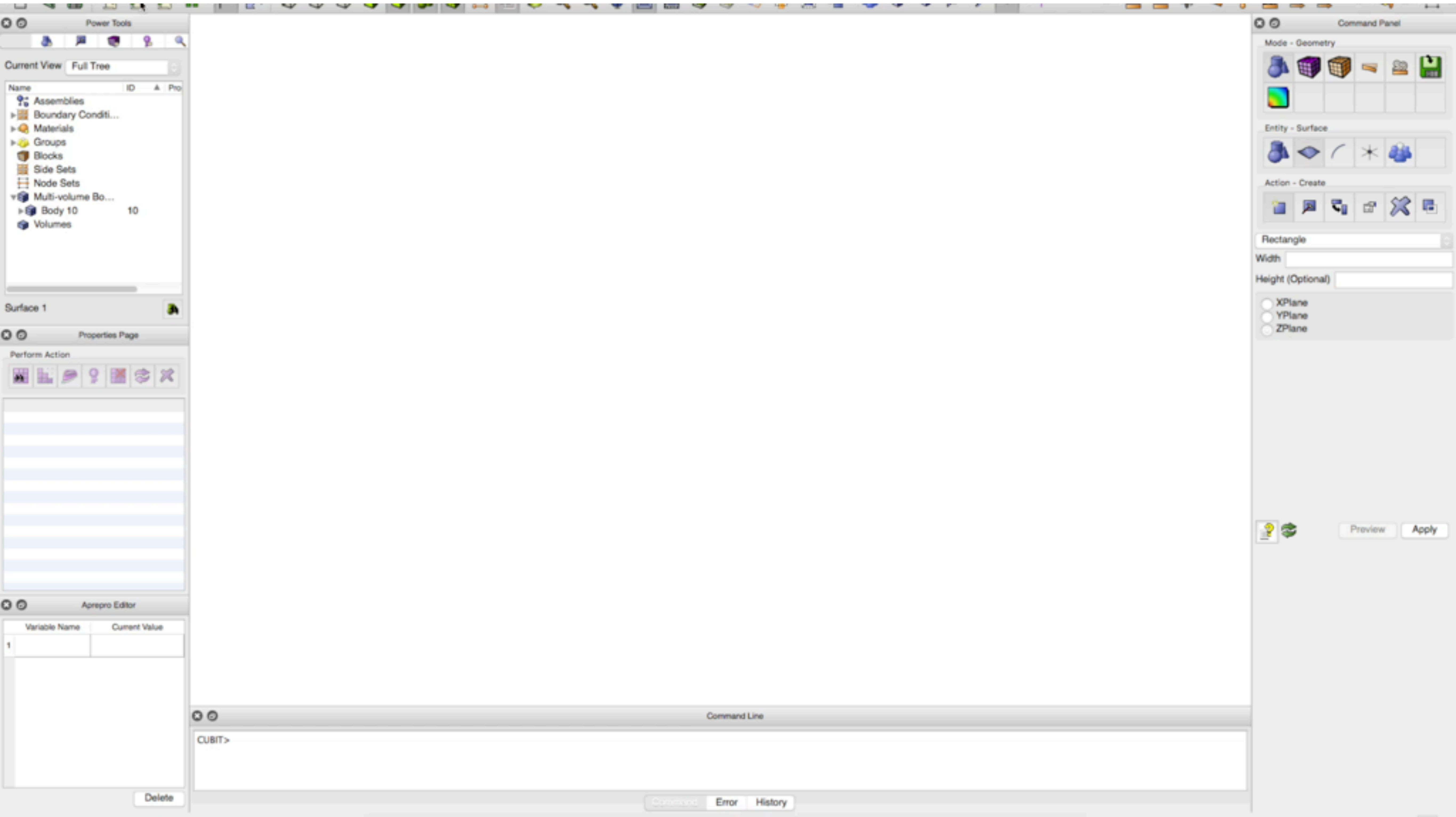
MOOSE.jl

Derek Gaston



Objective:

