High-Order Curvilinear Finite Elements for Lagrangian Hydrodynamics

Part II: Axisymmetric Formulation, Parallel Strategy and Numerical Results

Veselin Dobrev, Tzanio Kolev and Robert Rieben

Lawrence Livermore National Laboratory

Numerical Methods for Multi-Material Fluid Flows

Arcachon, France, Sept. 5–9, 2011

LLNL-PRES-497395

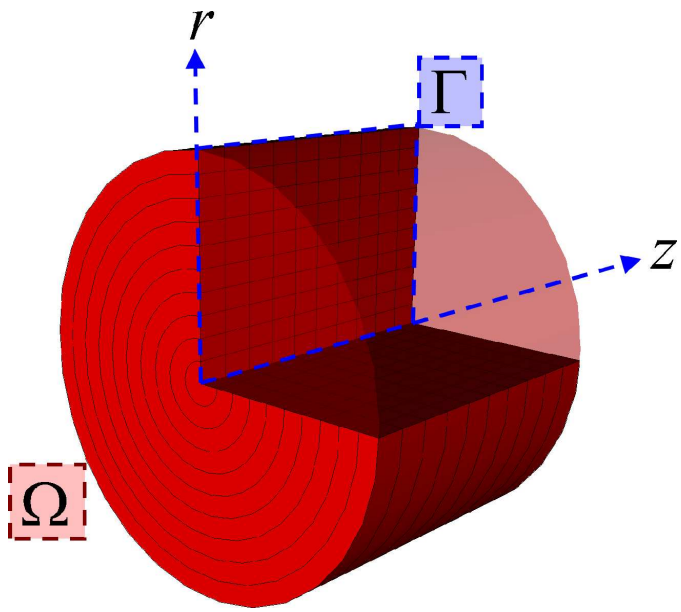
This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

Axisymmetric Problems

The evolution of the particles of a compressible fluid in a Lagrangian reference frame is governed by the following system of differential equations:

Euler's Equations

Momentum Conservation:	$\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \nabla \cdot \sigma$
Mass Conservation:	$\frac{1}{\rho}\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\nabla \cdot \vec{v}$
Energy Conservation:	$\rho \frac{\mathrm{d}e}{\mathrm{d}t} = \sigma : \nabla \vec{v}$
Equation of State:	$p = EOS(e, \rho)$
Equation of Motion:	$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{v}$

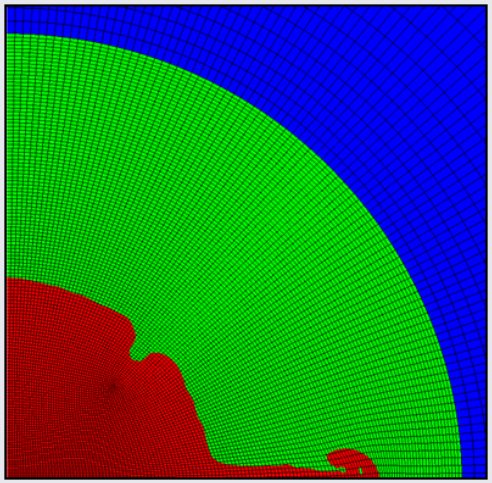


- For 3D problems with axial symmetry, the reduction to a 2D meridian cut Γ provides a significant computational advantage
- Maintaining both *symmetry preservation* and *energy conservation* has proven challenging
- **We will present an extension of our general finite element framework from Part I, which conserves total energy by construction while maintaining good symmetry.**

What Can Go Wrong?

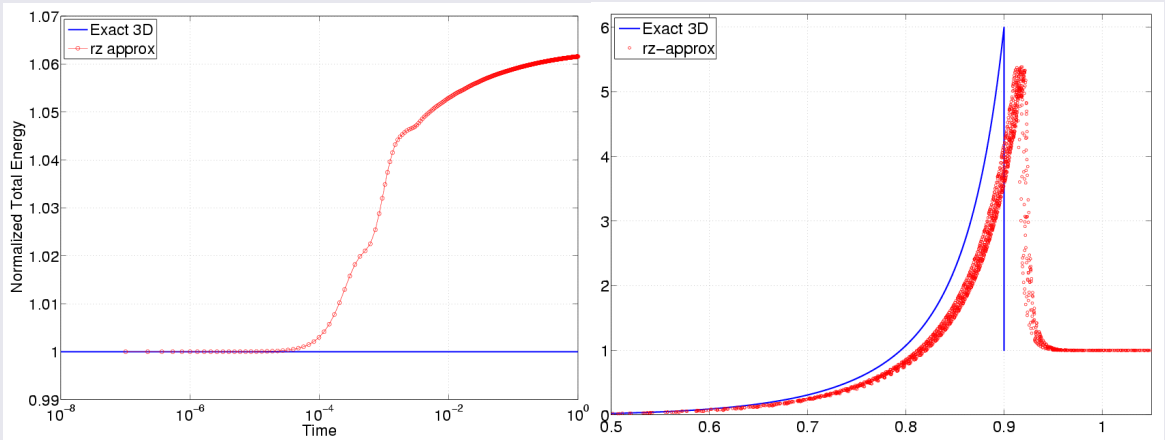
Axisymmetric ICF Test

- ICF-like implosion with radial pressure drive.
- Unstructured butterfly mesh with symmetric initial conditions.
- **Axis jet is numerical and gets worse as mesh is refined.**



Axisymmetric Spherical Sedov Test

- Spherical Sedov blast wave in axisymmetric mode.
- Total energy should remain 1.0 for all time.
- Traditional SGH methods use the Wilkin's *area weighted* approach for computing accelerations
- This preserves symmetry of accelerations but the corresponding energy update may not be conservative.
- **6% spurious gain in energy leads to incorrect shock speed and does not improve under mesh refinement.**



Symmetry breaking and lack of energy conservation lead to non-physical results

Overview of Axisymmetric Methods

Staggered-grid hydro (SGH) methods

- **M. L. Wilkins**, *Calculations of elastic-plastic flow*, Meth. Comput. Phys., 3, 1964.
- **P. Whalen**, *Algebraic limitations on two dimensional hydrodynamics simulations*, J. Comput. Phys. 124, pp. 46-54, 1996.
- **E. Caramana, D. Burton, M. Shashkov and P. Whalen**, *The construction of compatible hydrodynamics algorithms utilizing conservation of total energy*, J. Comput. Phys., 146, pp. 227–262, 1998.
- **L. Margolin and M. Shashkov**, *Using a curvilinear grid to construct symmetry-preserving discretizations for Lagrangian gas dynamics*, J. Comput. Phys. 149, pp. 389-417, 1999.
- **A. Barlow, D. Burton and M. Shashkov**, *Compatible, energy and symmetry preserving 2D Lagrangian hydrodynamics in rz cylindrical coordinates*, Proc. Comp. Sci., 1(1), ICCS 2010, pp. 1893–1901, 2010.

Finite element-based methods

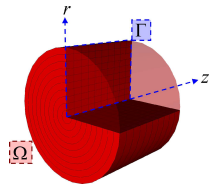
- **P. Matejovic and V. Adamik**, *A one-point integration quadrilateral with hourglass control in axisymmetric geometry*, Comp. Meth. Appl. Mech. Eng., 70(3), pp. 301–320, 1988.
- **P. Matejovic**, *Quadrilateral with high coarse-mesh accuracy for solid mechanics in axisymmetric geometry*, Comp. Meth. Appl. Mech. Eng., 88(2), pp. 241–258, 1991.

Cell-centered methods

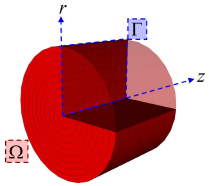
- **P.-H. Maire**, *A high-order cell-centered Lagrangian scheme for two-dimensional compressible fluid flows on unstructured meshes*, J. Comput. Phys., 228 (7), pp. 2391–2425, 2009.

Position and Strong Mass Conservation

- We introduce a 2D Curvilinear FEM mesh on $\Gamma(t)$ with zones $\Gamma_z(t)$ and denote the 2D position vector, parametric mapping and Jacobian with $\mathbf{x}(t)$, Φ_z and $\mathbf{J}_z$.



Position and Strong Mass Conservation



- We introduce a 2D Curvilinear FEM mesh on $\Gamma(t)$ with zones $\Gamma_z(t)$ and denote the 2D position vector, parametric mapping and Jacobian with $\mathbf{x}(t)$, Φ_z and $\mathbf{J}_z$.

Axisymmetric strong mass conservation

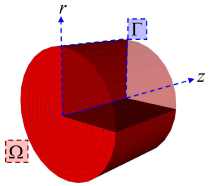
- Let $\Omega'(t)$ be the revolution of an arbitrary set $\Gamma'(t) \subset \Gamma(t)$. Then

$$\int_{\Omega'(t)} \rho(t) = \int_{\Omega'(t_0)} \rho(t_0) \quad \longrightarrow \quad 2\pi \int_{\Gamma'(t)} r\rho(t) = 2\pi \int_{\Gamma'(t_0)} r\rho(t_0)$$

- Therefore the **strong mass conservation principle** in RZ takes the form

$$r(t)\rho(t)|\mathbf{J}_z(t)| = r(t_0)\rho(t_0)|\mathbf{J}_z(t_0)|$$

Position and Strong Mass Conservation



- We introduce a 2D Curvilinear FEM mesh on $\Gamma(t)$ with zones $\Gamma_z(t)$ and denote the 2D position vector, parametric mapping and Jacobian with $\mathbf{x}(t)$, Φ_z and $\mathbf{J}_z$.

Axisymmetric strong mass conservation

- Let $\Omega'(t)$ be the revolution of an arbitrary set $\Gamma'(t) \subset \Gamma(t)$. Then

$$\int_{\Omega'(t)} \rho(t) = \int_{\Omega'(t_0)} \rho(t_0) \quad \longrightarrow \quad 2\pi \int_{\Gamma'(t)} r\rho(t) = 2\pi \int_{\Gamma'(t_0)} r\rho(t_0)$$

- Therefore the **strong mass conservation principle** in RZ takes the form

$$r(t)\rho(t)|\mathbf{J}_z(t)| = r(t_0)\rho(t_0)|\mathbf{J}_z(t_0)|$$

Axisymmetric mass matrices

- Let $\mathbf{w}$ and ϕ be the kinematic and thermodynamic finite element basis functions on Γ.
- Define the weighted axisymmetric mass matrices

$$\mathbf{M}_\mathbf{v}^{\mathbf{rz}} = \int_{\Gamma(t)} r\rho \mathbf{w}\mathbf{w}^\mathbf{T} \quad \text{and} \quad \mathbf{M}_\mathbf{e}^{\mathbf{rz}} = \int_{\Gamma(t)} r\rho\phi\phi^\mathbf{T}$$

- The RZ strong mass conservation principle implies that these are **constant in time**:

$$\frac{\mathrm{d}\mathbf{M}_\mathbf{v}^{\mathbf{rz}}}{\mathrm{d}t} = \mathbf{0}, \qquad \frac{\mathrm{d}\mathbf{M}_\mathbf{e}^{\mathbf{rz}}}{\mathrm{d}t} = \mathbf{0}$$

Axisymmetric Momentum Equation

Reducing the 3D momentum equation to the axisymmetric cut plane Γ we get

$$\int_{\Omega(t)} \left(\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) \cdot \vec{w}_i = -\int_{\Omega(t)} \sigma : \nabla \vec{w}_i \quad \longrightarrow \quad 2\pi \int_{\Gamma(t)} r \left(\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) \cdot \vec{w}_i = -2\pi \int_{\Gamma(t)} r\, \sigma_{rz} : \nabla_{rz} \vec{w}_i$$

Axisymmetric Momentum Equation

Reducing the 3D momentum equation to the axisymmetric cut plane Γ we get

$$\int_{\Omega(t)} \left(\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) \cdot \vec{w}_i = -\int_{\Omega(t)} \sigma : \nabla \vec{w}_i \quad \longrightarrow \quad \cancel{2\pi} \int_{\Gamma(t)} r \left(\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) \cdot \vec{w}_i = -\cancel{2\pi} \int_{\Gamma(t)} r\, \sigma_{rz} : \nabla_{rz} \vec{w}_i$$

Axisymmetric tensors

The axisymmetric gradient of a vector field is given by

$$\nabla_{rz} \vec{v} = \begin{pmatrix} \frac{\partial v_z}{\partial z} & \frac{\partial v_z}{\partial r} & 0 \\ \frac{\partial v_r}{\partial z} & \frac{\partial v_r}{\partial r} & 0 \\ 0 & 0 & \frac{v_r}{r} \end{pmatrix}_{z-r-\theta} = \begin{pmatrix} \nabla_{2d} \vec{v} & 0 \\ 0 & \frac{v_r}{r} \end{pmatrix}$$

Therefore, for $\sigma = -p\mathbf{I} + \mu \nabla \vec{v}$, the axisymmetric stress tensor is

$$\sigma_{rz} = \begin{pmatrix} -p\mathbf{I} + \mu \nabla_{2d} \vec{v} & 0 \\ 0 & -p + \mu \frac{v_r}{r} \end{pmatrix}$$

Axisymmetric Momentum Equation

Reducing the 3D momentum equation to the axisymmetric cut plane Γ we get

$$\int_{\Omega(t)} \left(\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) \cdot \vec{w}_i = -\int_{\Omega(t)} \sigma : \nabla \vec{w}_i \quad \longrightarrow \quad 2\pi \int_{\Gamma(t)} r \left(\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) \cdot \vec{w}_i = -2\pi \int_{\Gamma(t)} r\, \sigma_{rz} : \nabla_{rz} \vec{w}_i$$

Axisymmetric tensors

The axisymmetric gradient of a vector field is given by

$$\nabla_{rz} \vec{v} = \begin{pmatrix} \frac{\partial v_z}{\partial z} & \frac{\partial v_z}{\partial r} & 0 \\ \frac{\partial v_r}{\partial z} & \frac{\partial v_r}{\partial r} & 0 \\ 0 & 0 & \frac{v_r}{r} \end{pmatrix}_{z-r-\theta} = \begin{pmatrix} \nabla_{2d} \vec{v} & 0 \\ 0 & \frac{v_r}{r} \end{pmatrix}$$

Therefore, for $\sigma = -p\mathbf{I} + \mu \nabla \vec{v}$, the axisymmetric stress tensor is

$$\sigma_{rz} = \begin{pmatrix} -p\mathbf{I} + \mu \nabla_{2d} \vec{v} & 0 \\ 0 & -p + \mu \frac{v_r}{r} \end{pmatrix}$$

The axisymmetric momentum equation then becomes

$$\int_{\Gamma(t)} r \left(\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) \cdot \vec{w}_i = -\int_{\Gamma(t)} r \begin{pmatrix} \sigma_{2d} & 0 \\ 0 & -p + \mu \frac{v_r}{r} \end{pmatrix} : \begin{pmatrix} \nabla_{2d} \vec{w}_i & 0 \\ 0 & \frac{w_r}{r} \end{pmatrix}$$

$$= -\int_{\Gamma(t)} r(\sigma_{2d} : \nabla_{2d} \vec{w}_i) - p w_r + \mu \frac{v_r w_r}{r}$$

The $\frac{1}{r}$ term is never evaluated at $r = 0$ (quadrature points are interior and $\lim_{r \to 0} \frac{v_r w_r}{r} = 0$).

Axisymmetric Stress Tensor

$$\sigma(\vec{x}) = -p(\vec{x})\mathbf{I} + \sigma_a(\vec{x}) + s(\vec{x})$$

Axisymmetric Stress Tensor

$$\sigma(\vec{x}) = -p(\vec{x})\mathbf{I} + \sigma_a(\vec{x}) + s(\vec{x})$$

Artificial viscosity

- Consider our default option: $\sigma_a = \mu_{\vec{s}_1}\varepsilon(\vec{v})$.
- Shock direction, $\vec{s}_1$, directional length scale $\ell_{\vec{s}_1}$ and measure of compression, $\Delta_{\vec{s}_1}\vec{v}$, are computed the same way as in 2D (ignore the $\vec{\mathbf{e}}_\theta$ eigenvector).
- Vorticity/compression measure uses RZ gradient: $\psi_0 = |\nabla_{rz} \cdot \vec{v}|/\|\nabla_{rz}\vec{v}\|$.

Axisymmetric Stress Tensor

$$\sigma(\vec{x}) = -p(\vec{x})\mathbf{I} + \sigma_a(\vec{x}) + s(\vec{x})$$

Artificial viscosity

- Consider our default option: $\sigma_a = \mu_{\vec{s}_1}\varepsilon(\vec{v})$.
- Shock direction, $\vec{s}_1$, directional length scale $\ell_{\vec{s}_1}$ and measure of compression, $\Delta_{\vec{s}_1}\vec{v}$, are computed the same way as in 2D (ignore the $\vec{\mathbf{e}}_\theta$ eigenvector).
- Vorticity/compression measure uses RZ gradient: $\psi_0 = |\nabla_{rz}\cdot\vec{v}|/\|\nabla_{rz}\vec{v}\|$.

Stress deviator

- The axisymmetric stress deviator matrix has the form

$$s = \begin{pmatrix} s_{zz} & s_{zr} & 0 \\ s_{rz} & s_{rr} & 0 \\ 0 & 0 & s_{\theta\theta} \end{pmatrix}_{z-r-\theta} = \begin{pmatrix} s_{2d} & 0 \\ 0 & s_{\theta\theta} \end{pmatrix}$$

  with $s_{zr} = s_{rz}$ and $s_{\theta\theta} = -(s_{zz} + s_{rr})$ since s_{rz} is *symmetric* and *traceless*.

