Phillip**

*AMR++, fluid-  
elastic interactions, multigrid*

# Introduction To Overture and PADRE Outline

---

---

- Introduction to OVERTURE
- Introduction to PADRE
- Interfaces
- Performance
- Applications
- Future work

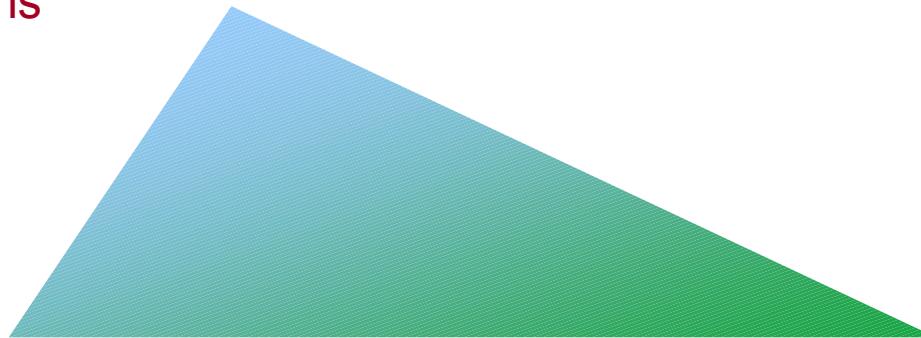
# Motivation

---

---

## Algorithm Complexity

- MultiGrid
- AMR
- Implicit/Explicit Equations
- Conservation
- Higher Order Methods



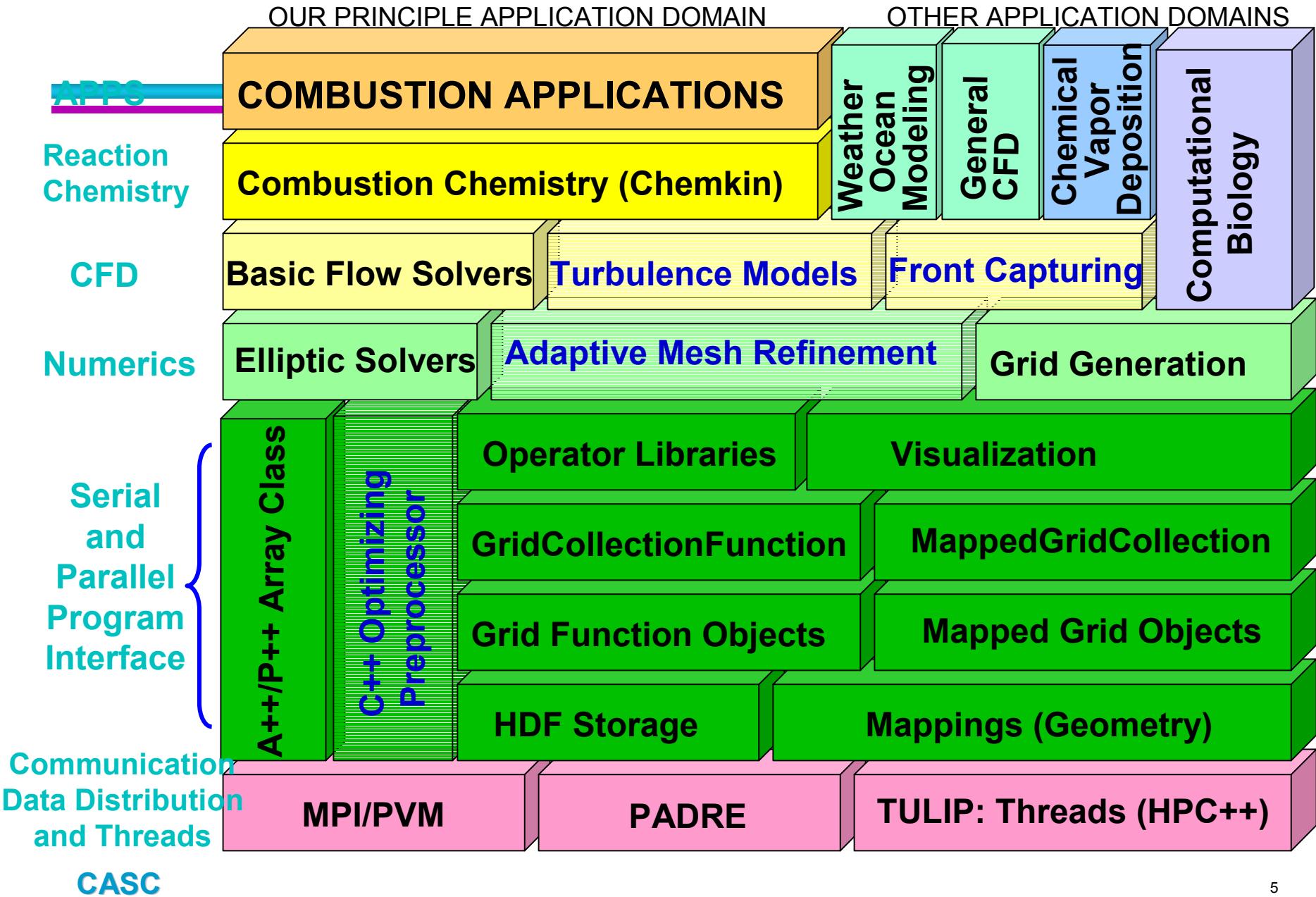
## Application Complexity

- Geometry
- Physics

## Architecture Complexity

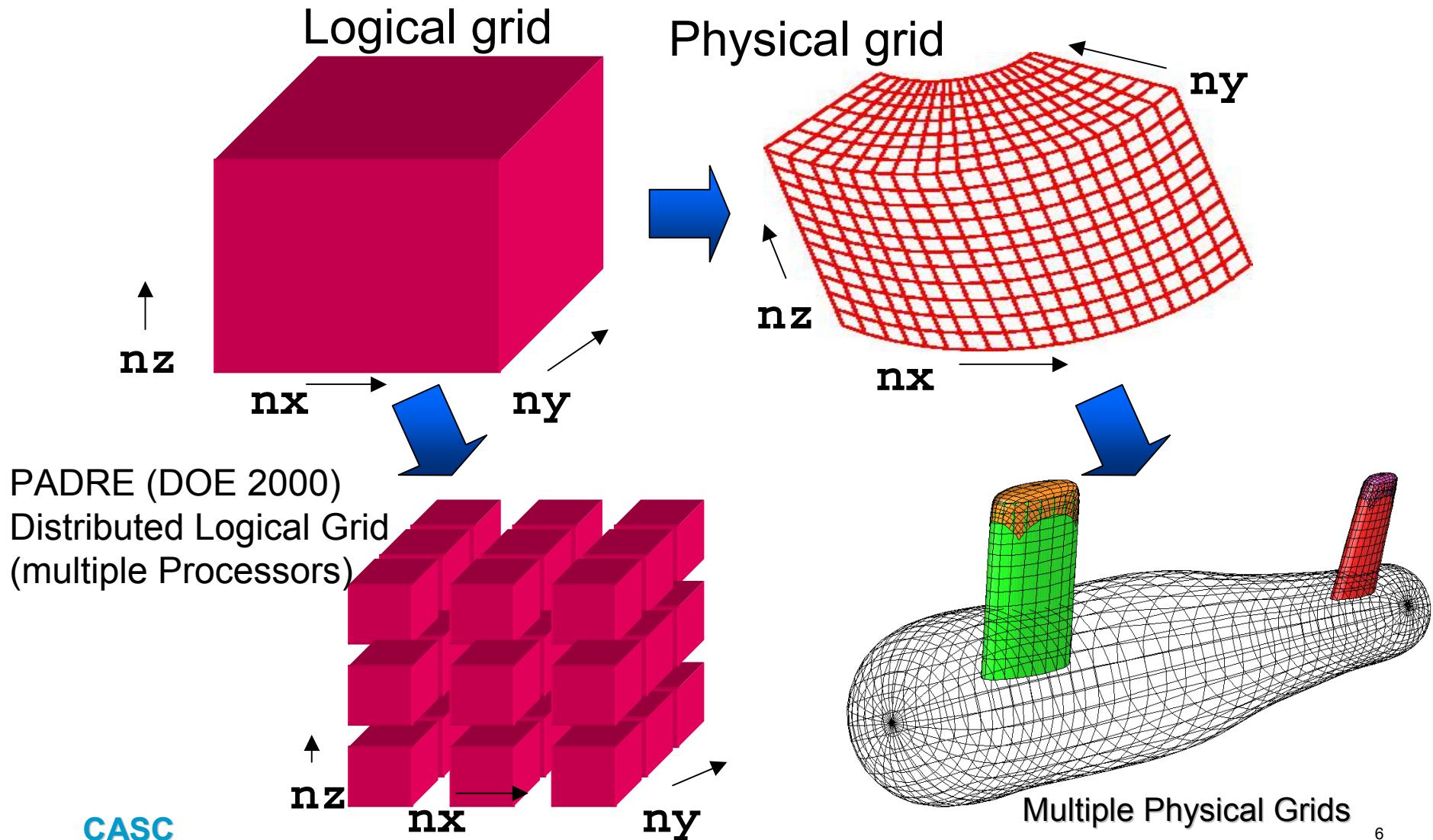
- Cache
- Vector
- Shared Memory Parallelism
- Distributed Memory Parallelism

# Design of the OVERTURE Framework



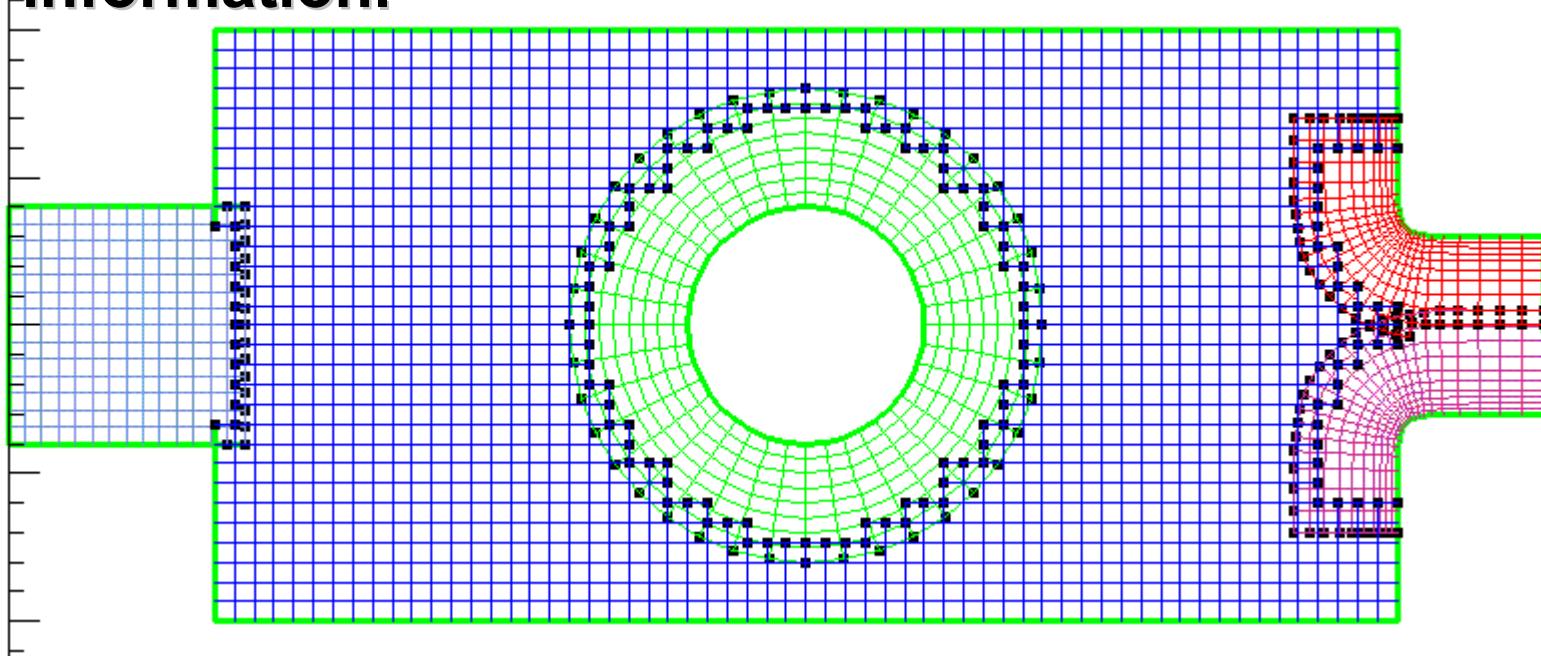
# Grid Generation

## Diverse Geometries

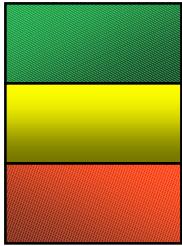


An *overlapping grid* consists of a set of logically rectangular curvilinear grids that overlap where they meet and which completely cover the computational domain.

The *Overture* grid generator, *Ogen*, automatically computes the overlap and connectivity information.



