SAND2012-1207 C

Template-based Generic Programming Applied to Large-scale Multiphysics Simulation

Roger Pawlowski, Eric Cyr, Patrick Notz, Eric Phipps,
Andrew Salinger, and John Shadid
Sandia National Laboratories

SIAM Conference on Parallel Processing
MS30 Directed Acyclic Graph Approaches for Parallel Scientific Computing
Thursday Feb 16th, 2012

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.

Outline

- Motivation
- Requirements
  - Analysis Requirements
  - Complexity Requirements
- Solution:
  - DAG Implementation
  - Template-based Generic Programming
- Examples
- Conclusions

Motivation

Achieving Scalable Predictive Simulations of Complex Highly Nonlinear Multi-physics PDE Systems

- Multiphysics systems are characterized by a myriad of complex, interacting, nonlinear multiple time- and length-scale physical mechanisms:
  - Dominated by **short dynamical time-scales**
  - Widely separated time-scales **(stiff system)**
  - Evolve a solution on a **long time scale relative to component time scales**
  - Balance to produce **steady-state** behavior.

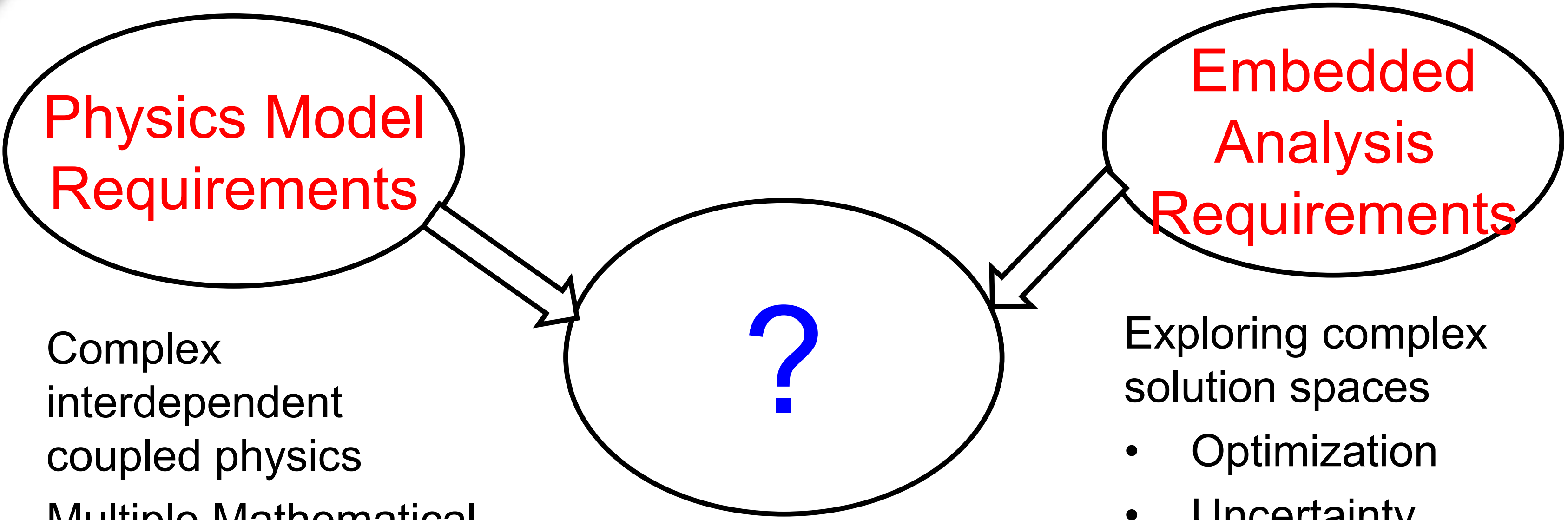
Explicit Methods

Typically requires some form of Implicit Methods

e.g. Nuclear Fission / Fusion Reactors; Conventional /Alternate Energy Systems; High Energy Density Physics; Astrophysics; etc ….

- Our approach:
  - Stable and higher-order accurate implicit formulations and discretizations
  - Robust, scalable and efficient prec. for fully-coupled Newton-Krylov methods
  - Integrate sensitivity and error-estimation to enable UQ capabilities.

Complexity in Multiphysics Simulation



Physics Model Requirements

?

Embedded Analysis Requirements

- Complex interdependent coupled physics
- Multiple Mathematical Models
- Multiple Numerical Formulations

Flexible, extensible, maintainable and EFFICIENT!

Exploring complex solution spaces

- Optimization
- Uncertainty Quantification
- Bifurcation analysis

- Supporting multiplicity in models and solution techniques often leads to complex code with **complicated logic** and **fragile software designs!**

What does *embedded* mean?

- We used to call this *intrusive*
- Generally anything that requires more of a simulation code than just running it
  - i.e., not black-box or non-intrusive
- Why do this?
  - By asking for more, improvements can be made
    - Increased efficiency, scalability, robustness
    - Greater understanding through deeper analysis

Analysis Requirements (1)

- Model problem

$$f(\dot{x}, x, p) = 0, \quad \dot{x}, x \in \mathbb{R}^n, \quad p \in \mathbb{R}^m, \quad f : \mathbb{R}^{2n+m} \rightarrow \mathbb{R}^n$$

- Direct to steady-state, implicit time-stepping, linear stability analysis

$$\left(\alpha \frac{\partial f}{\partial \dot{x}} + \beta \frac{\partial f}{\partial x} \right) \Delta x = -f$$

- Steady-state parameter continuation

$$\begin{aligned} f(x^{(n)}, p^{(n)}) &= 0 \\ g(x^{(n)}, p^{(n)}) &= v_x^T (x^{(n)} - x^{(n-1)}) + v_p^T (p^{(n)} - p^{(n-1)}) - \Delta s_n = 0 \\ \longrightarrow \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial p} \\ v_x^T & v_p^T \end{bmatrix} &\begin{bmatrix} \Delta x^{(n)} \\ \Delta p^{(n)} \end{bmatrix} = - \begin{bmatrix} f \\ g \end{bmatrix} \end{aligned}$$

- Bifurcation analysis

$$\begin{aligned} f(x, p) &= 0, \\ \sigma(x, p) &= 0, \end{aligned} \quad \sigma = -u^T J v, \quad \frac{\partial \sigma}{\partial x} = -u^T \frac{\partial}{\partial x}(Jv), \quad \frac{\partial \sigma}{\partial p} = -u^T \frac{\partial}{\partial p}(Jv),$$

$$\begin{bmatrix} J & a \\ b^T & 0 \end{bmatrix} \begin{bmatrix} v \\ s_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} J^T & b \\ a^T & 0 \end{bmatrix} \begin{bmatrix} u \\ s_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Analysis Requirements (2)

- Steady-state sensitivity analysis

$$
\begin{aligned}
&f(x^*, p) = 0, \quad s^* = g(x^*, p) \implies \\
&\qquad \frac{ds^*}{dp} = -\frac{\partial g}{\partial x}(x^*, p) \left(\frac{\partial f}{\partial x}(x^*, p) \right)^{-1} \frac{\partial f}{\partial p}(x^*, p)) + \frac{\partial g}{\partial p}(x^*, p)
\end{aligned}
$$

- Transient sensitivity analysis

$$
\begin{gathered}
f(\dot{x}, x, p) = 0, \\
\frac{\partial f}{\partial \dot{x}} \frac{\partial \dot{x}}{\partial p} + \frac{\partial f}{\partial x} \frac{\partial x}{\partial p} + \frac{\partial f}{\partial p} = 0
\end{gathered}
$$

Analysis Requirements (3)

- **Steady-state stochastic problem (for simplicity):**

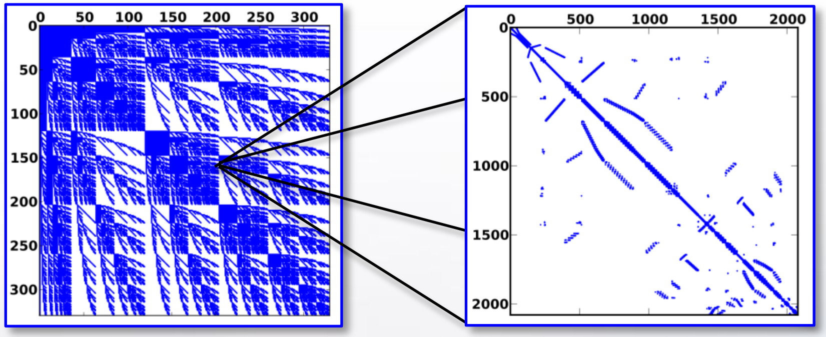
$$\text{Find } u(\xi) \text{ such that } f(u,\xi) = 0,\ \xi : \Omega \to \Gamma \subset R^M, \text{ density } \rho$$

- **Stochastic Galerkin method (Ghanem and many, many others…):**

$$\hat{u}(\xi) = \sum_{i=0}^{P} u_i \psi_i(\xi) \to F_i(u_0, \dots, u_P) = \frac{1}{\langle \psi_i^2 \rangle} \int_\Gamma f(\hat{u}(y), y) \psi_i(y) \rho(y) dy = 0, \quad i = 0, \dots, P$$

- **Method generates new coupled spatial-stochastic nonlinear problem (intrusive)**

$$0 = F(U) = \begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_P \end{bmatrix}, \quad U = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_P \end{bmatrix} \qquad \frac{\partial F}{\partial U} :$$



Stochastic sparsity

Spatial sparsity

- **Advantages:**
  - **Many fewer stochastic degrees-of-freedom for comparable level of accuracy**
- **Challenges:**
  - **Computing SG residual and Jacobian entries in large-scale, production simulation codes**
  - **Solving resulting systems of equations efficiently**

Physics Model Requirements

- Changing Models:
  - New equation sets
  - New material models/source terms
- Arbitrary Precision
- Block Operators for physics-based and block-aggregate preconditioning
- Integration with Third Party Libraries

Example:
Incomp. flow + Energy Cons.