- The semi-discrete stress deviator equation is

$$\frac{ds_{2d}}{dt} = g_{rz} \equiv 2\mu_s\left(\varepsilon_{2d}(\vec{v}) - \frac{1}{3}\nabla_{rz}\cdot\vec{v}\right) + \frac{s_{2d}(\nabla_{2d}\vec{v} - \vec{v}\nabla_{2d}) - (\nabla_{2d}\vec{v} - \vec{v}\nabla_{2d})s_{2d}}{2}$$

- We do not keep track of $s_{\theta\theta}$, since the plastic-yield factor can be computed directly:

$$f(s, Y) = \sqrt{\frac{2}{3}\frac{Y^2}{Tr(s^2)}} = \sqrt{\frac{Y^2}{3(s_{zz}^2 + s_{zz}s_{rr} + s_{rr}^2 + s_{rz}^2)}}$$

Semi-discrete Axisymmetric Method

Axisymmetric generalized corner force matrix:

$$(\mathbf{F}^{rz})_{ij} = \int_{\Gamma(t)} r\,(\sigma_{rz} : \nabla_{rz}\vec{w}_i)\,\phi_j$$

Axisymmetric stress deviator rate:

$$(\mathbf{g}^{rz}_{mn})_j = \int_{\Gamma(t)} r\rho\,(g_{rz})_{mn}\,\phi_j$$

Semi-discrete axisymmetric finite element method

Momentum Conservation:	$\mathbf{M}^{rz}_{\mathbf{v}} \frac{d\mathbf{v}}{dt} = -\mathbf{F}^{rz} \cdot \mathbf{1}$
Energy Conservation:	$\mathbf{M}^{rz}_{\mathbf{e}} \frac{d\mathbf{e}}{dt} = (\mathbf{F}^{rz})^{\mathbf{T}} \cdot \mathbf{v}$
Equation of Motion:	$\frac{d\mathbf{x}}{dt} = \mathbf{v}$
Stress Deviator Rate:	$\mathbf{M}^{rz}_{\mathbf{e}} \cdot \frac{d\mathbf{s}}{dt} = \mathbf{g}^{rz}$

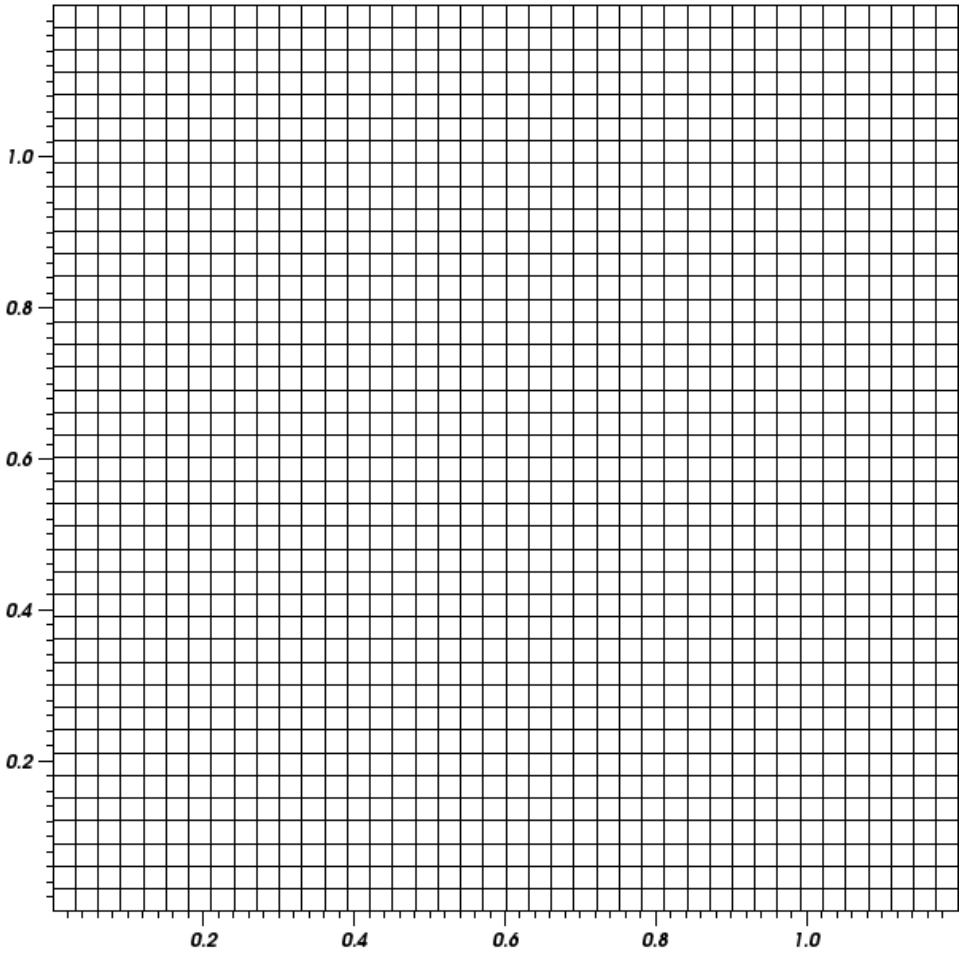
By strong mass conservation, we get **exact semi-discrete energy conservation**:

$$\frac{dE}{dt} = \frac{d}{dt}\left(\int_{\Omega(t)} \rho\frac{|\vec{v}|^2}{2} + \rho e\right) = \frac{d}{dt}\left(2\pi\int_{\Gamma(t)} r\rho\frac{|\vec{v}|^2}{2} + r\rho e\right)$$
$$= 2\pi\frac{d}{dt}\left(\frac{\mathbf{v}\cdot\mathbf{M}^{rz}_{\mathbf{v}}\cdot\mathbf{v}}{2} + \mathbf{1}\cdot\mathbf{M}^{rz}_{\mathbf{e}}\cdot\mathbf{e}\right) = 2\pi\left(-\mathbf{v}\cdot\mathbf{F}^{rz}\cdot\mathbf{1} + \mathbf{1}\cdot(\mathbf{F}^{rz})^{\mathbf{T}}\cdot\mathbf{v}\right) = 0.$$

This holds for any choice of velocity and energy spaces!

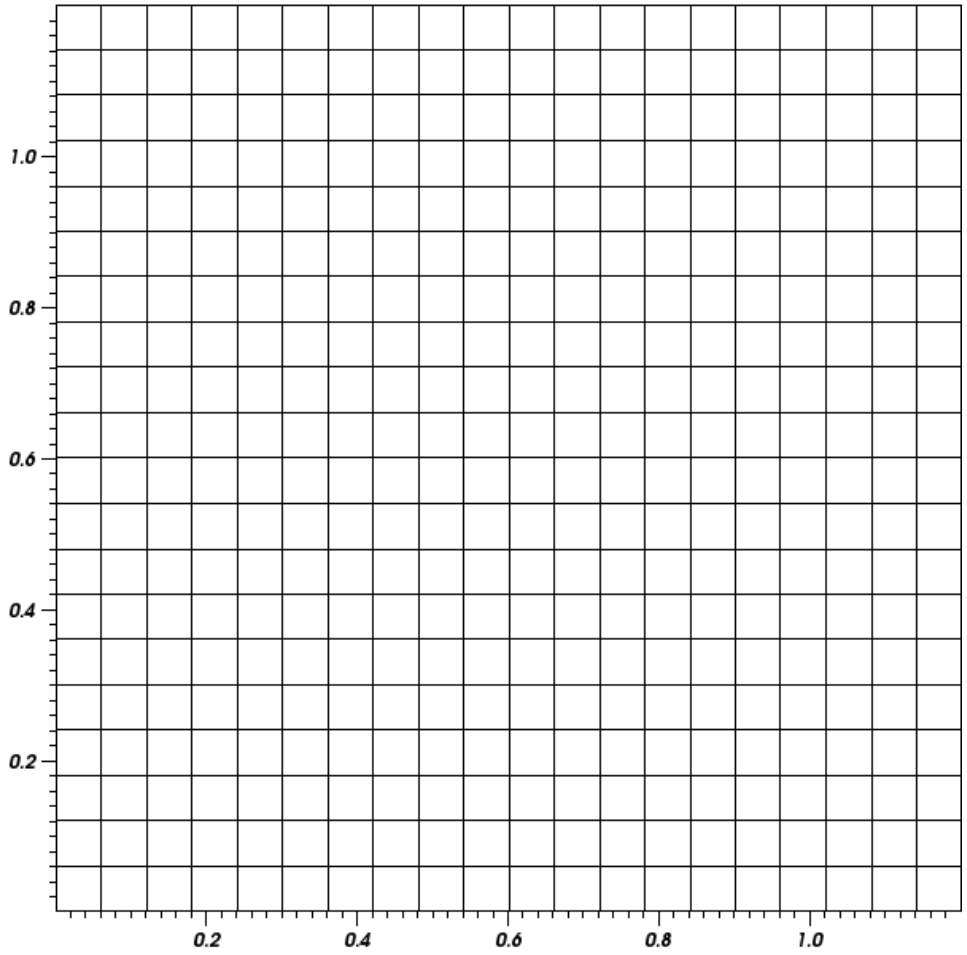
Axisymmetric Sedov Explosion

40x40 Lagrangian SGH - Density



- Symmetry is not preserved
- Mesh distorted near the origin

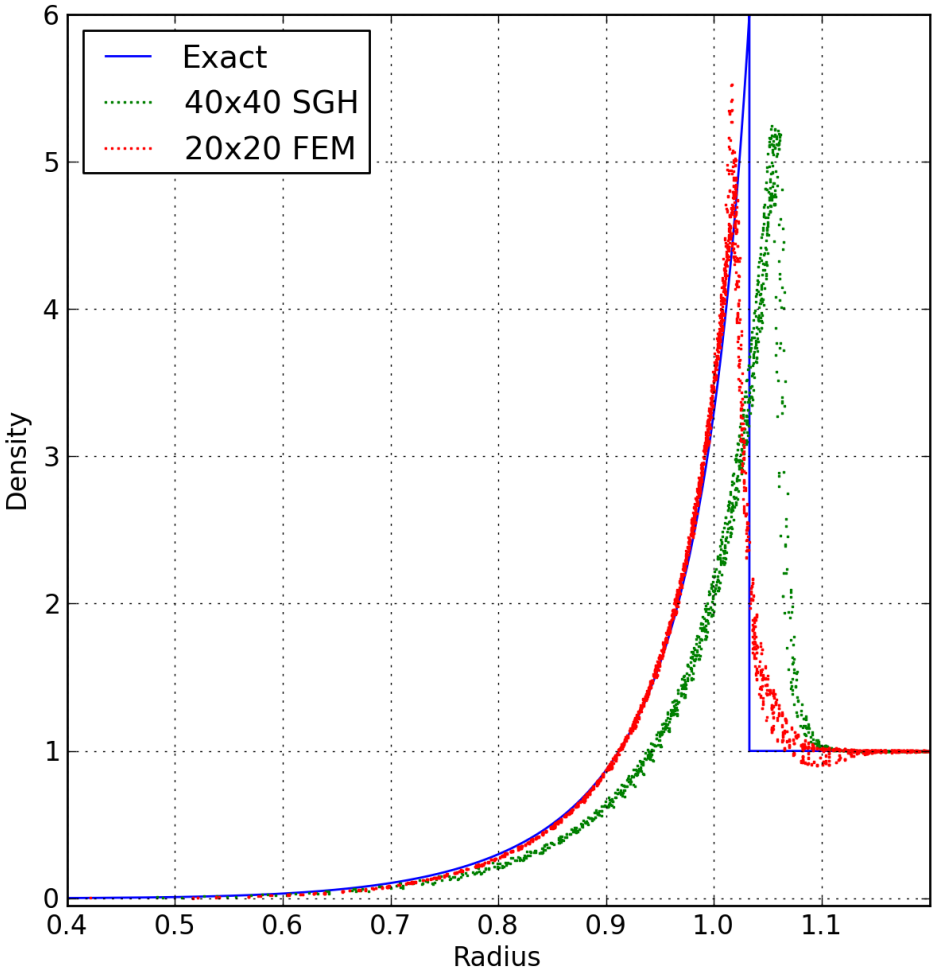
20x20 Lagrangian FEM - Density



- Symmetry is preserved
- Curvilinear zones match physics
- FE pressure treatment

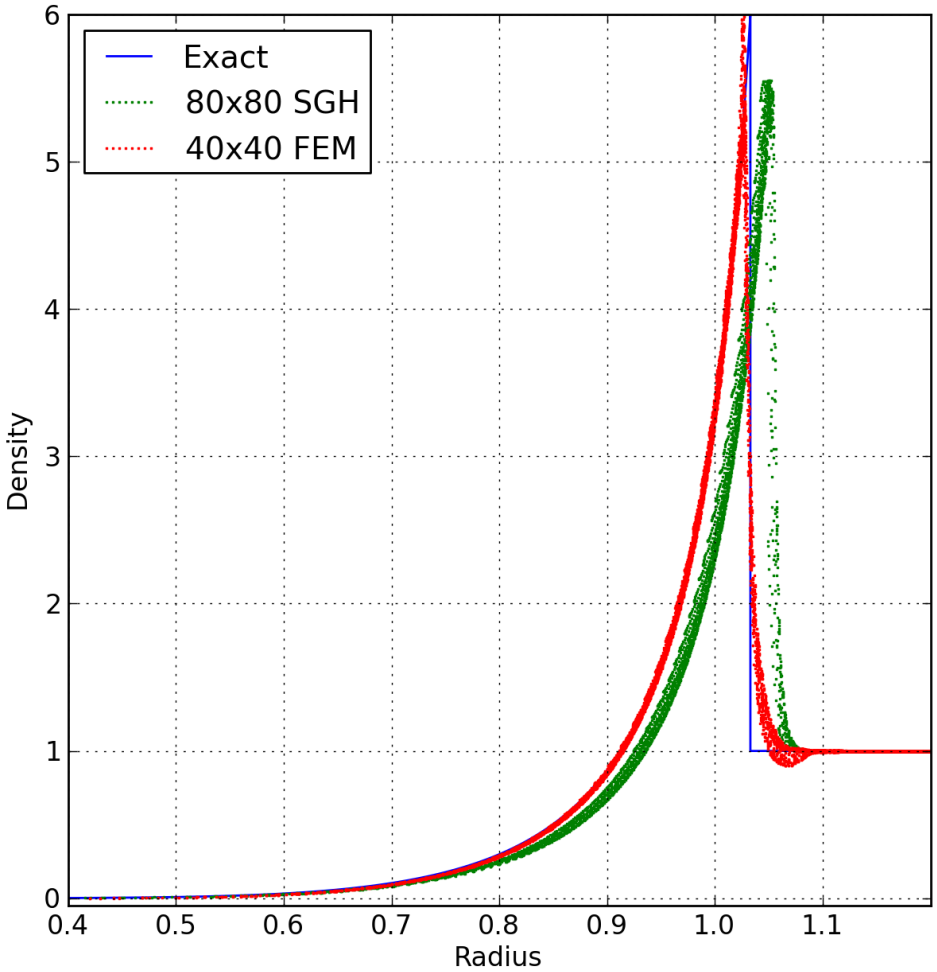
Axisymmetric Sedov - Scatter Plots

Coarse Mesh Scatter Plot of Density vs Radius



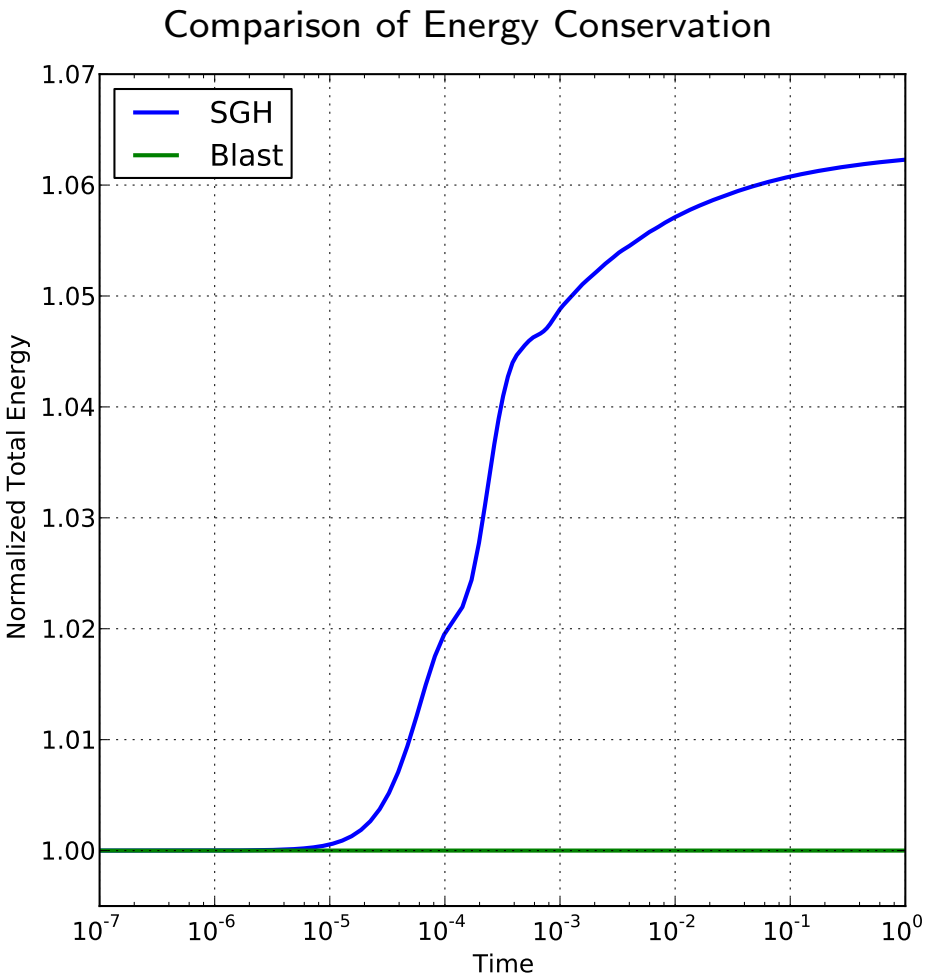
- SGH shock is too fast
- FEM is good with only 20x20 zones

Fine Mesh Scatter Plot of Density vs Radius

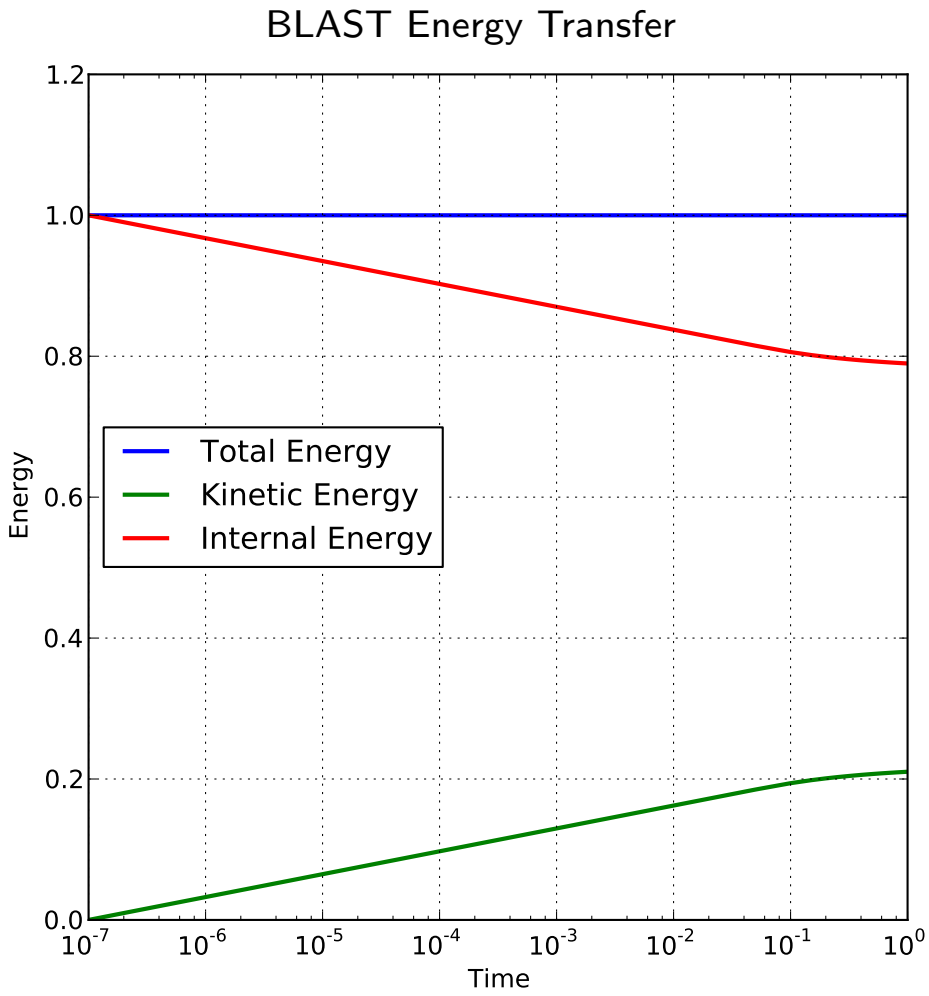


- SGH does not improve under refinement
- FEM matches exact solution very closely