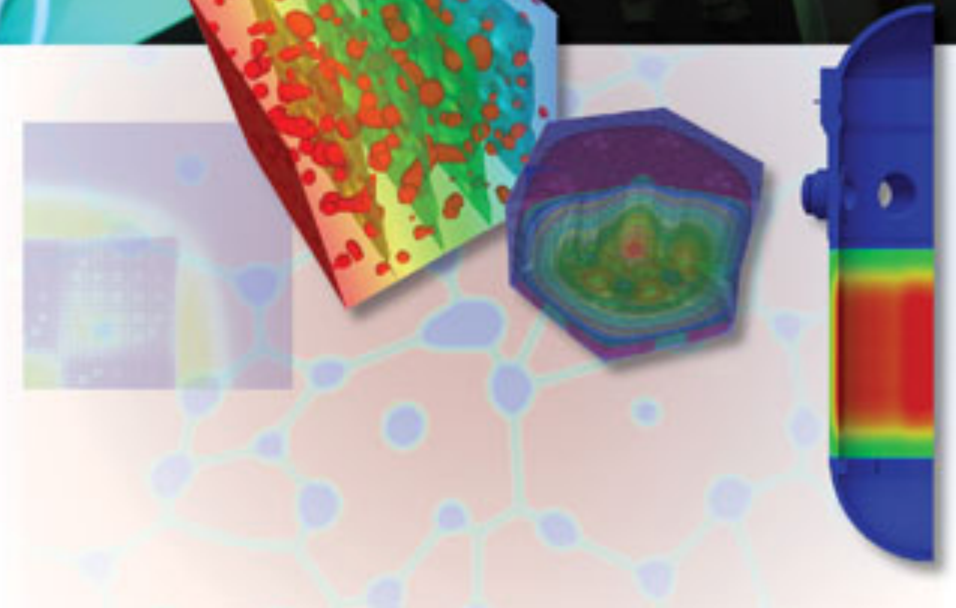
Create a scalable, finite-element multiphysics framework in Julia