# Overlapping grids provide flexible, efficient grid generation and solver algorithms



*Excels*

*Adequate*

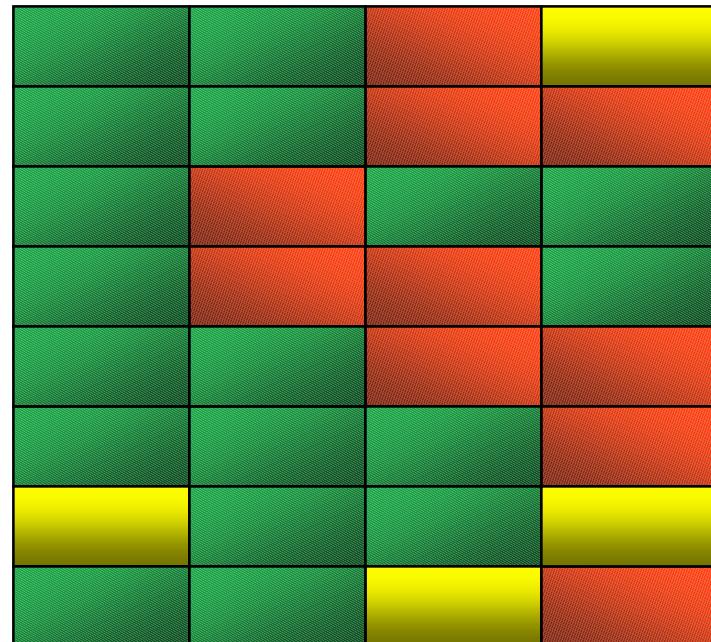
*Difficult or unavailable*

Overlapping

Embedded

Unstructured

Multi-block



*Efficient solver storage*

*Efficient Moving Components*

*Body-fitted coordinates*

*Boundary Layer control*

*Component-based grid generation*

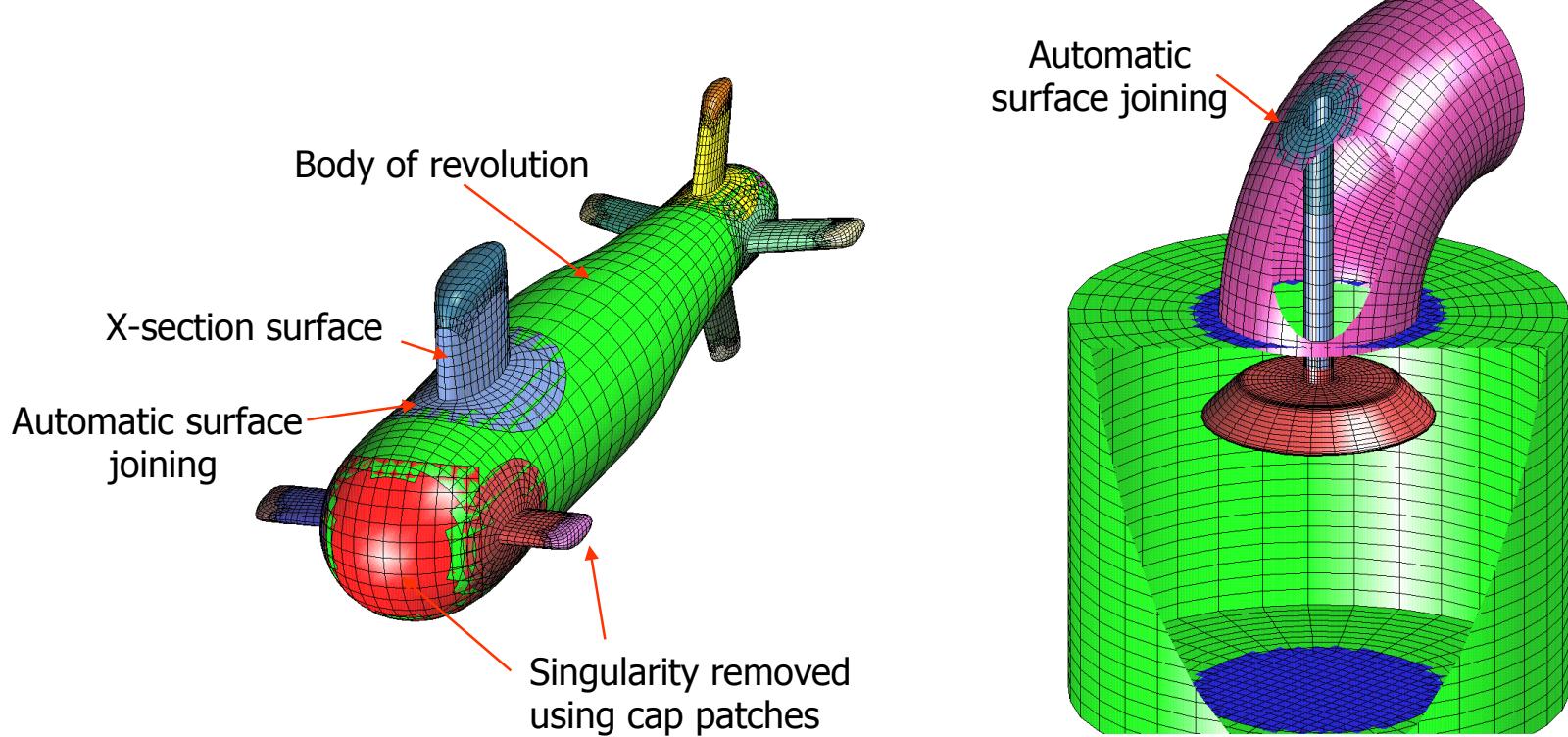
*Volume Partitioning NOT required*

*Expectations for automatic grid-generation*

*Full re-grid not required for added components*

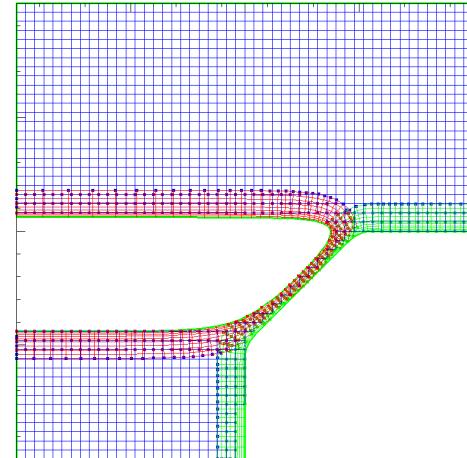
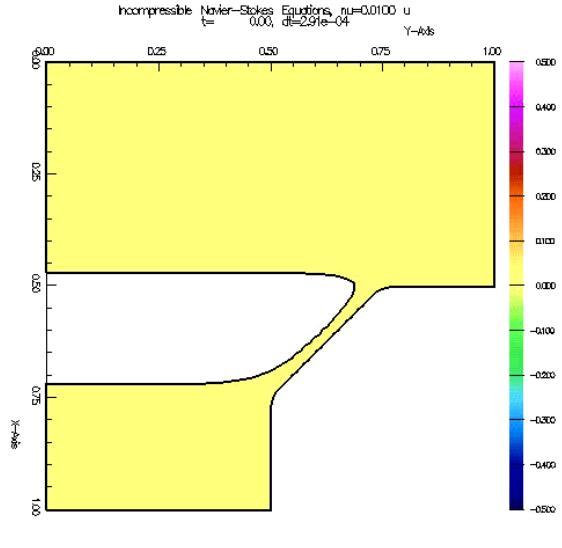
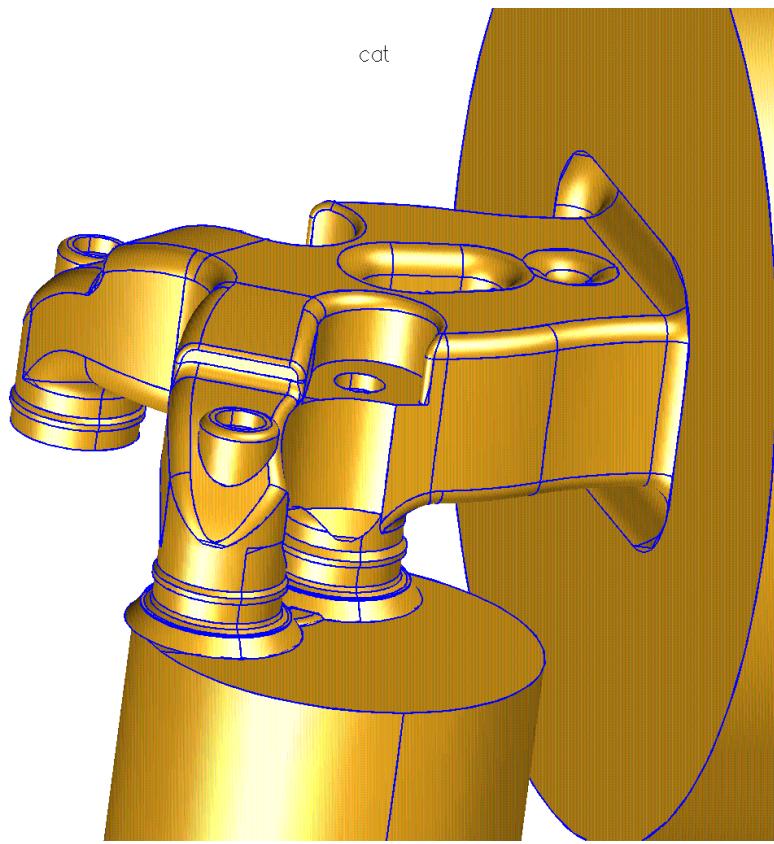
# Overset Grid Generation Capabilities

---



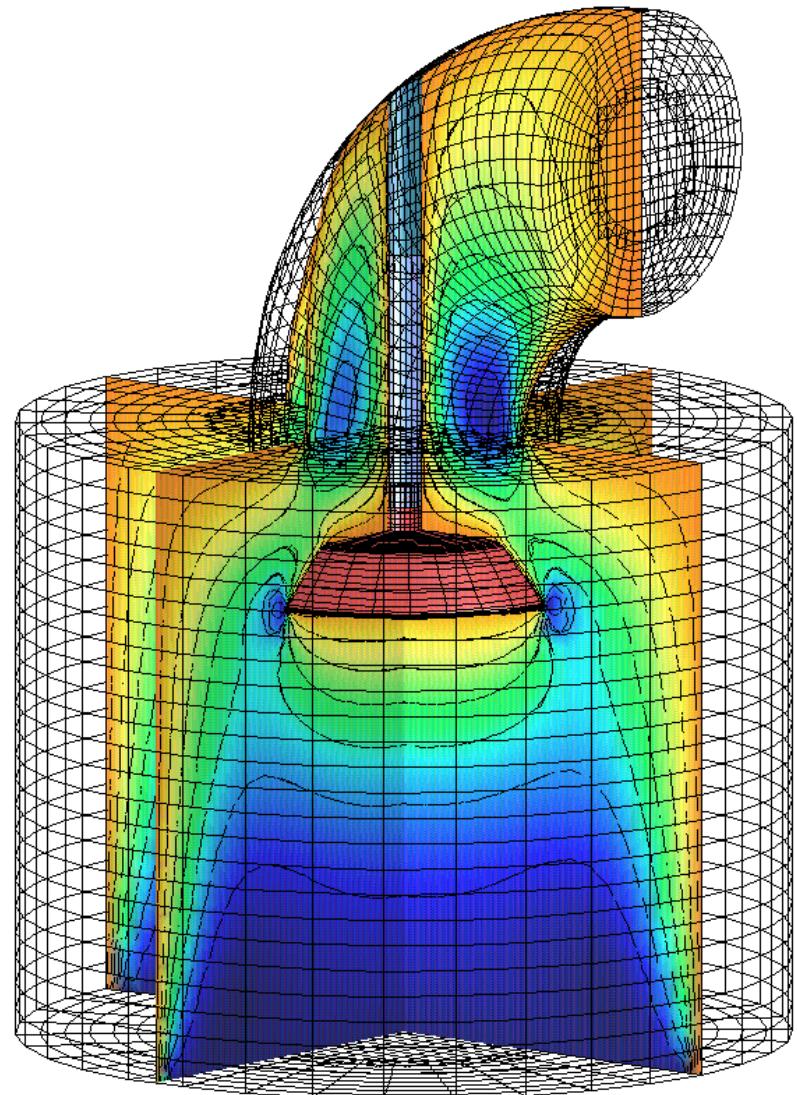
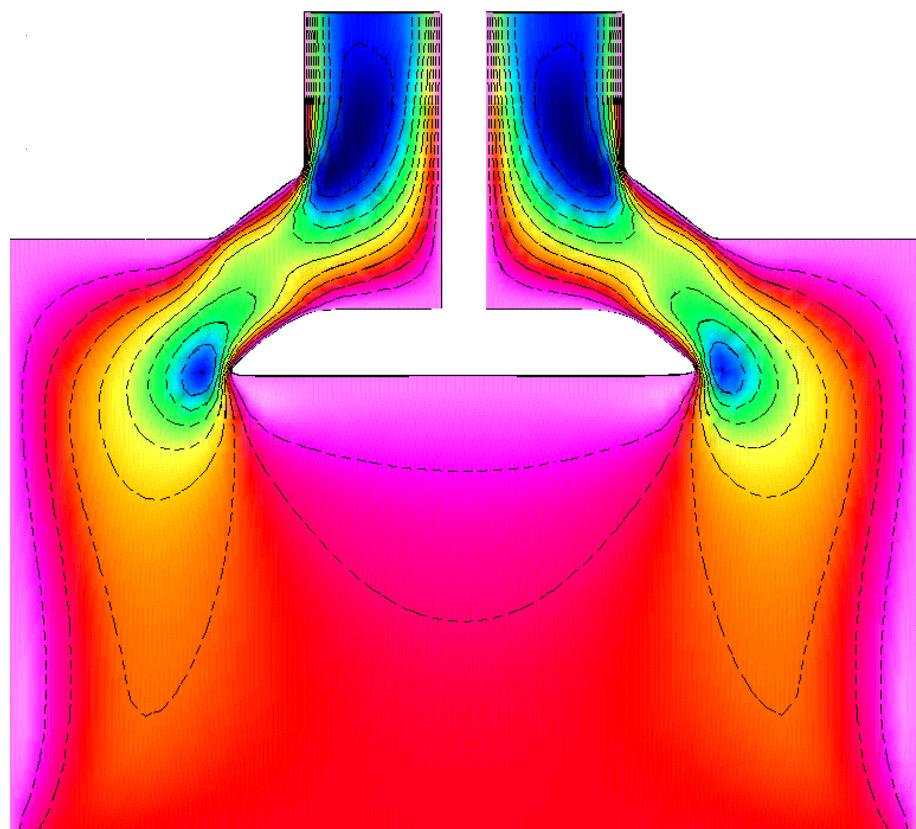
*Bill Henshaw*

# Simulations in complex moving geometry present many challenges

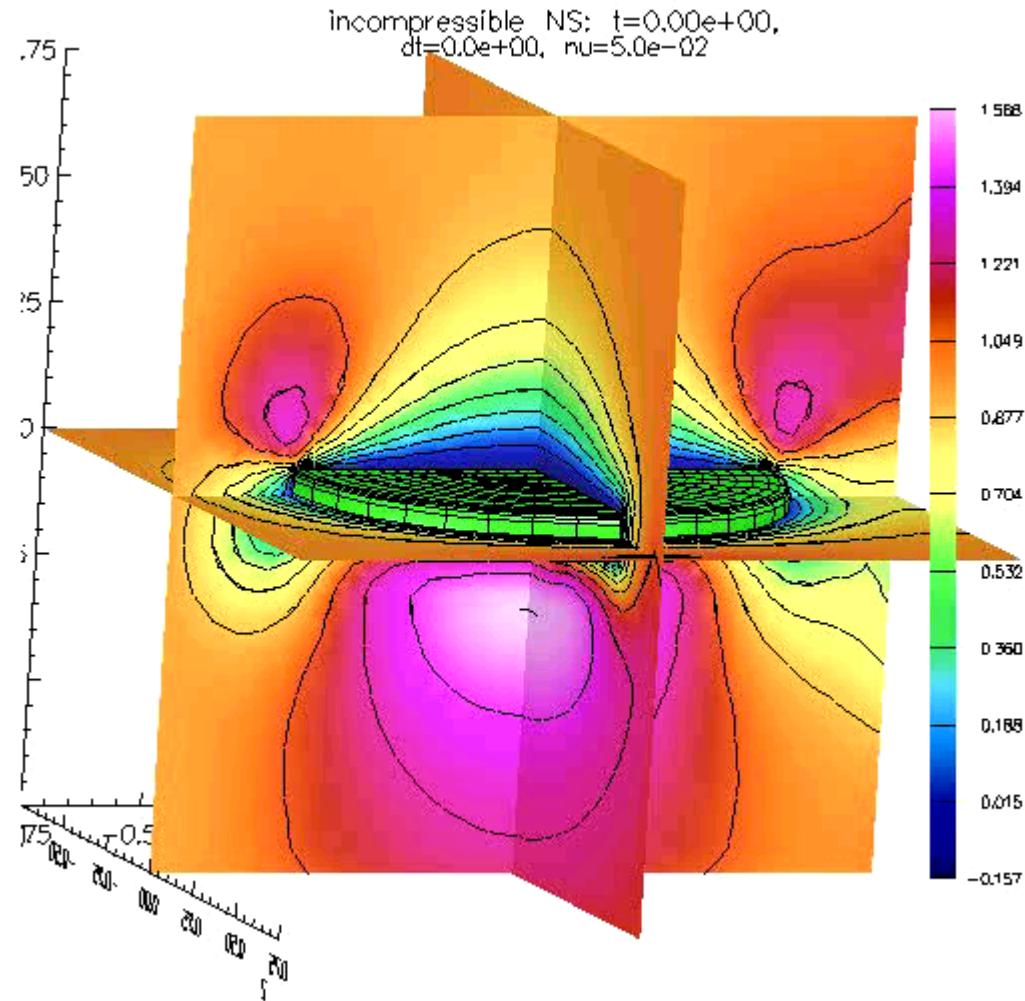


# Moving valve computations

---



# Simulations in complex moving geometry present many challenges



# The fundamental building block for the *Overture* framework is the P++ array class

---

Stencil operations on structured grids are naturally expressed in terms of array operations

Details of parallel implementation can be hidden from the user by the array class

Like F90  
Index Triplet

Parallel communication  
occurs at the =

```
Index I,J;  
floatArray u,v,w;
```

```
// update stencil and communicate between processors
```

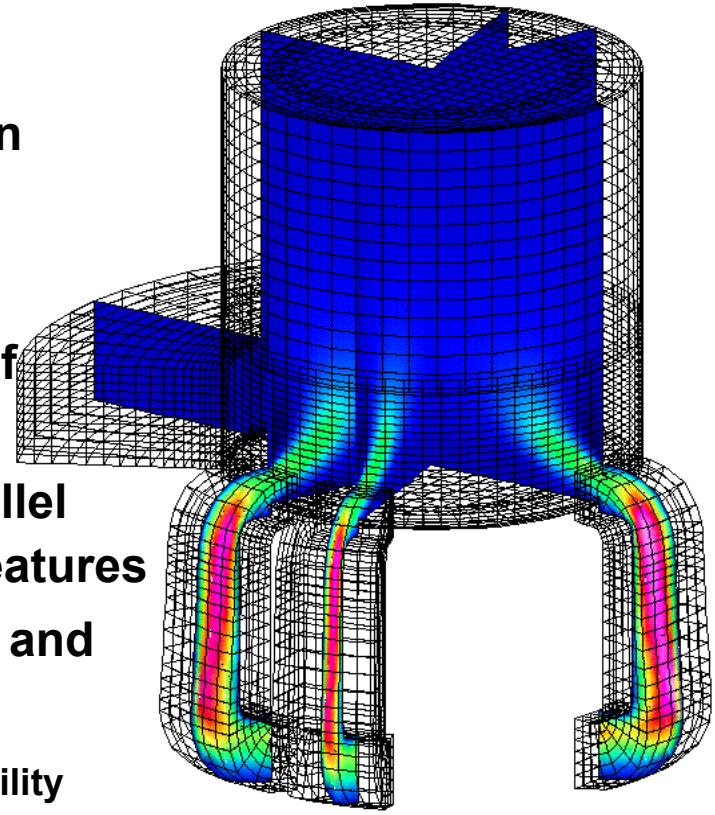
```
u(I,J) += .25*(u(I-1,J) + u(I+1,J) + u(I,J-1) + u(I,J+1));
```

Like F90  
Arrays

# Purpose of PADRE: INTEROPERABILITY

- Encapsulate distribution mechanisms
- Interface for Load Balancing
- Encapsulate message scheduling between distributions
- Tool for building parallel libraries
  - Simplifies design and development of parallel libraries
  - Simplifies maintenance of existing parallel libraries while providing additional features
  - Permits interoperability with existing and future parallel libraries
    - Use of PADRE is not required for interoperability

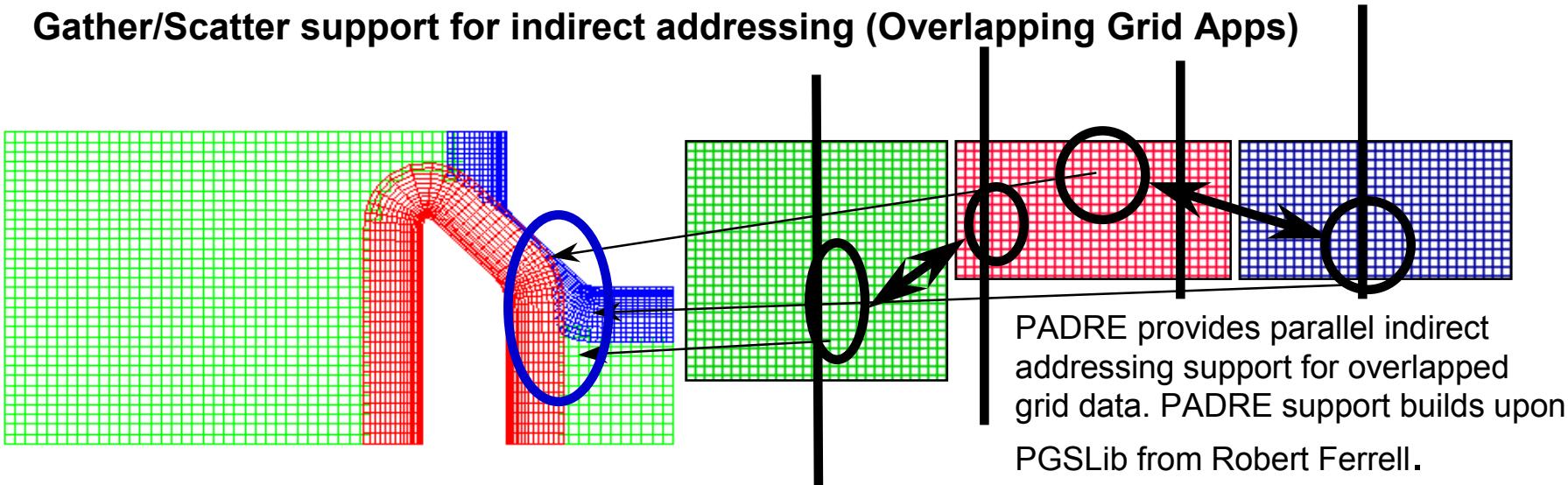
Incompressible Navier–Stokes v  
 $t = 0.90$   $dt = 0.36E-02$   $\nu = .01000$