Axisymmetric Sedov - Energy Conservation



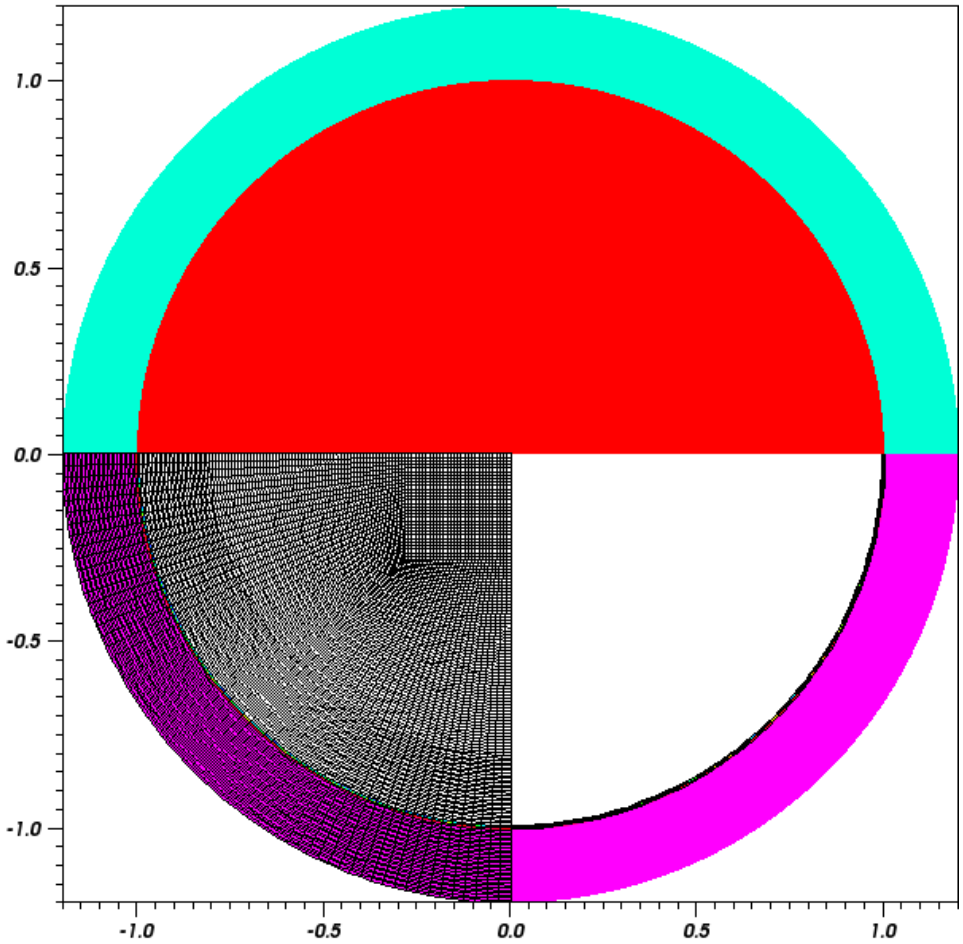
- SGH gains 6% energy
- BLAST conserves energy to machine precision



- BLAST converts IE to KE without loss

Simple Velocity Driven ICF-like Test

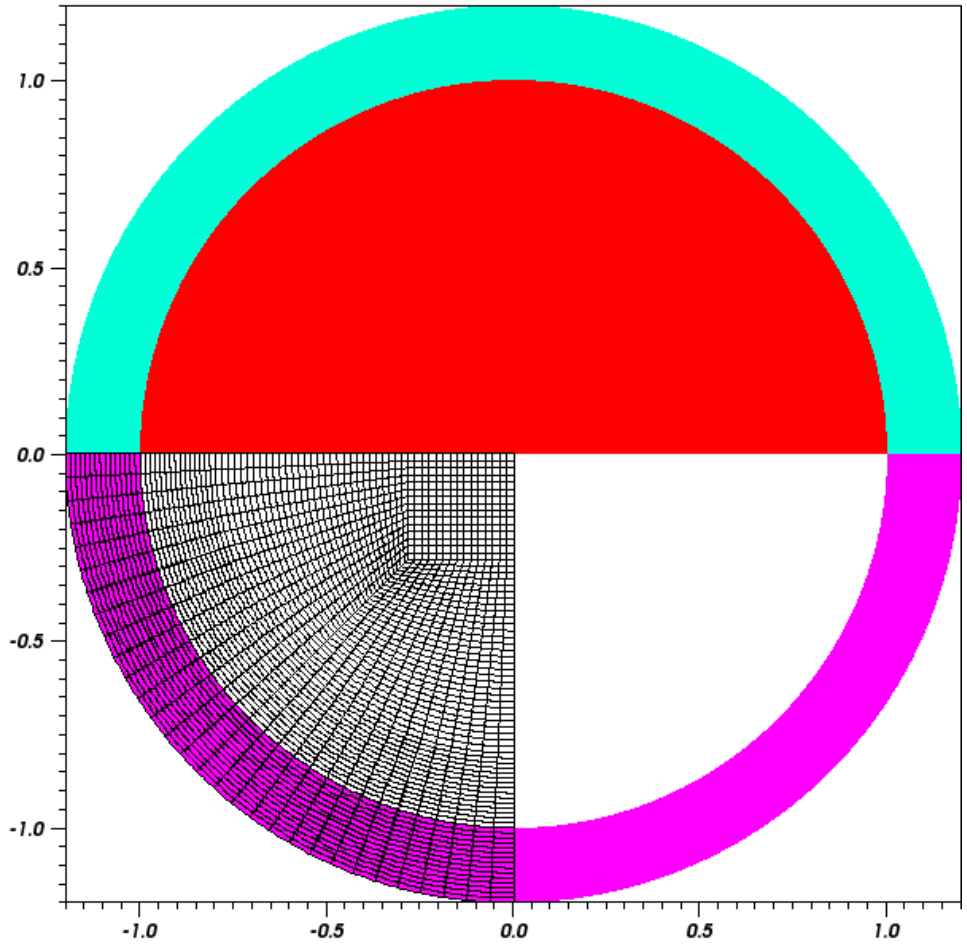
Internal Energy



log(Density)

ALE Staggered Grid Hydro

Internal Energy

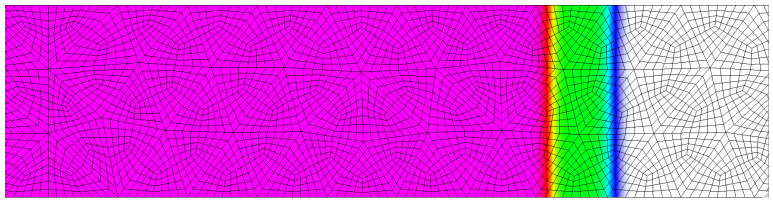


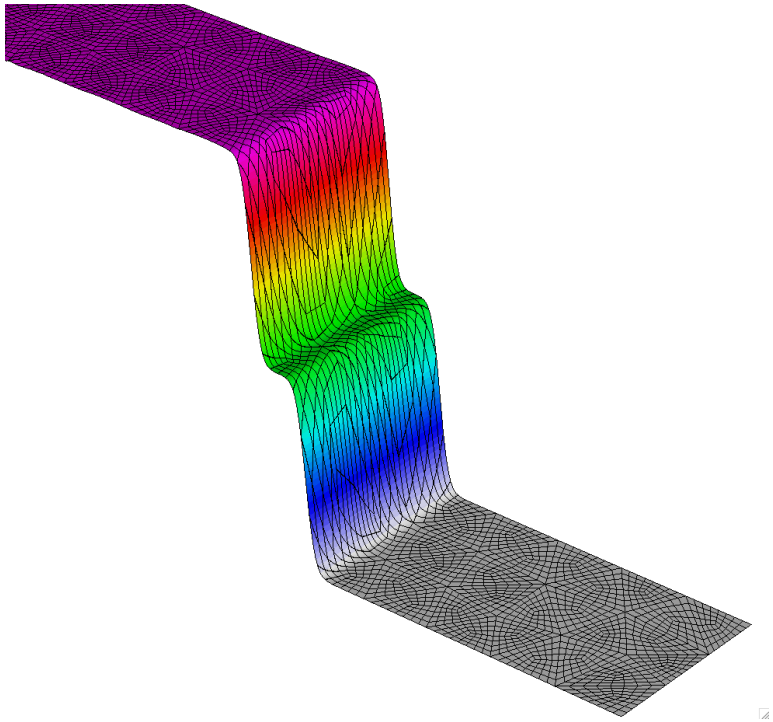
log(Density)

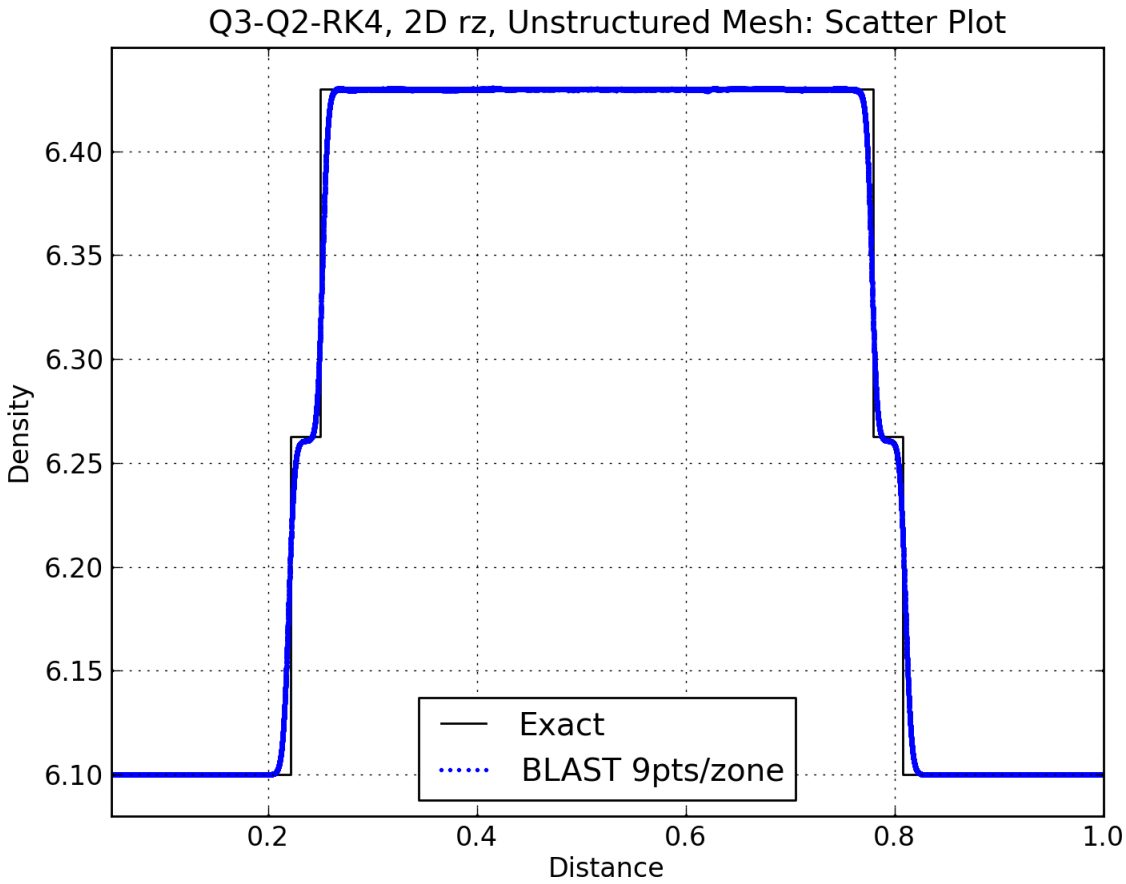
Pure Lagrangian FEM

S. Galera, P-H. Maire, J. Breil, *A two-dimensional unstructured cell-centered multi-material ALE scheme using VOF interface reconstruction*, JCP, 2010.

Axisymmetric Elastic-Plastic Shock Wave





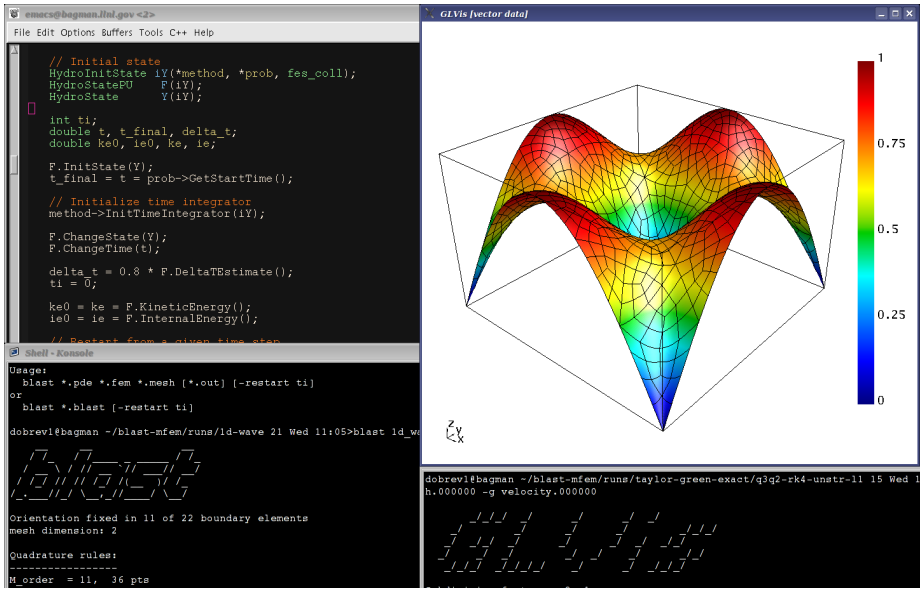


- Axisymmetric version of the problem from Talk I
- Q_3Q_2-RK4 method on highly unstructured 2D mesh
- 1D symmetry is preserved
- No artifacts at axis of rotation

Parallelism in Our Research Codes

Our Research Codes

- **BLAST**: C++ high-order curvilinear FEM Lagrangian hydrocode. Solves XY/RZ/3D problems on tri/quad/tet/hex meshes with many finite element options.
  www.llnl.gov/CASC/blast
- **MFEM**: modular C++ finite element library.
  mfem.googlecode.com
- **GLVis**: OpenGL visualization tool.
  glvis.googlecode.com



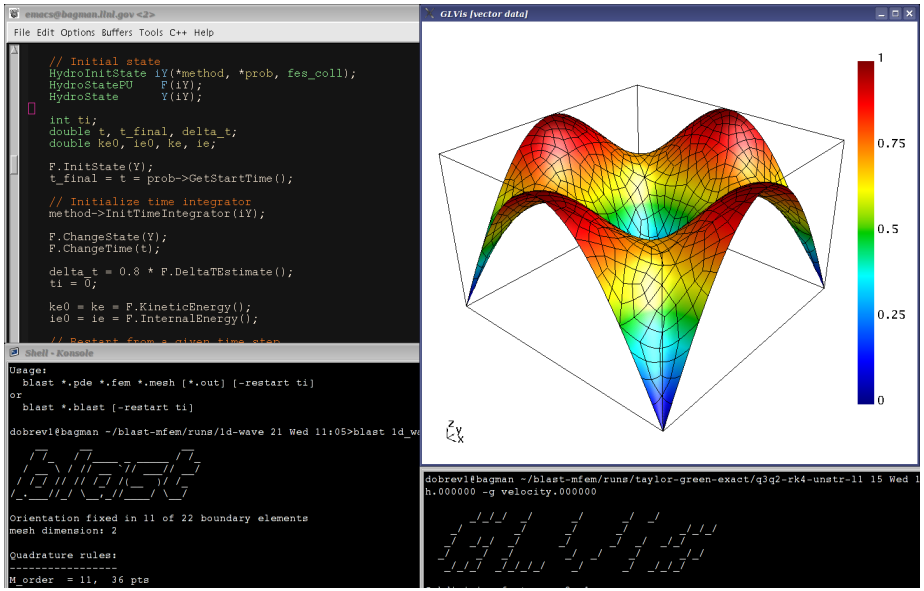
BLAST algorithm

```
Read mesh, material properties and input parameters
Loop over time steps:
  - Loop over the zones in the domain:
      - Loop over quadrature points in each zone:
          - Compute hydro forces associated with the quadrature point
  - Assemble zone contribution to global linear system and rhs
  - Solve global linear system for new accelerations
  - Integrate accelerations in time to get velocities and new mesh positions
  - Update internal energies due to hydrodynamic motion
```

Parallelism in Our Research Codes

Our Research Codes

- **BLAST**: C++ high-order curvilinear FEM Lagrangian hydrocode. Solves XY/RZ/3D problems on tri/quad/tet/hex meshes with many finite element options.
  www.llnl.gov/CASC/blast
- **MFEM**: modular C++ finite element library.
  mfem.googlecode.com
- **GLVis**: OpenGL visualization tool.
  glvis.googlecode.com



BLAST algorithm

```
Read mesh, material properties and input parameters
Loop over time steps:
  - Loop over the zones in the domain:
      - Loop over quadrature points in each zone:
          - Compute hydro forces associated with the quadrature point
  - Assemble zone contribution to global linear system and rhs
  - Solve global linear system for new accelerations
  - Integrate accelerations in time to get velocities and new mesh positions
  - Update internal energies due to hydrodynamic motion
```

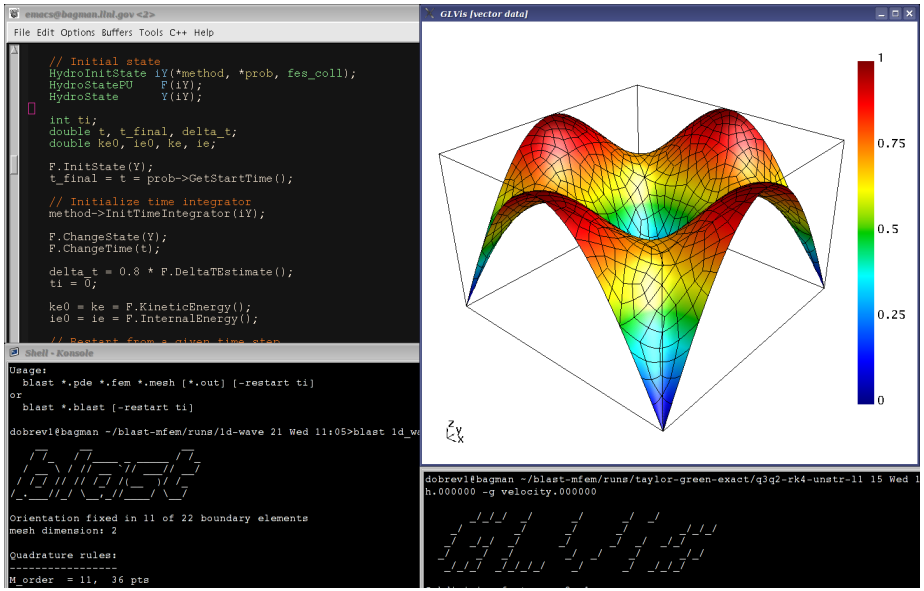
Two layers of parallelism:

- MPI-based parallel finite elements in MFEM – domain-decomposed between CPUs

Parallelism in Our Research Codes

Our Research Codes

- **BLAST**: C++ high-order curvilinear FEM Lagrangian hydrocode. Solves XY/RZ/3D problems on tri/quad/tet/hex meshes with many finite element options.
  www.llnl.gov/CASC/blast
- **MFEM**: modular C++ finite element library.
  mfem.googlecode.com
- **GLVis**: OpenGL visualization tool.
  glvis.googlecode.com



BLAST algorithm

```
Read mesh, material properties and input parameters
Loop over time steps:
  - Loop over the zones in the domain:
      - Loop over quadrature points in each zone:
          - Compute hydro forces associated with the quadrature point
  - Assemble zone contribution to global linear system and rhs
  - Solve global linear system for new accelerations
  - Integrate accelerations in time to get velocities and new mesh positions
  - Update internal energies due to hydrodynamic motion
```

Two layers of parallelism:

- MPI-based parallel finite elements in MFEM – domain-decomposed between CPUs
- CUDA-based parallel corner forces in BLAST – zone-decomposed on GPUs

First Parallel Layer: CPU/MPI Domain Decomposition

Parallel data decomposition in BLAST

- Each CPU is assigned a subdomain consisting of a number of zones
- MFEM handles the translation between local finite element bilinear forms / grid functions and global parallel matrices / vectors.
- Just a few MPI calls (`MPI_Bcast` and `MPI_Allreduce`).

