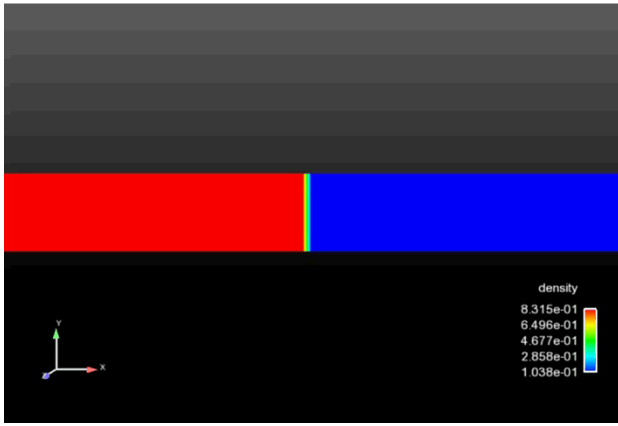
SAND2015-1456PE

*Exceptional service in the national interest*

Sandia National Laboratories

# MiniAero and Aero: An Overview






Ken Franko

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

# Outline

- Mathematical Problem/Numerical Method
- MiniAero Code Design including use of Kokkos
- Comparison with Full Application
- Asynchronous Many-Task Parallelism
- Conclusions

# Equations

Compressible Navier-Stokes Equations

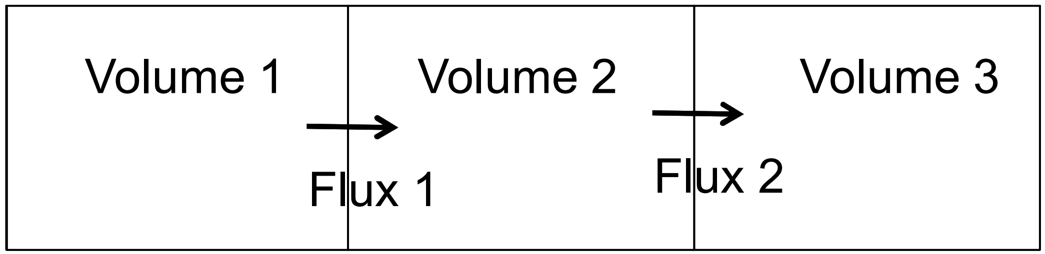
$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0$$ Mass

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho u_i u_j + P\delta_{ij}\right) = \frac{\partial \tau_{ij}}{\partial x_j}$$ Momentum

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho u_j H}{\partial x_j} = -\frac{\partial q_j}{\partial x_j} + \frac{\partial u_i \tau_{ij}}{\partial x_j}$$ Energy

# Solution Method

Finite Volume:




- Cell-centered(MiniAero) or Node-centered(Aero)
- 1$^{st}$ order or 2$^{nd}$ order in space
- Flux boundary conditions
- Inviscid/Viscous option.

Explicit RK4 Time Marching (classic RK4) (MiniAero)

Point Implicit Solver for Implicit Time Marching (Aero)

# Finite Volume Details

$$\int_{\Omega} \frac{\partial \mathbf{U}}{\partial t} dV + \int_{\delta\Omega} (\mathbf{F}_j - \mathbf{G}_j) dA_j = 0$$

$$\int_{\Omega} \frac{\partial \mathbf{U}}{\partial t} dV \approx \frac{\partial \mathbf{U}^c}{\partial t} V^c$$

$$\int_{\delta\Omega} (\mathbf{F}_j - \mathbf{G}_j) dA_j = \sum_{f=1}^{\#faces} (\mathbf{F}_j^f - \mathbf{G}_j^f) A_j^f$$

$$\frac{d\mathbf{U}^c}{dt} = \frac{1}{V^c} \sum_{f=1}^{\#faces} (\mathbf{F}_j^f - \mathbf{G}_j^f) A_j^f = \mathbf{R}(U^c)$$