	DOF
$R_{\mathbf{u}} = \dfrac{\partial(\rho\mathbf{v})}{\partial t} + \nabla\cdot(\rho\mathbf{v}\otimes\mathbf{v} + \mathbf{T}) - \rho\mathbf{g} = 0$	$\mathbf{v}$
$R_p = \dfrac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0$	P
$R_e = \dfrac{\partial(\rho e)}{\partial t} + \nabla\cdot[\rho\mathbf{v}e + \mathbf{q}] - T : \nabla\mathbf{v} = 0$	ρe

$$\mathbf{T} = P\mathbf{I} + \frac{2}{3}\mu(\nabla\cdot\mathbf{u})\mathbf{I} - \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$$

Example: Extra Operators for MHD

$$
\begin{array}{l|c}
 & \text{DOF} \\
R_{\mathbf{u}} = \frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla\cdot(\rho\mathbf{v}\otimes\mathbf{v} + \mathbf{T}\boxed{-\mathbf{T}_m}) - \rho\mathbf{g} = 0 & \mathbf{v} \\
R_p = \frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0 & P \\
R_e = \frac{\partial(\rho e)}{\partial t} + \nabla\cdot[\rho\mathbf{v}e + \mathbf{q}] - T : \nabla\mathbf{v}\boxed{-\eta\|\frac{1}{\mu_0}\nabla\times\mathbf{B}\|^2} = 0 & \rho e \\
\boxed{R_B = \frac{\partial\mathbf{B}}{\partial t} - \nabla\times(\mathbf{u}\times\mathbf{B}) + \nabla\times(\frac{\eta}{\mu_0}\nabla\times\mathbf{B}) = 0} & \boxed{\mathbf{B}}
\end{array}
$$

$$\mathbf{T} = P\mathbf{I} + \frac{2}{3}\mu(\nabla\cdot\mathbf{u})\mathbf{I} - \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$$

$$\boxed{\mathbf{T}_{\mathbf{M}} \quad = \quad \frac{1}{\mu_0}\mathbf{B}\otimes\mathbf{B} - \frac{1}{2\mu_0}\|\mathbf{B}\|^2\mathbf{I}}$$

- **New DOFs require new derivatives**
- **New material model requires new derivatives for ALL possible equations!**
- **Can we avoid explosion of derivative implementations for different DOFs?**

Example: Simplification in 2D Leads to Change of Variables

	DOF
$R_{\mathbf{u}} = \frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla\cdot(\rho\mathbf{v}\otimes\mathbf{v} + \mathbf{T} - \mathbf{T}_m) - \rho\mathbf{g} = 0$	$\mathbf{v}$
$R_p = \frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0$	P
$R_e = \frac{\partial(\rho e)}{\partial t} + \nabla\cdot[\rho\mathbf{v}e + \mathbf{q}] - T : \nabla\mathbf{v} - \eta\|\frac{1}{\mu_0}\nabla\times\mathbf{B}\|^2 = 0$	ρe
$\boxed{R_{A_z} = \frac{\partial A_z}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{A_z} - \frac{\eta}{\mu_0}\nabla^2\mathbf{A_z} = 0}$	$\boxed{A_z}$

$$\mathbf{T} = P\mathbf{I} + \frac{2}{3}\mu(\nabla\cdot\mathbf{u})\mathbf{I} - \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$$

$$\mathbf{T_M} = \frac{1}{\mu_0}\mathbf{B}\otimes\mathbf{B} - \frac{1}{2\mu_0}\|\mathbf{B}\|^2\mathbf{I}$$

$$\boxed{\mathbf{B} = \nabla\times\mathbf{A}}$$

- **Reuse MHD**
- **Added new equation and DOF**
- **Can we avoid Explosion of derivative implementations for different DOF?**

Example: Change of Variables for Compressible (Conservative) Form

	DOF
$R_{\mathbf{u}} = \frac{\partial(\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} + \mathbf{T}) - \rho \mathbf{g} = 0$	$\boxed{\rho \mathbf{v}}$
$R_p = \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$	$\boxed{\rho}$
$R_e = \frac{\partial(\rho e)}{\partial t} + \nabla \cdot [\rho \mathbf{v} e + \mathbf{q}] - T : \nabla \mathbf{v} = 0$	ρe

$$\mathbf{T} = P\mathbf{I} + \frac{2}{3}\mu(\nabla \cdot \mathbf{u})\mathbf{I} - \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)$$

- **Reuse basic equations**
- **New DOFs**
- **Can we avoid Explosion of derivative implementations for different DOF?**

Formulations/Equations of State (HydroMagnetic Thermal Cavity)

Changing the models changes the dependency chain, complicating the generation of sensitivities for implicit methods

Constant Density - Strictly incompressible

$$\rho = \rho_0 = \text{ constant}$$

Boussinesq Approximation

$$\rho \approx \rho_0 + \frac{\partial \rho}{\partial T}\Big|_0 (T - T_0) \text{ in momentum body force term}$$

$$\rho = \rho_0 \text{ and everywhere else}$$

Variable density Formulations

Low Flow Mach Number Approximation

$$\rho = f(P_{th}, T, Y_i) \text{ where } P_{th} \text{ is thermodynamic not hydrodynamic pressure (P)}$$

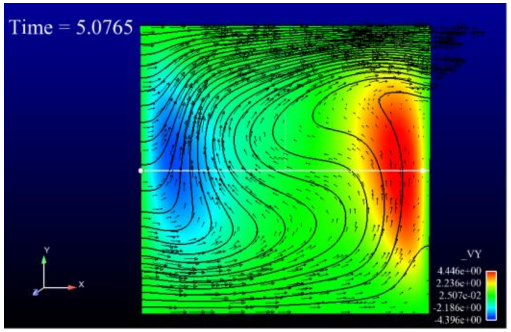
Anelastic Approximation

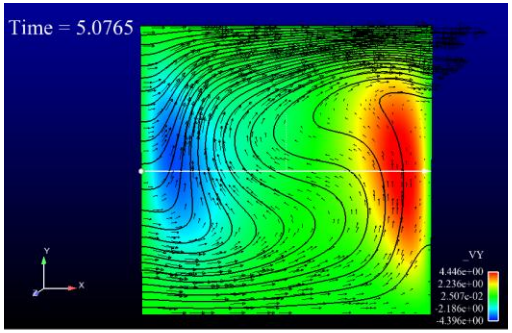
$$\rho = f(P, T, Y_i) \text{ and } \frac{\partial \rho}{\partial t} = 0 \text{ in continuity eq.}$$

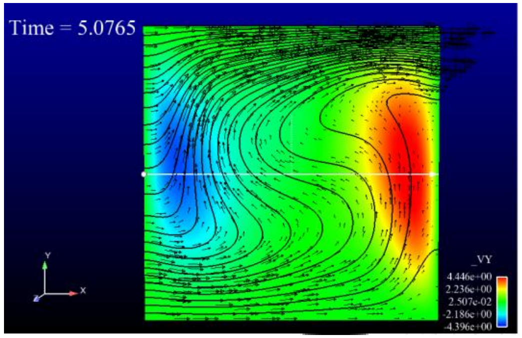
Compressible Fluid

$$\rho = f(P, T, Y_i) \Longrightarrow P = f(\rho, T, Y_i)$$

Density becomes a degree of freedom!







Challenges

- Many kinds of quantities required:
  - State and parameter derivatives
  - Various forms of second derivatives
  - polynomial chaos expansions
  - …

- Quickly integrate, adapt, and reuse models and equation sets while supporting requirements

- Incorporating these directly requires significant effort
  - **Combinatorial explosion of required sensitivities**
  - Time consuming, error prone
  - Gets in the way of physics/model development

- Requires code developers to understand requirements of algorithmic approaches
  - Limits embedded algorithm R&D on complex problems

A Solution

- Need a framework that
  - Allows simulation code developers to focus on complex physics development
  - Doesn’t make them worry about advanced analysis
  - Allows derivatives and other quantities to be easily and efficiently extracted
  - Is extensible to future embedded algorithm requirements

- **Directed Acyclic Graph based assembly**
  - Managers complexities with model dependencies
  - Maximize reuse of model code
  - Avoid complex switching during assembly

- **Template-based generic programming**
  - Code developers write physics code templated on scalar type
  - Operator overloading libraries provide tools to propagate needed embedded quantities (derivatives, stochastics, etc.)
  - Libraries connect these quantities to embedded solver/analysis tools

Lightweight DAG-based Expression Evaluation

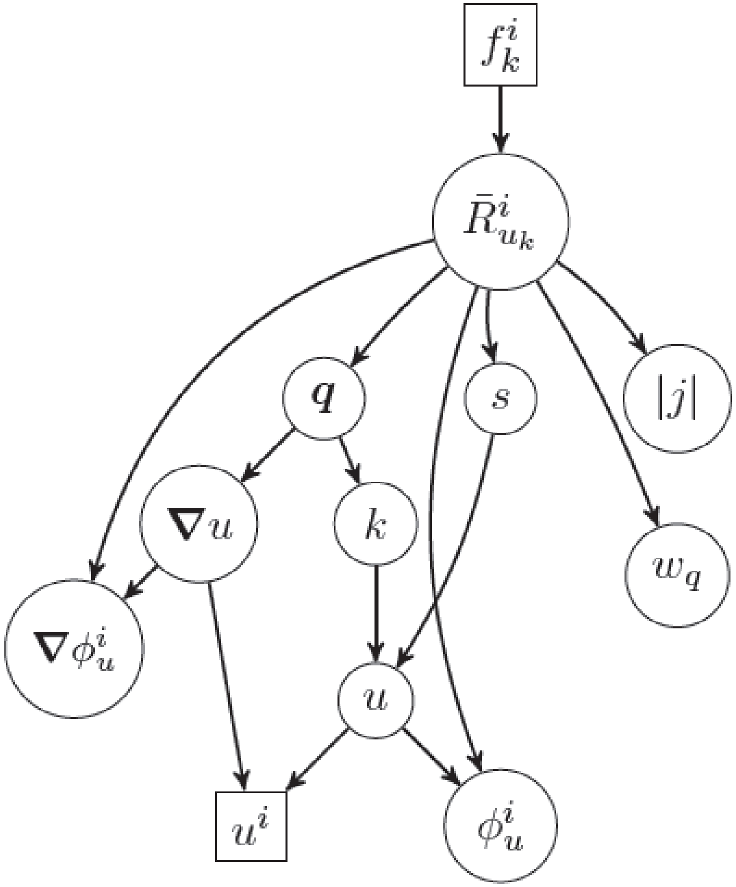
- Toolkit for handling complexity in Multiphysics

- Decompose a complex problem into a graph of simple tasks to support rapid development, separation of concerns and extensibility.