MPI-based parallel finite elements in MFEM

- Parallel mesh
- Parallel finite element space
- Parallel stiffness matrix and load vector

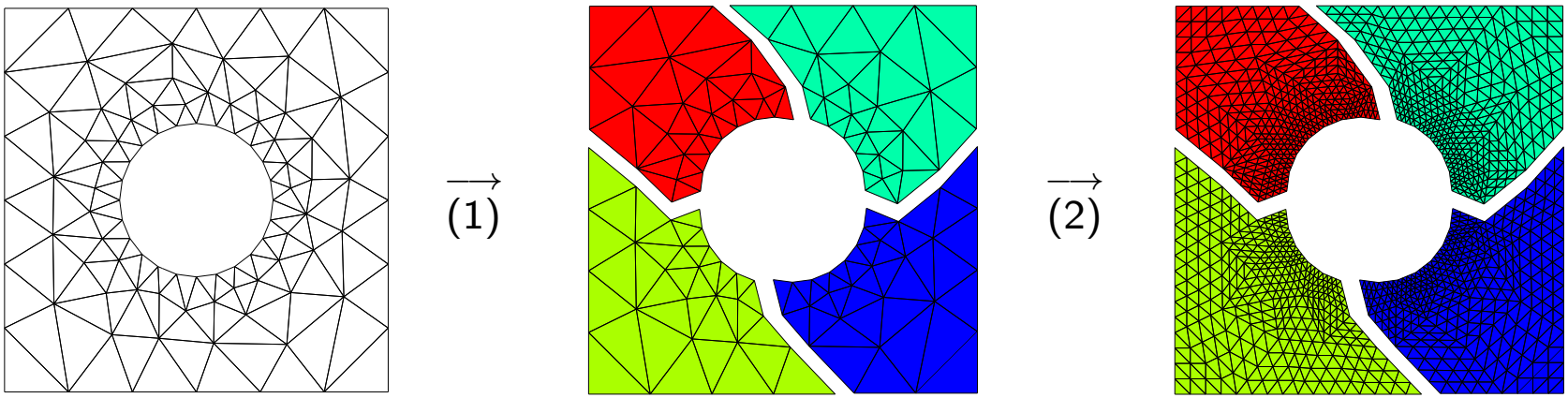
First Parallel Layer: CPU/MPI Domain Decomposition

Parallel data decomposition in BLAST

- Each CPU is assigned a subdomain consisting of a number of zones
- MFEM handles the translation between local finite element bilinear forms / grid functions and global parallel matrices / vectors.
- Just a few MPI calls (`MPI_Bcast` and `MPI_Allreduce`).

MPI-based parallel finite elements in MFEM

- Parallel mesh



(1) Parallel mesh splitting (domain decomposition using METIS).
(2) Parallel mesh refinement.

- Parallel finite element space
- Parallel stiffness matrix and load vector

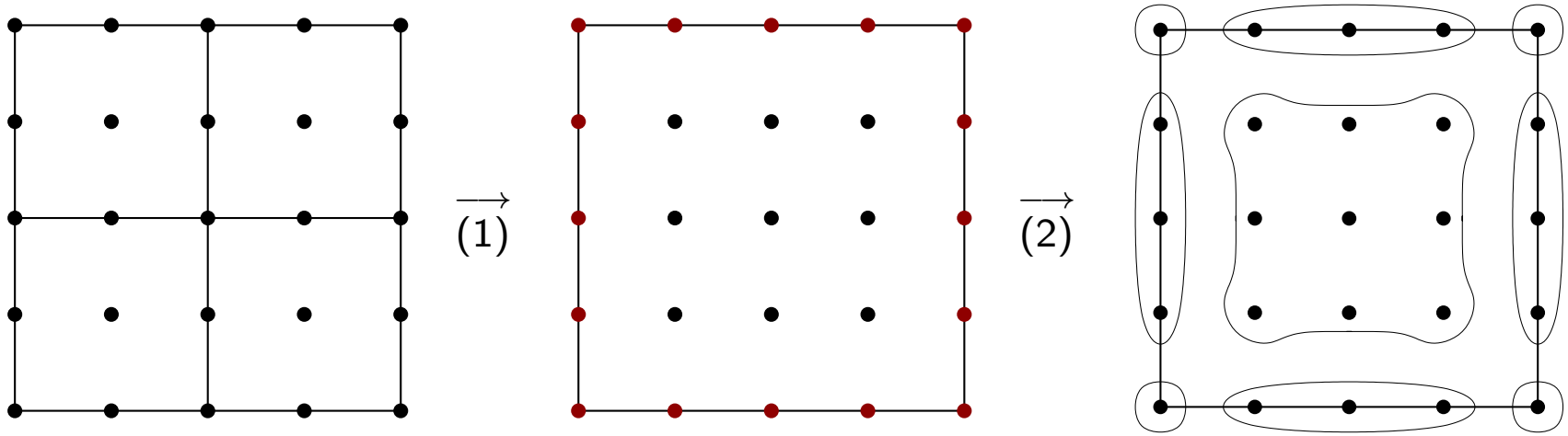
First Parallel Layer: CPU/MPI Domain Decomposition

Parallel data decomposition in BLAST

- Each CPU is assigned a subdomain consisting of a number of zones
- MFEM handles the translation between local finite element bilinear forms / grid functions and global parallel matrices / vectors.
- Just a few MPI calls (`MPI_Bcast` and `MPI_Allreduce`).

MPI-based parallel finite elements in MFEM

- Parallel mesh
- Parallel finite element space



(1) Find shared degrees of freedom (**dofs**).
(2) Form groups of dofs and assign ownership.
(3) Build a parallel Boolean matrix **P** = **dofs_truedofs** identifying each dof with a master (true) dof. We use the ParCSR format in the *hypre* library for parallel matrix storage.

- Parallel stiffness matrix and load vector

First Parallel Layer: CPU/MPI Domain Decomposition

Parallel data decomposition in BLAST

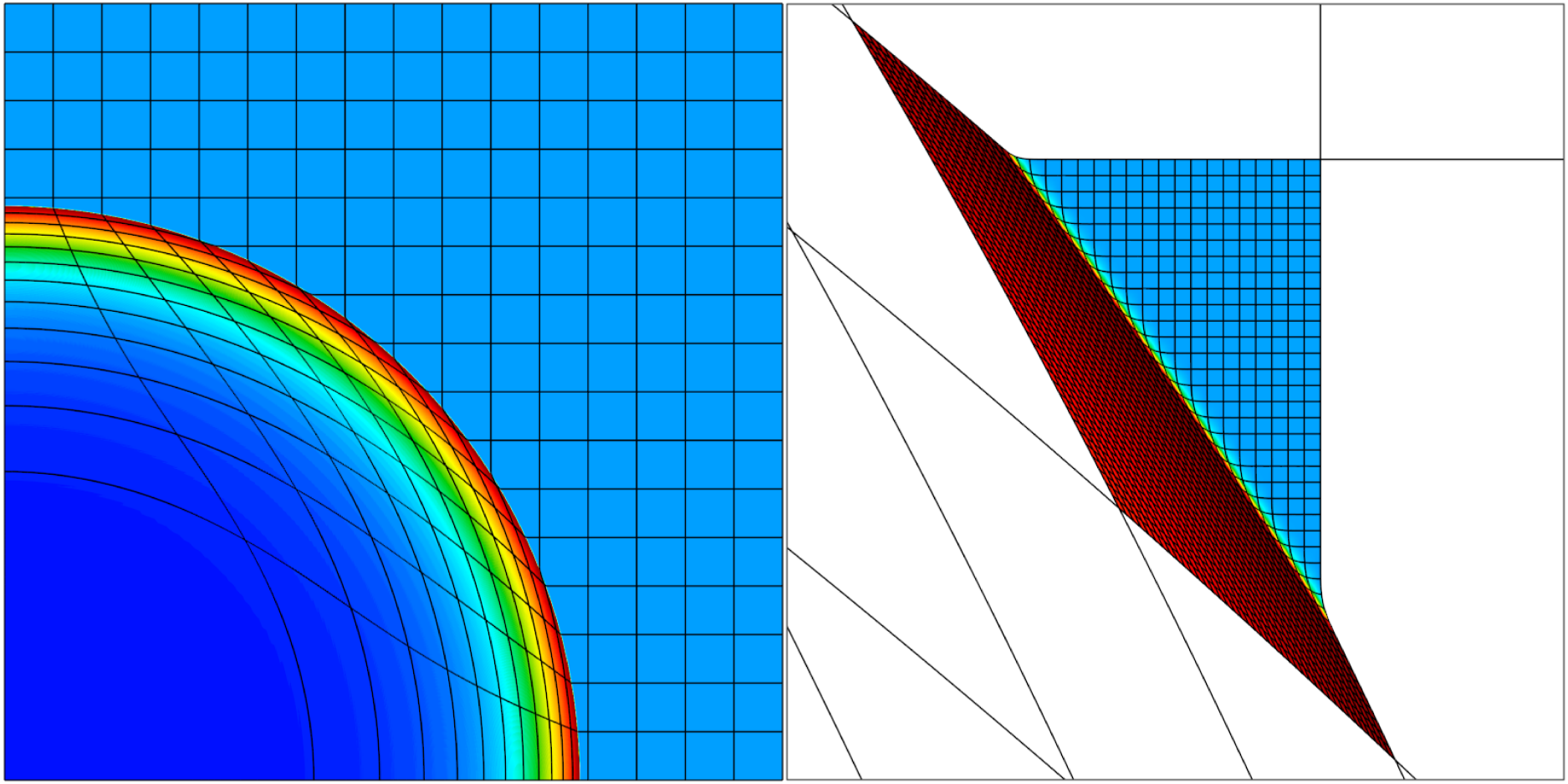
- Each CPU is assigned a subdomain consisting of a number of zones
- MFEM handles the translation between local finite element bilinear forms / grid functions and global parallel matrices / vectors.
- Just a few MPI calls (`MPI_Bcast` and `MPI_Allreduce`).

MPI-based parallel finite elements in MFEM

- Parallel mesh
- Parallel finite element space
- Parallel stiffness matrix and load vector
  (1) Assemble the stiffness matrix in each processor and form a block-diagonal matrix $\mathbf{A}_{\mathbf{dofs}}$.
  (2) Compute $\mathbf{A} = \mathbf{P}^T \mathbf{A}_{\mathbf{dofs}} \mathbf{P}$ (using *hypre*'s **RAP**).
  (3) Form $\mathbf{b}_{\mathbf{dofs}}$ by assembling the load vector in each processor.
  (4) Compute $\mathbf{b} = \mathbf{P}^T \mathbf{b}_{\mathbf{dofs}}$.
  (5) Global problem: $\mathbf{A}\mathbf{x} = \mathbf{b}$.
  (6) Restriction to each processor: $\mathbf{P}\mathbf{x}$.

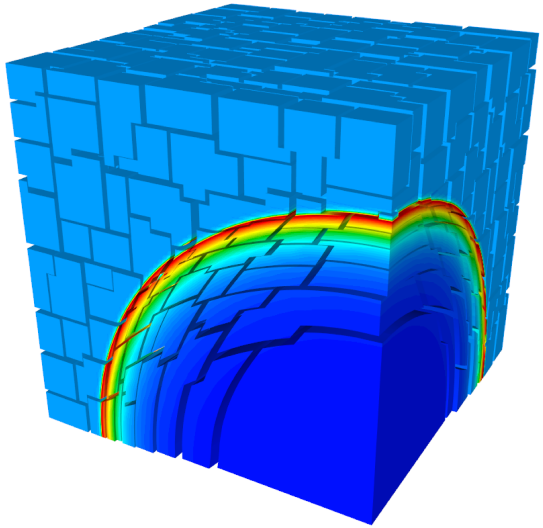
  No explicit communications needed!

Parallel Sedov Blast in 2D

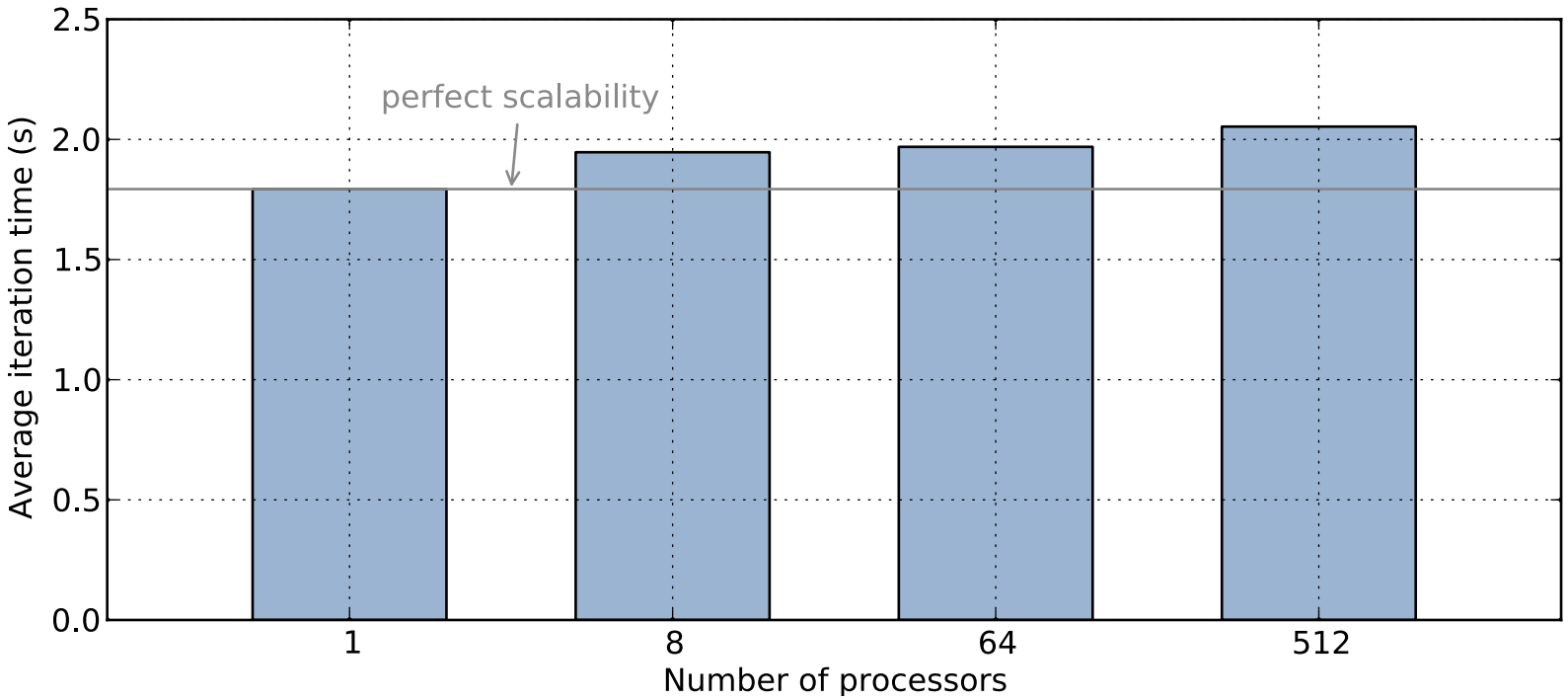


- 640×640 grid on 256 processors with uniform Cartesian partitioning
- Shown are the processor subdomains and the mesh inside one of the processors
- The shock is resolved in a singe zone

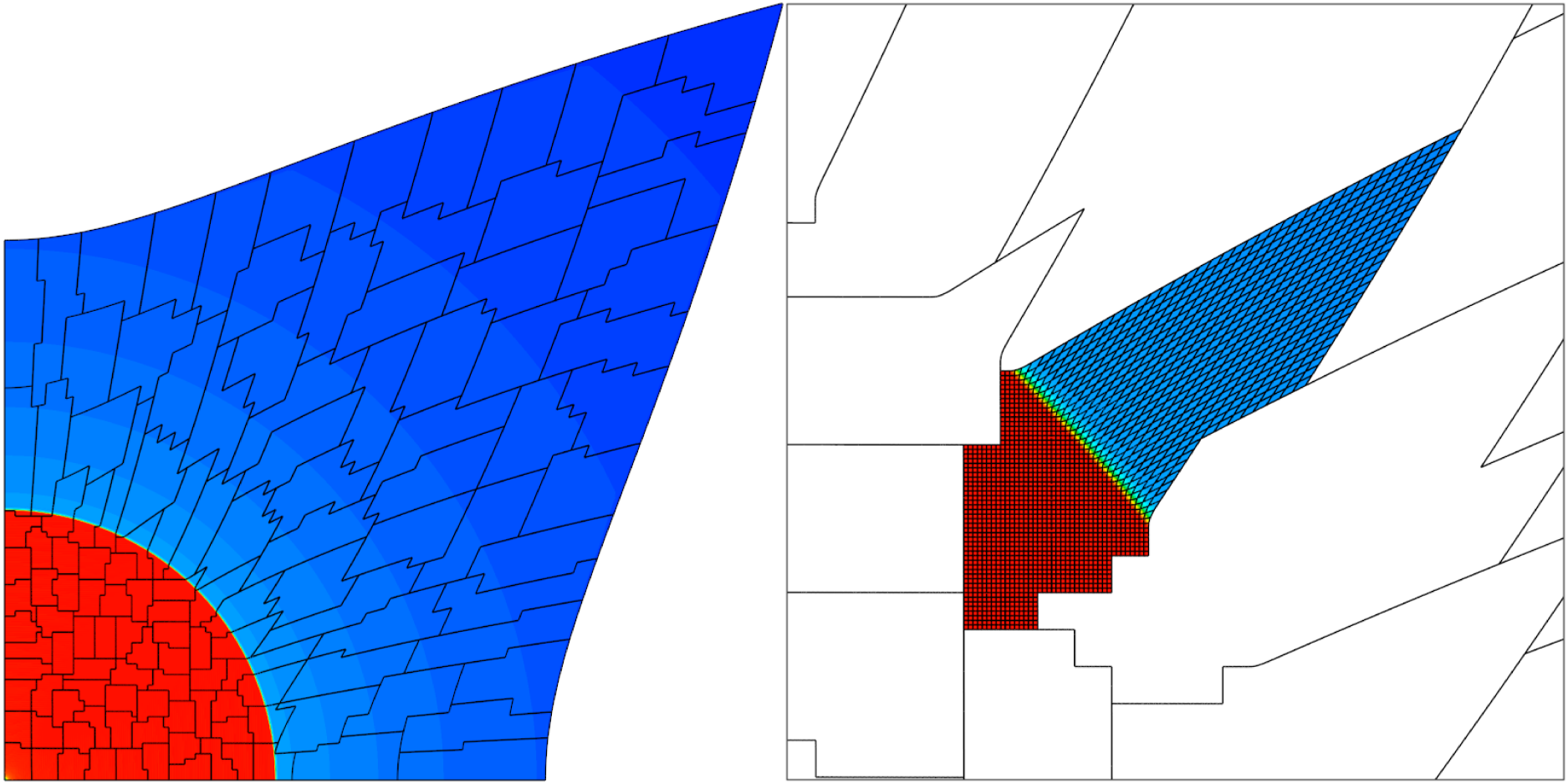
Parallel Weak Scalability in 3D



- Fixed problem size per processor (8^3)
- Shown is the 64^3 grid on 512 processors