Min = -0.18E+00 Max = 0.28E+01

# PADRE Services

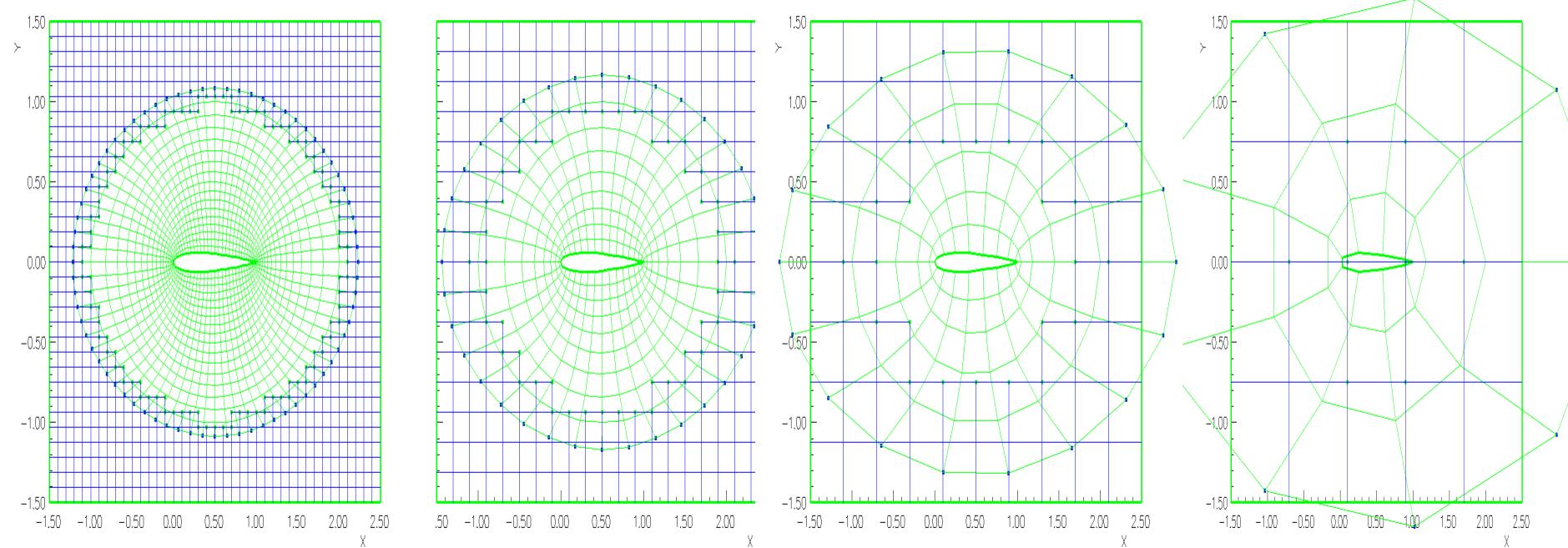
- Dynamic Distribution (redistribution, etc.)
- Generation and Execution of Communication Schedules
  - General Array Language Support
  - Cached schedules for performance
  - Lazy Evaluation
  - Message Latency Hiding
- Subarray operation support for different distributions (AMR)
- Gather/Scatter support for indirect addressing (Overlapping Grid Apps)



# Multigrid Solvers on Overlapping Grids

## Automated construction of coarsenings

- an advantage of keeping structured grids in the solution process



- Natural interpolation operators (AMG not a requirement)
- Structured Grids are CPU and memory efficient

# PADRE Overview

---

---

## Multiple Distribution Libraries

e.g. Multi-block PARTI (UM),  
KeLP (UCSD), PGSLib (LANL),  
Global Arrays (PNL)

PADRE

Communication Library

e.g. MPI, PVM

Application Libraries

e.g. P++, Overture, AMR++

# P++ provides array objects and operations in a parallel environment

---

```
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5
```

# P++ *Arrays* are distributed over many processors on a parallel machine

---

---

1.0 0.9 0.9 0.8	0.8 0.9 0.5 0.5 0.5
0.9 0.8 0.8 0.7	0.6 0.7 0.4 0.4 0.4
0.8 0.7 0.6 0.7	0.6 0.5 0.3 0.3 0.2
0.9 0.8 0.8 0.7	0.6 0.7 0.4 0.4 0.4
1.0 0.9 0.9 0.8	0.8 0.9 0.5 0.5 0.5
0.8 0.7 0.6 0.7	0.6 0.5 0.3 0.3 0.2
0.9 0.8 0.8 0.7	0.6 0.7 0.4 0.4 0.4
0.8 0.7 0.6 0.7	0.6 0.5 0.3 0.3 0.2
1.0 0.9 0.9 0.8	0.8 0.9 0.5 0.5 0.5
0.9 0.8 0.8 0.7	0.6 0.7 0.4 0.4 0.4
0.8 0.7 0.6 0.7	0.6 0.5 0.3 0.3 0.2
0.9 0.8 0.8 0.7	0.6 0.7 0.4 0.4 0.4
1.0 0.9 0.9 0.8	0.8 0.9 0.5 0.5 0.5

# P++ *Arrays* are distributed over many processors on a parallel machine

---

---

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	

# P++ *Arrays* are distributed over many processors on a parallel machine

---

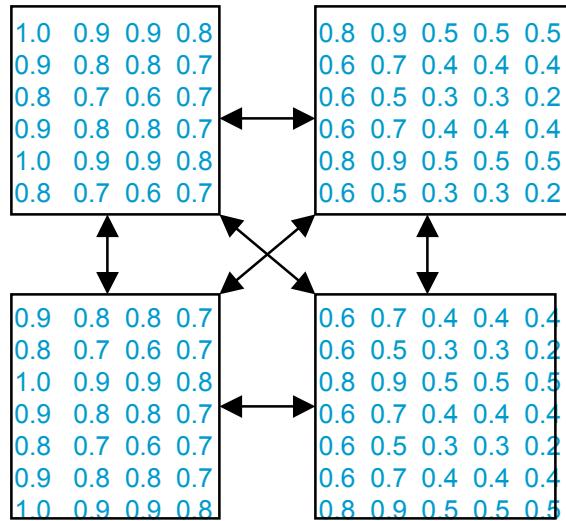
---

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

# P++ *Arrays* are distributed over many processors on a parallel machine

---



# P++ *Arrays* are distributed over many processors on a parallel machine

---

---

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5

# Associate an array with geometrical information to get a *MappedGridFunction*

---

---

1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	

0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2	
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4	
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5	

# Associate an array with geometrical information to get a *MappedGridFunction*

---

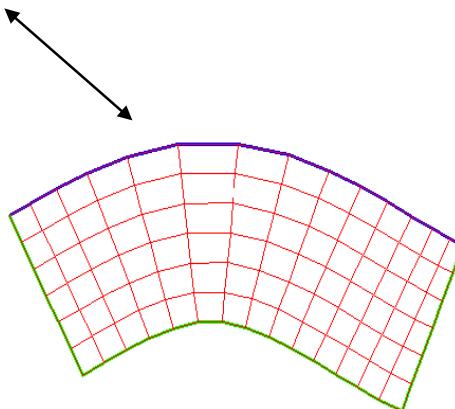
```
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5
```

# Associate an array with geometrical information to get a *MappedGridFunction*

---

---

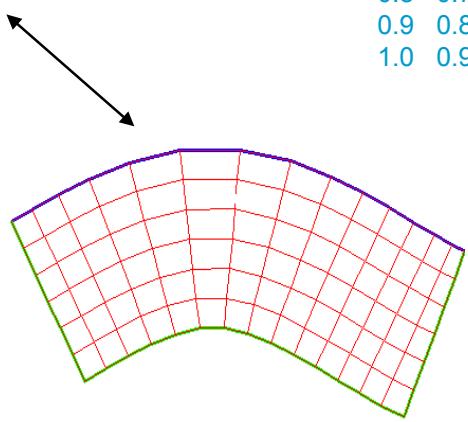
```
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
0.8 0.7 0.6 0.7 0.6 0.5 0.3 0.3 0.2  
0.9 0.8 0.8 0.7 0.6 0.7 0.4 0.4 0.4  
1.0 0.9 0.9 0.8 0.8 0.9 0.5 0.5 0.5
```



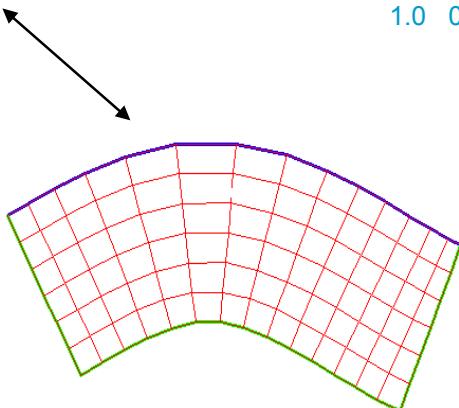
A floatMappedGridFunction  
is derived from a floatArray

A set of *MappedGridFunction*'s forms a  
*GridCollectionFunction*

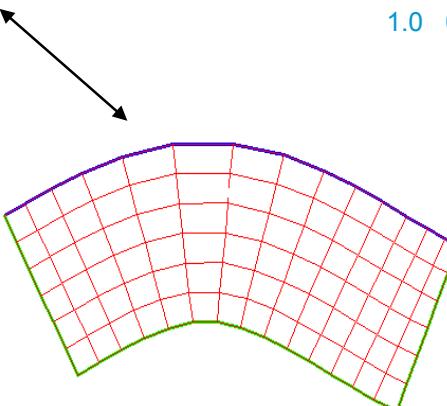
0.8	0.8	0.9	0.5	0.5	0.5
0.7	0.6	0.7	0.4	0.4	0.4
0.7	0.6	0.5	0.3	0.3	0.2
0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.8	0.9	0.5	0.5	0.5
0.7	0.6	0.5	0.3	0.3	0.2
0.7	0.6	0.7	0.4	0.4	0.4
0.7	0.6	0.5	0.3	0.3	0.2
0.8	0.8	0.9	0.5	0.5	0.5
0.7	0.6	0.7	0.4	0.4	0.4
0.7	0.6	0.5	0.3	0.3	0.2
0.8	0.8	0.9	0.5	0.5	0.5
0.7	0.6	0.7	0.4	0.4	0.4
0.7	0.6	0.5	0.3	0.3	0.2
0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.8	0.9	0.5	0.5	0.5



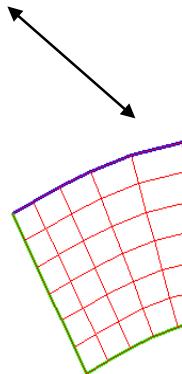
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5



1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3	0.3	0.2
0.9	0.8	0.8	0.7	0.6	0.7	0.4	0.4	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5	0.5	0.5



1.0	0.9	0.9	0.8	0.8	0.9	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3
0.9	0.8	0.8	0.7	0.6	0.7	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5
0.8	0.7	0.6	0.7	0.6	0.5	0.3
0.9	0.8	0.8	0.7	0.6	0.7	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3
1.0	0.9	0.9	0.8	0.8	0.9	0.5
0.9	0.8	0.8	0.7	0.6	0.7	0.4
0.8	0.7	0.6	0.7	0.6	0.5	0.3
0.9	0.8	0.8	0.7	0.6	0.7	0.4
1.0	0.9	0.9	0.8	0.8	0.9	0.5



# At the highest level, *Overture* code looks like the underlying mathematics.

---

Mathematical expressions involving differential operators such as

$$u_{new} = u - \delta t \cdot ((u \bullet \nabla)u - v \Delta u)$$

are expressed concisely using the Overture operator classes.

This example advances a convection-diffusion equation on an overlapping grid. All grid-dependent and parallel details are hidden at this level.

```
uNew = u - dt*( u.convectiveDerivative() - nu*u.laplacian());
```