- Basic Requirements of a graph “node”:
  - Generic name (“density”, “viscosity”)
  - Declared prerequisites (“temperature”, “pressure”)
  - Evaluation: evaluate()
  - Signature definition (scalar, vector, tensor, …)

- Separation of data (Fields) and kernels (Expressions) that operate on the data

$$R_u^i = \int_\Omega \left[\phi_u^i \dot{u} - \boldsymbol{\nabla} \phi_u^i \cdot \boldsymbol{q} + \phi_u^i s \right] \, \mathrm{d}\Omega$$



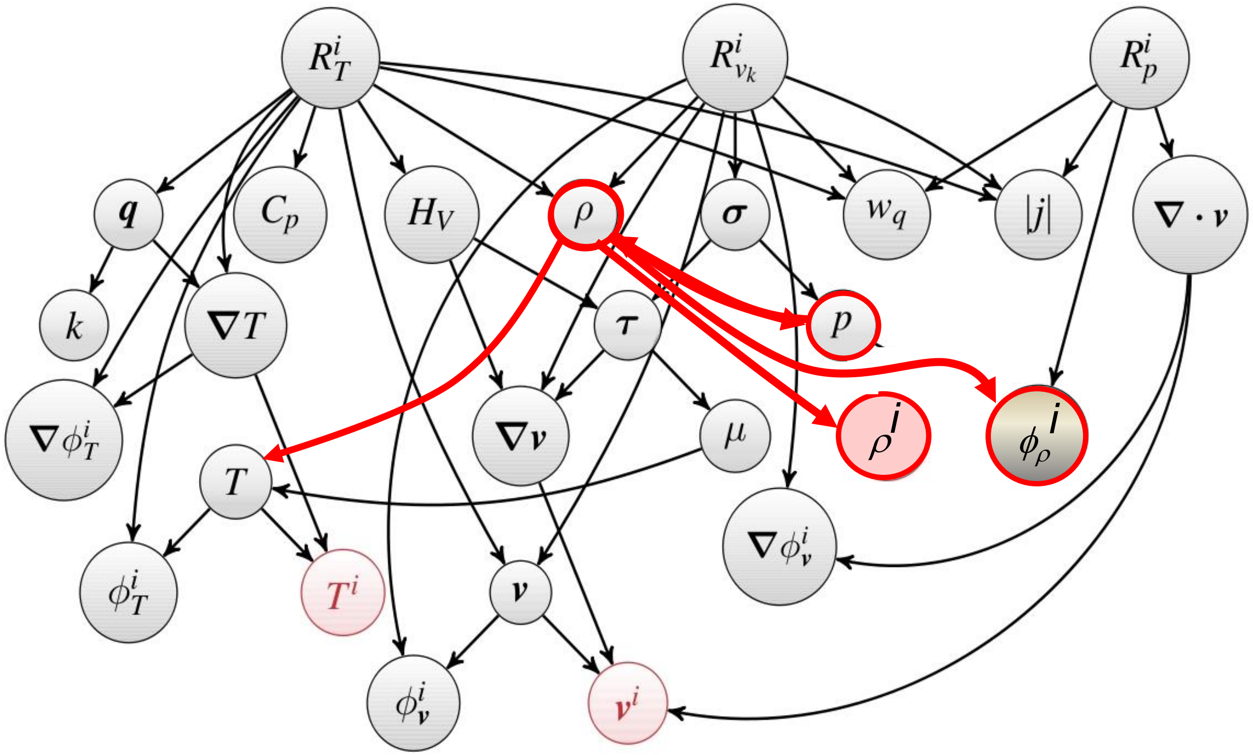
Navier-Stokes Example

- Graph-based equation description
  - Automated dependency tracking (Topological sort to order the evaluations)
  - Each node is a point of extension that can be swapped out
  - Easy to add equations
  - Easy to change models
  - Easy to test in isolation
  - User controlled granularity
  - No unique decomposition
- User controlled memory allocation of Field data
- Multi-core research:
  - Spatial decomposition
  - Algorithmic decomposition

$$R_T^i = \sum_{e=1}^{N_e}\sum_{q=1}^{N_q}\left[\left(\rho C_p \boldsymbol{v}\cdot\nabla T - H_V\right)\phi_T^i - \boldsymbol{q}\cdot\nabla\phi_T^i\right] w_q|j| = 0$$

$$R_{v_k}^i = \sum_{e=1}^{N_e}\sum_{q=1}^{N_q}\left[\rho\boldsymbol{v}\cdot\nabla\boldsymbol{v}\phi_{\boldsymbol{v}}^i + \boldsymbol{\sigma} : \nabla\left(\phi_{\boldsymbol{v}}^i \boldsymbol{e}_k\right)\right] w_q|j| = 0$$

$$R_p^i = \sum_{e=1}^{N_e}\sum_{q=1}^{N_q}\nabla\cdot\boldsymbol{v}\phi_p^i w_q|j| = 0$$



Template-based Generic Programming (TBGP)

- Template scalar type in the assembly process

- New Scalar types that ***overload the math operators***
    - Expression templates
    - Derivatives: FAD, RAD
    - Stochastic Galerkin: PCE

`double`	`Fad<double>`
Operation	Forward AD rule
$c = a \pm b$	$\dot{c} = \dot{a} \pm \dot{b}$
$c = ab$	$\dot{c} = a\dot{b} + \dot{a}b$
$c = a/b$	$\dot{c} = (\dot{a} - c\dot{b})/b$
$c = a^r$	$\dot{c} = ra^{r-1}\dot{a}$
$c = \sin(a)$	$\dot{c} = \cos(a)\dot{a}$
$c = \cos(a)$	$\dot{c} = -\sin(a)\dot{a}$
$c = \exp(a)$	$\dot{c} = c\dot{a}$
$c = \log(a)$	$\dot{c} = \dot{a}/a$

$$\dot{u} := \frac{du}{dz}$$

Fad: $\frac{df}{dx}(x_0)V$

$$V \in \mathbb{R}^{n \times p}$$

$$dx/dz = V$$

Seeding/initializing V

For J: $V = \mathbf{I}$

For Jw: $V = w$

TBGP Example

$$f_0 = 2x_0 + x_1^2$$
$$f_1 = x_0^3 + sin(x_1)$$

```
void computeF(double* x, double* f)
{
  f[0] = 2.0 * x[0] + x[1] * x[1];
  f[1] = x[0] * x[0] * x[0] + sin(x[1]);
}
```

```
template <typename ScalarT>
void computeF(ScalarT* x, ScalarT* f)
{
  f[0] = 2.0 * x[0] + x[1] * x[1];
  f[1] = x[0] * x[0] * x[0] + sin(x[1]);
}
```

```
void computeJ(double* x, double* J)
{
  // J(0,0)
  J[0] = 2.0;
  // J(0,1)
  J[1] = 2.0 * x[1];
  // J(1,0)
  J[2] = 3.0 * x[0] * x[0];
  // J(1,1)
  J[3] = cos(x[1]);
}
```

```
double* x;
double* f;
…
computeF(x,f);
```

```
DFad<double>* x;
DFad<double>* f;
…
computeF(x,f);
```

Same accuracy as writing analytic derivative:
No differencing error involved!

Generic Programming

(using data types from Trilinos/Sacado: E. Phipps)

Field Manager is templated on Evaluation Type

Concept: Evaluation Types		Scalar Types
Residual	$F(x,p)$	`double`
Jacobian	$J = \frac{\partial F}{\partial x}$	`DFad<double>`
Hessian	$\frac{\partial^2 F}{\partial x_i \partial x_j}$	`DFad< DFad<double> >`
Parameter Sensitivities	$\frac{\partial F}{\partial p}$	`DFad<double>`
Jv	Jv	`DFad<double>`
Stochastic Galerkin Residual		`Sacado::PCE::OrthogPoly<double>`
Stochastic Galerkin Jacobian		`Sacado::Fad::DFad< Sacado::PCE::OrthogPoly<double> >`

NOTES:

1. **Not tied to double (can do arbitrary precision)**
2. **Not tied to any one scalar type can use multiple scalar types in any evaluation type!**

TBGP in Multiphysics PDE Assembly

PDE Equation: $\dot{u} + \nabla \cdot \mathbf{q} + s = 0$ $\quad\quad$ $\mathbf{q} = -k\nabla u$

Galerkin Weak form ignoring boundary terms for simplicity:

$$R_u^i = \int_\Omega \left[\phi_u^i \dot{u} - \boldsymbol{\nabla} \phi_u^i \cdot \boldsymbol{q} + \phi_u^i s \right] \mathrm{d}\Omega$$

FEM Basis: $u = \sum_{i=1}^{N_u} \phi_u^i u^i$

Residual Equation:

$$\hat{R}_u^i = \sum_{e=1}^{N_E} \sum_{q=1}^{N_q} \left[\phi_u^i \dot{u} - \boldsymbol{\nabla} \phi_u^i \cdot \boldsymbol{q} + \phi_u^i s \right] w_q |j| = 0$$

TBGP + DAG: Global Evaluation

$$f(x) = \sum_{k=1}^{N_w} S_k^f \bar{R}^i_{u_k}(G_k^f x)$$

Gather/Seed

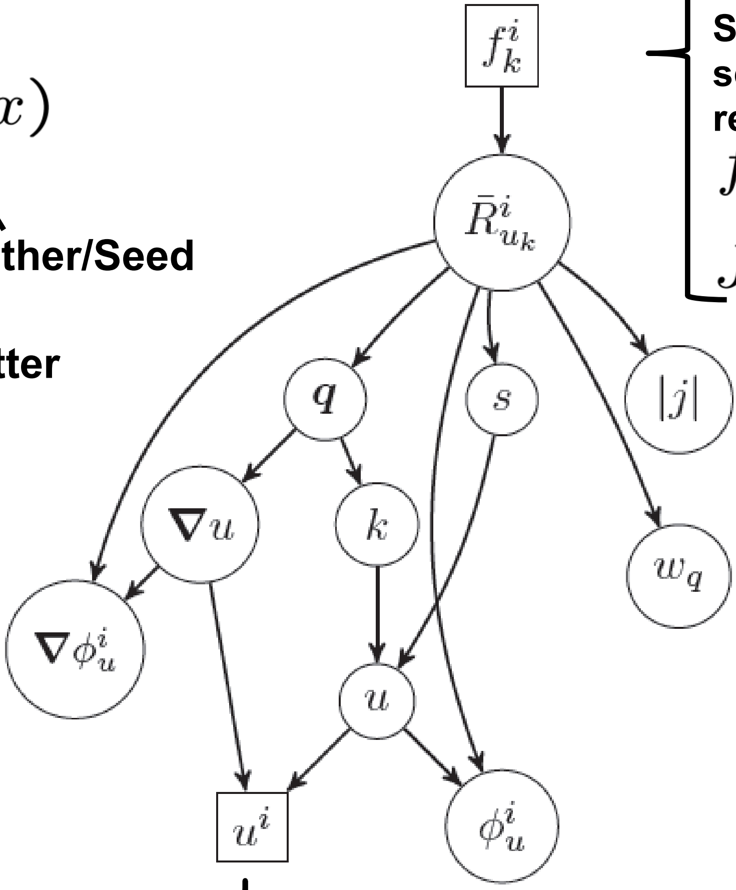
Extract/Scatter

Break mesh into worksets of elements

Extract values from Scalar type and scatter to global residual

$$f_k \leftarrow S_k^f \bar{R}^i_{u_k}$$

$$J_k \leftarrow S_k^J \bar{R}^i_{u_k}$$



$$u_k \leftarrow G_k^f x$$

$$u_k \leftarrow G_k^J x$$

Gather solution values and seed Scalar type

- Only have to specialize two expressions for evaluation type:
  - Gather/Seed
  - Extract/Scatter
- All other code is reused
  - Other code is the multiphysics equation sets
- Achieved separation of concerns!