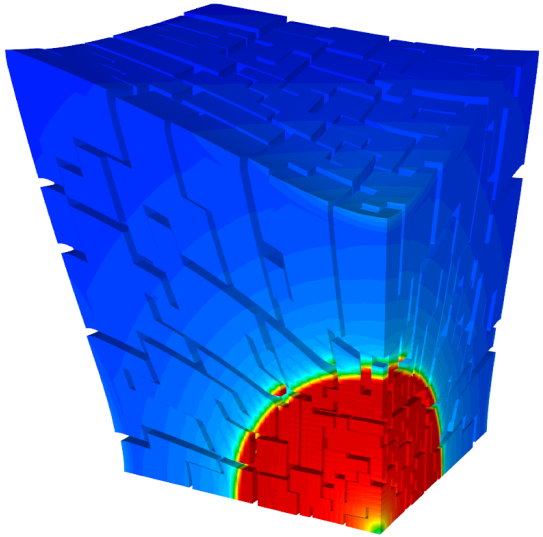


Parallel Noh Implosion in 2D

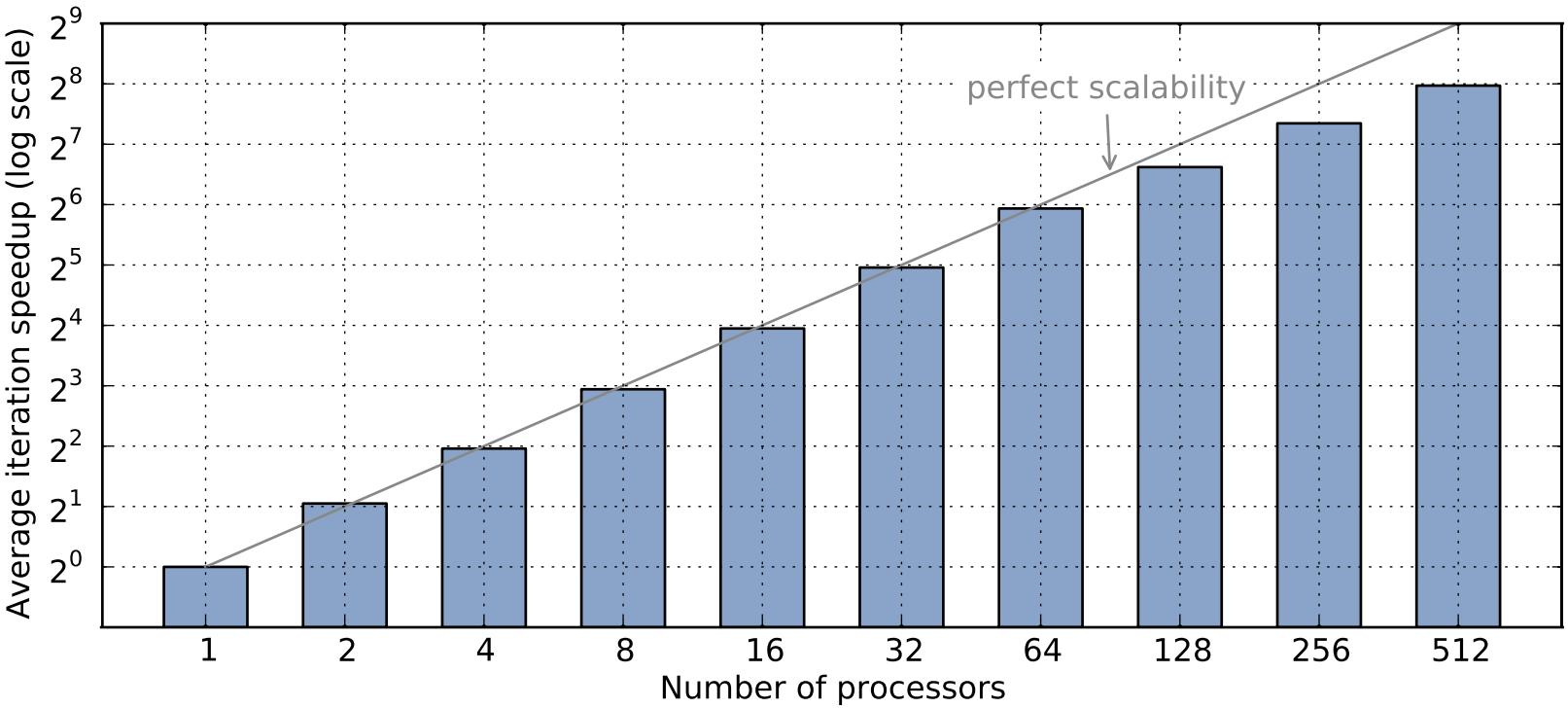


- 512×512 grid on 128 processors with non-uniform partitioning (from METIS).
- Shown are the processor subdomains and the mesh inside one of the processors
- The shock is resolved in a singe zone

Parallel Strong Scalability in 3D

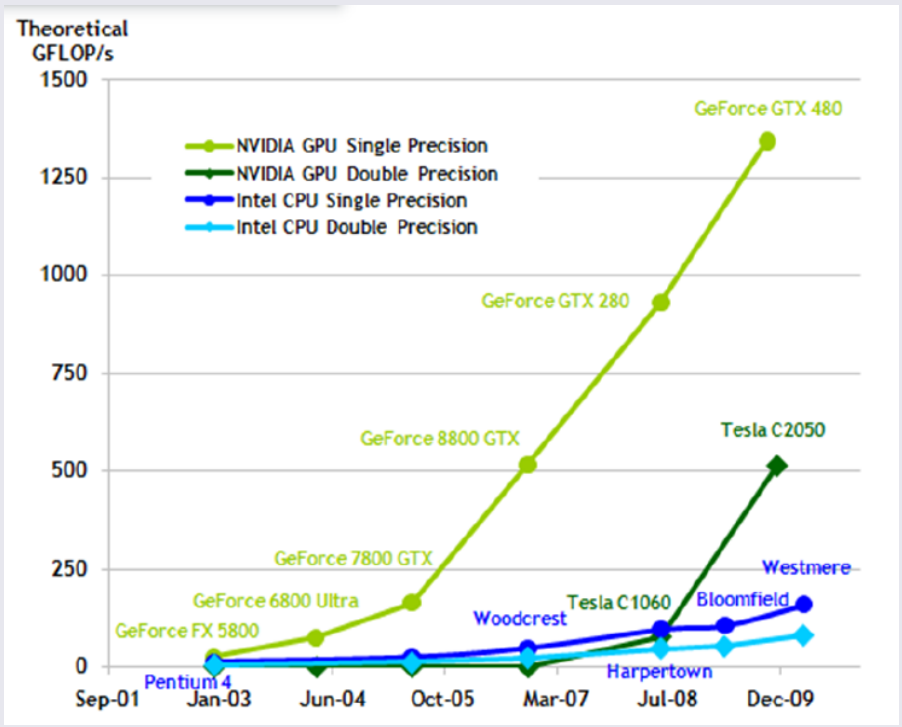


- Fixed total problem size
- Shown is the 32^3 grid on 128 processors
- Good performance on 512 processors with only 64 zones/processor

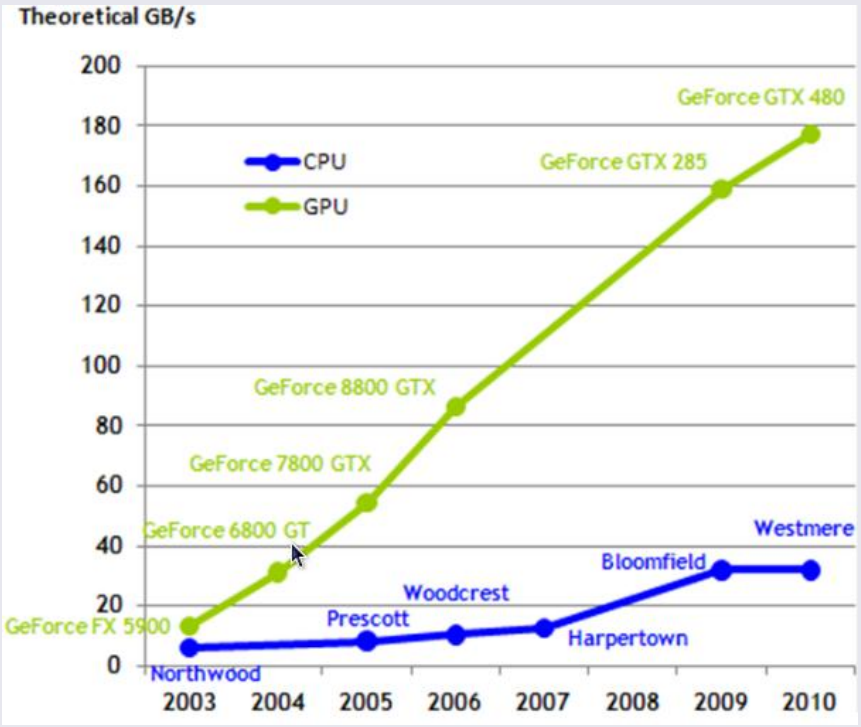


Second Parallel Layer: GPU/CUDA Zone Decomposition

GFLOPs



Bandwidth



- GPUs offer unprecedented computational power and memory bandwidth.
- Profiling results show that zonal calculations, such as the computation of the corner force matrix, have high flops/bytes ratios and are the dominant cost in BLAST.
- Together with Tingxing Dong (UTK), we developed a second, CUDA-based, parallel layer in BLAST to assist the CPU with some of these computations.

CUDA Kernels in BLAST

- Consider the semi-discrete finite element method in BLAST (without strength):

Momentum Conservation: $$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbf{M_v}^{-1}\mathbf{F} \cdot \mathbf{1}$$

Energy Conservation: $$\frac{\mathrm{d}\mathbf{e}}{\mathrm{d}t} = \mathbf{M_e}^{-1}\mathbf{F}^{\mathsf{T}} \cdot \mathbf{v}$$

Equation of Motion: $$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}$$

- We used CUDA to accelerate the following computations on the GPU:

CUDA Kernels in BLAST

- Consider the semi-discrete finite element method in BLAST (without strength):

Momentum Conservation: $$\frac{d\mathbf{v}}{dt} = -\mathbf{M_v}^{-1}\mathbf{F} \cdot \mathbf{1}$$

Energy Conservation: $$\frac{d\mathbf{e}}{dt} = \mathbf{M_e}^{-1}\mathbf{F}^T \cdot \mathbf{v}$$

Equation of Motion: $$\frac{d\mathbf{x}}{dt} = \mathbf{v}$$

- We used CUDA to accelerate the following computations on the GPU:
  1. Evaluation of **F**.

CUDA Kernels in BLAST

- Consider the semi-discrete finite element method in BLAST (without strength):

$$\textbf{Momentum Conservation:} \quad \frac{d\mathbf{v}}{dt} = -\mathbf{M_v}^{-1}\mathbf{F} \cdot \mathbf{1}$$

$$\textbf{Energy Conservation:} \quad \frac{d\mathbf{e}}{dt} = \mathbf{M_e}^{-1}\mathbf{F}^T \cdot \mathbf{v}$$

$$\textbf{Equation of Motion:} \quad \frac{d\mathbf{x}}{dt} = \mathbf{v}$$

- We used CUDA to accelerate the following computations on the GPU:
  1. Evaluation of $\mathbf{F}$.
  2. Evaluation of $\mathbf{F} \cdot \mathbf{1}$ and $\mathbf{F}^T \cdot \mathbf{v}$.

CUDA Kernels in BLAST

- Consider the semi-discrete finite element method in BLAST (without strength):

Momentum Conservation: $$\frac{d\mathbf{v}}{dt} = -\mathbf{M_v}^{-1}\mathbf{F} \cdot \mathbf{1}$$

Energy Conservation: $$\frac{d\mathbf{e}}{dt} = \mathbf{M_e}^{-1}\mathbf{F}^T \cdot \mathbf{v}$$

Equation of Motion: $$\frac{d\mathbf{x}}{dt} = \mathbf{v}$$

- We used CUDA to accelerate the following computations on the GPU:
  1. Evaluation of $\mathbf{F}$.
  2. Evaluation of $\mathbf{F} \cdot \mathbf{1}$ and $\mathbf{F}^T \cdot \mathbf{v}$.
  3. CG solver for $\mathbf{M_v}^{-1}$ and sparse matvec for $\mathbf{M_e}^{-1}$ based on CUBLAS/CUSPARSE.

CUDA Kernels in BLAST

- Consider the semi-discrete finite element method in BLAST (without strength):

$$\textbf{Momentum Conservation:} \quad \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbf{M_v}^{-1}\mathbf{F}\cdot\mathbf{1}$$

$$\textbf{Energy Conservation:} \quad \frac{\mathrm{d}\mathbf{e}}{\mathrm{d}t} = \mathbf{M_e}^{-1}\mathbf{F}^{\mathsf{T}}\cdot\mathbf{v}$$

$$\textbf{Equation of Motion:} \quad \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}$$

- We used CUDA to accelerate the following computations on the GPU:
  1. Evaluation of $\mathbf{F}$.
  2. Evaluation of $\mathbf{F}\cdot\mathbf{1}$ and $\mathbf{F}^{\mathsf{T}}\cdot\mathbf{v}$.
  3. CG solver for $\mathbf{M_v}^{-1}$ and sparse matvec for $\mathbf{M_e}^{-1}$ based on CUBLAS/CUSPARSE.
- Zonal corner forces $\{\mathbf{F_z}\}$ and the sparse matrices $\mathbf{M_v}$ and $\mathbf{M_e}^{-1}$ are stored on the GPU.
- Input/output vectors are transferred between the CPU and the GPU.

Generalized Corner Forces on the GPU

- $\mathbf{F}$ can be assembled from $\{\mathbf{F}_z\}$, which require a high order quadrature $\{(\alpha_k, \hat{\vec{q}}_k)\}_k$:

$$(\mathbf{F}_z)_{ij} = \int_{\Omega_z(t)} (\sigma : \nabla \vec{w}_i)\, \phi_j \approx \sum_k \alpha_k \hat{\sigma}(\hat{\vec{q}}_k) : \mathbf{J}_z^{-1}(\hat{\vec{q}}_k) \hat{\nabla} \hat{\vec{w}}_i(\hat{\vec{q}}_k)\, \hat{\phi}_j(\hat{\vec{q}}_k) |\mathbf{J}_z(\hat{\vec{q}}_k)| \,.$$

- Two-level concurrency: hydro forces for *each zone* (different z) and *each quadrature point* (different k) can be computed in parallel.

Generalized Corner Forces on the GPU

- $\mathbf{F}$ can be assembled from $\{\mathbf{F}_z\}$, which require a high order quadrature $\{(\alpha_k, \hat{\vec{q}}_k)\}_k$:

$$(\mathbf{F}_z)_{ij} = \int_{\Omega_z(t)} (\sigma : \nabla \vec{w}_i)\, \phi_j \approx \sum_k \alpha_k \hat{\sigma}(\hat{\vec{q}}_k) : \mathbf{J}_z^{-1}(\hat{\vec{q}}_k) \hat{\nabla} \hat{\vec{w}}_i(\hat{\vec{q}}_k)\, \hat{\phi}_j(\hat{\vec{q}}_k) |\mathbf{J}_z(\hat{\vec{q}}_k)| \,.$$

- Two-level concurrency: hydro forces for *each zone* (different z) and *each quadrature point* (different k) can be computed in parallel.

- Note that $\mathbf{F_z} = \mathbf{A_z} \mathbf{B_z^T}$, where

$$(\mathbf{A_z})_{ik} = \alpha_k \hat{\sigma}(\hat{\vec{q}}_k) : \mathbf{J}_z^{-1}(\hat{\vec{q}}_k) \hat{\nabla} \hat{\vec{w}}_i(\hat{\vec{q}}_k) |\mathbf{J}_z(\hat{\vec{q}}_k)| \quad \text{and} \quad (\mathbf{B_z})_{jk} = \hat{\phi}_j(\hat{\vec{q}}_k) \,.$$

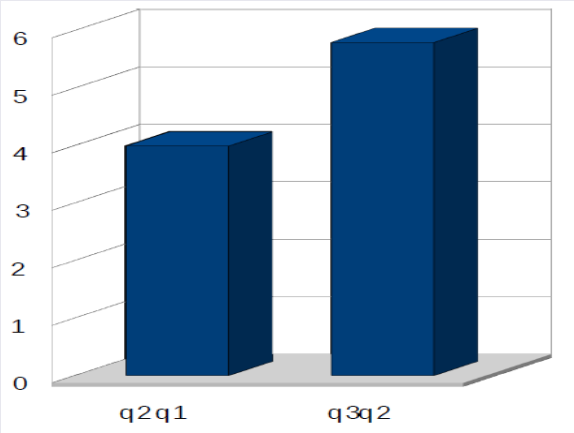
- **CUDA Kernel 1: loop over the quadrature points**
  - Each thread block corresponds to one or more zones (tunable).
  - Each thread works on one quadrature point and computes a column of the matrix $\mathbf{A_z}$.
  - The weights α_k do not change in time and are kept in GPU's constant memory.

- **CUDA Kernel 2: loop over the zones**
  - Each thread block corresponds to one zone and performs the multiplication $\mathbf{A_z} \mathbf{B_z^T}$.
  - Each thread evaluates one row of the resulting matrix $\mathbf{F_z}$ (a kinematic dof).
  - We store $\mathbf{A_z}$ in the shared and $\mathbf{B_z}$ in the constant GPU memory (hinted by the profiler).

Serial GPU Performance

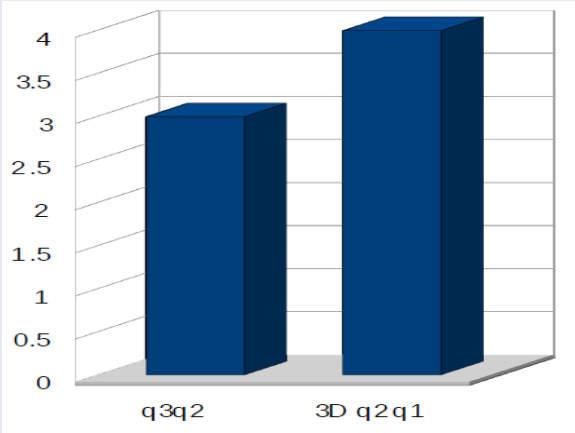
Tesla C1060

CPU: Xeon E5520



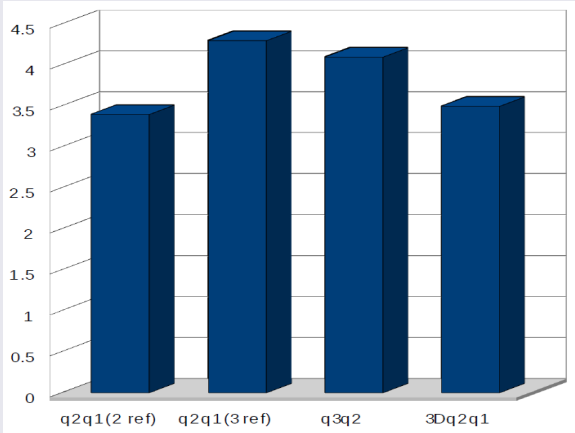
Tesla C2050

CPU: Xeon X5660



Quadro 5000

CPU: Xeon X5675

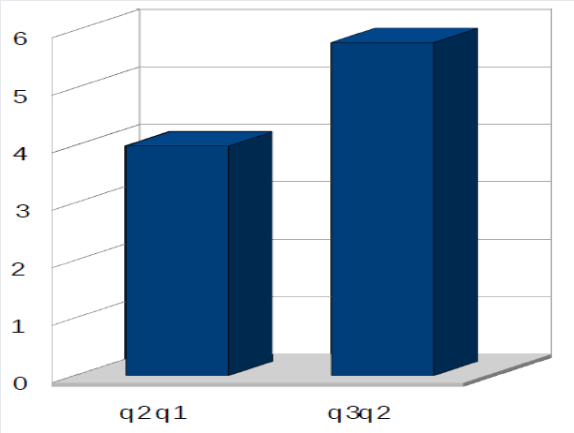


- Compare CPU and CPU+GPU code on one processor.
- Three test problems: 2D Q_2Q_1, 2D Q_3Q_2 and 3D Q_2Q_1.
- Good speedup across several different CPU/GPU pairings.
- GPU implementation required replacing LAPACK calls with hand-coded eigensolvers for 2×2 and 3×3 matrices, which also accelerated the CPU code significantly (2–4$\times$).