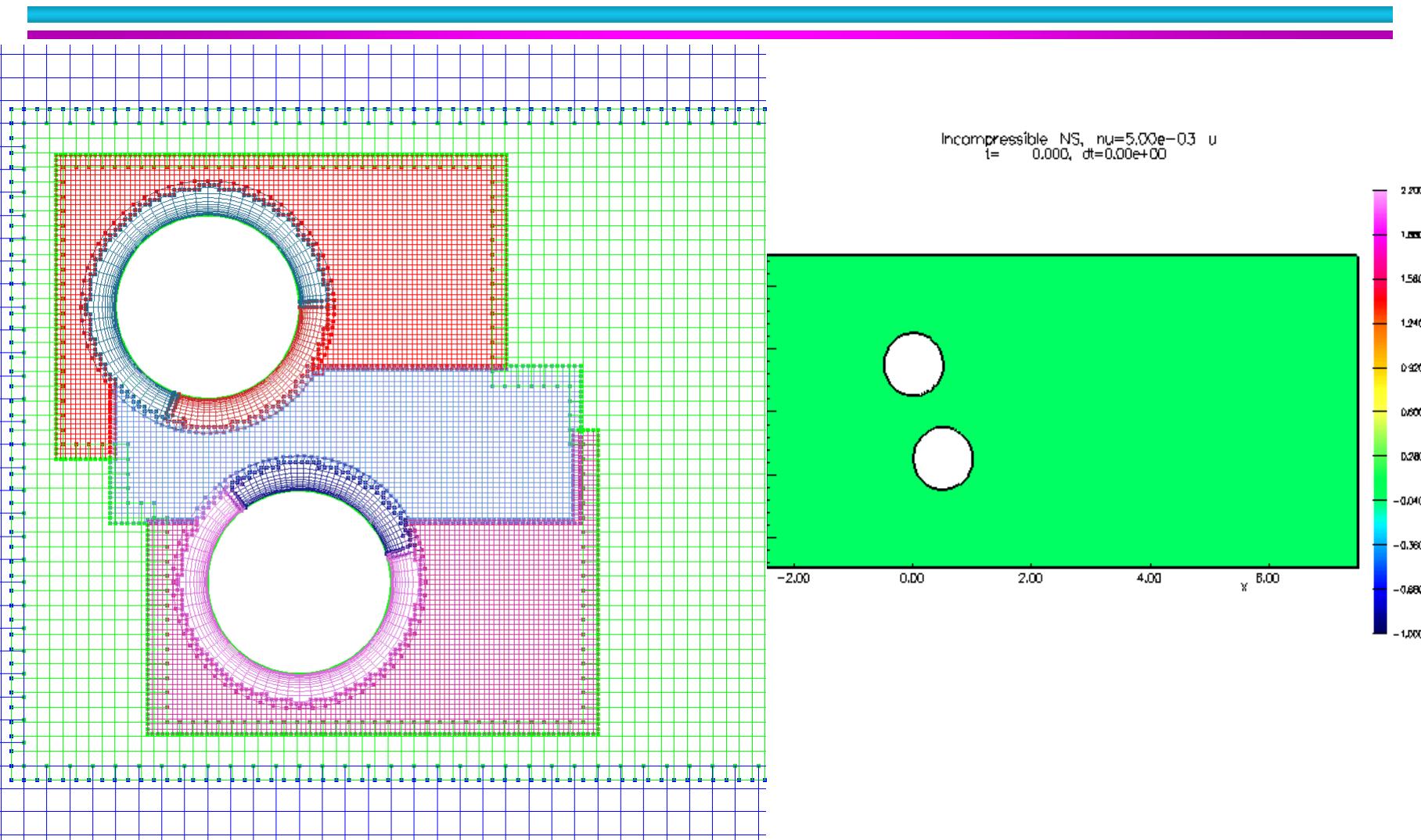
# 3D Incompressible Navier-Stokes

High Level Operations on Overset Grids

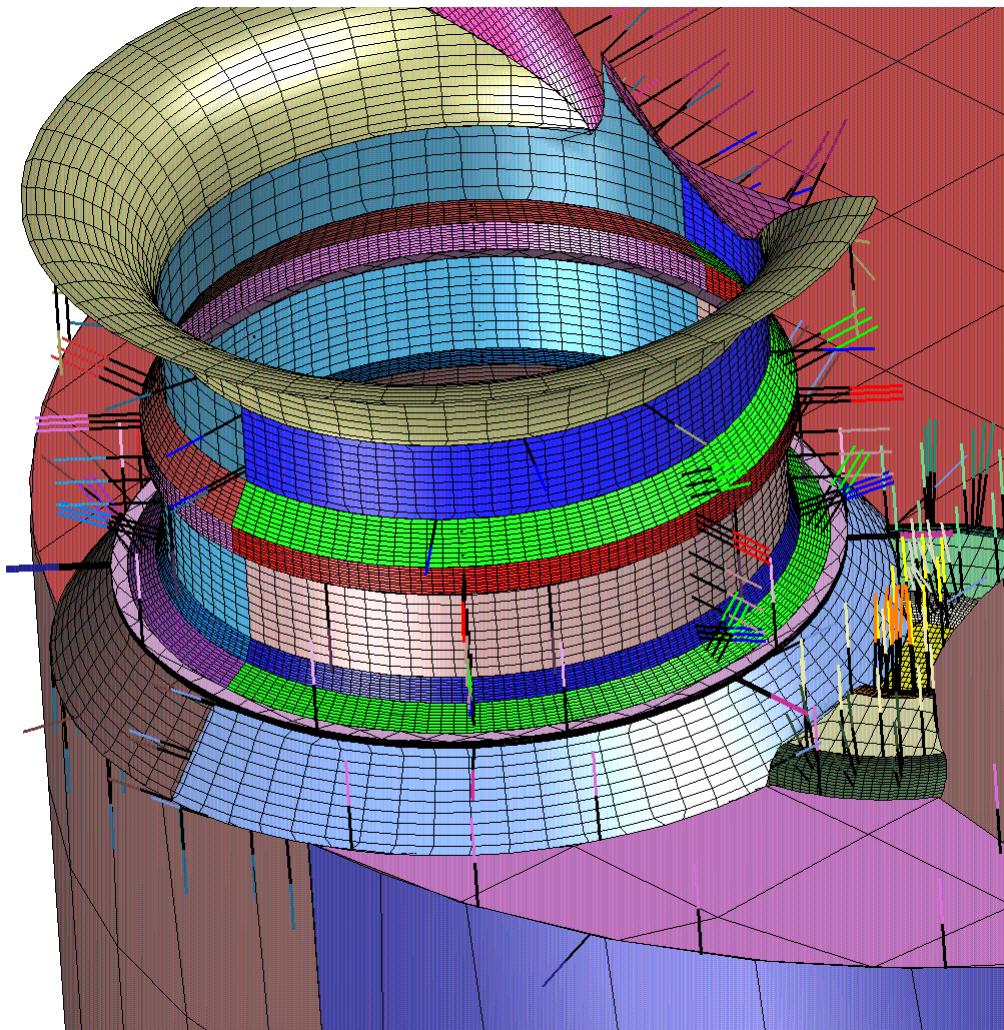
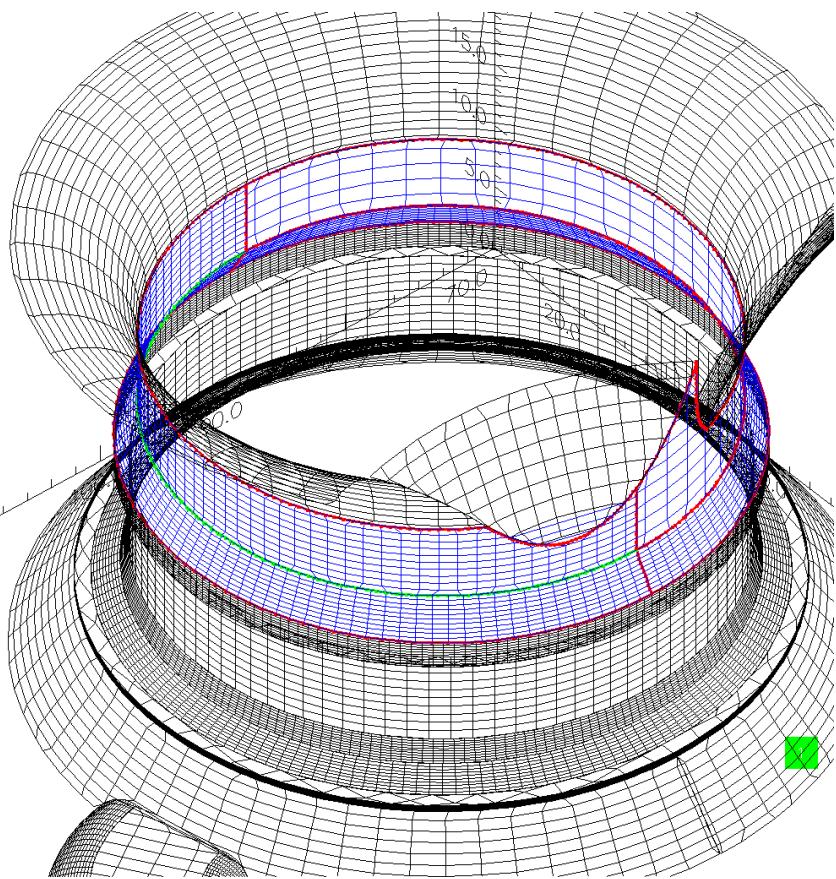
```
float dt=.0005, viscosity=.05;

for (int step=0; step < 100; step++)
{
    // ... forward Euler
    u += dt*( -u.convectiveDerivative() + viscosity*u.laplacian());
    u.applyBoundaryCondition (allVelocityComponents, dirichlet, wall);
    u.interpolate();
    // ... correct by enforcing incompressibility constraint
    u = projection.project (u);
    // ... visualize
    if (step % 10 == 0) ps.streamLines (u);
}
```

# Incompressible Flow Past Double Cylinder Re=200

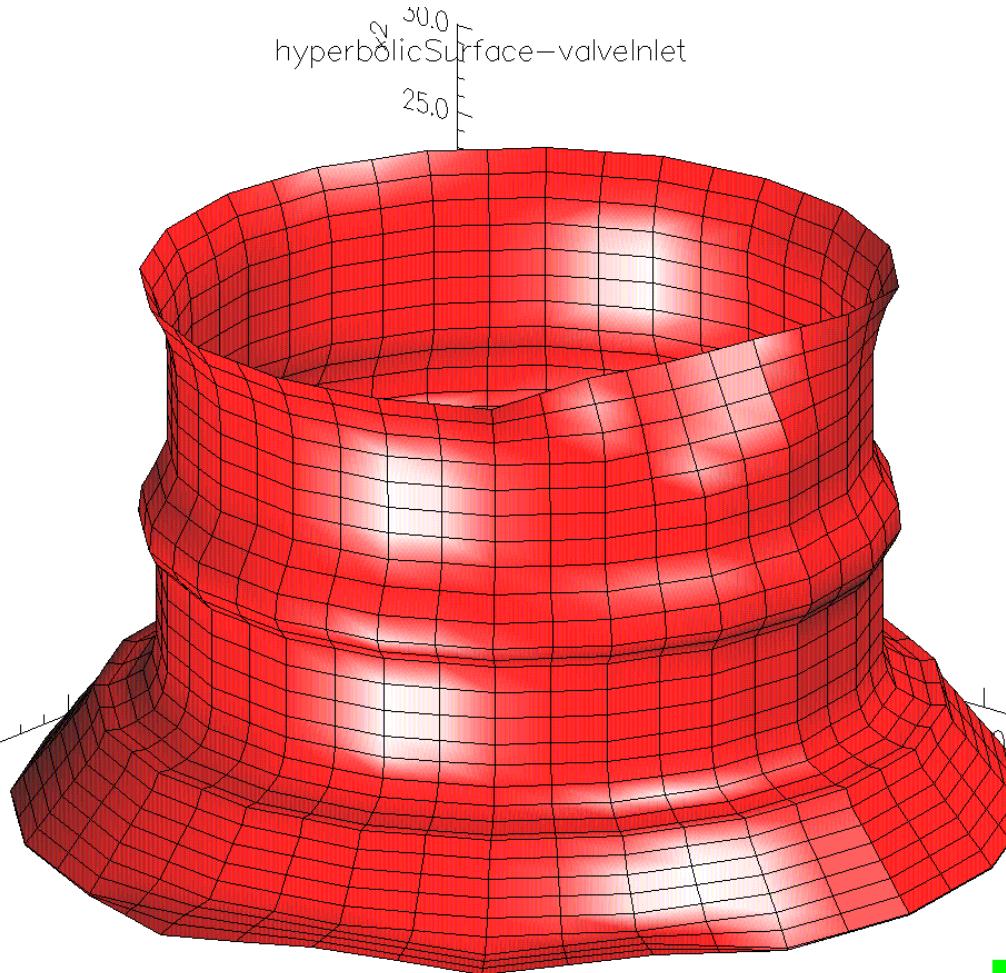
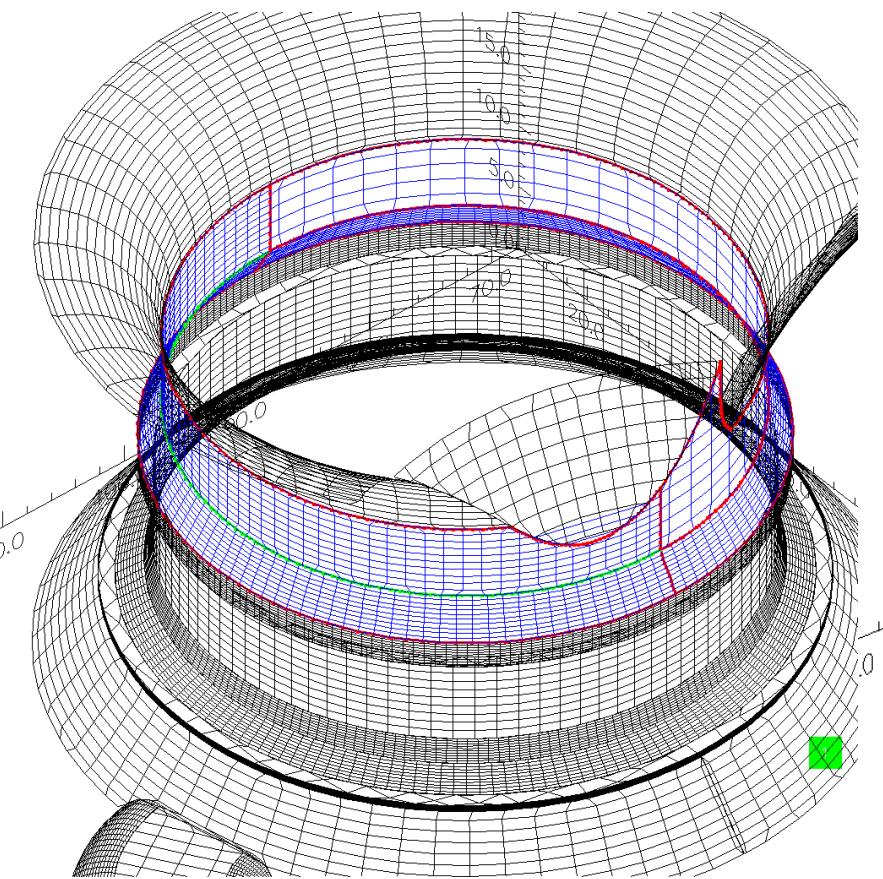