Physics/Block Preconditioning

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u} + p\mathbb{I} + \mathbf{\Pi}) - \frac{1}{\mu_0}\nabla\times\mathbf{B}\times\mathbf{B} = 0$$

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0$$

$$\frac{\partial\mathbf{B}}{\partial t} - \nabla\times(\mathbf{u}\times\mathbf{B}) + \nabla\times(\frac{\eta}{\mu_0}\nabla\times\mathbf{B}) = 0$$

$$\Downarrow$$

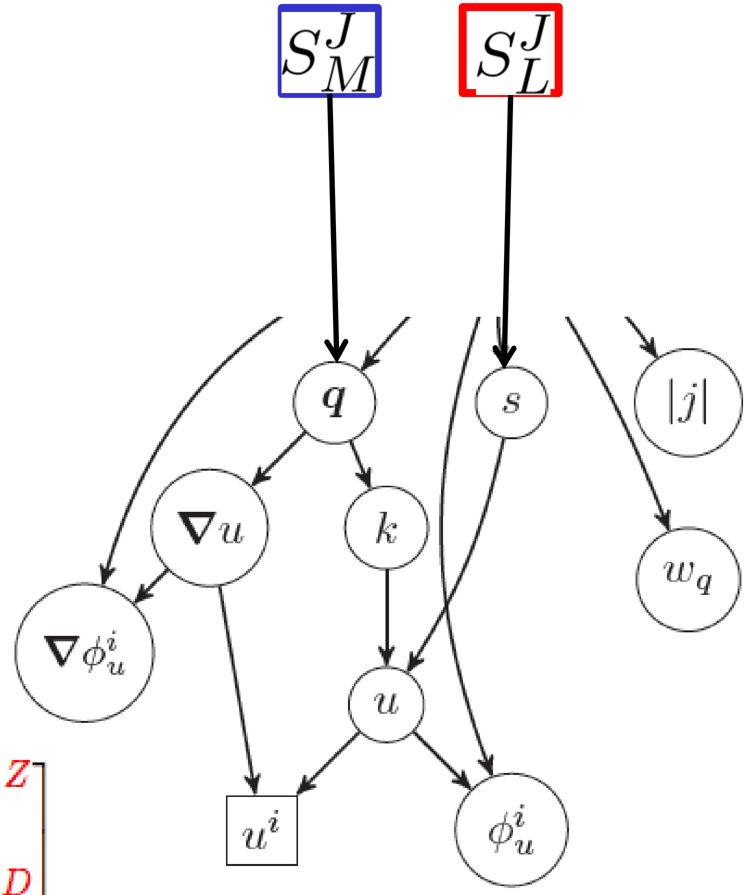
$$\begin{bmatrix} \begin{bmatrix} F & B^T \\ B & C \end{bmatrix} & \begin{bmatrix} Z \\ 0 \end{bmatrix} \\ \begin{bmatrix} Y & 0 \end{bmatrix} & D \end{bmatrix} \begin{bmatrix} u \\ p \\ b \end{bmatrix} = \begin{bmatrix} f \\ g \\ h \end{bmatrix}$$

$$\Downarrow$$

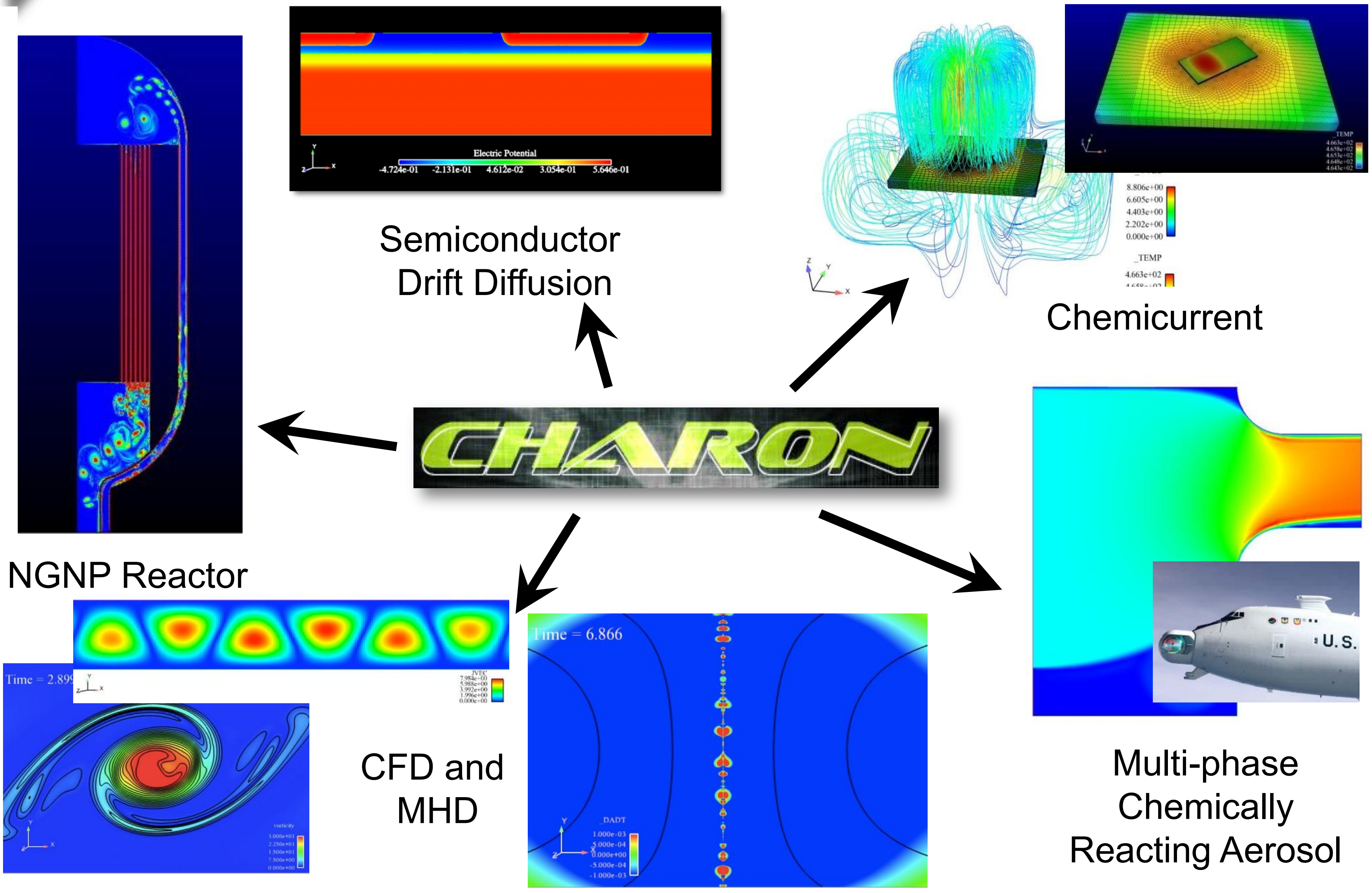
$$\begin{bmatrix} F & B^T & Z \\ B & C & 0 \\ Y & 0 & D \end{bmatrix}$$

$$\approx \begin{bmatrix} F & & Z \\ & I & \\ Y & & D \end{bmatrix} \begin{bmatrix} F^{-1} & & \\ & I & \\ & & I \end{bmatrix} \begin{bmatrix} F & B^T & \\ B & C & \\ & & I \end{bmatrix} = \begin{bmatrix} F & B^T & Z \\ B & C & \\ Y & \boxed{YF^{-1}B^T} & D \end{bmatrix}$$

JFNK + Block Scattering for Preconditiitoning



Rapid Development of New Physics
(Single driver and collection of interchangeable evaluators)



Semiconductor Drift Diffusion

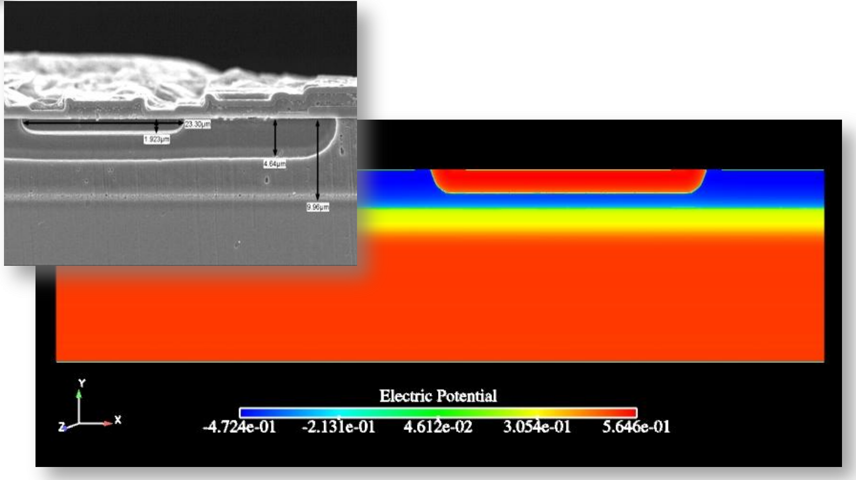
Chemicurrent

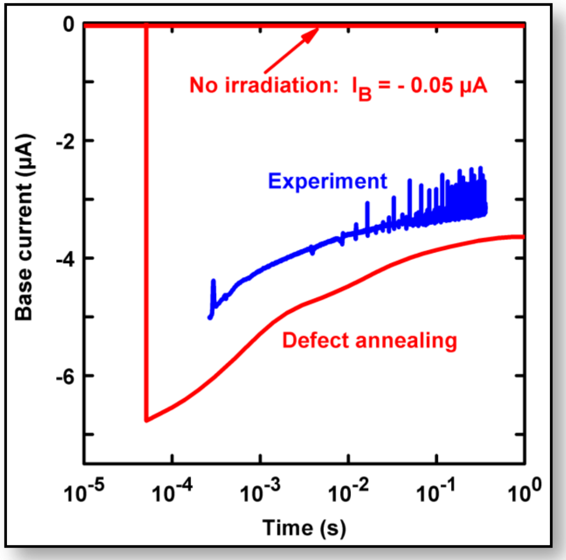
NGNP Reactor

CFD and MHD

Multi-phase Chemically Reacting Aerosol

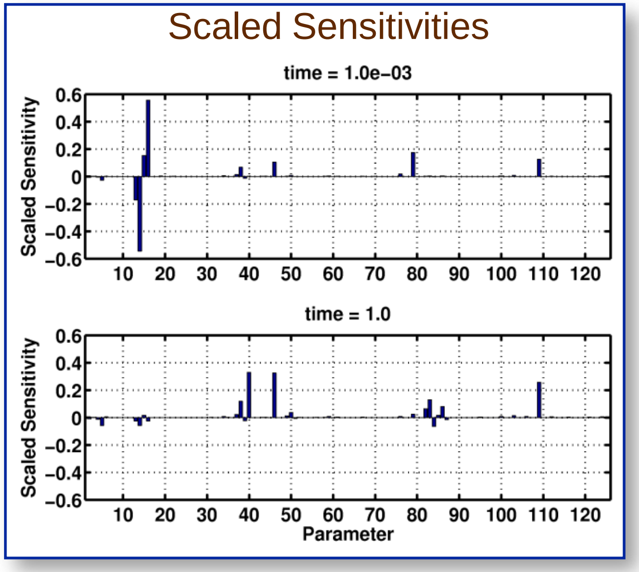
Transient Sensitivities of Radiation Damage in Semiconductor Devices





Comparison to FD:

- ✓ **Sensitivities at all time points**
- ✓ **More accurate**
- ✓ **More robust**
- ✓ **14x faster!**



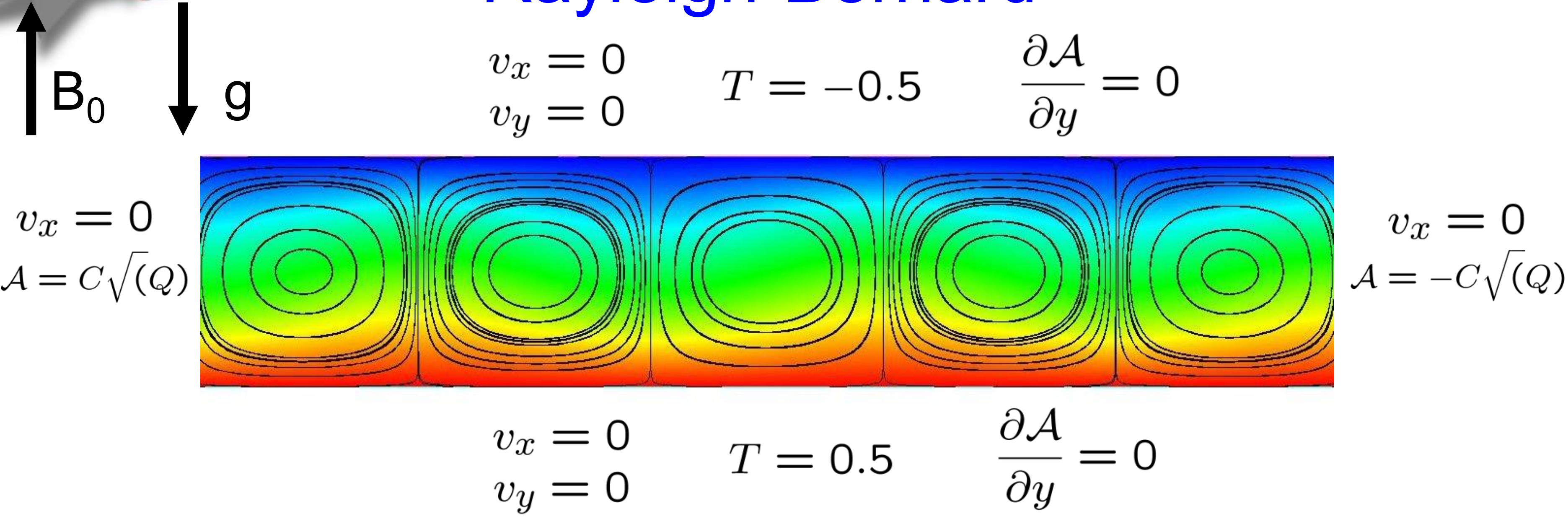
CHARON

QASPR

QUALIFICATION ALTERNATIVES TO SPR

Sandia National Laboratories

Hydromagnetic Rayleigh-Bernard



Top: $v_x = 0$, $v_y = 0$ $\quad T = -0.5 \quad \dfrac{\partial \mathcal{A}}{\partial y} = 0$

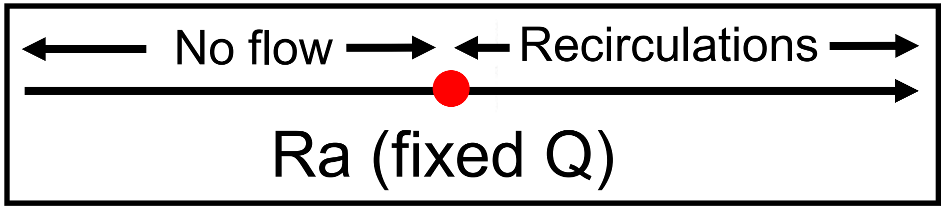
Left: $v_x = 0$, $\mathcal{A} = C\sqrt(Q)$

Right: $v_x = 0$, $\mathcal{A} = -C\sqrt(Q)$

Bottom: $v_x = 0$, $v_y = 0$ $\quad T = 0.5 \quad \dfrac{\partial \mathcal{A}}{\partial y} = 0$

Parameters:

- Q ~ B_0^2 (Chandresekhar number)
- Ra (Rayleigh number)



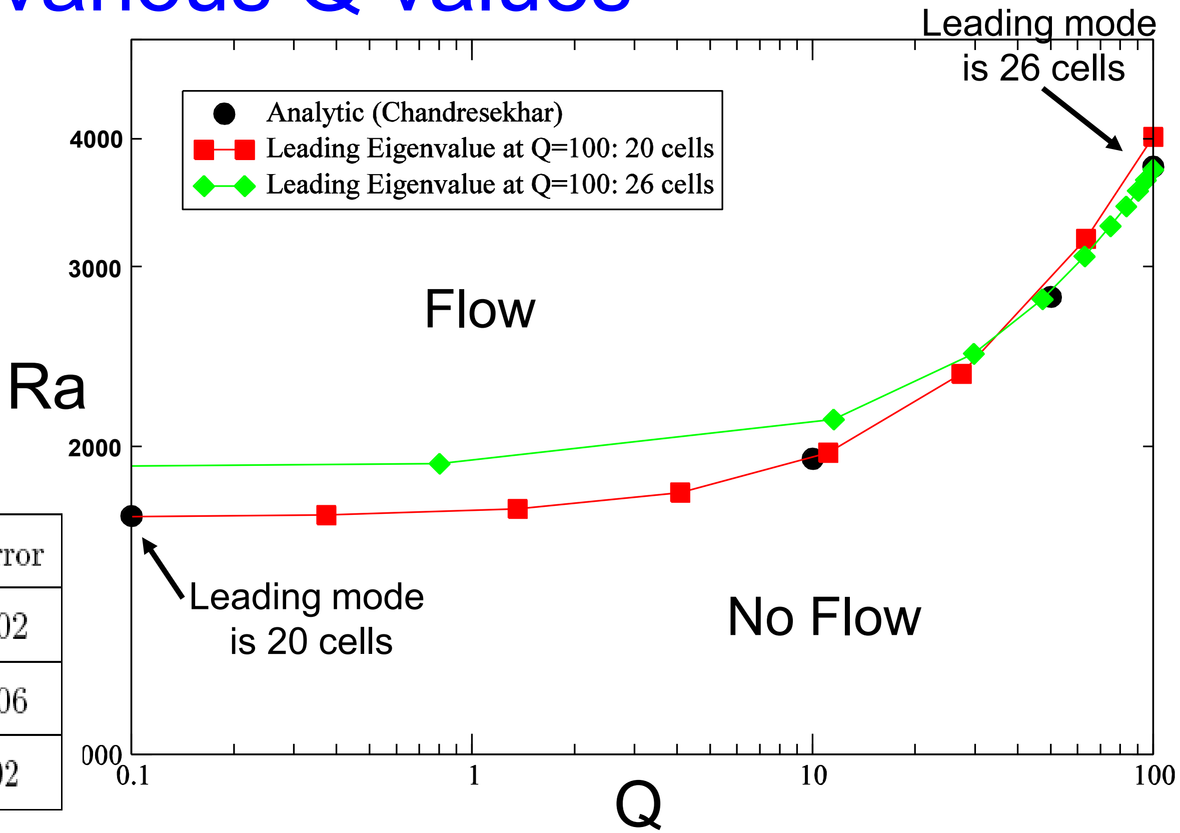
$$Q = \frac{B_0^2 d^2}{\mu_0 \rho \nu \eta} \qquad Ra = \frac{g\beta \Delta T d^3}{\nu\alpha} \qquad Pr = \frac{\nu}{\alpha} \qquad Prm = \frac{\nu}{\eta}$$

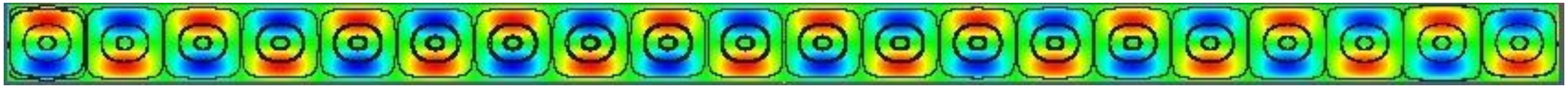
- Buoyancy driven instability initiates flow at high Ra numbers.
- Increased values of Q delay the onset of flow.
- Domain: 1x20

Leading Mode is different for various Q values

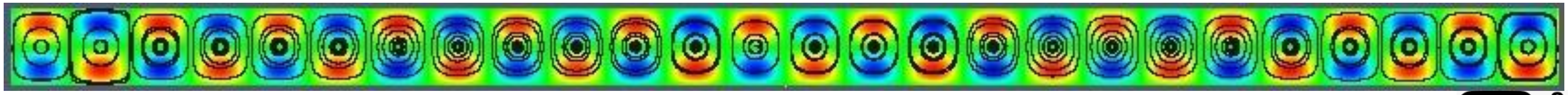
- Analytic solution is on an infinite domain with two bounding surfaces (top and bottom)
- Multiple modes exist, mostly differentiated by number of cells/wavelength.
- Therefore tracking the same eigenmode does not give the stability curve!!!
- Periodic BCs will not fix this

Q	Ra*	Ra_{cr} [Chandrasekhar[]]	% error
0	1707.77	1707.8	0.002
10^1	1945.78	1945.9	0.006
10^2	3756.68	3757.4	0.02