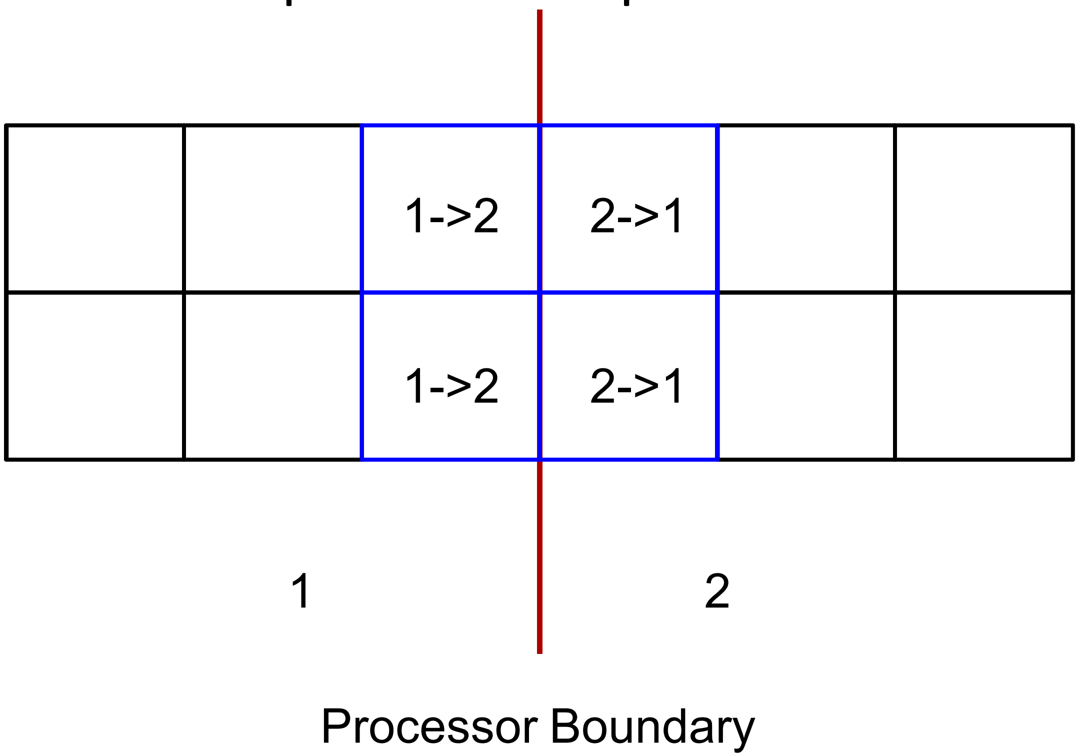
# Aero/MiniAero Summary

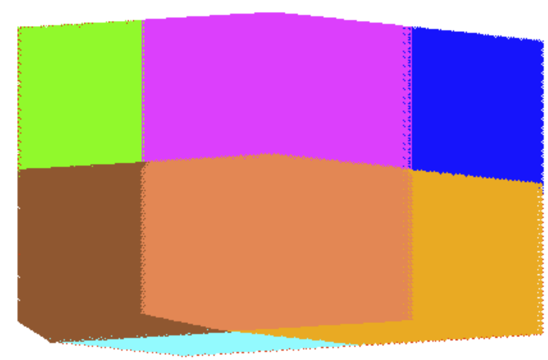
- Fully 3D unstructured finite volume
- Explicit or Implicit time marching
- Inviscid Roe Flux
- Newtonian Viscous Flux
- Ideal Gas Model

# MiniAero MPI Implementation

Sandia National Laboratories

- Decompose in multiple directions and use ghosting.