# *Ogen* interactively picks starting curves for hyperbolic surface grid generation



# The Hyperbolic surface grid is generated by marching over the composite surface

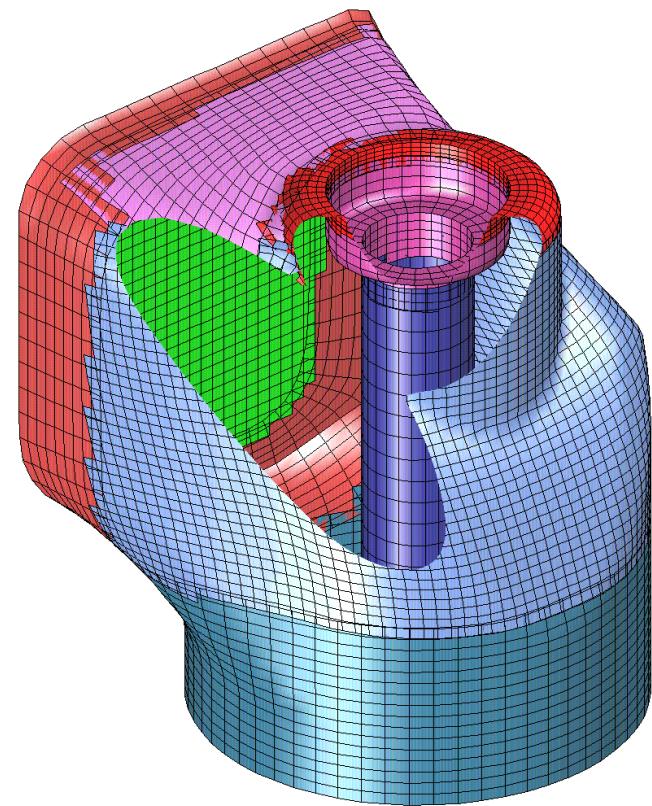
---



# Automatic grid generation for complex geometry is a difficult task with any approach

---

CAD surface

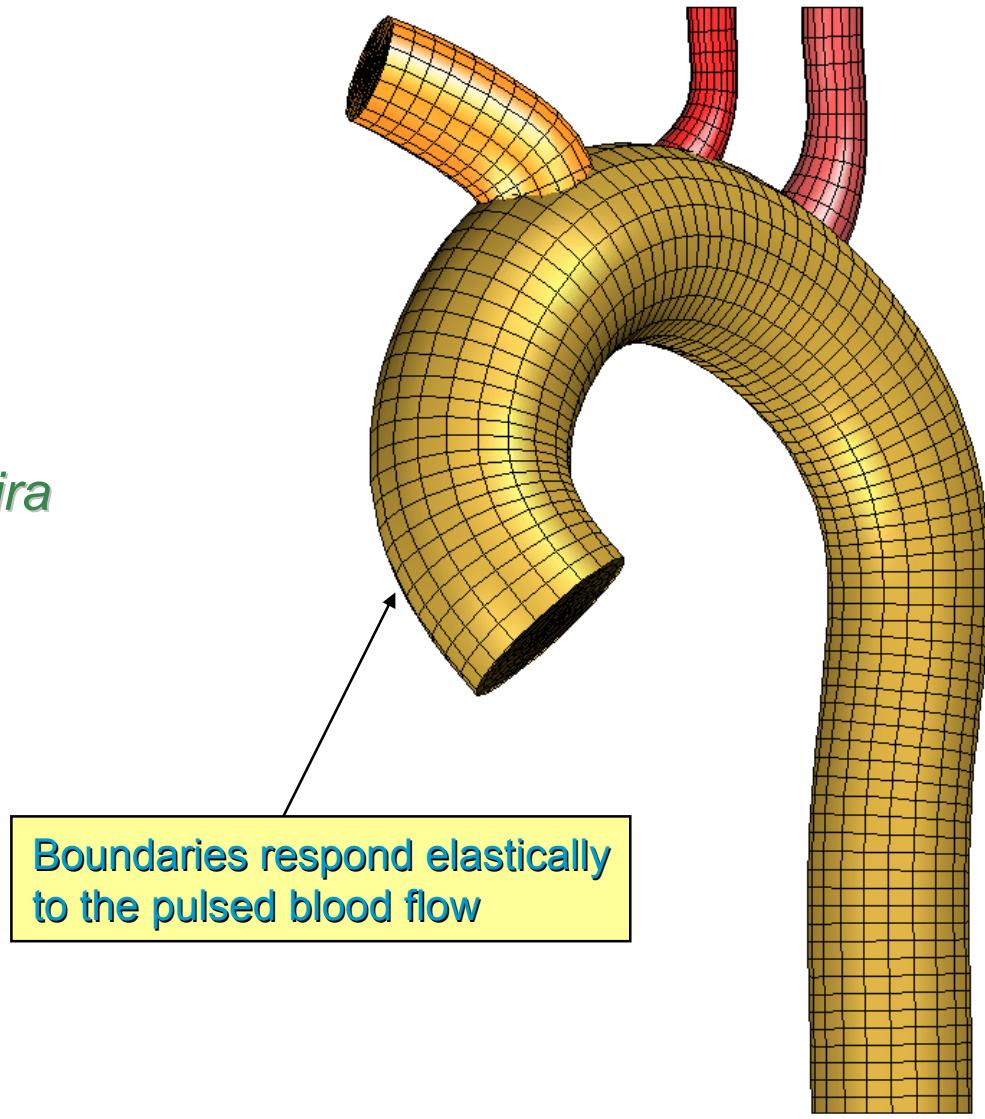


Volume mesh

# Simulation of Arterial blood flow

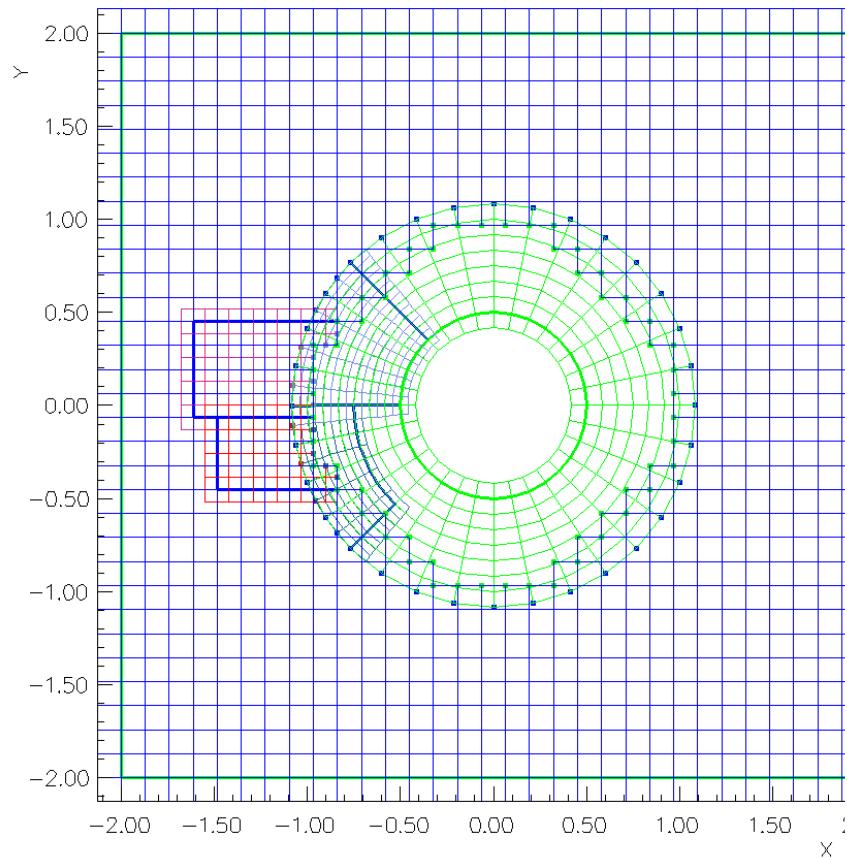
---

*Angela Cheer  
Harry Dwyer  
Thomas Rutaganira  
UC Davis*

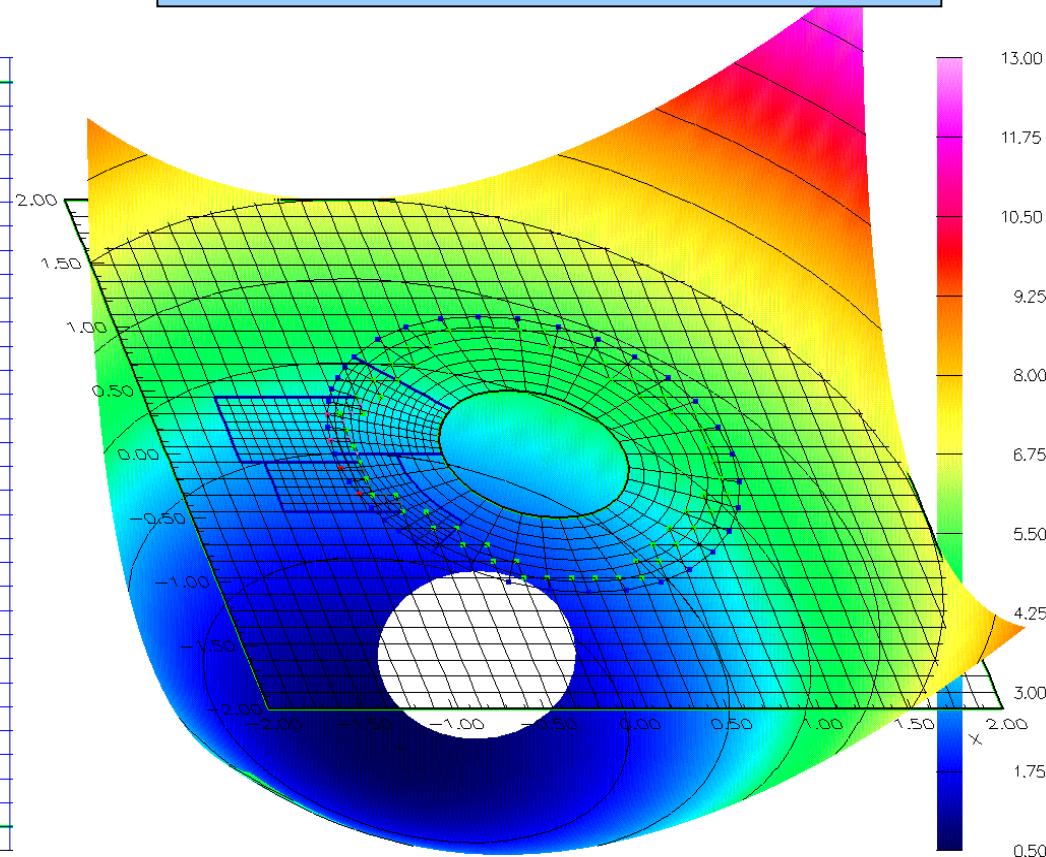


# The infrastructure for supporting AMR within Overture has been implemented

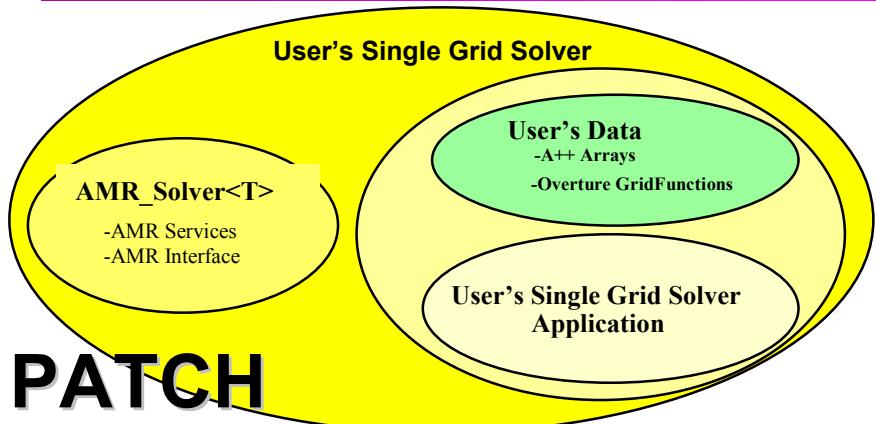
Adaptive Overlapping Grid



Elliptic Solution on Adaptive Overlapping Grid

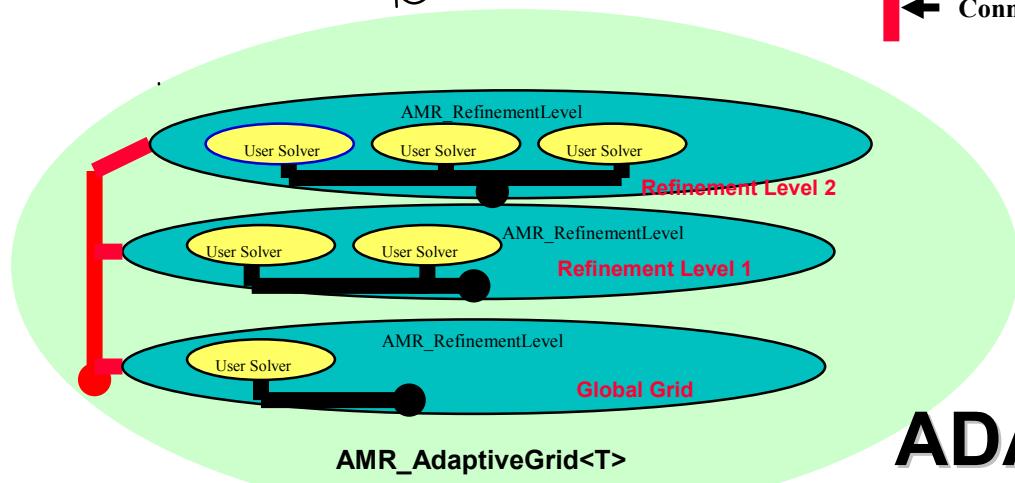
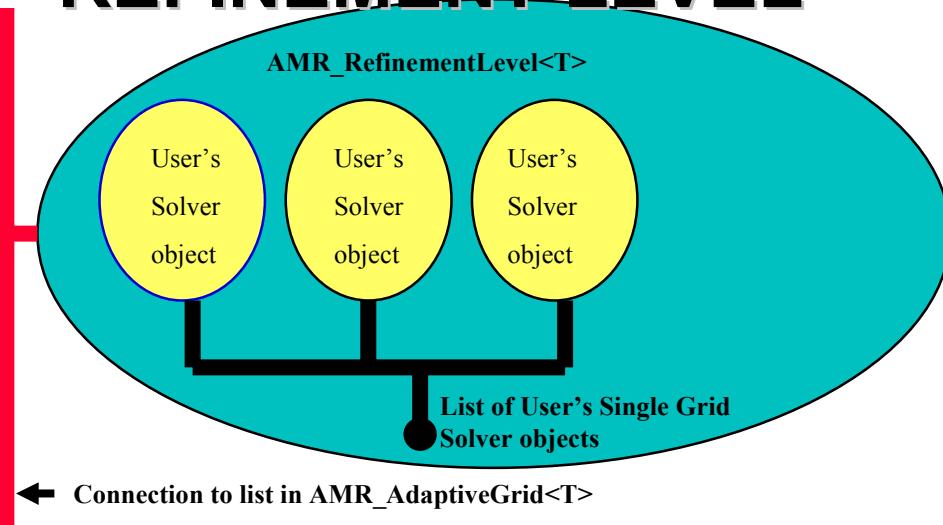


# Structure of AMR++



**PATCH**

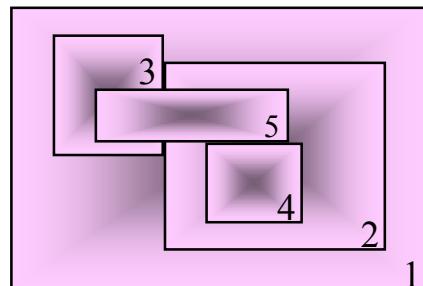
**REFINEMENT LEVEL**



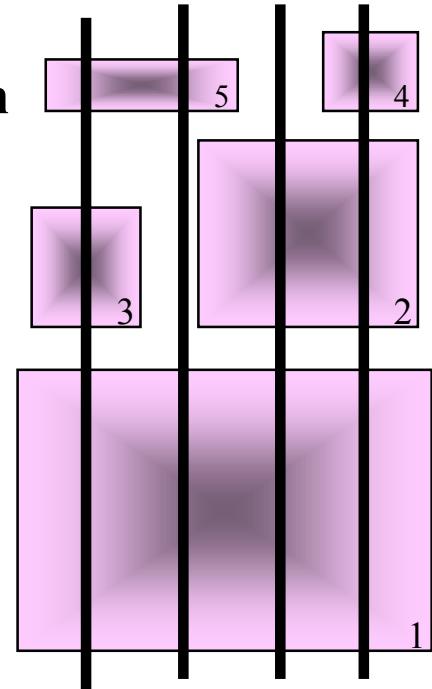
**ADAPTIVE GRID**

# Different Distributions of Adaptive Grids

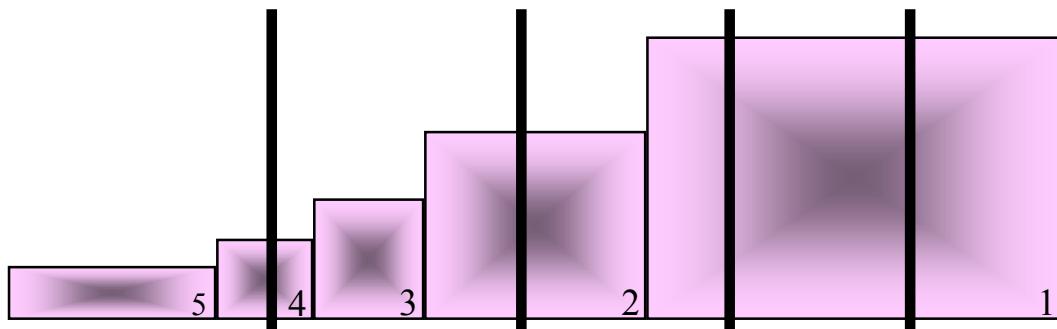
Adaptive Grid



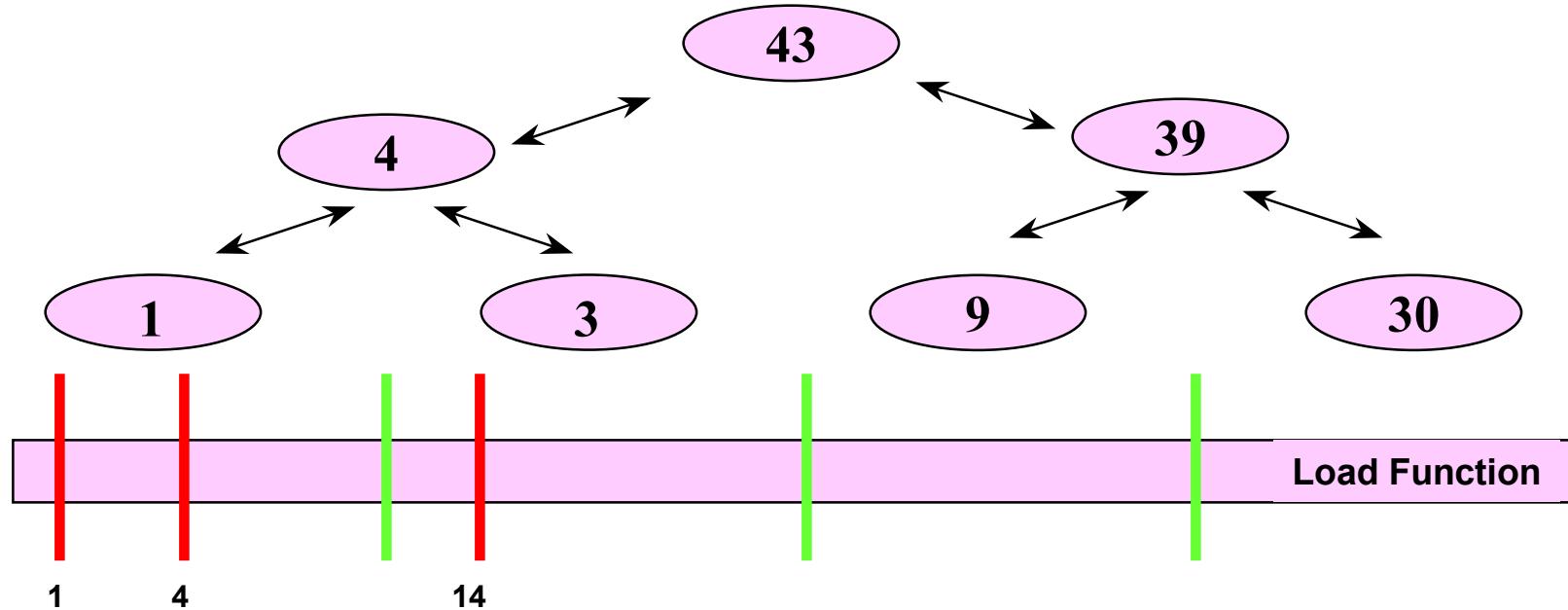
Serial Solution  
of Levels



Parallel Solution of Levels



# Multilevel Load Balancing -- MLB

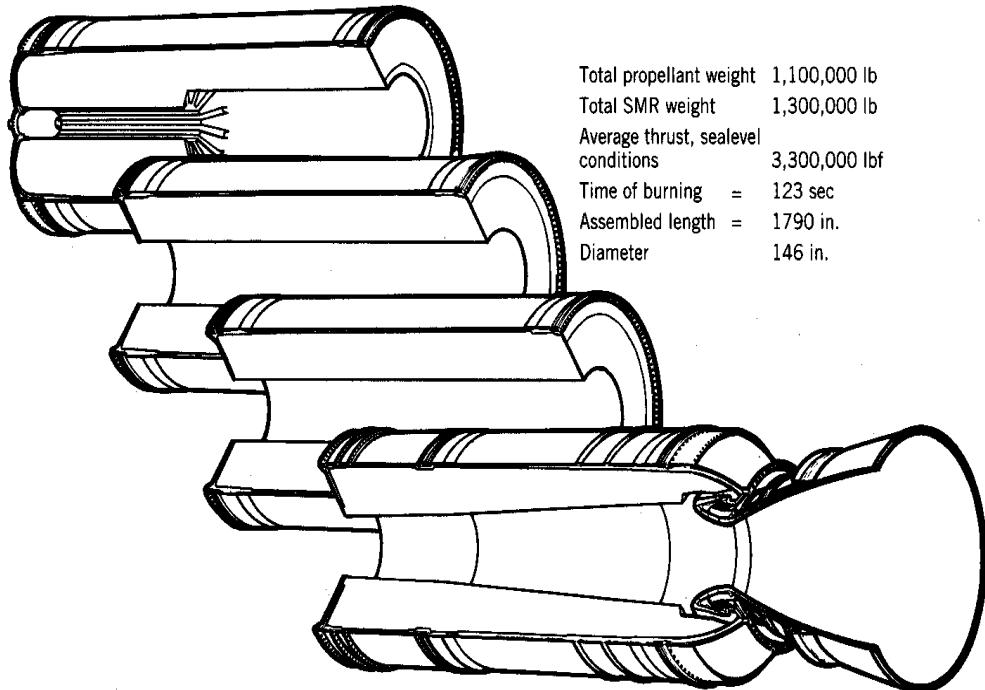


- Multidimensional Load Balancing
- Structured Data Distribution
- Low Complexity/Fast Execution/Direct Method
- Low Complexity Data Movement -  $O(\log p)$