Mode: 20 Cells: Q=100, Ra=4017



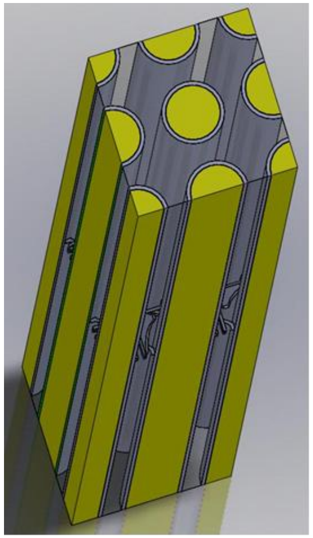
Mode: 26 Cells: Q=100, Ra=3757

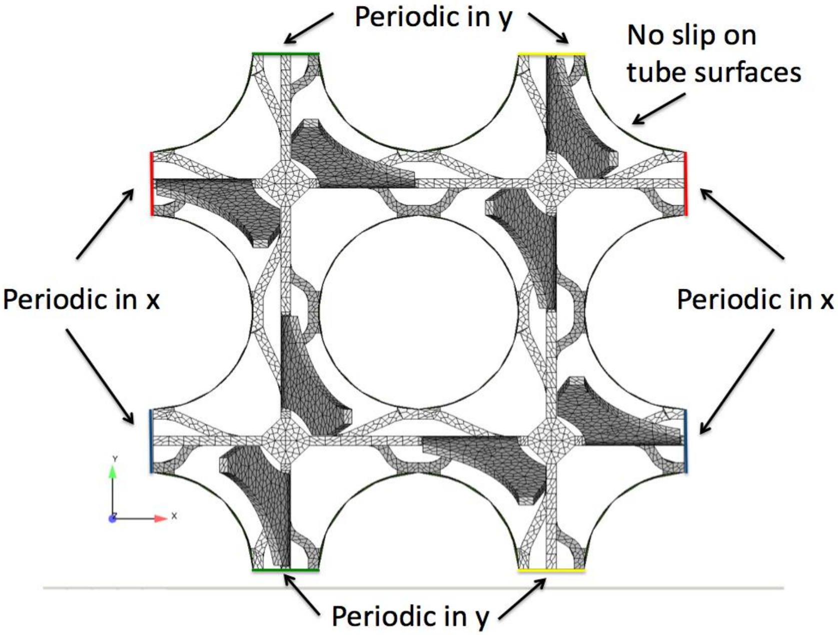
Sandia National Laboratories

Problem Description

- 3x3 Rod bundle
- Isothermal
- Fluid: Water
  - T: 394K
  - Viscosity: 2.32×10^{-4} Pa sec
  - Density: 924 kg/m^3
- Symmetry on sides
- No slip (v=0) on rods
- Inflow on bottom
  - 5 m/sec
- Outflow on top:

$$\mathbf{T}\cdot\mathbf{n}=0$$





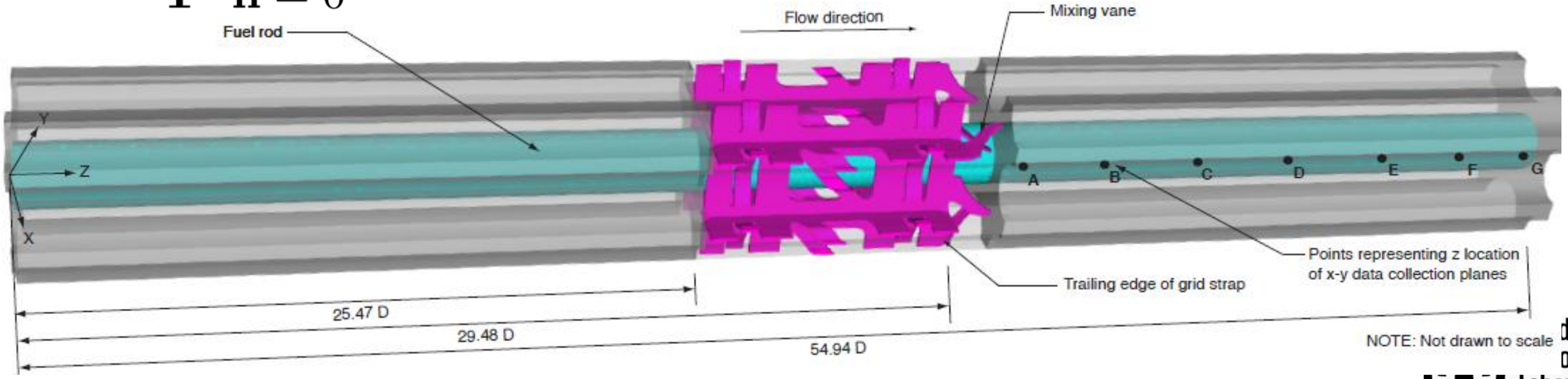
Periodic in y

No slip on tube surfaces

Periodic in x

Periodic in x

Periodic in y



Fuel rod

Flow direction

Mixing vane

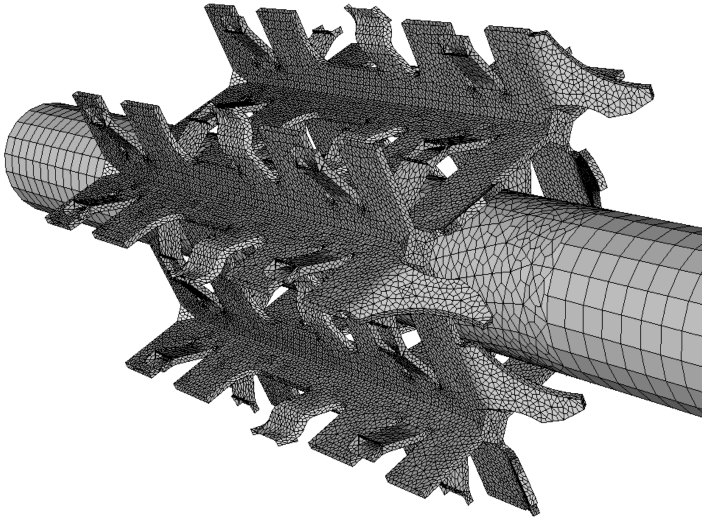
Points representing z location of x-y data collection planes

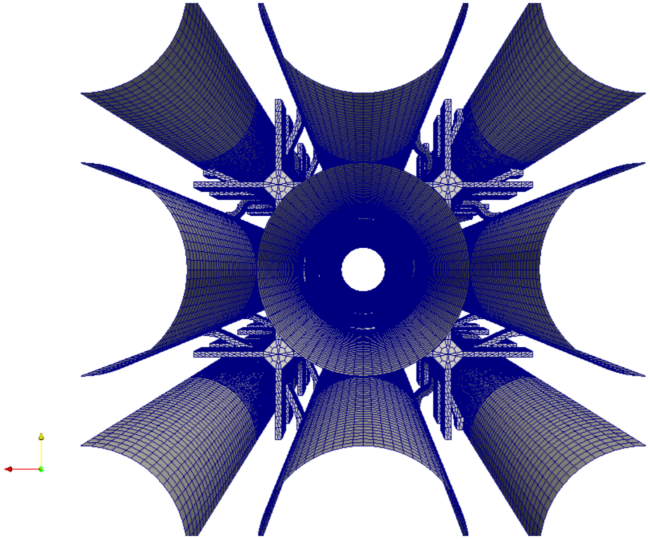
Trailing edge of grid strap

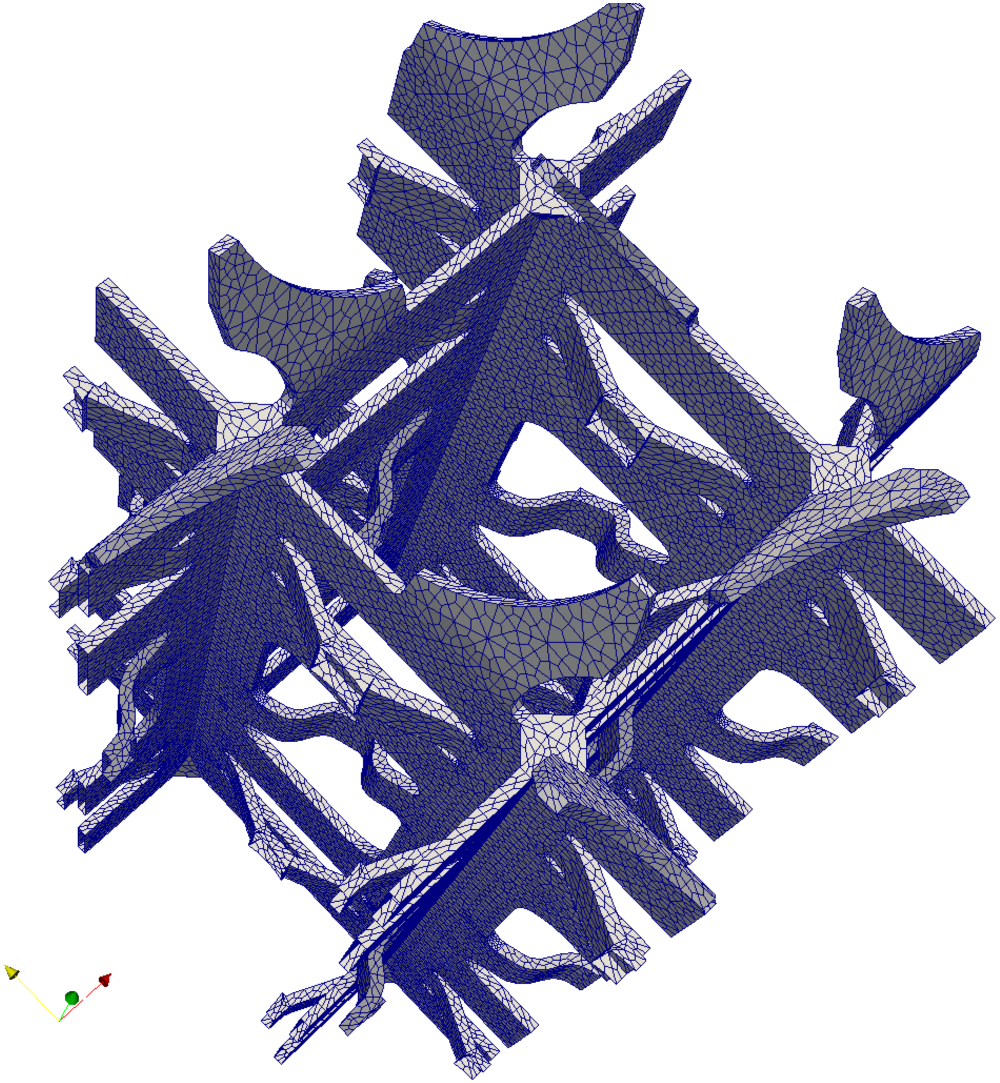
NOTE: Not drawn to scale

Geometry

(Complex geometry including mixing vanes)