Serial GPU Performance

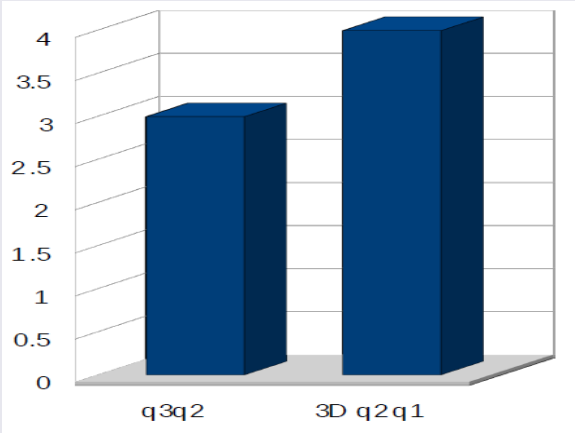
Tesla C1060

CPU: Xeon E5520



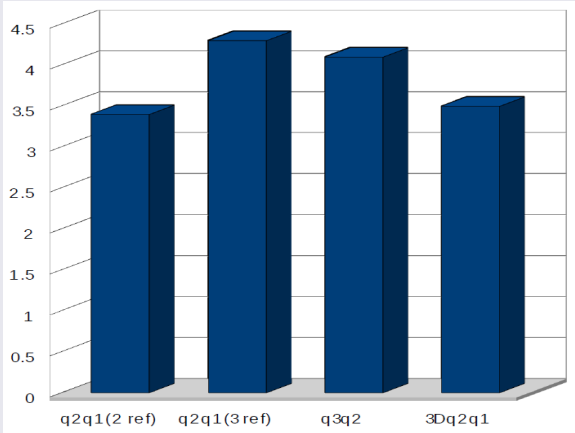
Tesla C2050

CPU: Xeon X5660

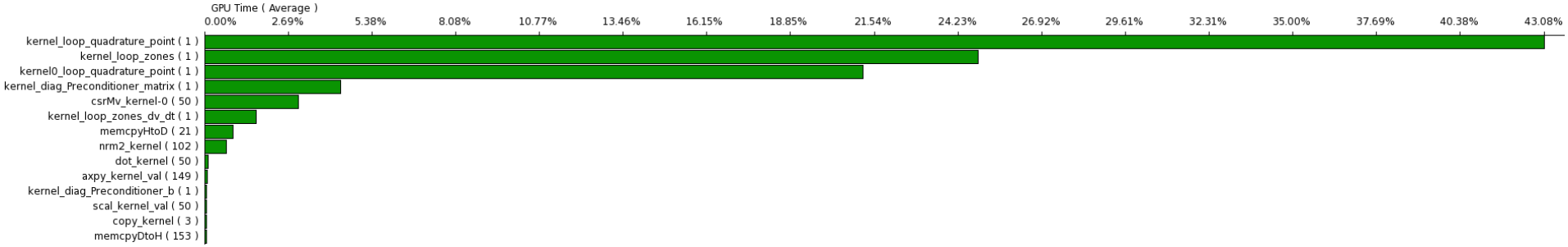


Quadro 5000

CPU: Xeon X5675

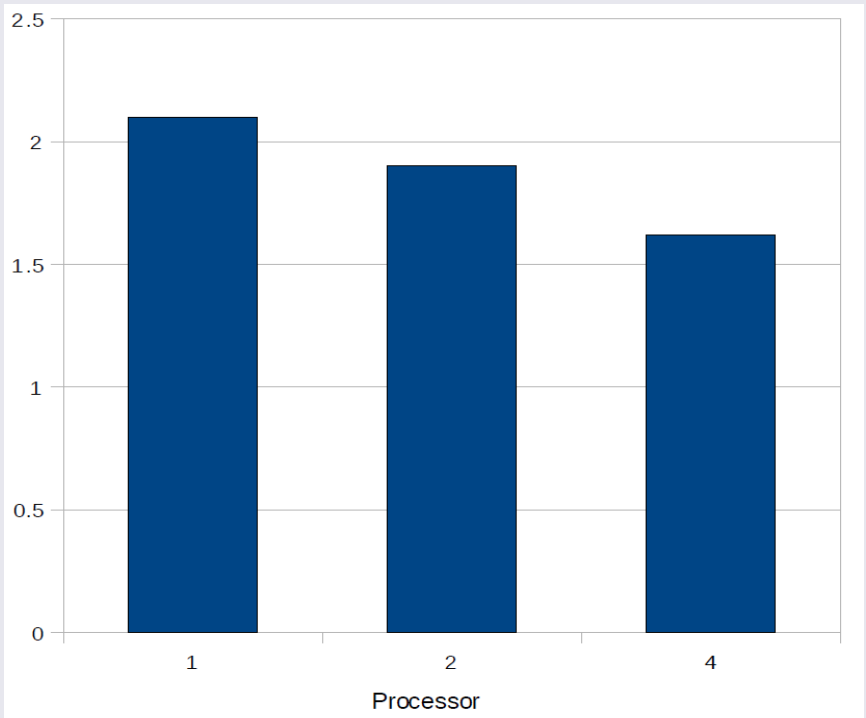


- Compare CPU and CPU+GPU code on one processor.
- Three test problems: 2D Q_2Q_1, 2D Q_3Q_2 and 3D Q_2Q_1.
- Good speedup across several different CPU/GPU pairings.
- GPU implementation required replacing LAPACK calls with hand-coded eigensolvers for 2×2 and 3×3 matrices, which also accelerated the CPU code significantly (2–4$\times$).
- High ratio of kernel time to memory transfer.



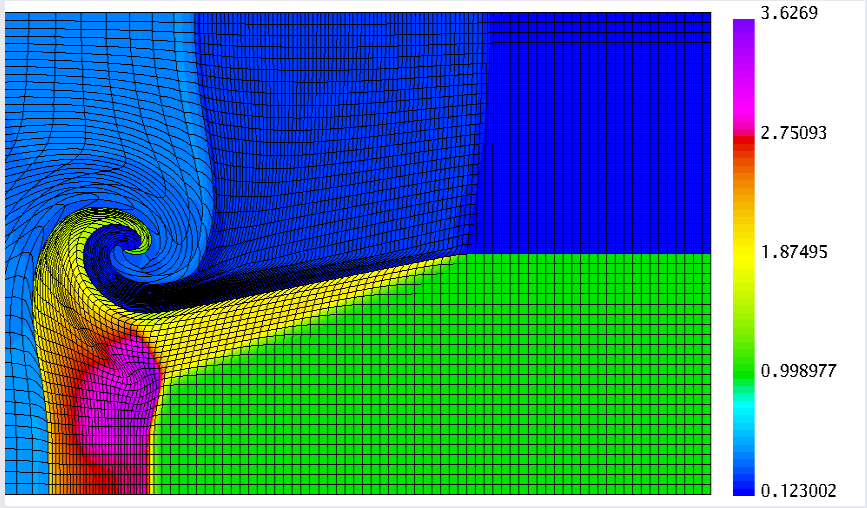
Parallel GPU Performance

MPI+CUDA on Tesla C2050

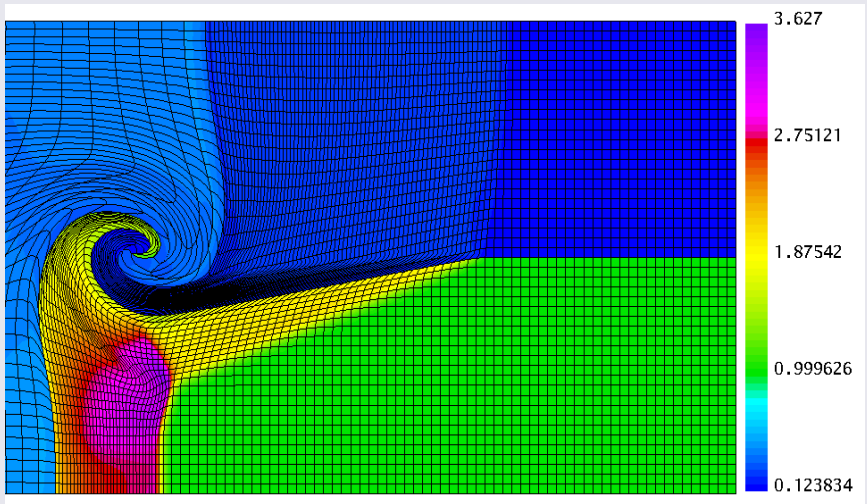


- Compare MPI and MPI+GPU code on 1, 2 and 4 processors.
- Different floating point CPU/GPU implementation lead to slight differences in the numerical results.

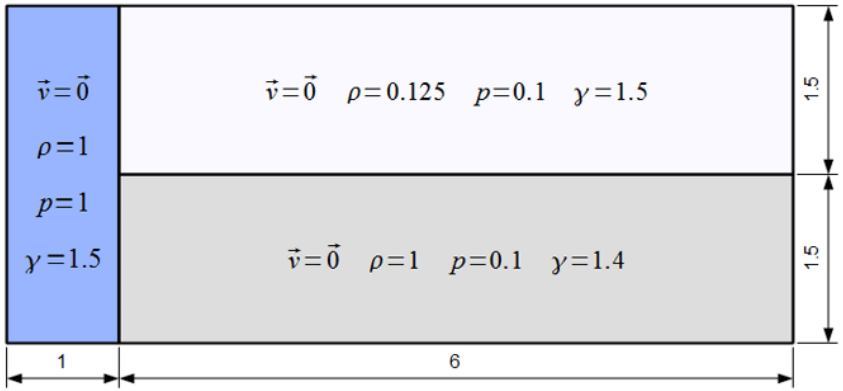
MPI Result on 4 CPUs



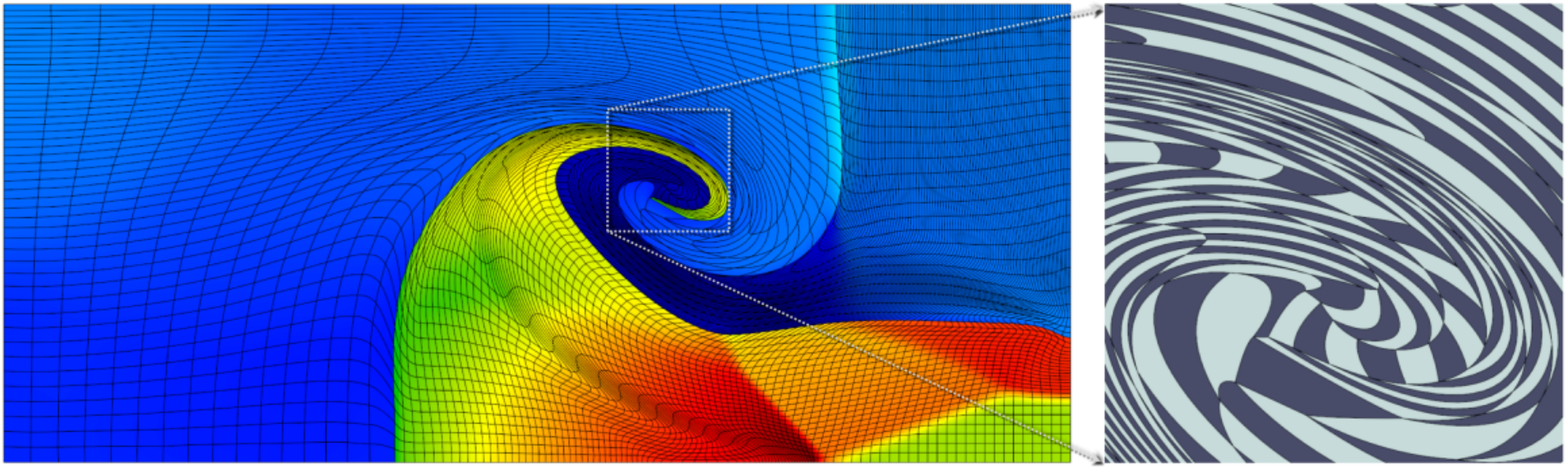
MPI Result on 4 CPUs + 2 GPUs



2D Shock triple-point interaction

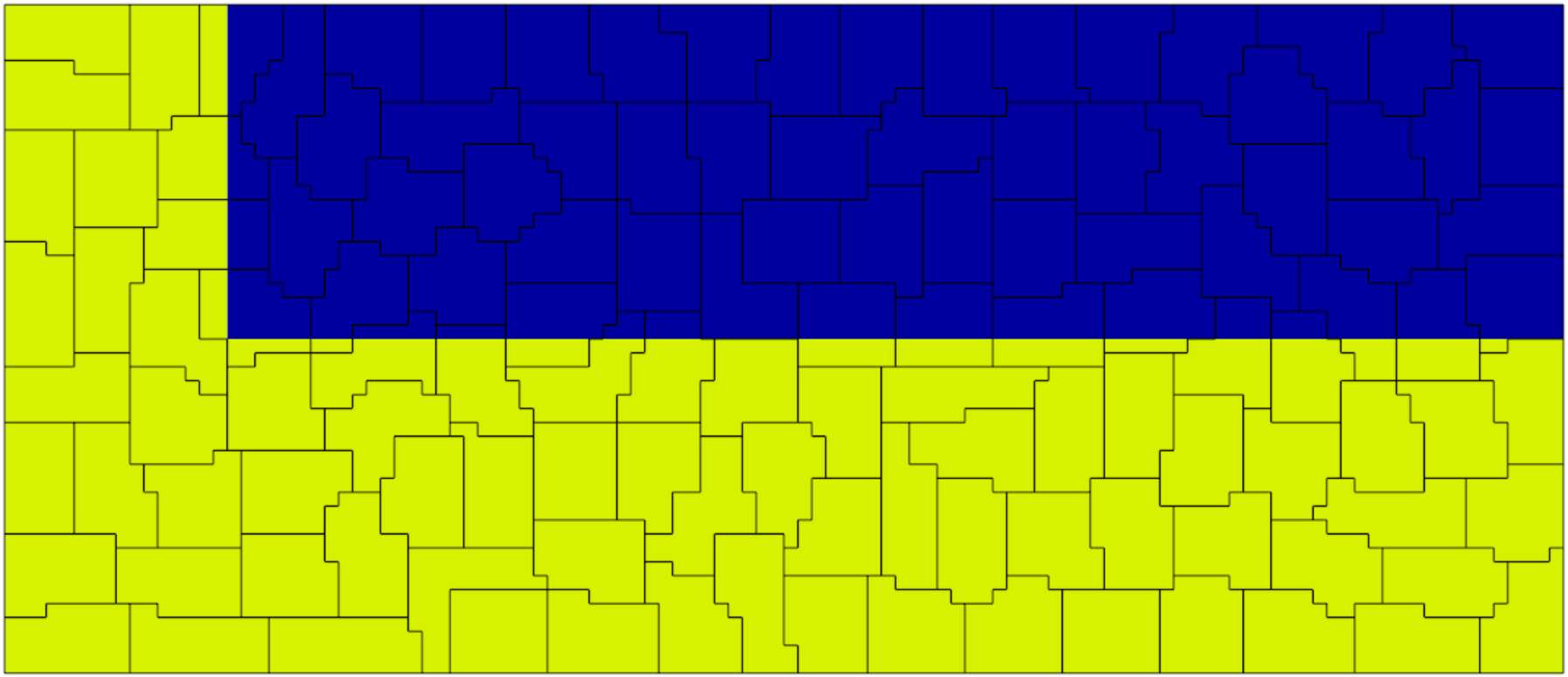


S. Galera, P-H. Maire, J. Breil,
A two-dimensional unstructured cell-centered multi-material ALE scheme using VOF interface reconstruction, JCP, 2010.