MOOSE

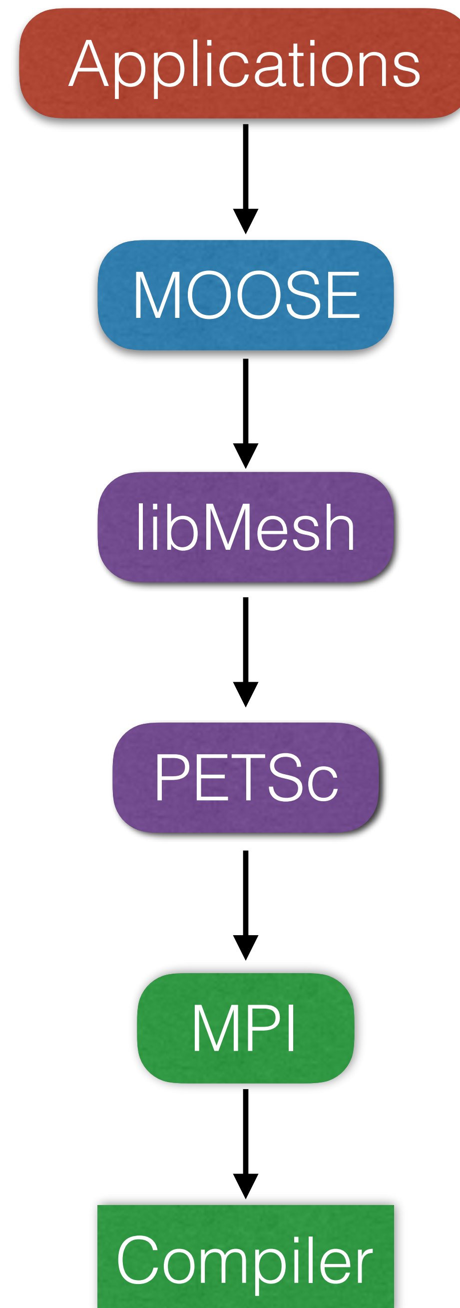
Idaho National Laboratory

Multiphysics Object Oriented Simulation Environment

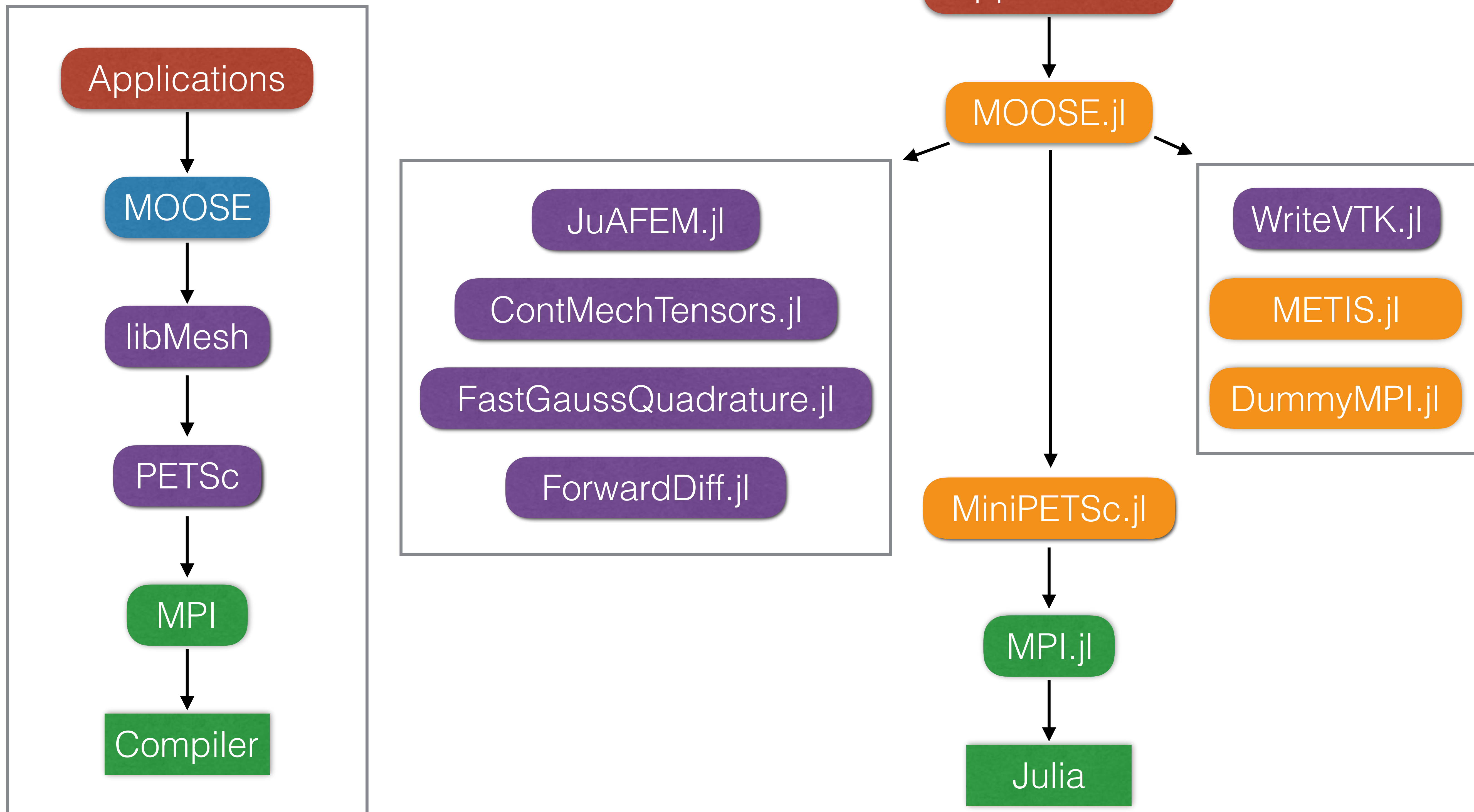
- MOOSE is a finite-element, multiphysics framework that **simplifies the development** of numerical applications.
- Provides a high-level interface to **sophisticated nonlinear solvers** and **massively parallel computational capability**.
- Used to model thermomechanics, neutronics, geomechanics, reactive transport, microstructure, computational fluid dynamics…
- **Open source** and freely available at **mooseframework.org**
- High honors:
  - Early career award from President Obama
  - R&D 100 from R&D Magazine
  - Hundreds of publications, thousands of citations