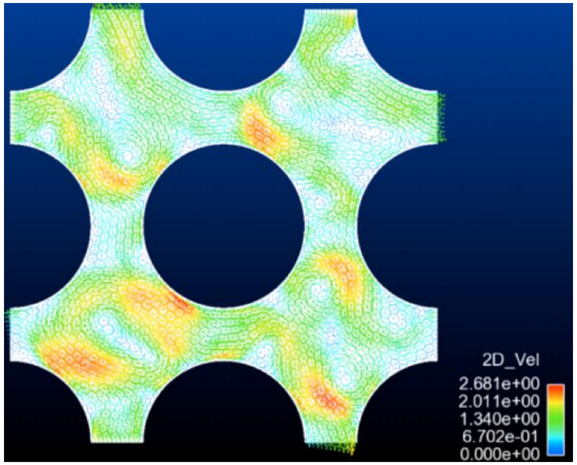


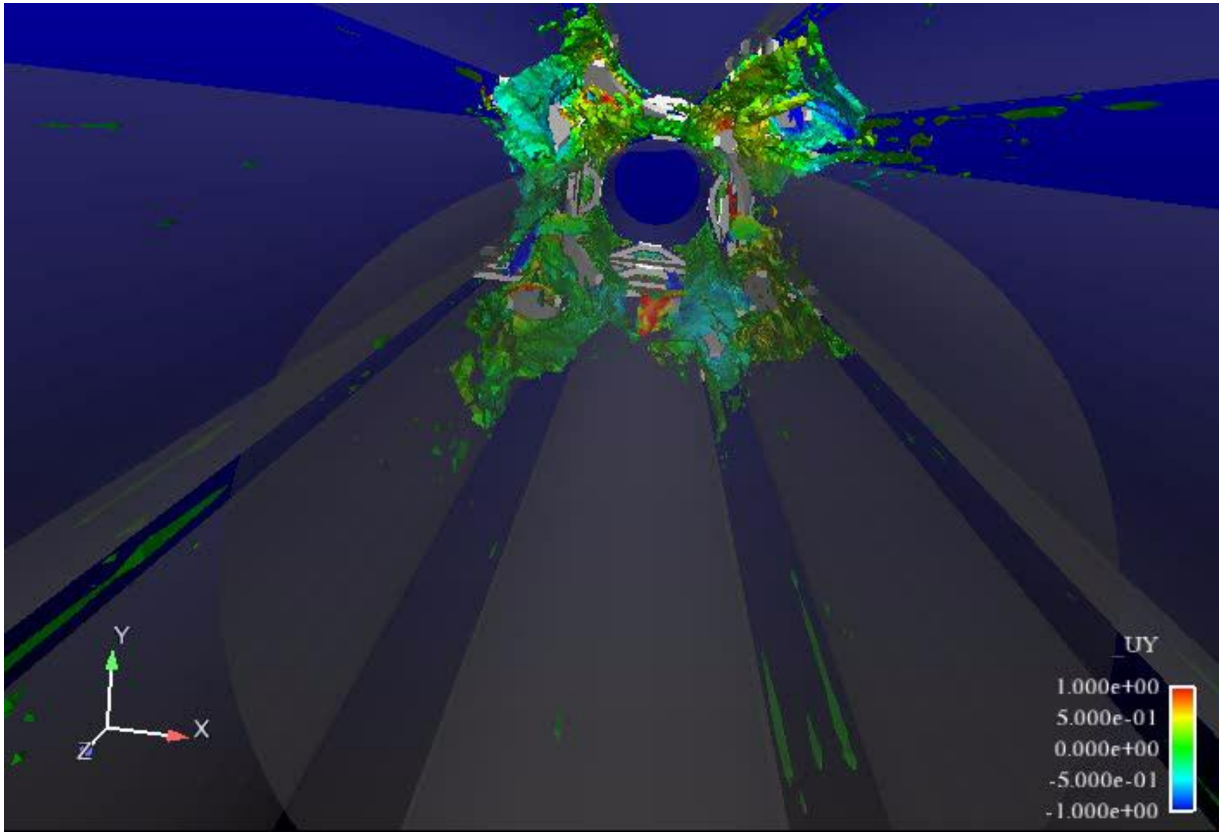


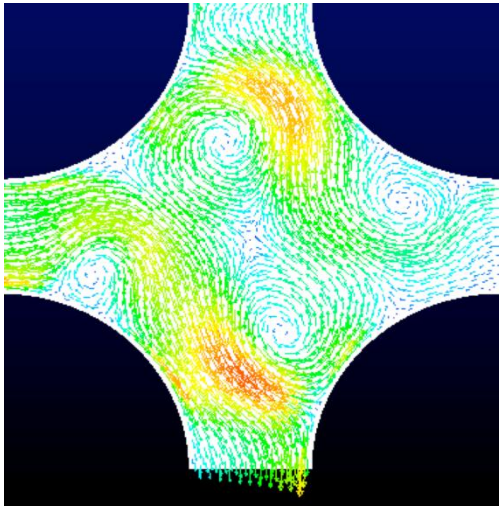


Solution Profiles

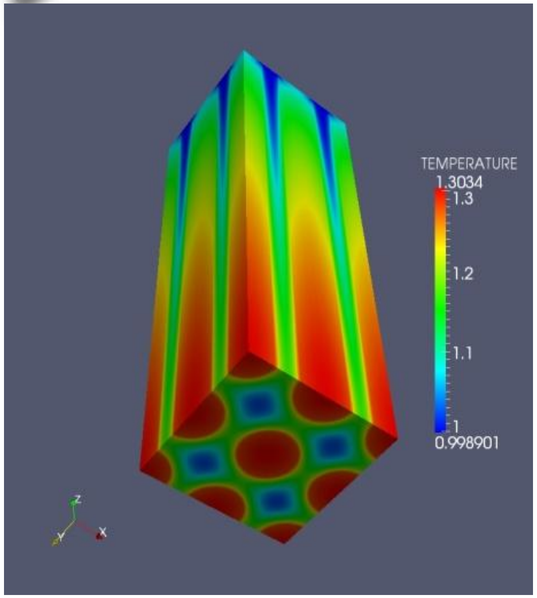
- Sandia Redsky platform, 256-1000 processes
- Oak Ridge Jaguar platform, 1200-9600 processes
- 2nd order BFD time integration
- Linear Lagrange elements (2nd order in space)