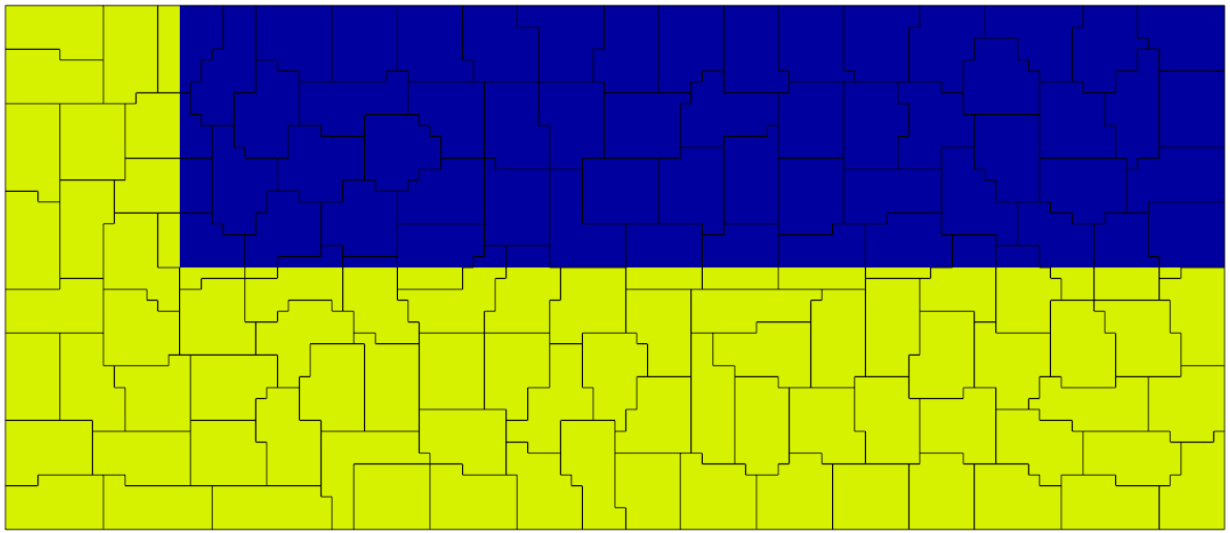
Curved zones with high aspect ratios develop naturally in Lagrangian simulations and are impossible to represent using elements with straight edges

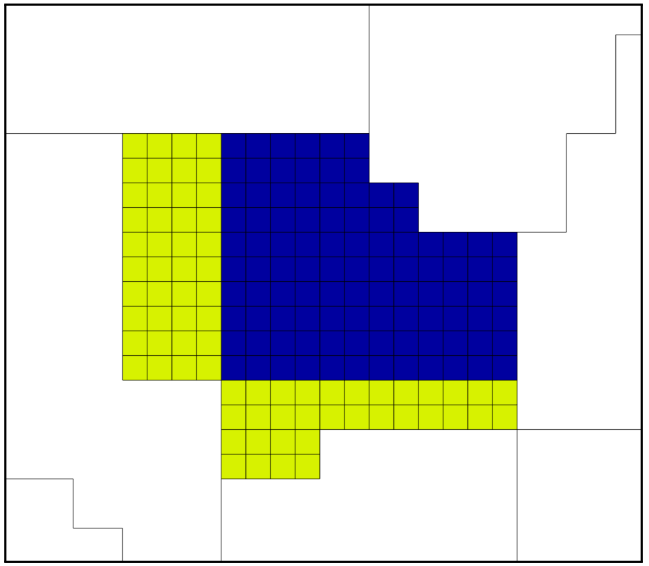
Parallel shock triple-point interaction

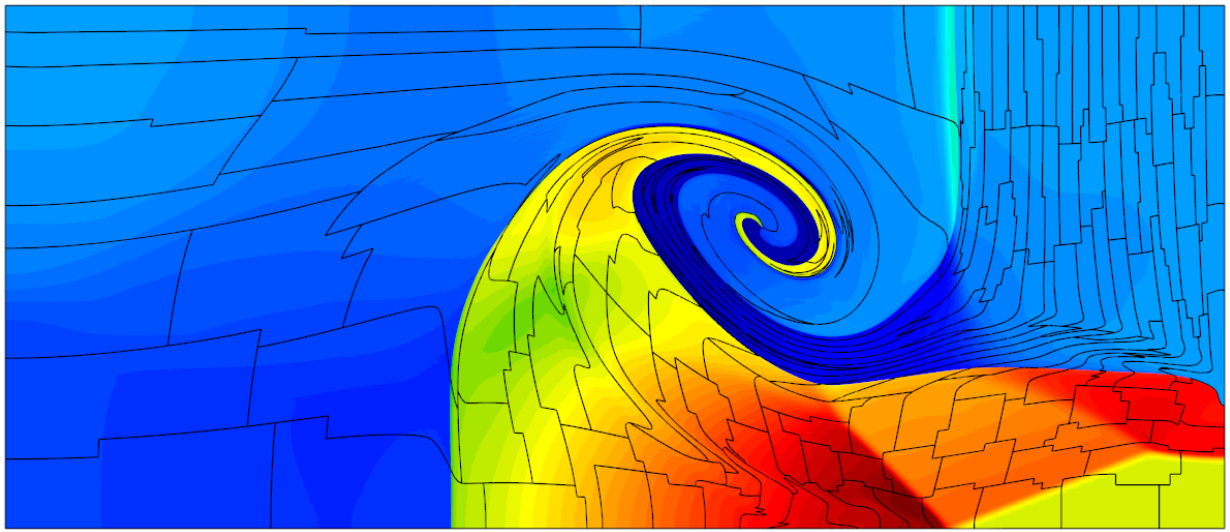


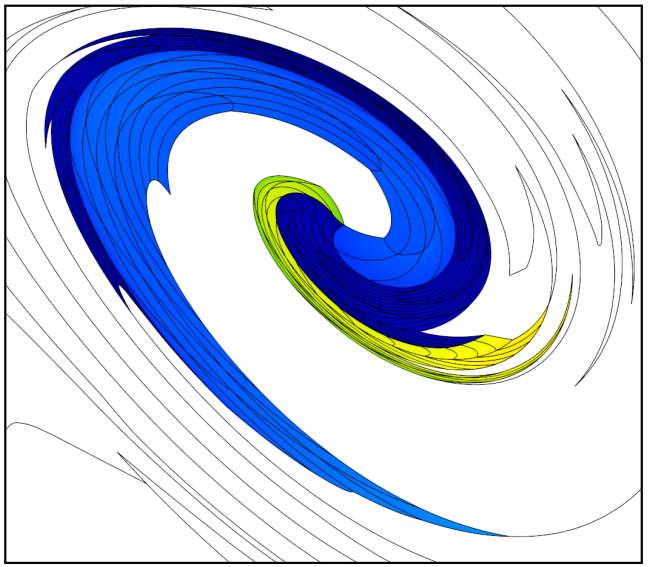
- Run on 128 processors with non-uniform partitioning (from the METIS graph partitioner)

Parallel shock triple-point interaction









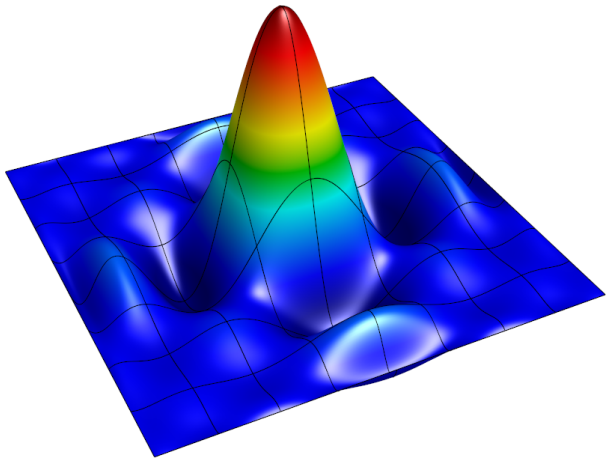
- Parallel subdomains undergo significant deformations.

Parallel high-order shock triple-point interaction

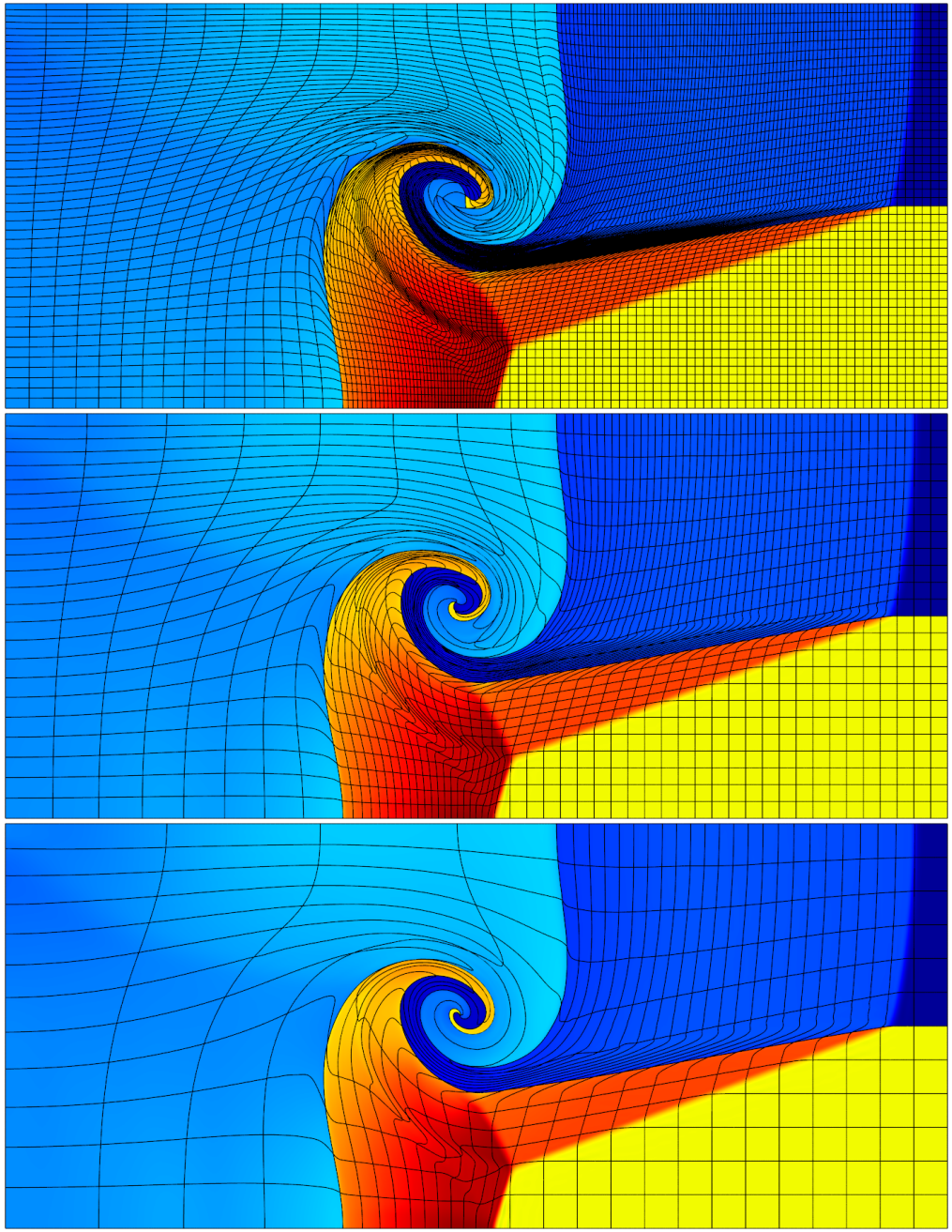
- Compare Q_2Q_1-RK2Avg, Q_4Q_3-RK4, Q_8Q_7-2×RK4.
- Same number of unknowns, 12 CPUs, $t = 3.3$.

	Q_4/Q_2	Q_8/Q_2
t_{cycle}	2.15	13.9
n_{cycles}	0.76	0.75
$\lvert\{\hat{\vec{q}}_k\}\rvert$	$6^2/4^2$	$12^2/4^2$
$nnz(\mathbf{M_v})$	2.24	6.19

- Directional length scale is divided by polynomial order.
- Higher order methods are more efficient than expected.



One of the Q_8 basis functions

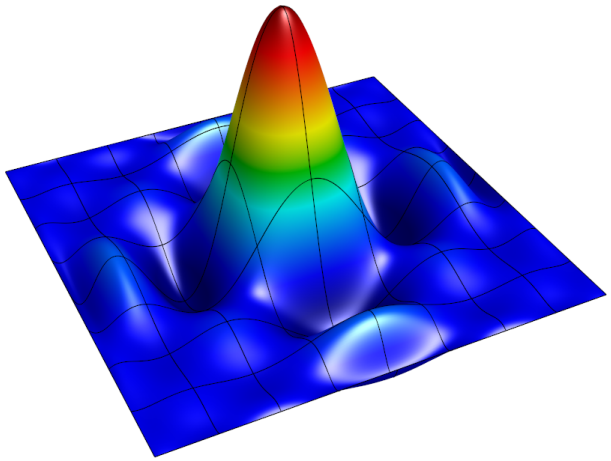


Parallel high-order shock triple-point interaction

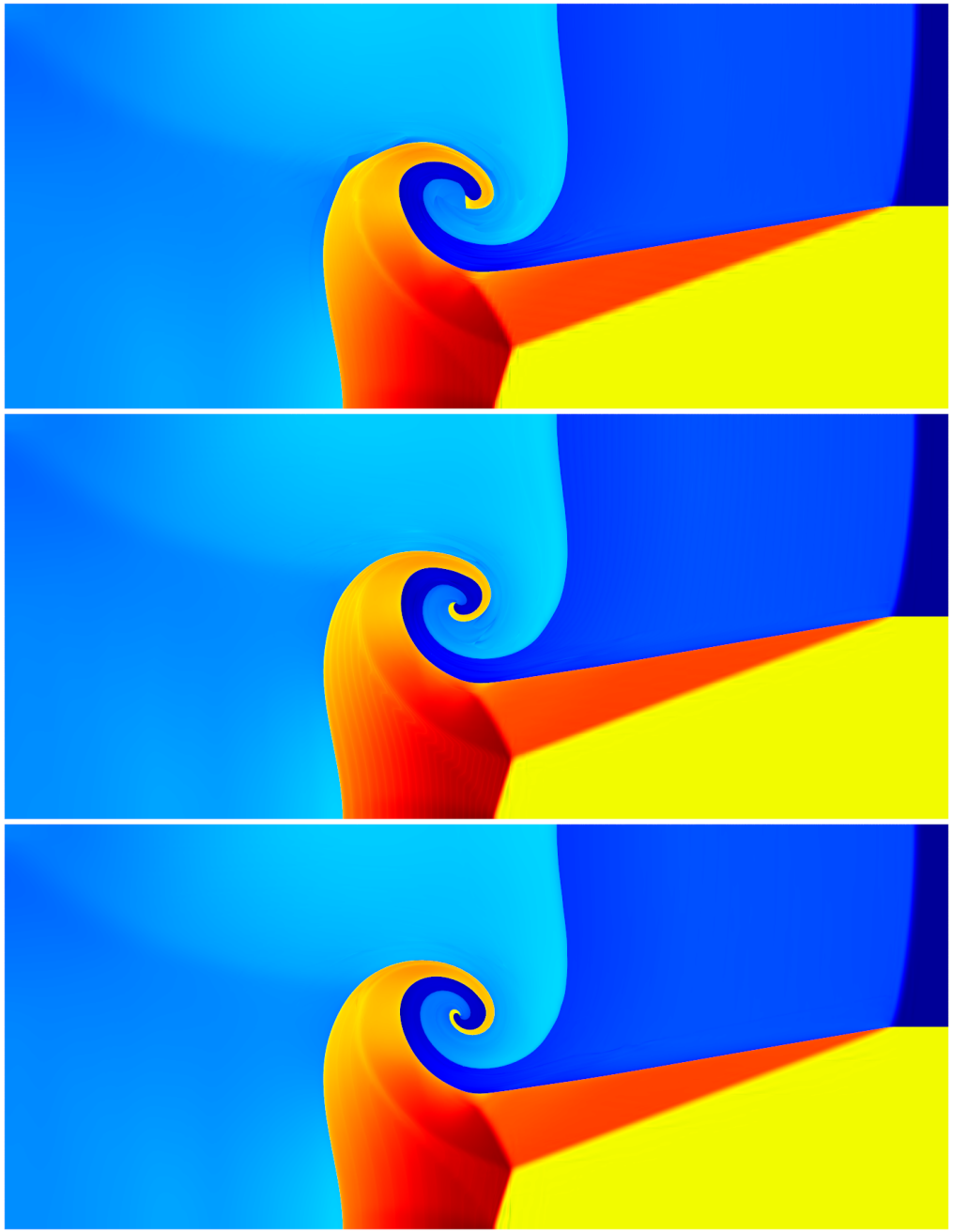
- Compare Q_2Q_1-RK2Avg, Q_4Q_3-RK4, Q_8Q_7-2×RK4.
- Same number of unknowns, 12 CPUs, $t = 3.3$.

	Q_4/Q_2	Q_8/Q_2
t_{cycle}	2.15	13.9
n_{cycles}	0.76	0.75
$\lvert\{\hat{\vec{q}}_k\}\rvert$	$6^2/4^2$	$12^2/4^2$
$nnz(\mathbf{M_v})$	2.24	6.19

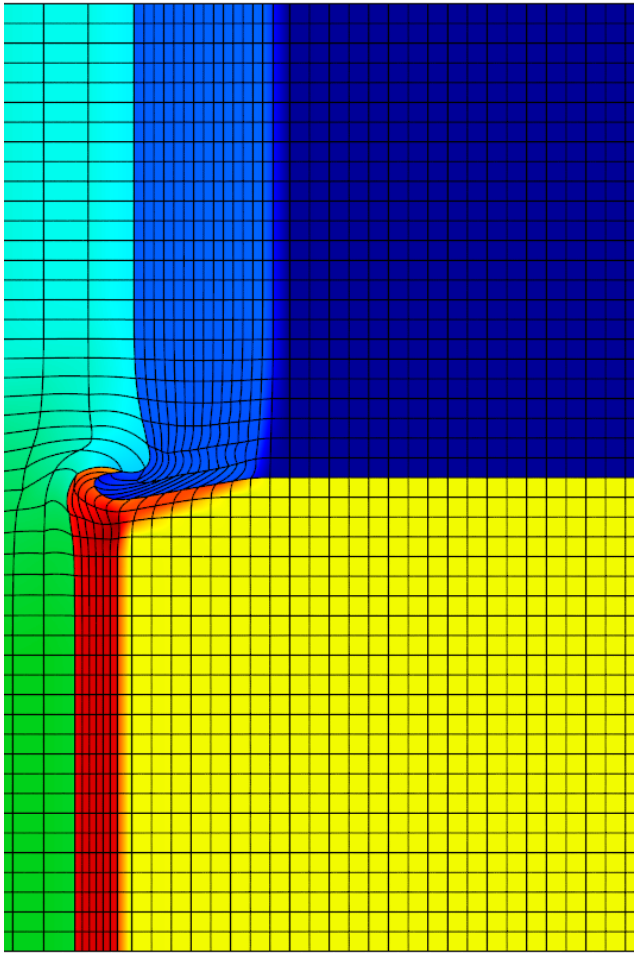
- Directional length scale is divided by polynomial order.
- Higher order methods are more efficient than expected.