MOOSE Anatomy



MOOSE.jl Anatomy

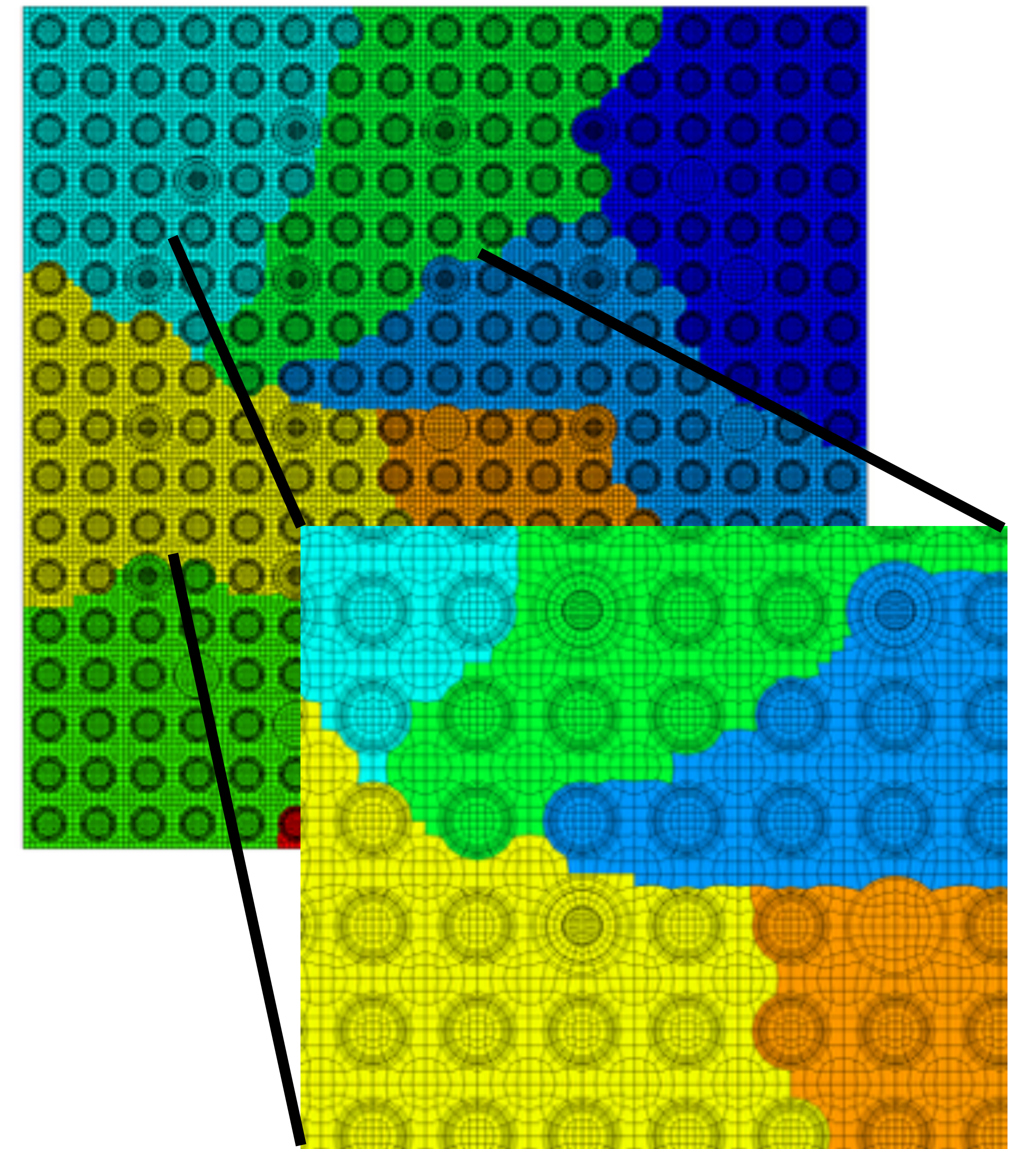


MiniPETSC.jl

- PETSc is a suite of parallel linear/nonlinear solvers written in C
- PETSc.jl already existed: year and a half of bit-rot
- MiniPETSc.jl: Small interface to PETSc:
  - Parallel Vector
  - Parallel Matrix
  - Krylov Solvers
- Wraps PETSc objects up in Julia standard interfaces:
  - AbstractVector
  - AbstractMatrix