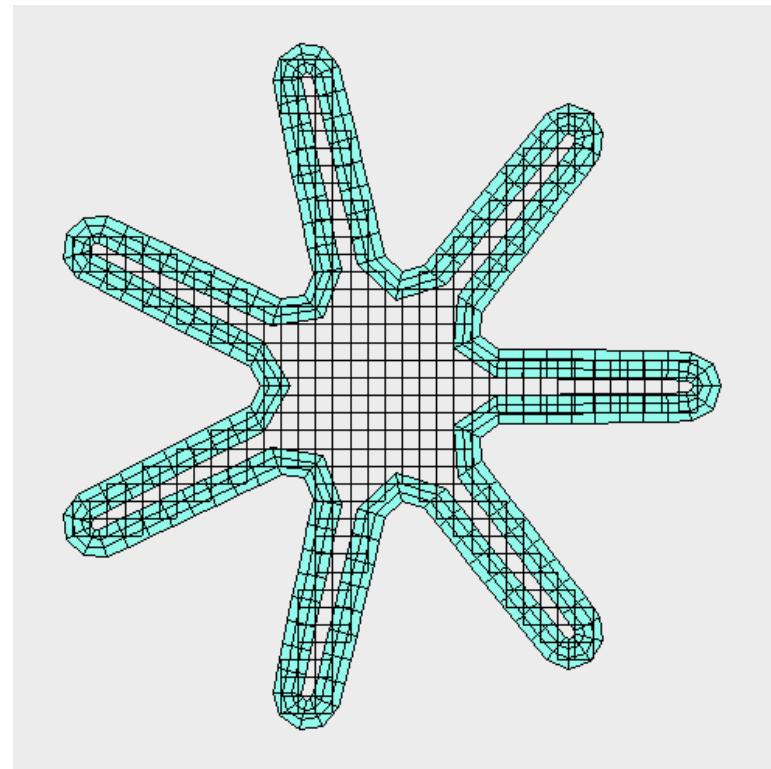
— Old Distribution  
— New Distribution

# Computational Model

SPACE SHUTTLE BOOSTER

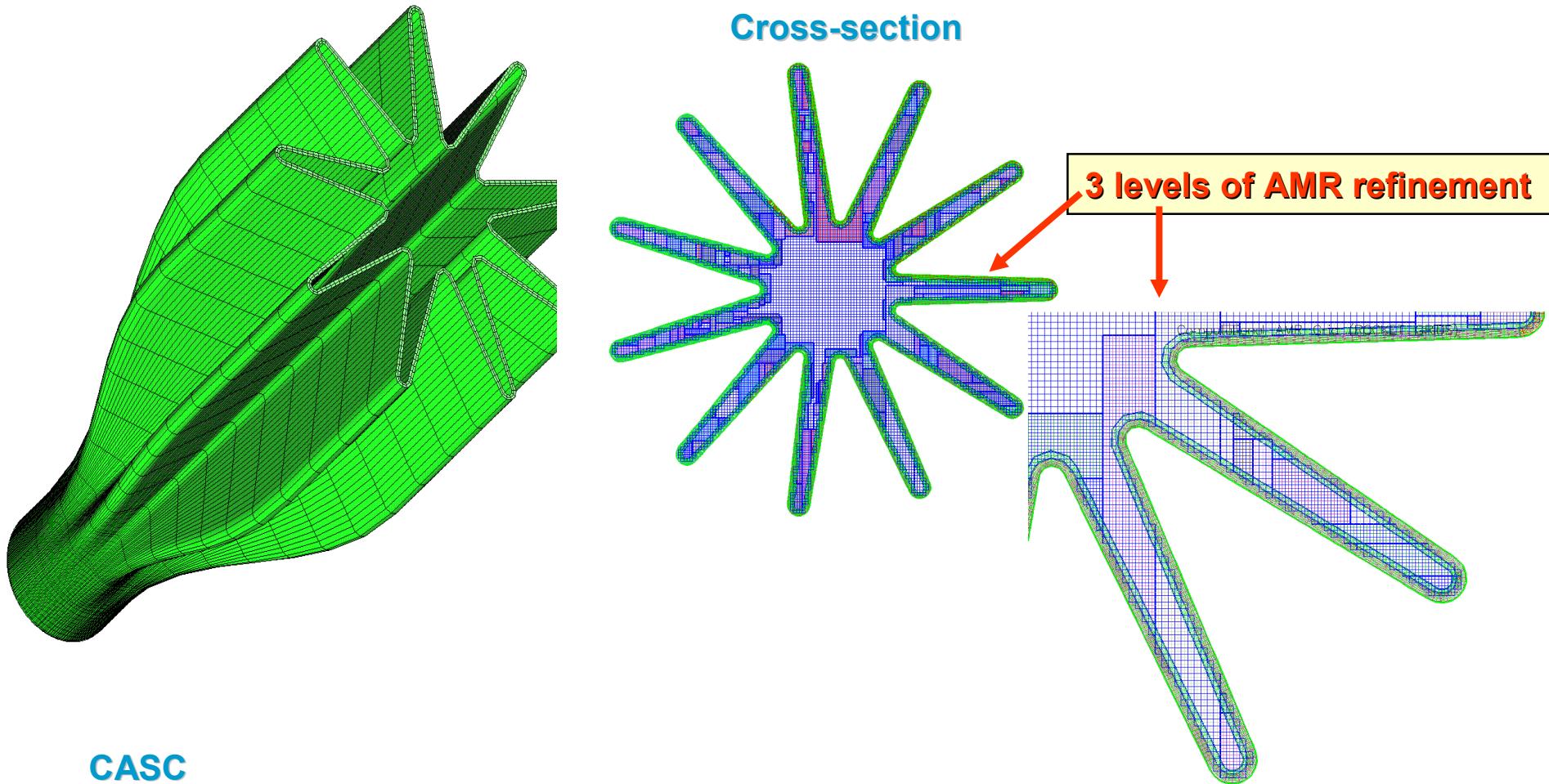


**Fig. 14-2.** Simplified diagram of the four segments of the Space Shuttle solid rocket motor. Details of the thrust vector actuating mechanisms or the ignition system are not shown. (Courtesy of NASA and Thiokol, Inc.)



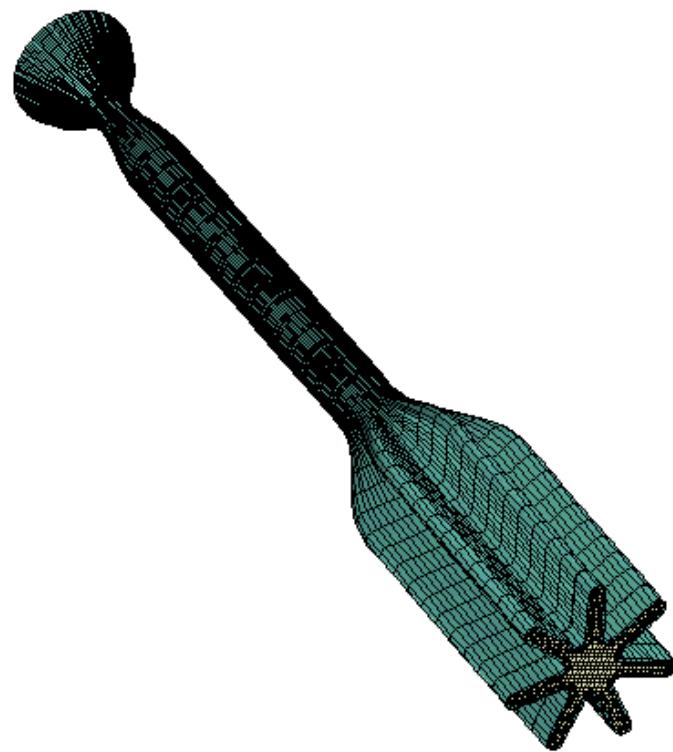
# Solid Rocket Booster (SRB) Fuel Grain

3D Boundary Grid for Interior of SRB



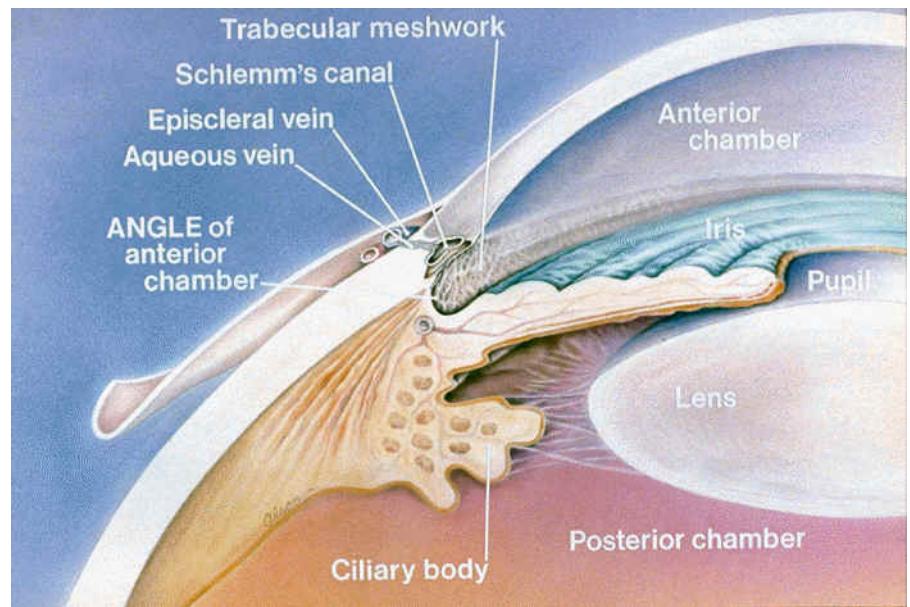
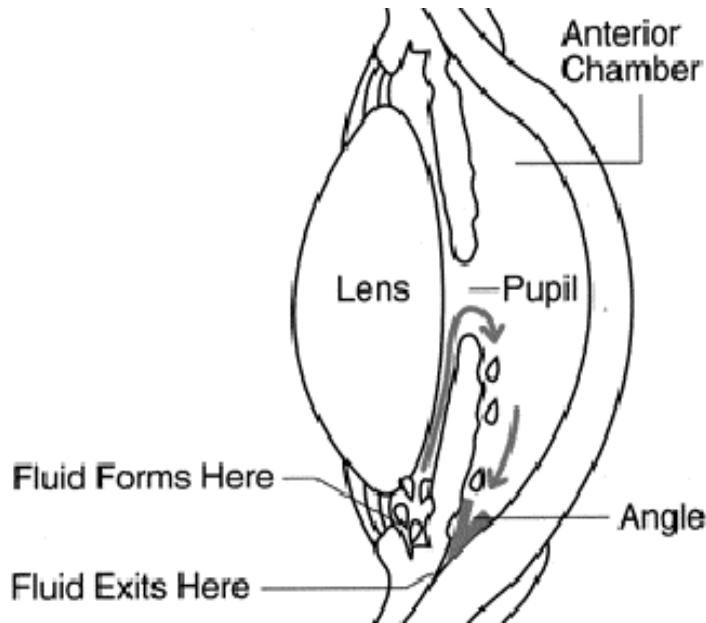
# 3D Rocket Booster Model

---

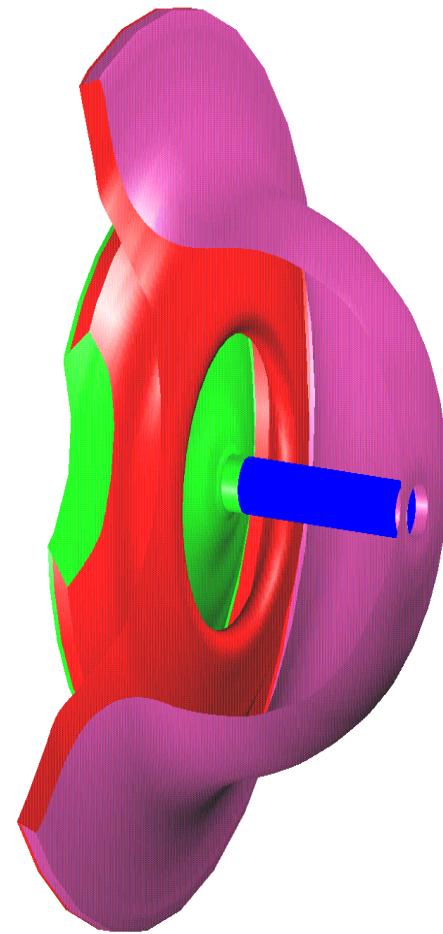
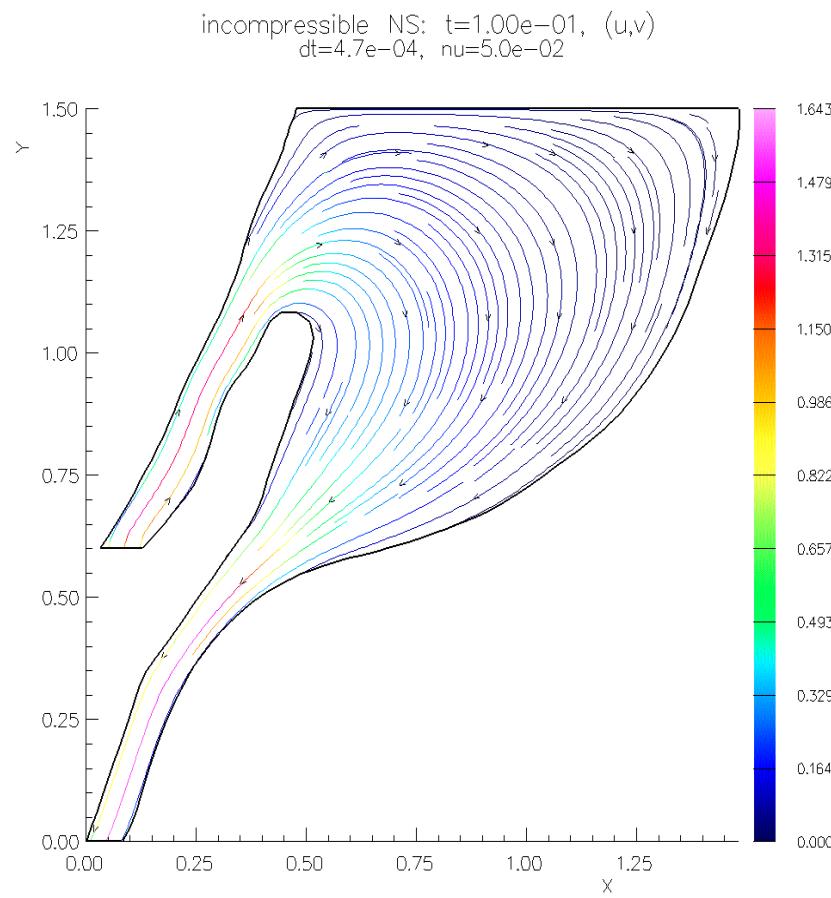


# APPLICATION PROBLEM

## Fluid Flow in The Eye

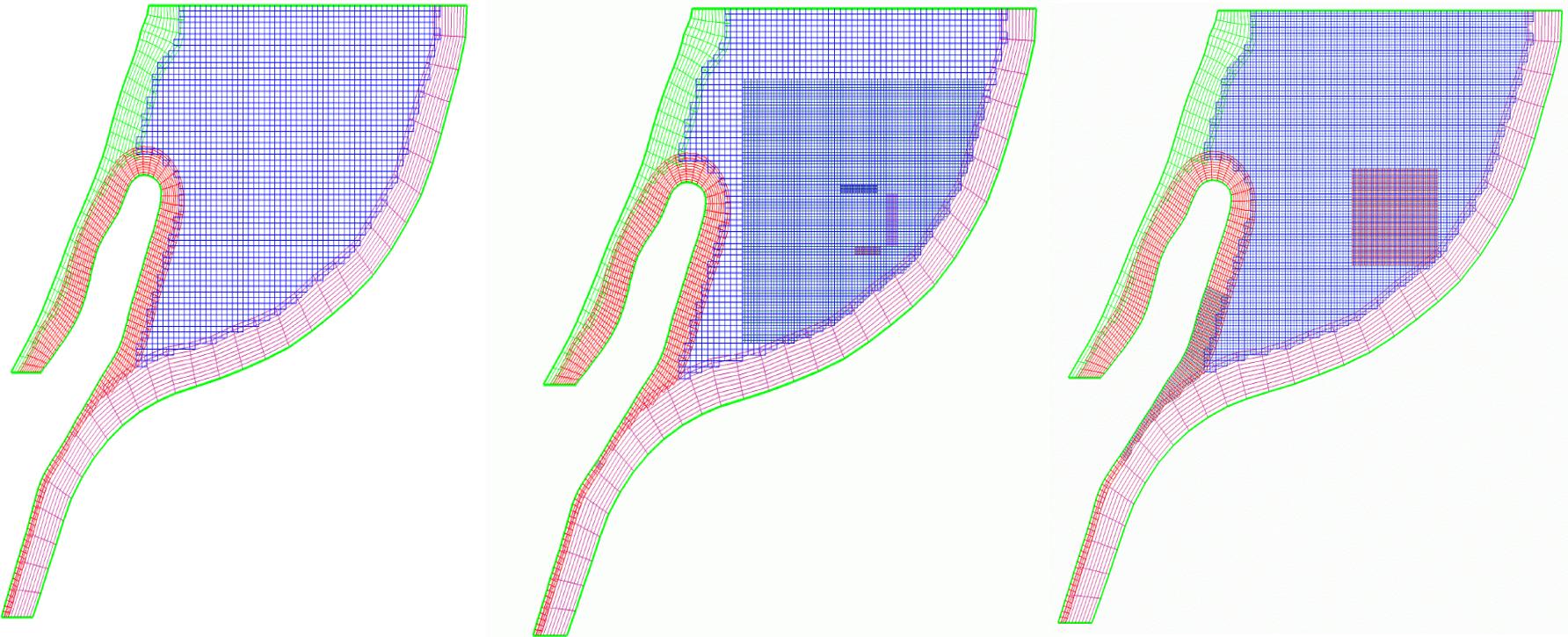


## Preliminary Computations and Computational Grids



# Computational grid with different levels of refinement

---



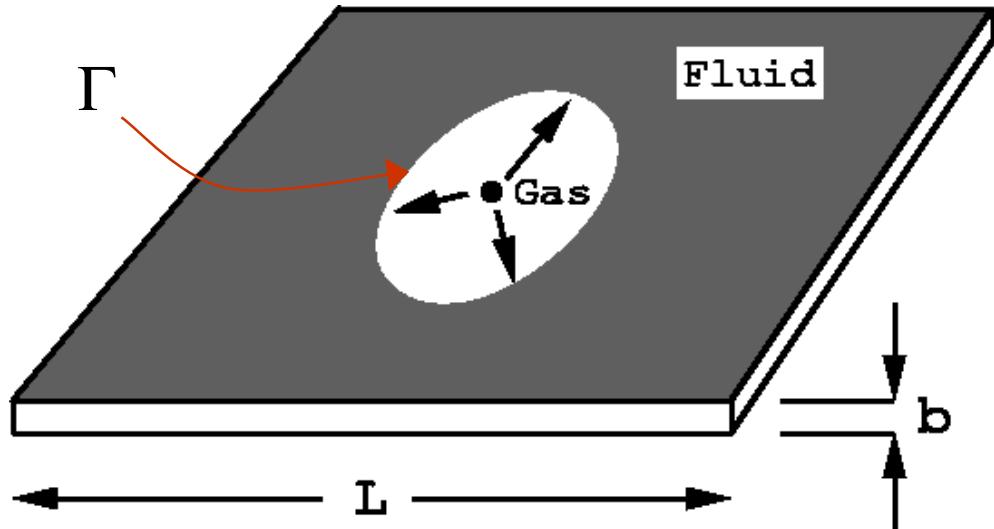
# Moving Overlapping grids have been used to study nonlinear Hele-Shaw flow

---

Unstable expansion of air bubble into non-Newtonian shear-thinning fluid between flat plates