# Physics Kernels

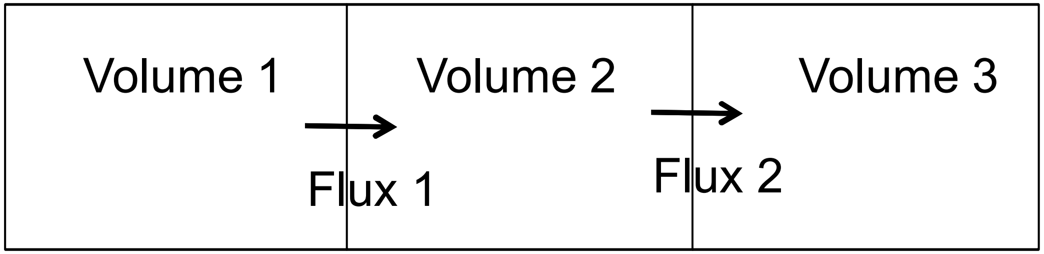
**MiniAero**

- Kokkos::Views are used to store the multi-dimensional arrays that are needed.
  - Heavy object-oriented design needs to be avoid.
  - Everything is an array (connectivity, flow state, indices)
- Functors apply a specific operation based on an index
  - **Parallel_for**, parallel_reduce, and parallel_scan.
- Heavy use of template for polymorphism.

**Aero**

- Uses Sierra Toolkit (STK).
- Loop-based – main loop is edge loop.
- Heavy use of class and virtual functions for polymorphism.

# MiniAero Assembly – Thread safety




Options

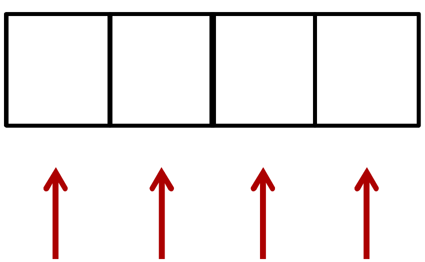
1. Store fluxes at Faces. Two loops – over faces and then over volumes. Downside: Performance and need to store flux direction for each volume.
2. **Stores fluxes for each face on volume. Two loops – over faces and then over volumes. AKA: Gather-sum**
   **Downside: Increased memory use**
3. **Atomic operations. Single loop over faces and no additional memory required.**
   **Downside: Could be slow with large number of conflicts.**

# MiniAero and Aero

| MiniAero | Aero |
|---|---|
| Cell-centered | Node-centered |
| Explicit time-marching only | Explicit and implicit |
| 1st and 2nd order in space | 1st and 2nd order in space |
| Green-Gauss Gradients | Green-Gauss Gradient |
| MPI+X | MPI only |
| Kernel Based | Loop Based |
| Threaded | Not threaded |
| Array-based mesh | Uses STK mesh |