Going Parallel

1. Interface to parallel solver (MiniPETSc.jl)
2. Partition mesh (METIS.jl)
3. Assign contiguous degree of freedom (DoF) numberings
4. Create “ghosted” vectors
5. Iterate over local elements, scatter into parallel Matrix and Vector
6. Solve!
7. Serialize solution to processor 0, write to VTK



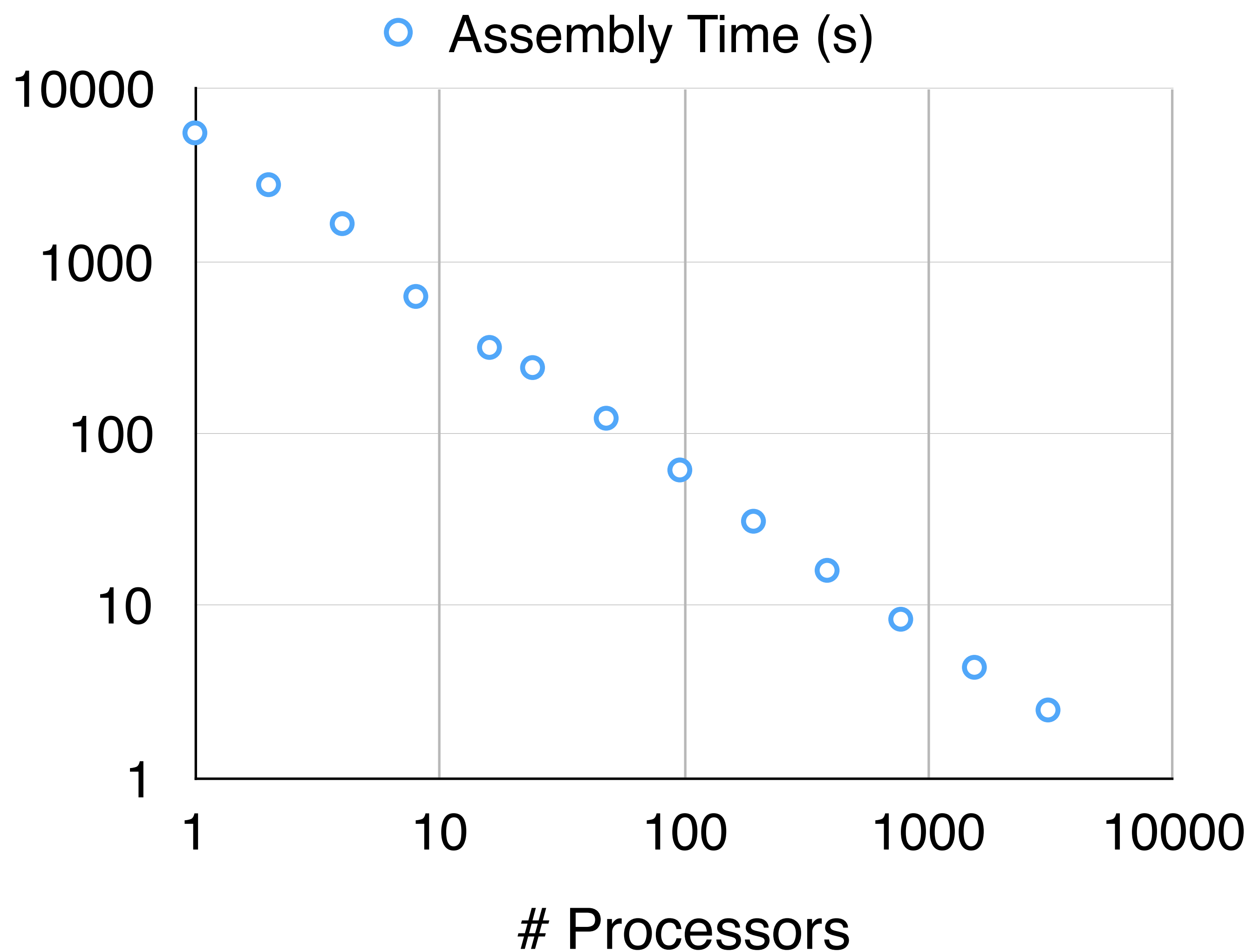
Domain decomposed nuclear reactor geometry for 8 processors