One of the Q_8 basis functions

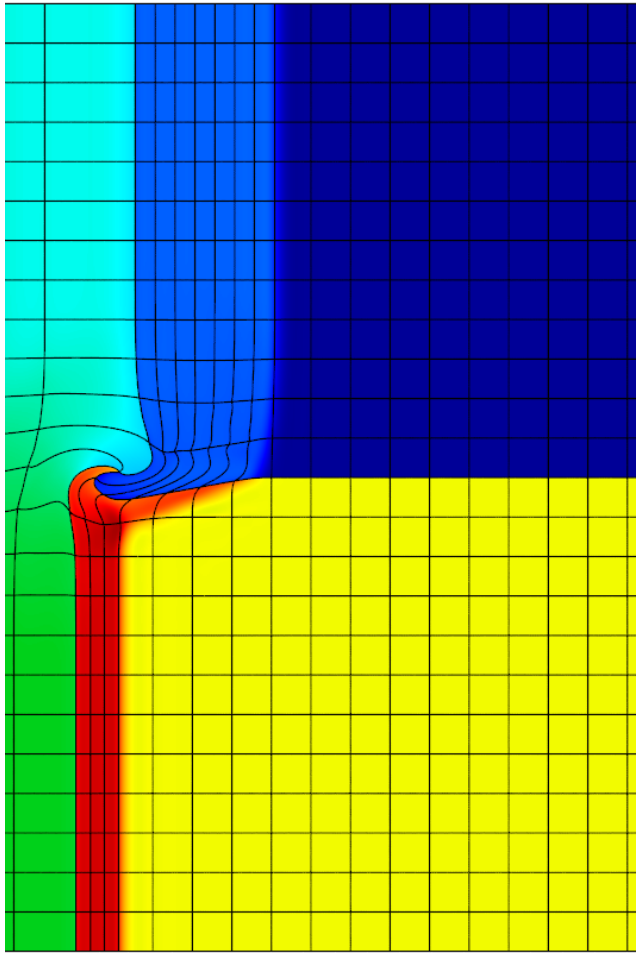


Parallel high-order shock triple-point interaction



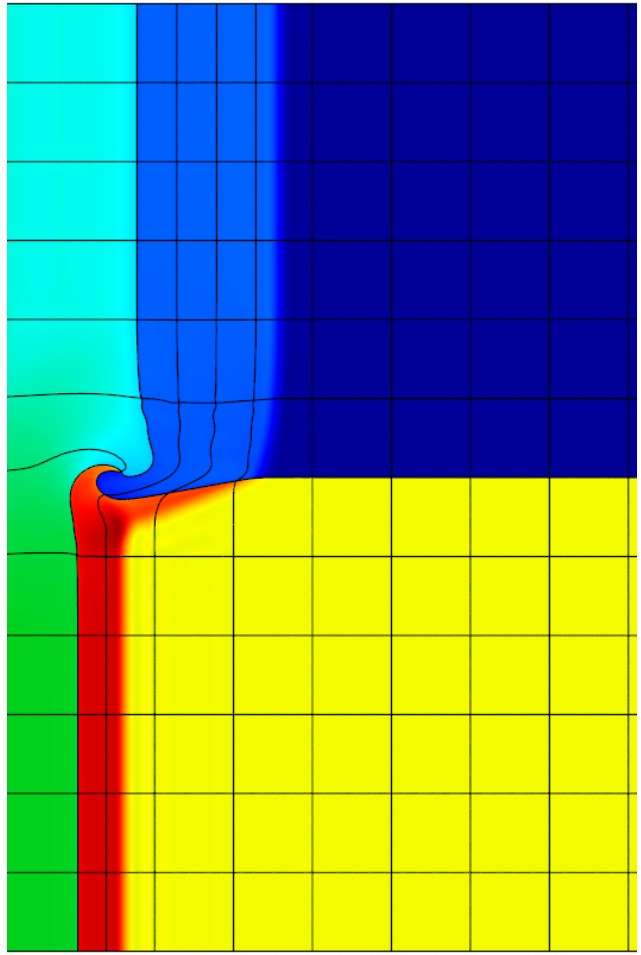
$Q_2\,Q_1$

$\mathbf{A_z}$ is 18×16
$\mathbf{B_z}$ is 4×16
$\mathbf{F_z}$ flops/bytes ≈ 29



$Q_4\,Q_3$

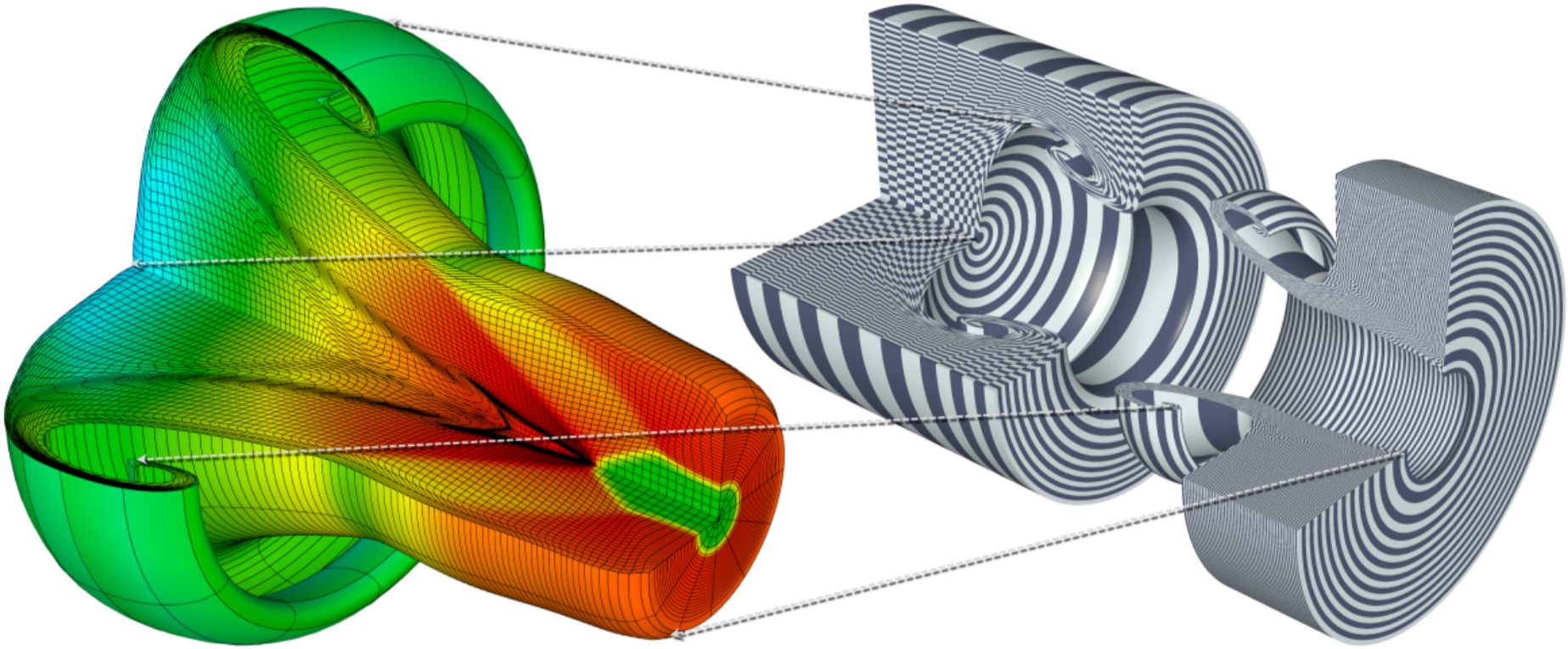
$\mathbf{A_z}$ is 50×36
$\mathbf{B_z}$ is 16×36
$\mathbf{F_z}$ flops/bytes ≈ 248



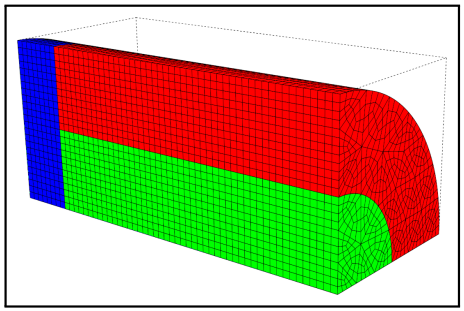
$Q_8\,Q_7$

$\mathbf{A_z}$ is 162×144
$\mathbf{B_z}$ is 64×144
$\mathbf{F_z}$ flops/bytes ≈ 3848

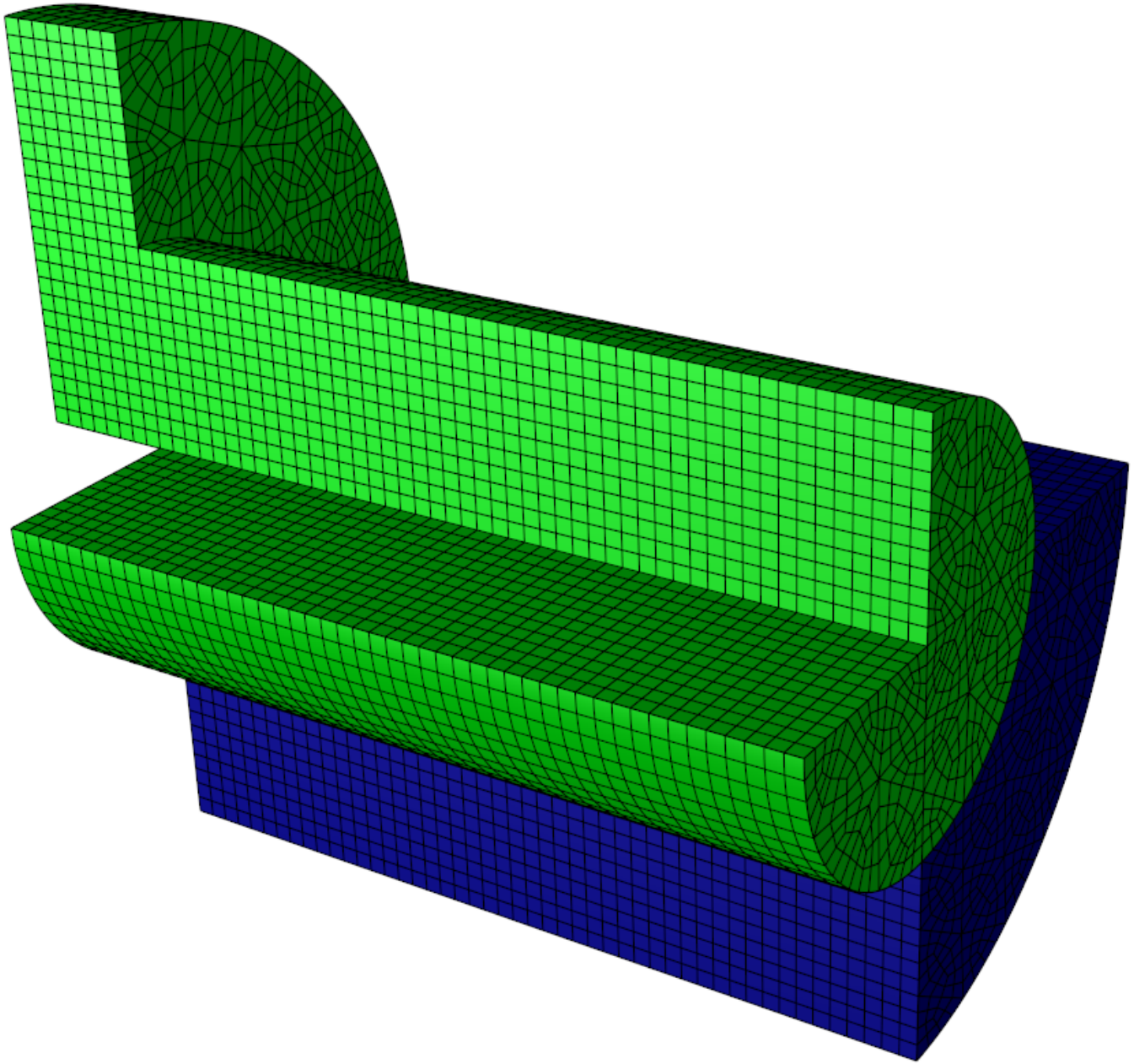
Axisymmetric shock triple-point interaction



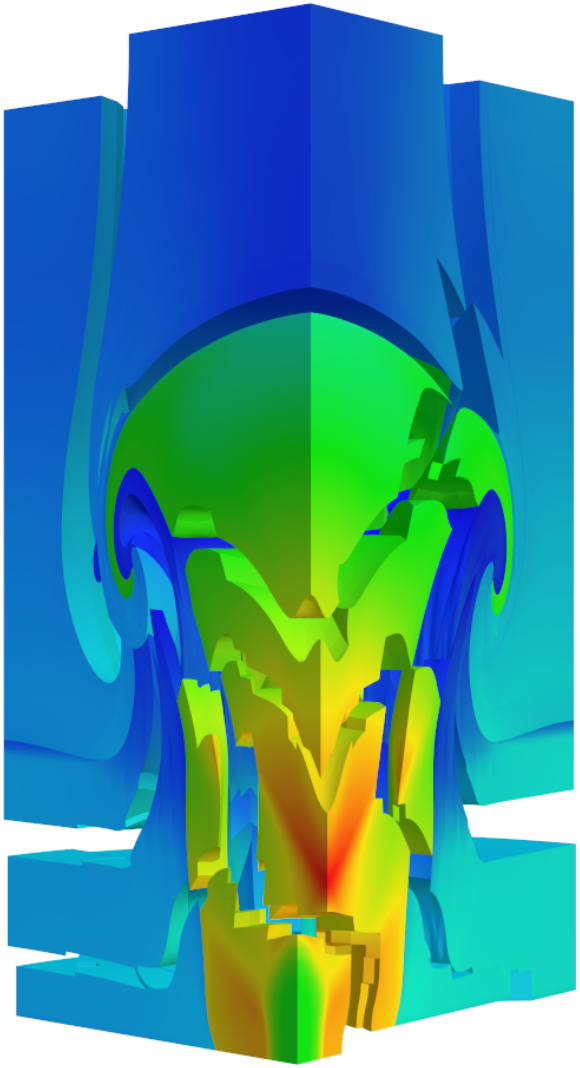
Parallel 3D shock triple-point interaction

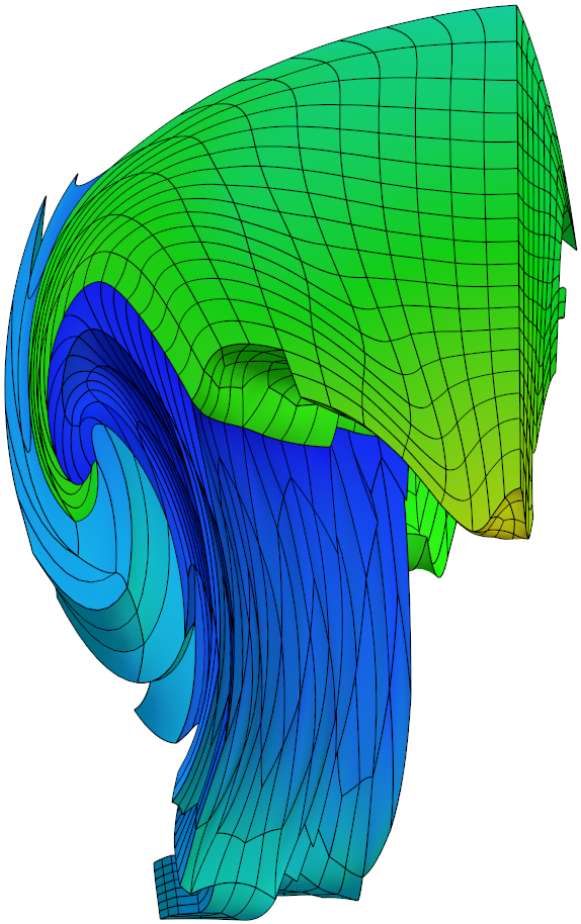


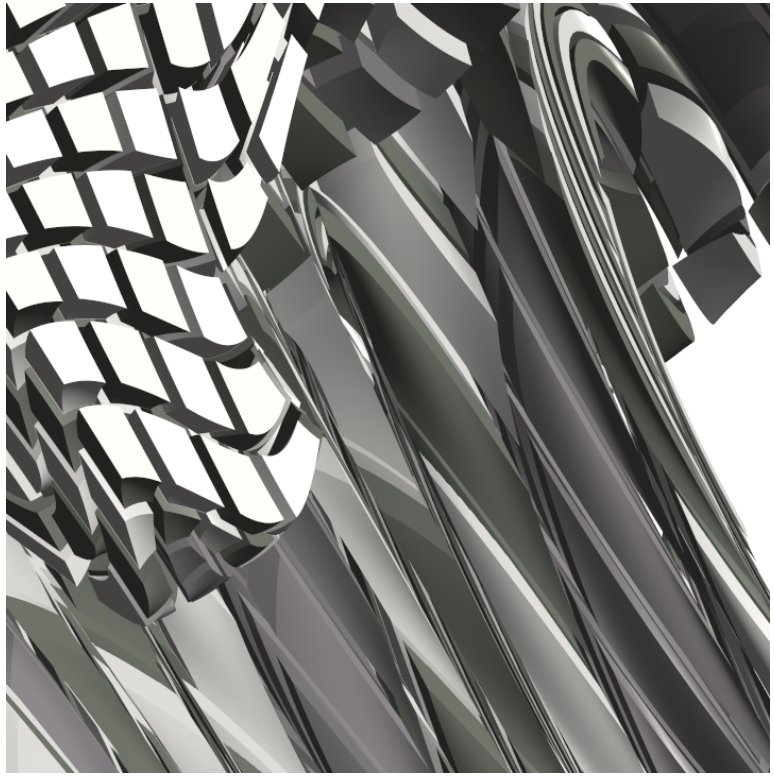
- Initial 3D mesh is unstructured in the z-orthogonal plane
- Parallel run on 10 processors with the $Q_2 Q_1$-RK2Avg method.
- Circular arcs are fitted in Q_2 position dofs which are then refined once.
- Shown is the revolved density in the three materials (logarithmic scale).

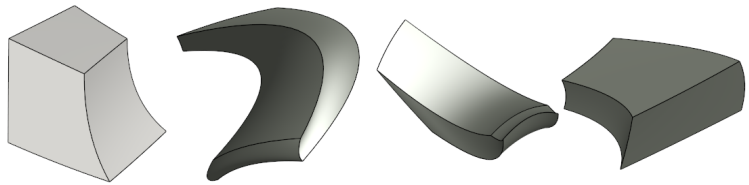


Parallel 3D shock triple-point interaction









- Unstructured parallel data decomposition.
- The robustness of the curvilinear zones extends to 3D multi-material problems.

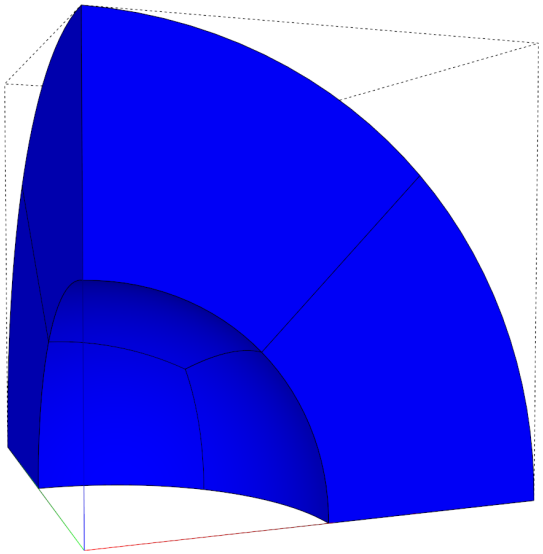
Conclusions and Future Directions

Some benefits of our high order discretization methods:

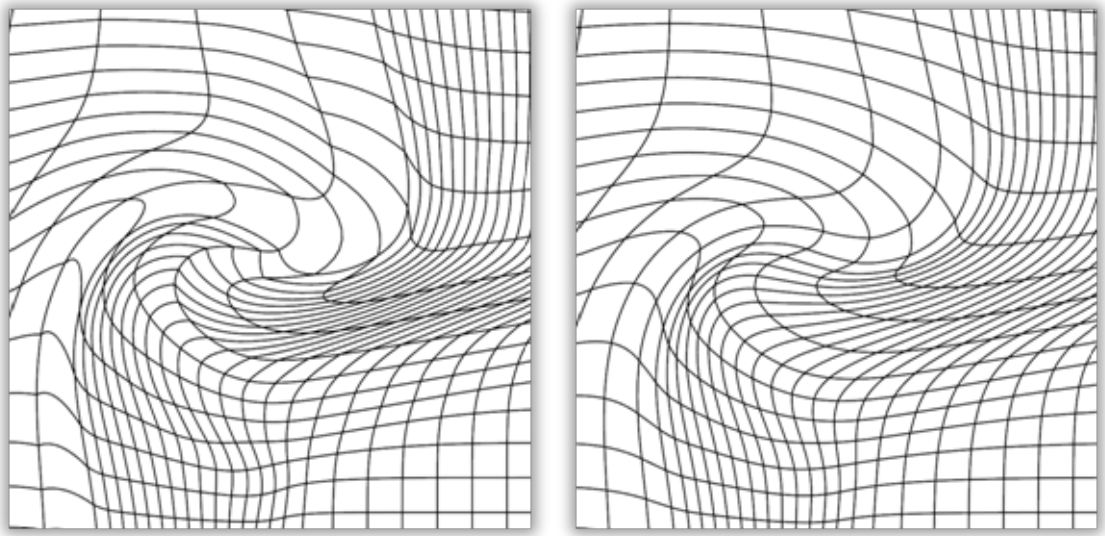
- More accurate capturing of flow features using curvilinear zones.
- Exact total energy conservation by construction.
- Substantial reduction in mesh imprinting and improved symmetry preservation.
- Same framework for 2D, 3D and axisymmetric problems.
- Locally FLOP-intensive algorithms excel on modern parallel CPU+GPU architectures.

Future research directions:

- NURBS-based hydro (collaboration with UCSD and LANL).
- ALE: curvilinear mesh optimization; high-order field remap; multi-material zone treatment.



NURBS mesh in MFEM representing a spherical shell.



Original (left) and smoothed (right) curvilinear mesh obtained by high-order extension of local harmonic smoothing.