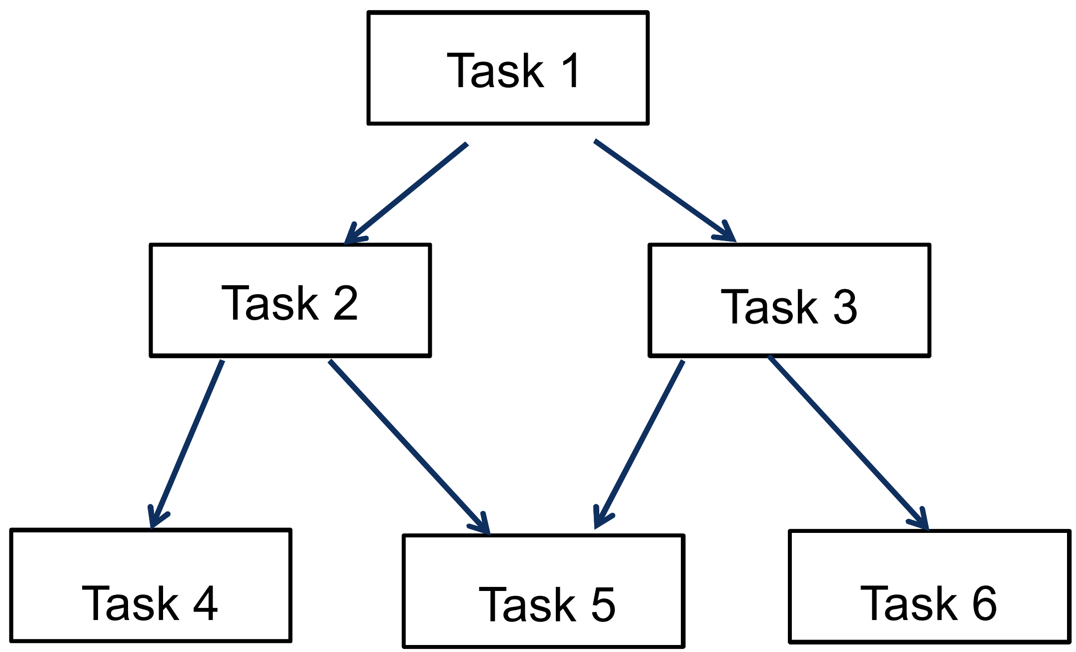
# MiniAero: Data versus Task Parallel

## Data



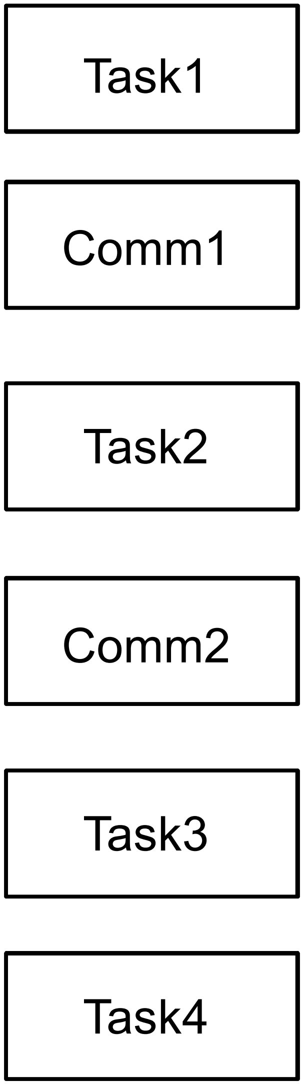
Threads/vectorization

## Task

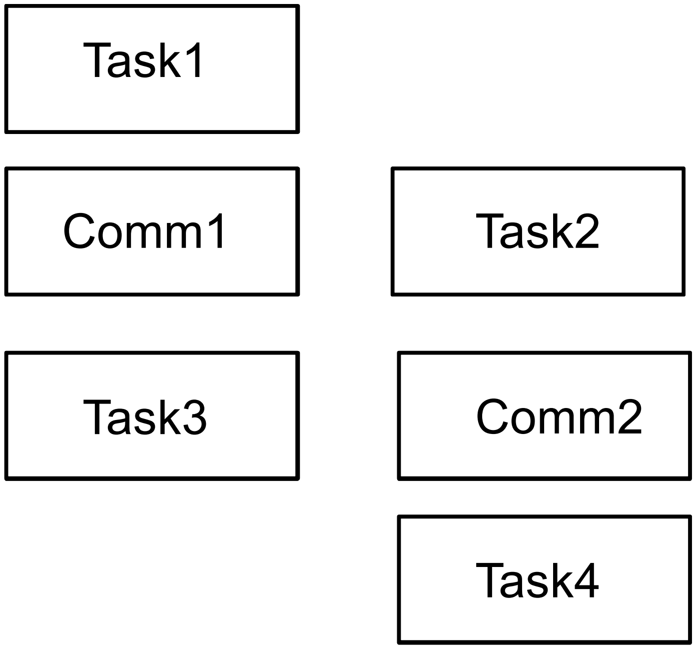


# Why Task Parallel?

## Standard MPI




## Task Parallel



# Task-Parallel Mini-Aeros

- Different approach than data-parallel, can be complementary.
- Part of L2 milestone with Dharma project(based in Sandia-CA)
- Evaluate different existing task-parallel programming models
  - Uintah
  - Legion
  - Charm++

# Conclusions

- MiniAero and Aero solve similar equations with similar methods
- Largest difference is between explicit and implicit time marching.
- Programming models are very different
  - MiniAero: heavy template use, kernels as functors, and threaded.
  - Aero: heavy run-time polymorphism, loop heavy, and MPI-only

# Questions