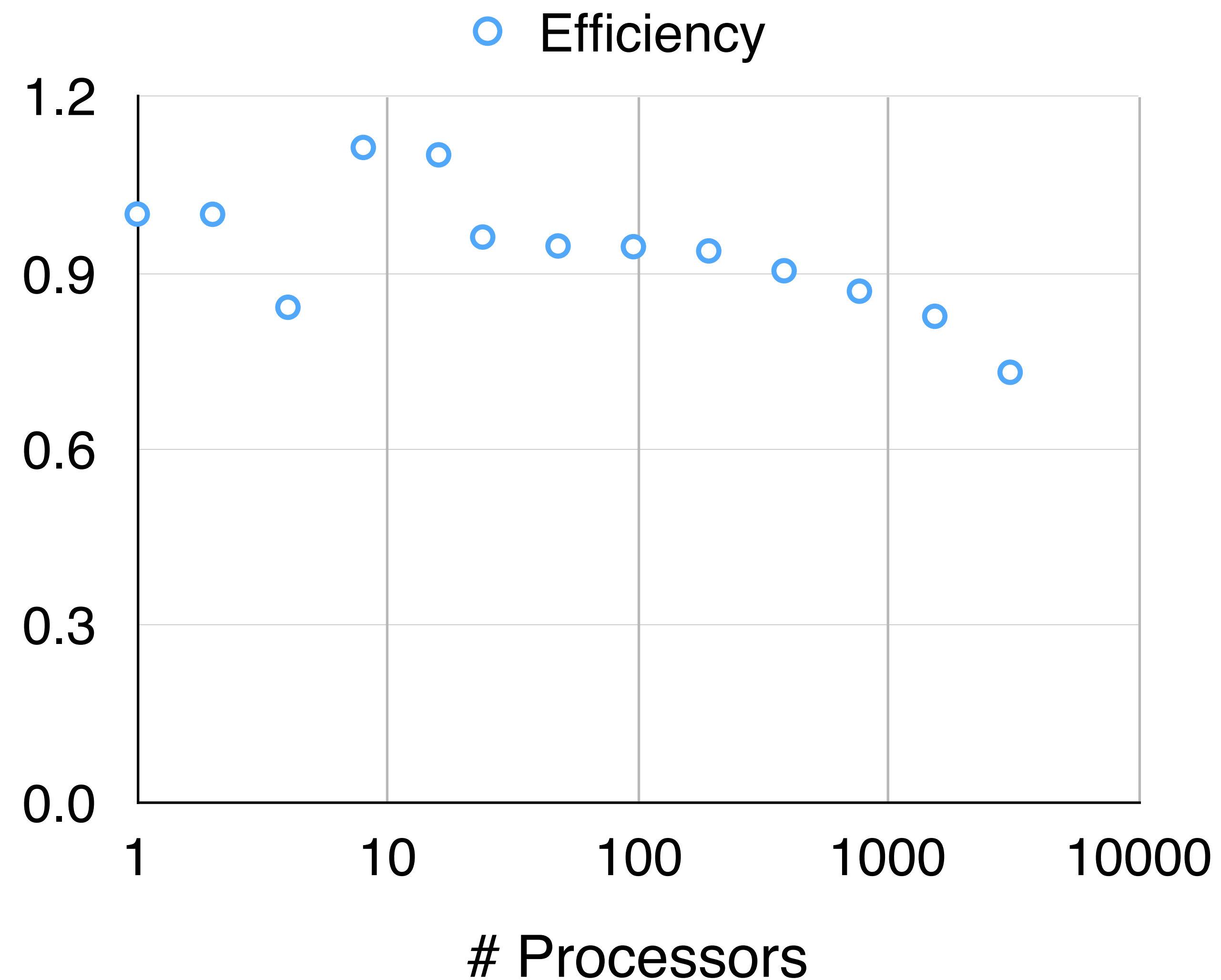
Scalability

$$-\nabla \cdot \nabla u_i + \sum_{k,k\neq i} \nabla u_k \nabla u_i = 0$$

$$u_i = 0, u_i \in S_{left}$$

$$u_i = 1, u_i \in S_{right}$$





Strong scaling: 400x400 elements, 25 coupled PDEs
Falcon Cluster at Idaho National Laboratory

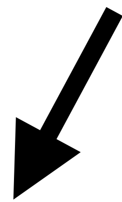
Using MOOSE.jl

Strong Form: $-\nabla \cdot \nabla u = 0$

Weak Form: $$\int_V \nabla u \cdot \nabla \phi_i dV - \int_S (\nabla u \cdot \vec{n}) \phi_i dS = 0$$