Nonlinear Darcy's Law

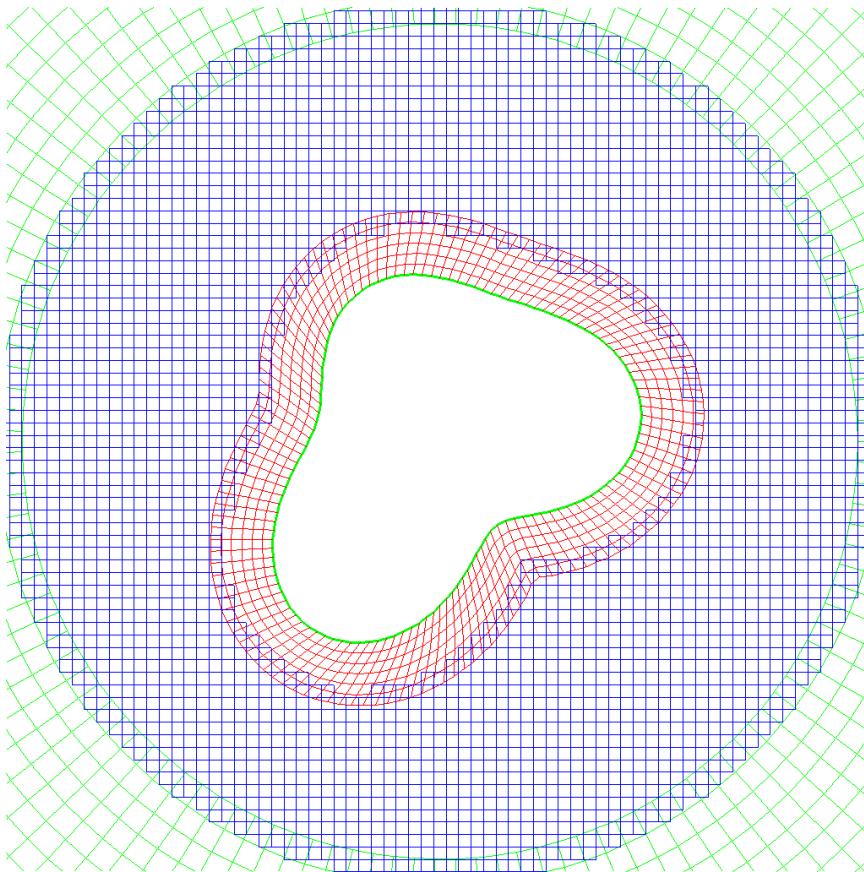
$$\mathbf{u} = \frac{-\nabla p}{\mu (\text{We}^2 |\nabla p|^2)}$$



Petri Fast  
Mike Shelley NYU

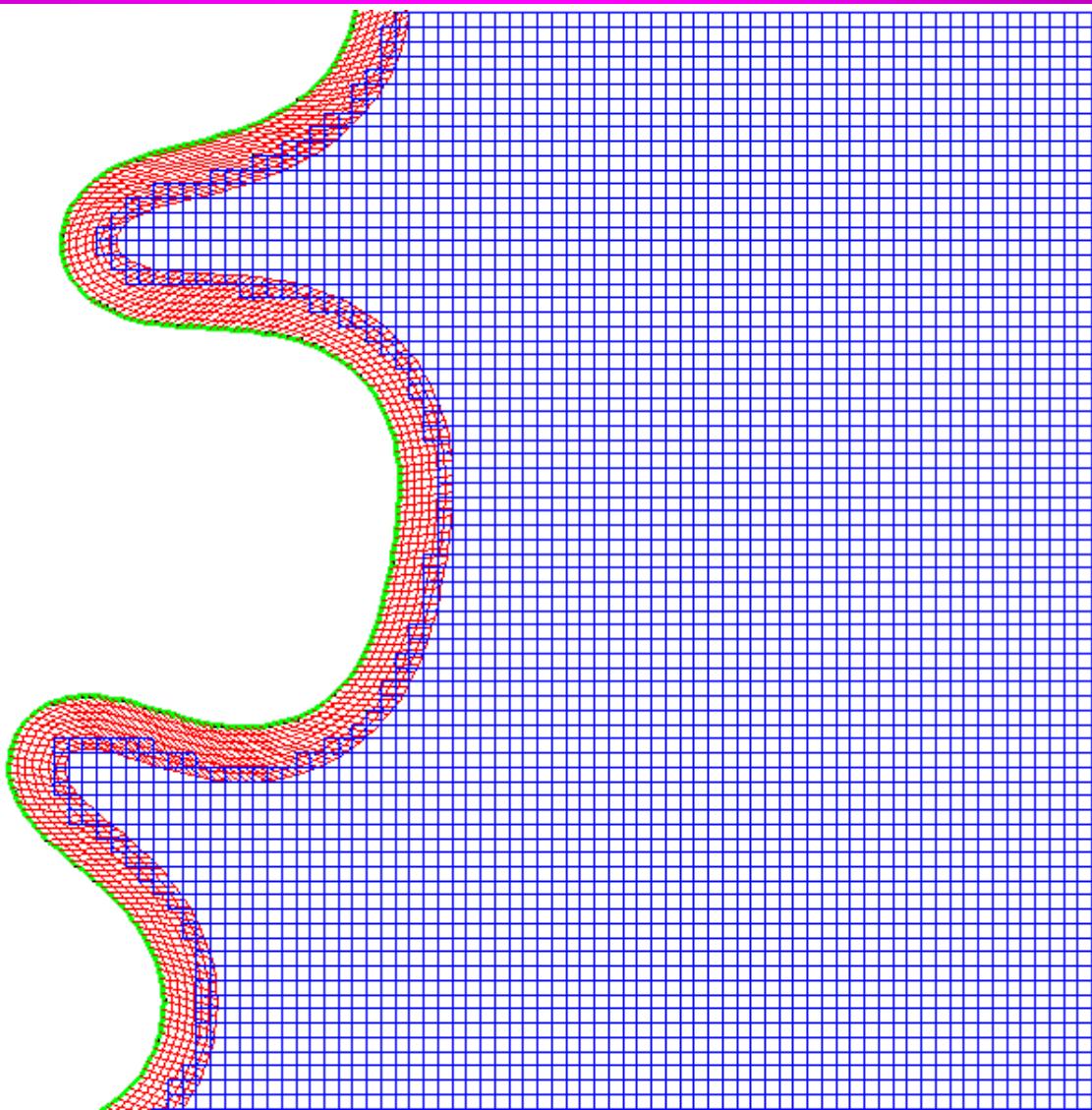
# Fluid region near interface represented with a body-fitted time-evolving grid

---



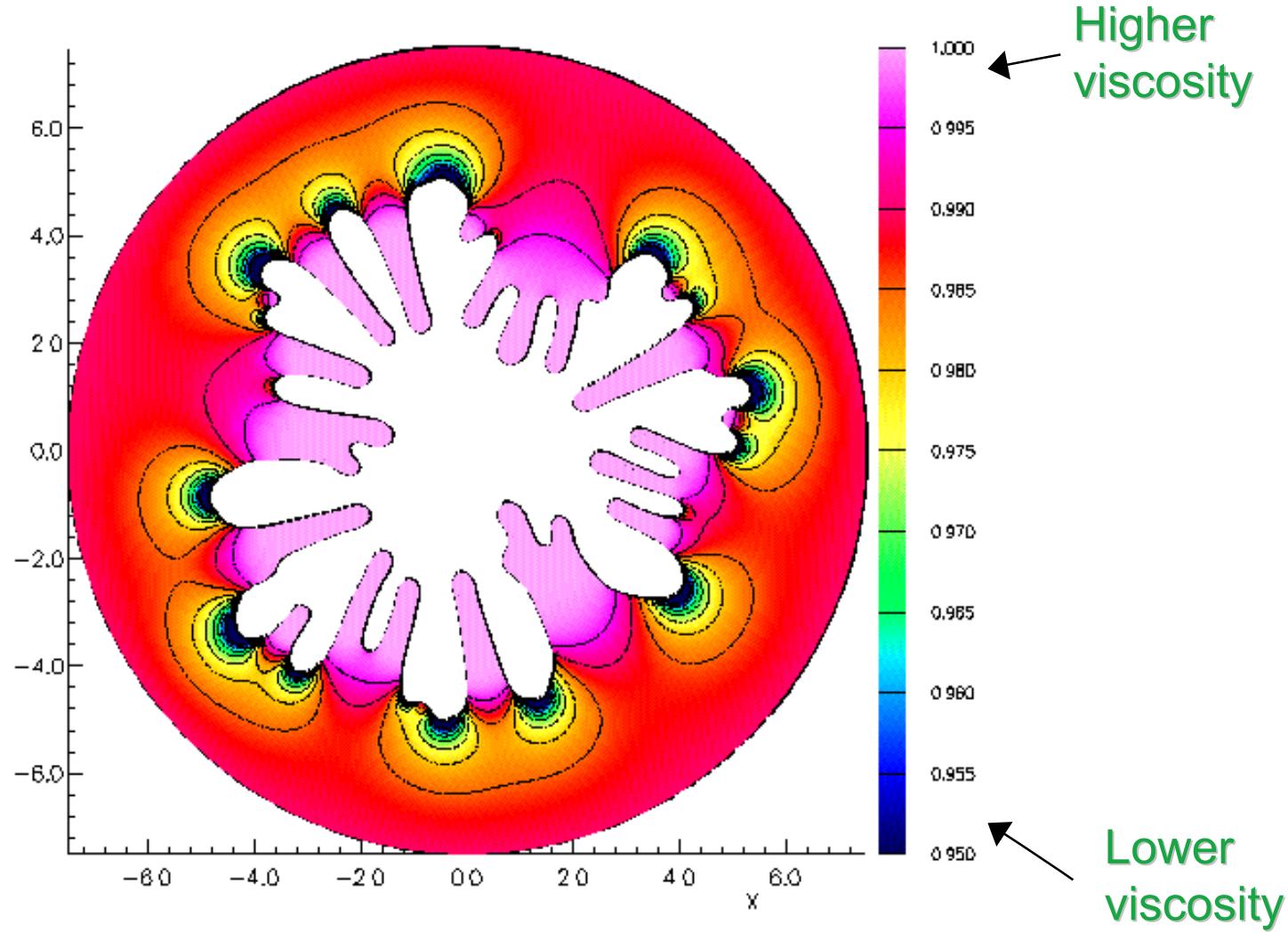
Time = 0.00

# Overlapping grid approach allows resolution of spatial curvature and accurate solution of the pressure equation



Petri Fast

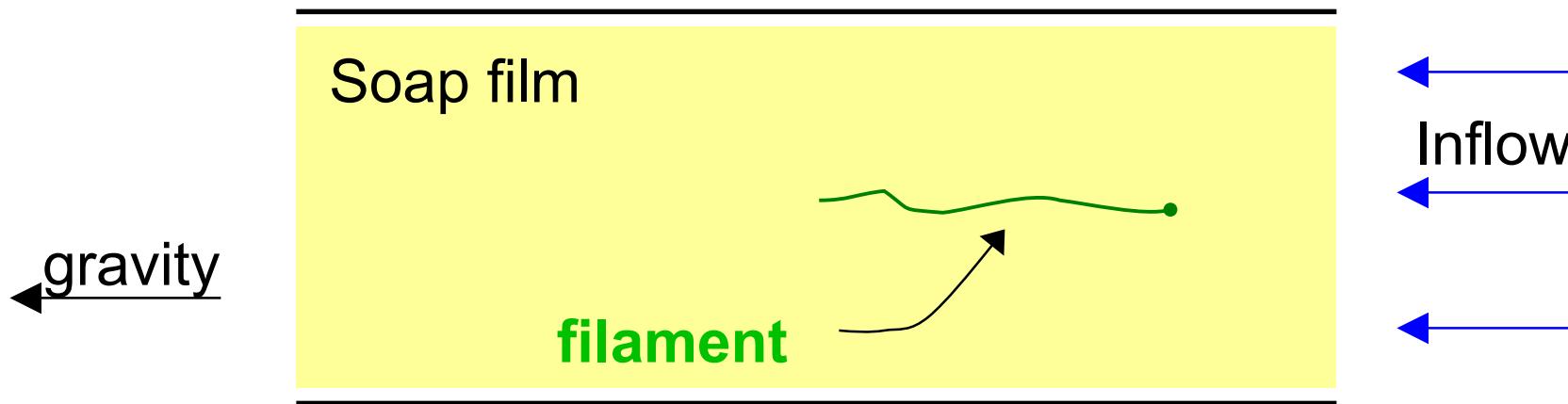
# Shear thinning effect is observed near tips in the overlapping grid computation



Petri Fast

CASC

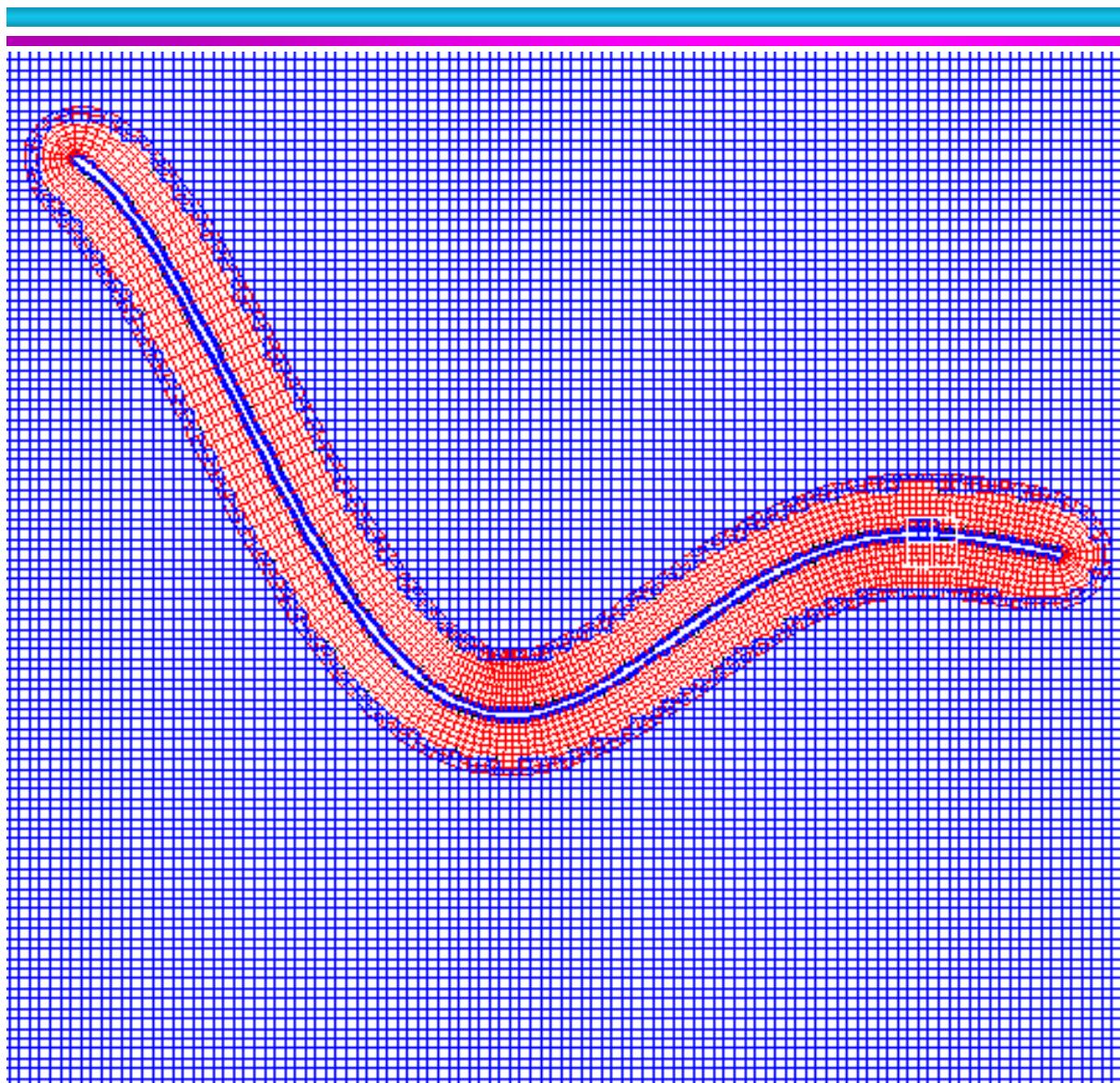
# *Overture will be used to study dynamics of an elastic filament in a soap film*