Kernel

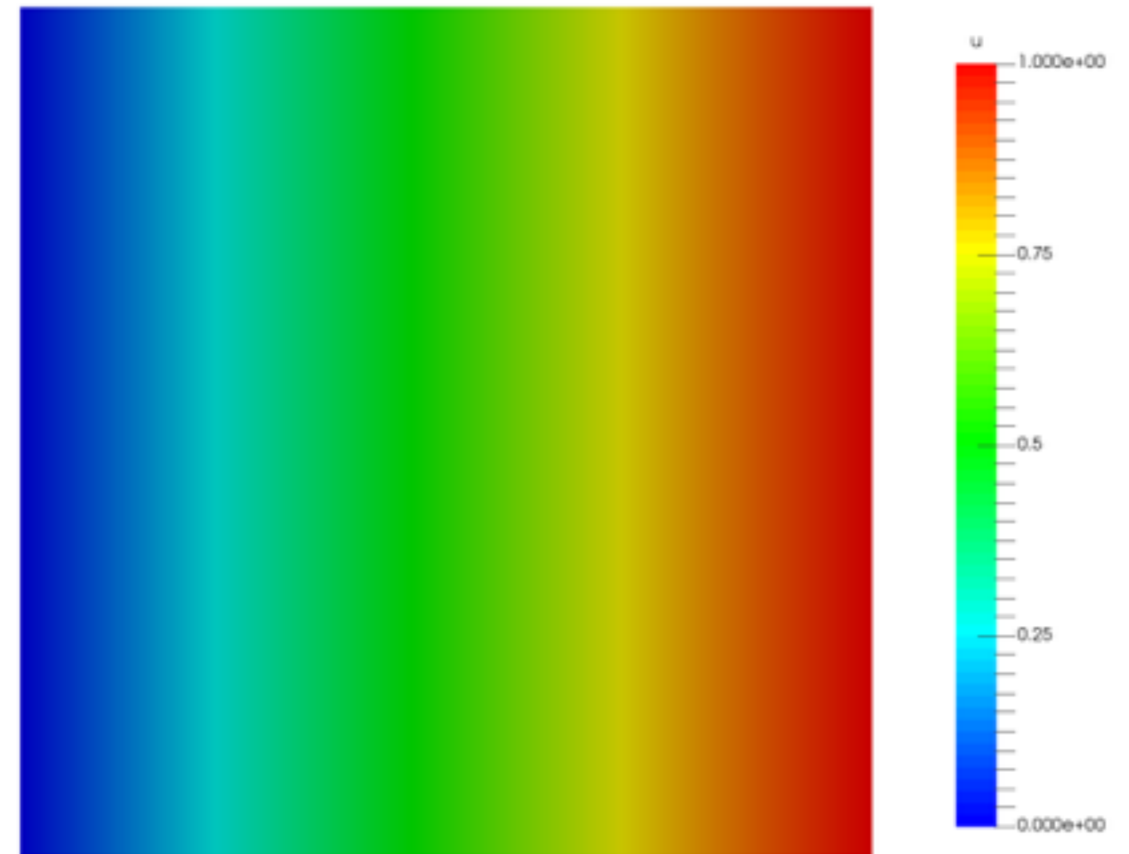
BoundaryCondition

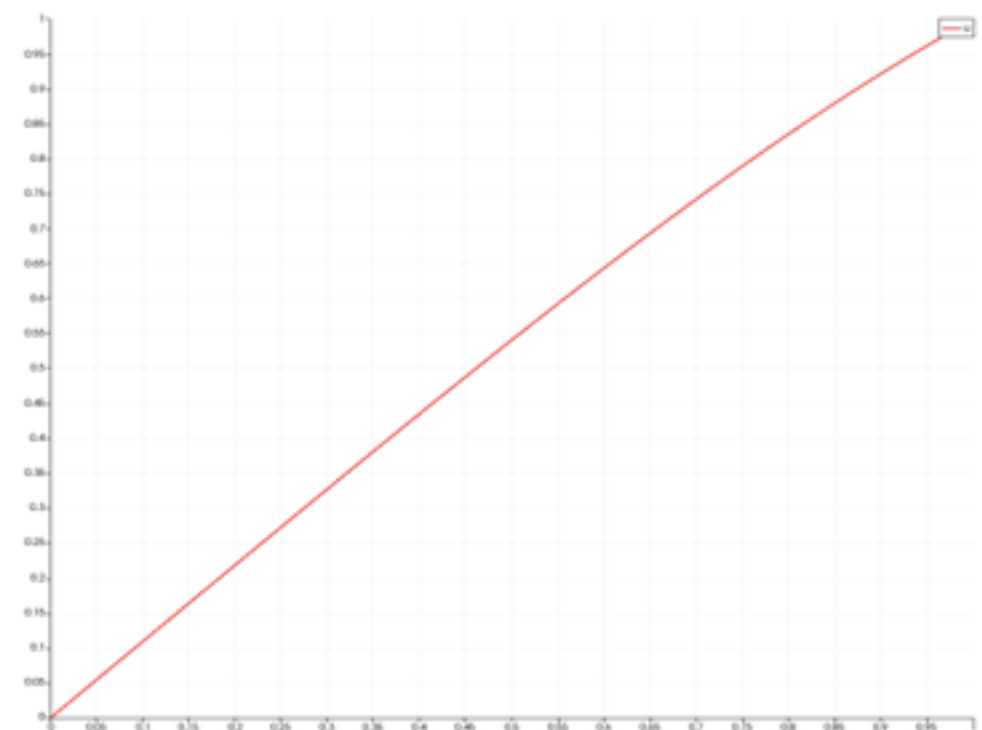


```
@inline function computeQpResidual(kernel::Diffusion, qp::Integer, i::Integer)
    u = kernel.u

    return u.grad[qp] · u.grad_phi[qp][i]
end
```

Example:

```
using MOOSE

include("NonlinearForce.jl")

# Create the Mesh
mesh = buildSquare(0, 1, 0, 1, 2, 2)

# Create the System to hold the equations
diffusion_system = System{Float64}(mesh)

# Add a variable to solve for
u = addVariable!(diffusion_system, "u")

# Apply the Laplacian operator to the variable
addKernel!(diffusion_system, Diffusion(u))

# Apply the NonlinearForce operator
addKernel!(diffusion_system, NonlinearForce(u))

# u = 0 on the Left
addBC!(diffusion_system, DirichletBC(u, [4], 0.0))

# u = 1 on the Right
addBC!(diffusion_system, DirichletBC(u, [2], 1.0))

# Initialize the system of equations
initialize!(diffusion_system)

# Create a solver and solve
solver = PetscNonlinearImplicitSolver(diffusion_system)
solve!(solver, nl_max_its=5)

# Output
out = VTKOutput()
output(out, solver, "nonlinear_force_out")
```





https://github.com/friedmud/MOOSE.jl