*Petri Fast  
Bill Henshaw  
Mike Shelley  
J. Zhang  
C. Wiggins  
LLNL&NYU*

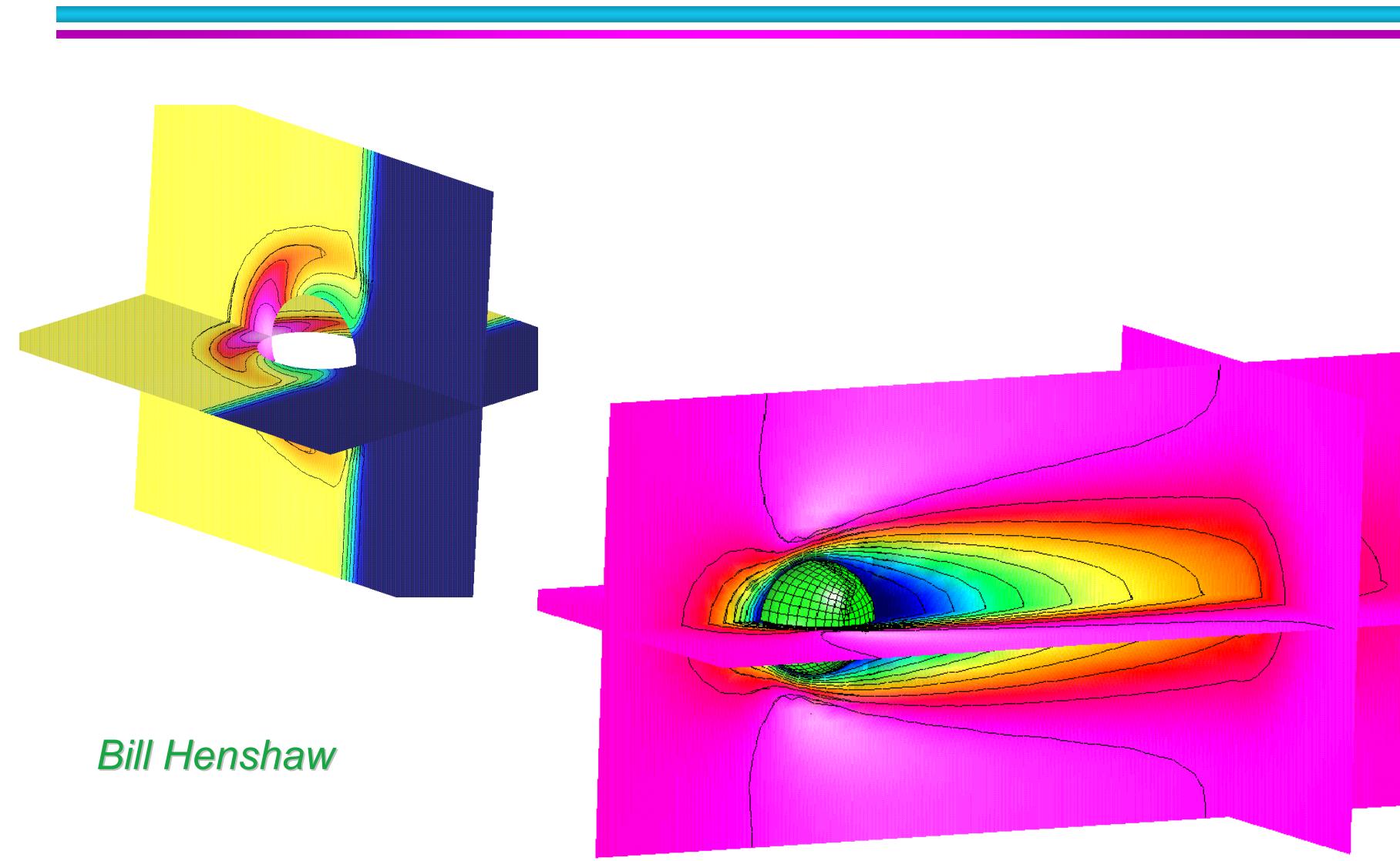
- Experimental realization of 2D flow (Couder et al, Phys. D, 1989)
- Flow is governed by 2D incompressible NS with thickness='density'
- Elastic filament couples to the flow & has dynamics
- Prototype of fluid-structure interactions
- Motivated by insect flight
  - Elastic, moving & deformable wings in viscous flow
- PROBLEMS: moving, complex geom.; stiffness from elasticity

# Filament modeled with a moving boundary responding elastically to forces from the flow



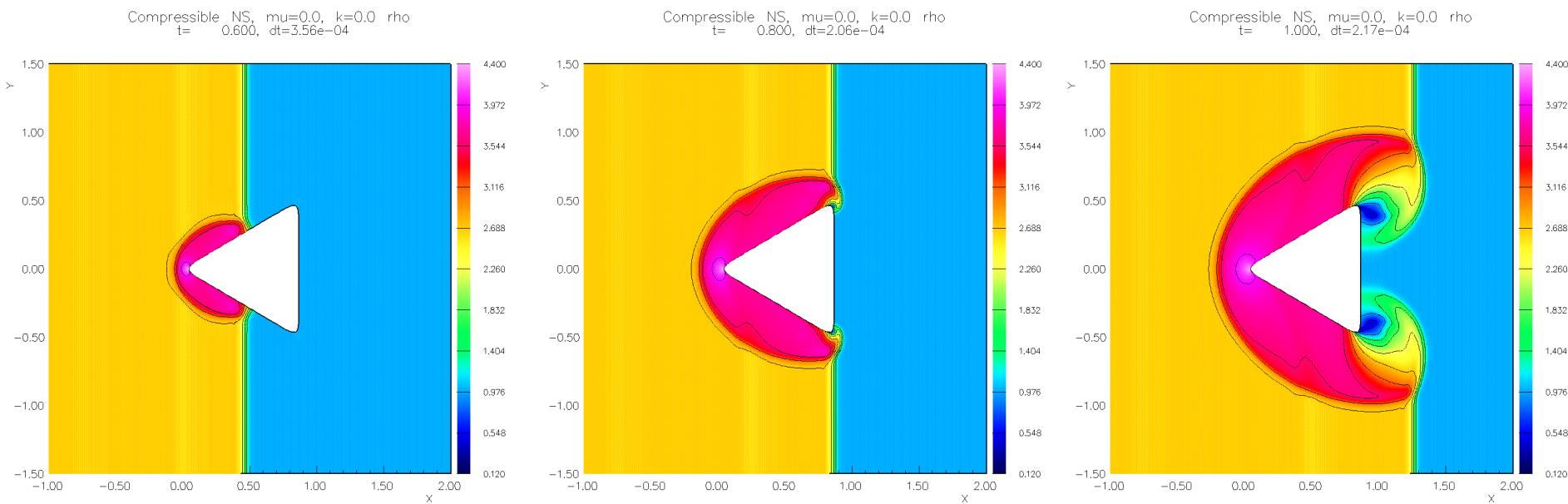
Petri Fast

# Compressible and Incompressible Flow past Sphere



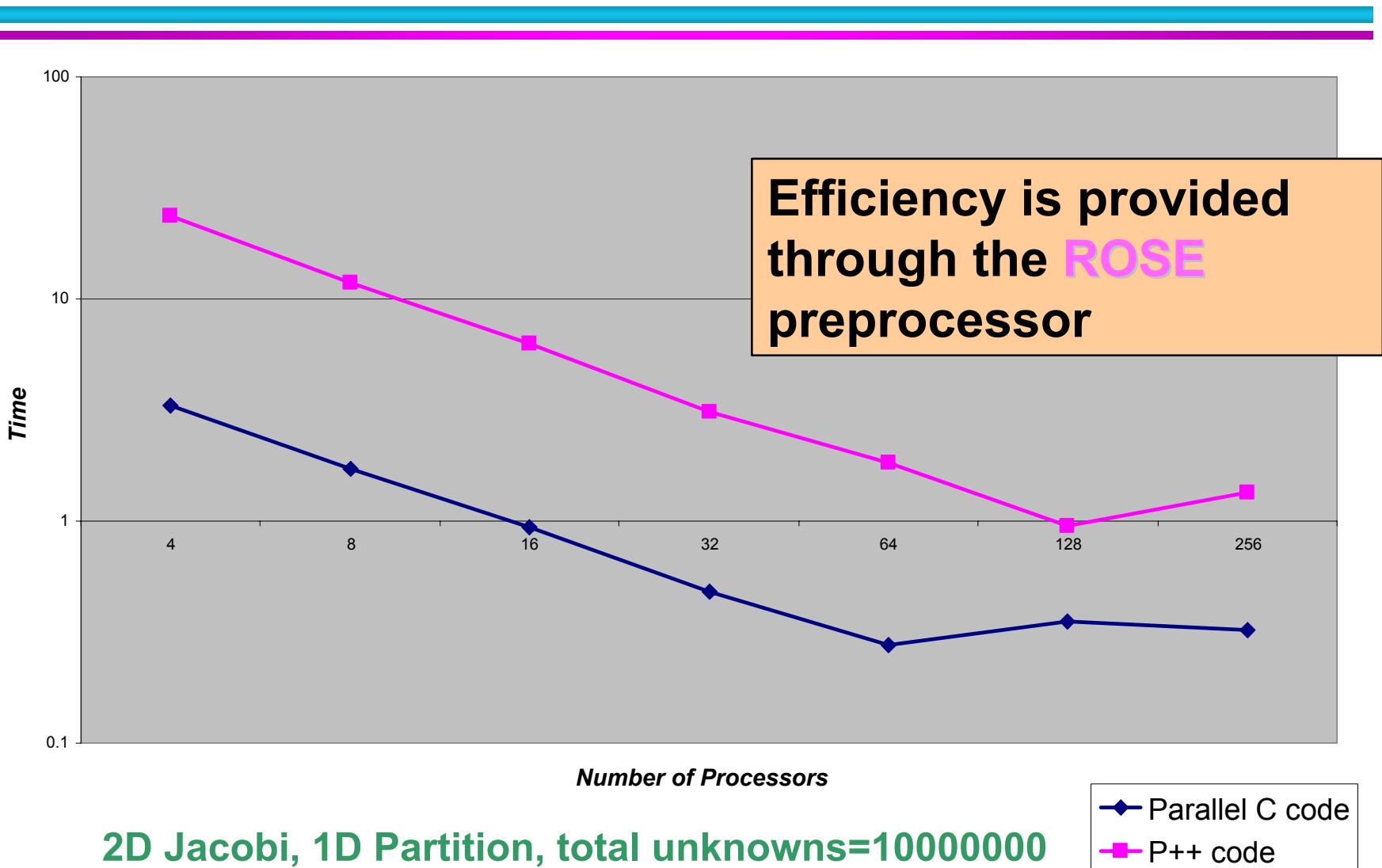
*Bill Henshaw*

# Compressible flow past rounded triangle



Bill Henshaw

# P++ Stencil Operations Scale Well on ASCI Blue Pacific



# **ROSE** transforms high-level *Overture* statements into low-level code the compiler can optimize

P++ Code

```
Index I (1,n,1);
doubleArray Solution(n+1);
doubleArray RHS(n+1);
Solution(I) =
    ((h*h)*RHS(I) + Solution(I+1) + Solution(I-1)) / 0.5;
```

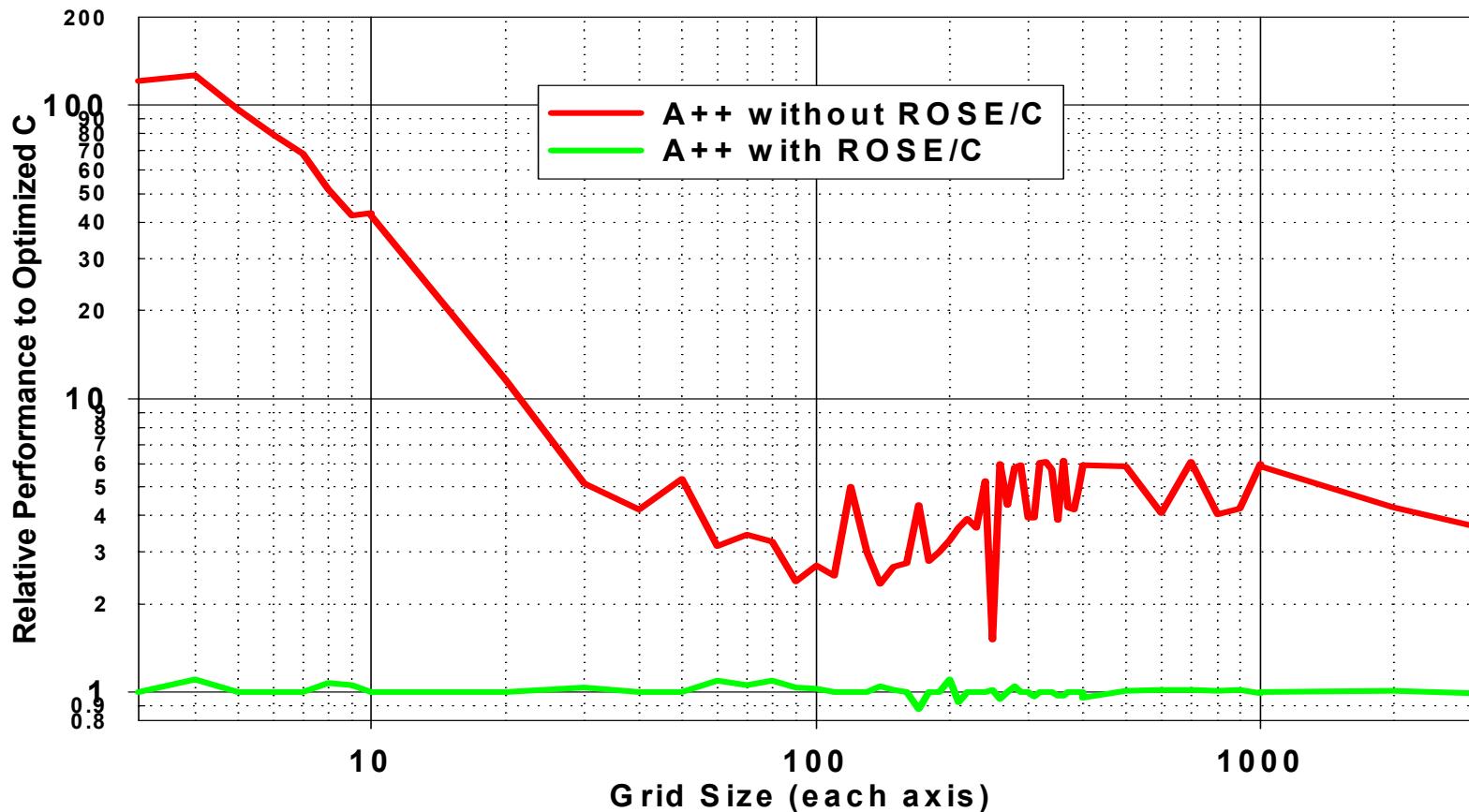
## Automated ROSE Transformation

```
Index I (1,n,1);
doubleArray RHS(n+1);
doubleArray Solution(n+1);
double* restrict RHS_data      = RHS      .getDataPointer();
double* restrict Solution_data = Solution.getDataPointer();

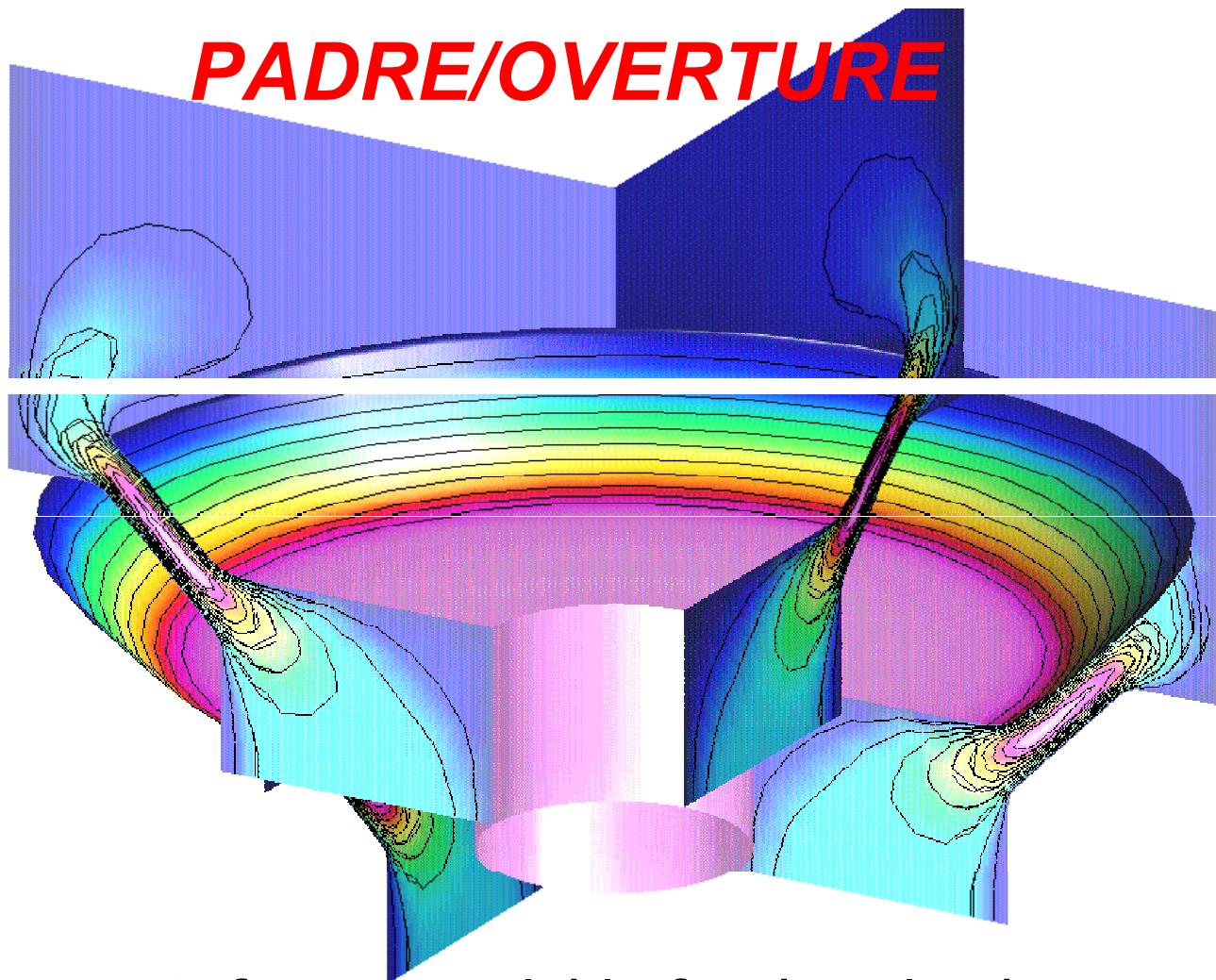
int I_index = 0;
int I_base = I.getBase();
int I_bound = I.getBound();

for (I_index = I_base; I_index < I_bound; I_index++)
    Solution_data[I_index] = ((h*h)*RHS_data[I_index] +
        Solution_data[I_index+1] +
        Solution_data[I_index-1]) / 0.5;
```

# A++ Performance with and without ROSE (Sun Ultra)



# **PADRE/OVERTURE**



Software available for download at  
<http://www.llnl.gov/casc/Overture/>