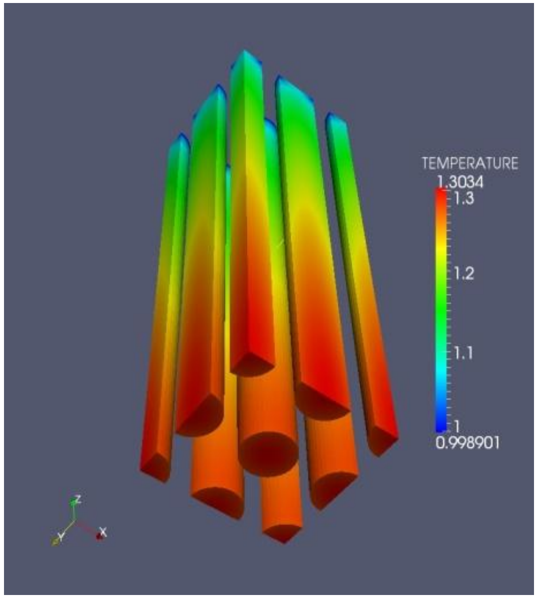


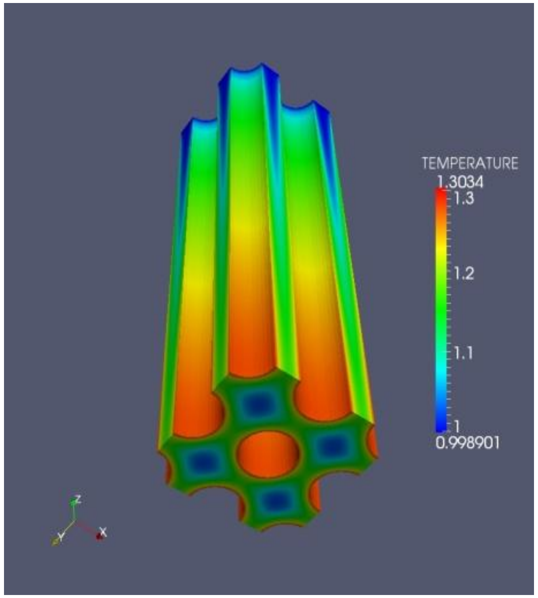


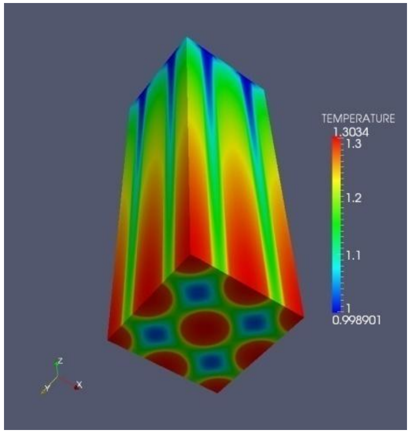


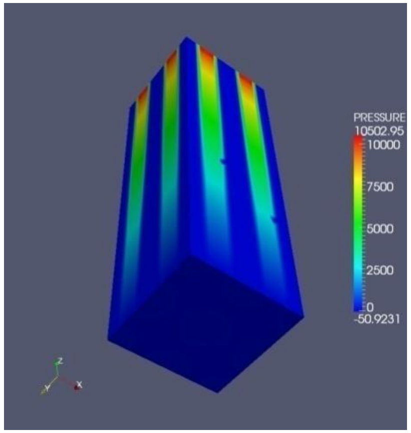
Conjugate Heat Transfer

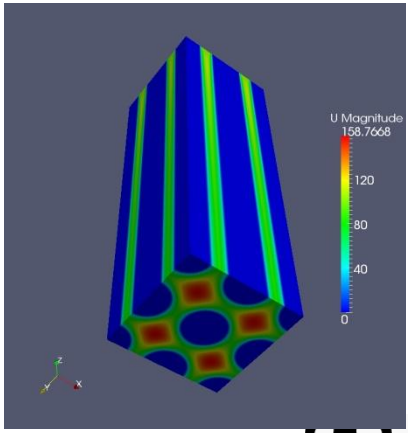






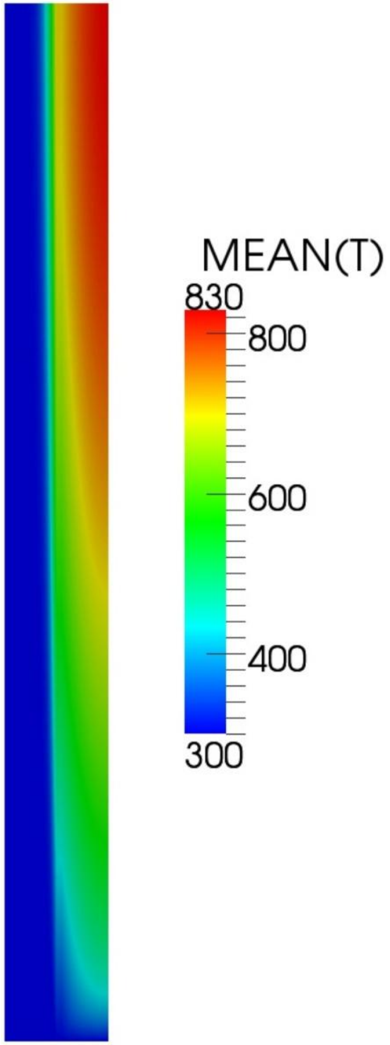


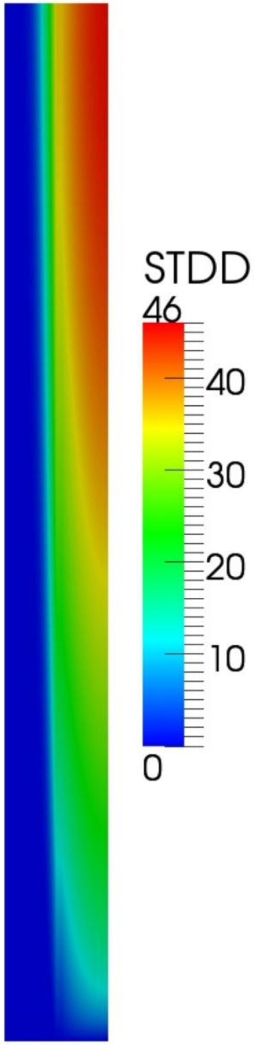


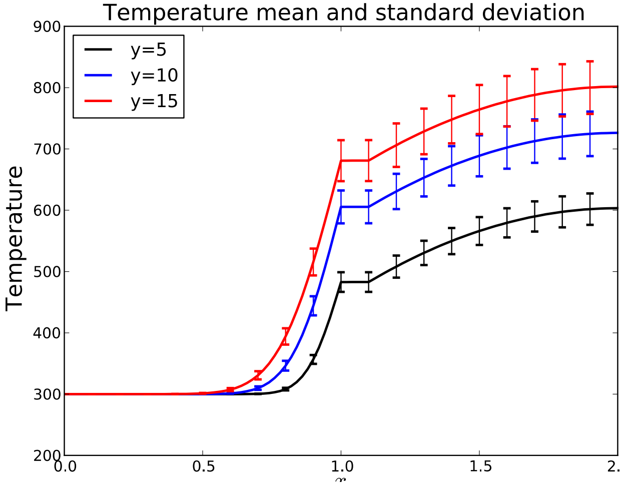


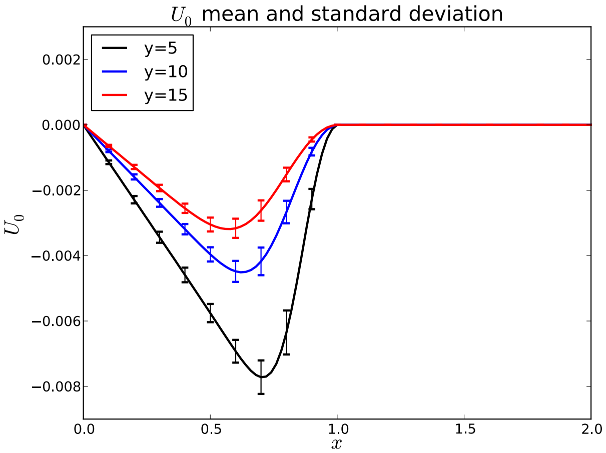
Embedded UQ in Drekar:

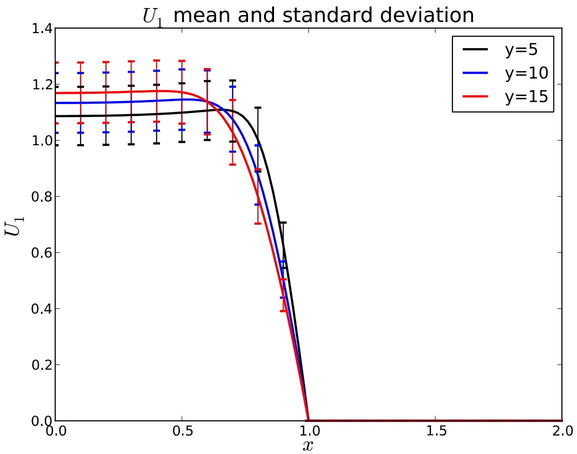
Rod to Fluid Heat Transfer











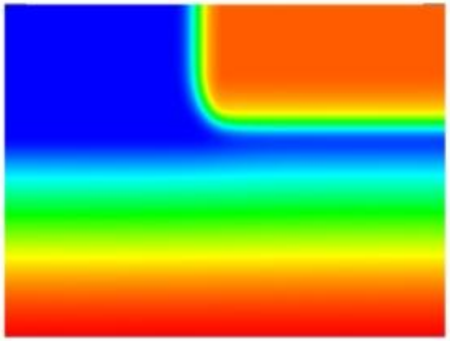
Sandia National Laboratories

Large-Scale Semiconductor Device Simulations on IBM Blue Gene Platform

- **Generic programming (via AD tools) is applied at the element level, not globally.**
- **Weak scaling to 65k cores and two billion DOF: Jacobian evaluation via AD scales!**
- **Using all four cores per node with MPI process on each core.**

Largest run to date: Solving linear systems of 2 billion unknowns on 147,000 cores

cores	DOF	Jacobian time
256	7.93m	52.19
1024	31.5m	52.28
4096	126m	52.09
8192	253m	52.82
16384	504m	52.74
32768	1.01b	52.96
65536	2.01b	52.94



Conclusions

DAG + TBGP =

- We can write very advanced multiphysics software that is efficient, flexible and maintainabile but templates are crucial
- Decoupling algorithms from equations is powerful:
  - We don’t write Jacobians anymore - enormous savings of manpower!
- Generic programming allows:
  - Segregation of technologies
  - Easily adaptive environment (from SE standpoint)
- Machine precision accuracy of required quantities is achieved

Trilinos Tools for PDEs Supporting TBGP

- Intrepid: Tools for discretizations of PDEs
  - Basis functions, quadrature rules, …
  - All Intrepid classes/functions templated on scalar type
    - Derivatives w.r.t. DOFs
    - Derivatives w.r.t. coordinates
- Phalanx: Local field evaluation kernels
  - DAG for multiphysics complexity
  - Explicitly manages fields/evaluators for different scalar types
- Shards
  - Templated multi-dimensional array
- Stokhos
  - PCE classes, overloaded operators
  - Simultaneous ensemble propagation classes, overloaded operators
  - Tools and data structures for forming, solving embedded SG systems
- Sacado
  - Parameter library – tools to manage model parameters
  - Template manager – tools to manage instantiations of a template class on multiple scalar types
  - MPL – simple implementation of some metaprogramming constructs

Thank you!

Domain Model for Multiphysics

<u>A Theory Manual for Multiphysics Code Coupling in LIME</u>, R. Pawlowski, R. Bartlett, R. Schmidt, R. Hooper, and N. Belcourt, SAND2011-2195

$$f(\dot{x}, x, \{p_l\}, t) = 0$$

Residual

$$\dot{x} = \frac{\partial x}{\partial t}$$

State (DOF)

Set of parameters

Time

$x \in \mathbb{R}^{n_x}$ is the vector of state variables (unknowns being solved for),
$\dot{x} = \partial x / \partial t \in \mathbb{R}^{n_x}$ is the vector of derivatives of the state variables with respect to time,
$\{p_l\} = \{p_0, p_1, \ldots, p_{N_p - 1}\}$ is the set of N_p independent parameter sub-vectors,
$t \in [t_0, t_f] \in \mathbb{R}^1$ is the time ranging from initial time t_0 to final time t_f,

$$f(\dot{x}, x, \{p_l\}, t) : \mathbb{R}^{\left(2n_x + \left(\sum_{l=0}^{N_p-1} n_{p_l}\right) + 1\right)} \rightarrow \mathbb{R}^{n_x}$$

$$g_j(\dot{x}, x, \{p_l\}, t) = 0, \text{ for } j = 0, \ldots, N_g - 1$$

Response Function

$g_j(\dot{x}, x, \{p_l\}, t) : \mathbb{R}^{\left(2n_x + \left(\sum_{l=0}^{N_p-1} n_{p_l}\right) + 1\right)} \rightarrow \mathbb{R}^{n_{g_j}}$ is the j^{th} response function.

Sandia National Laboratories

Jacobian-Free Newton-Krylov (JFNK)

$$x_{k+1} = x_k + \alpha \Delta x$$

$$J_k \Delta x = -F_k$$

<u>Iterative Linear Solver – GMRES</u>

Krylov Subspace of the form:

$$\mathcal{K}(A, v) \equiv \text{span}\{v, Av, A^2 v, ..., A^{m-1} v\}$$

In the inner iteration of the linear solve, we only need the action of the Jacobian on a vector:

$$Jv \approx \frac{F(x + \delta v) - F(x)}{\delta}$$

Only require an explicit matrix for preconditioning – does NOT have to be exact!

Advantages:

- **<u>Same as Newton, but no Jacobian is required</u>!**
- **Residual Based!**

Disadvantage:

- **Accuracy/convergence issues due to scalar perturbation factor:**

$$\delta = \lambda \left(\lambda + \frac{||\mathbf{x}||}{||\mathbf{v}||} \right)$$

- **Solution vector scaling is critical**

Example: JFNK
(2D Diffusion/Rxn System: 2 eqns)

- **JFNK (FD)**

$$Jv \approx \frac{F(x+\delta v)-F(x)}{\delta}$$

$$t \approx (\text{num\_Its}) * cost(F)$$

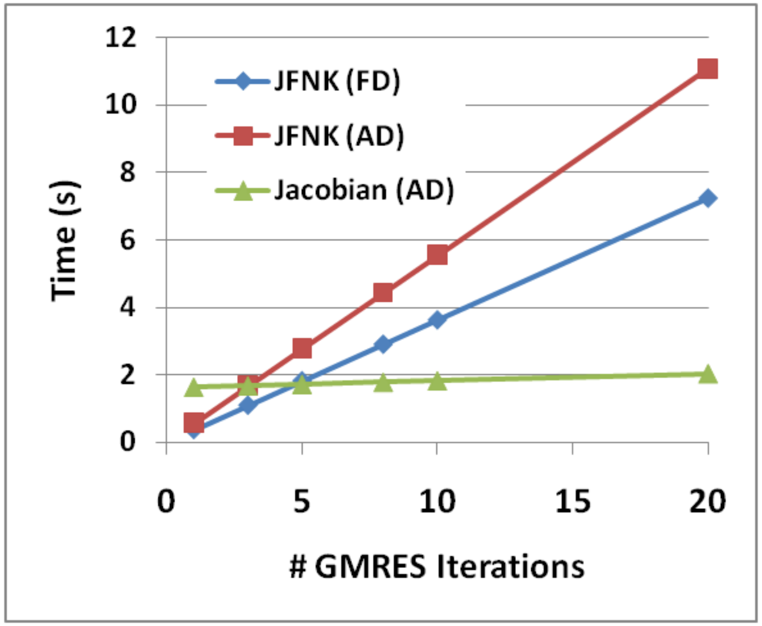
- **JFNK (AD)**
  - **Machine precision accurate**
  - **Ex: Solution varies 10^12 over domain**

$$Jv \leq 2.5 * cost(F)$$
$$t \approx 1.53 * (num\_Its) * cost(F)$$

- **Explicit Jacobian (AD generated)**
  - **Machine precision accurate**
  - **Complexity ideas allow for storing individual operators for preconditioning!**
  - **Larger memory requirements**

$$J(x) \leq 13 * cost(F)$$
$$t \approx 4.45 + (num\_Its) * cost(Mv)$$

	Relative times	
JFNK (AD)	**F(x)**	**1.00**
Explicit J (AD)	**J(x)**	**4.45**
JFNK (AD)	**Jv (AD)**	**1.53**
Explicit J (AD)	**Mv (matvec)**	**0.06**



Example: Modify for Turbulence

$$R_{\mathbf{u}} = \frac{\partial(\bar{\rho}\bar{\mathbf{v}})}{\partial t} + \nabla\cdot(\bar{\rho}\bar{\mathbf{v}}\otimes\bar{\mathbf{v}} + \mathbf{T}) = 0$$

$$R_p = \frac{\partial\bar{\rho}}{\partial t} + \bar{\rho}\nabla\cdot\bar{\mathbf{v}} = 0$$

over bar denotes spatial filtering in LES

DOF

$\bar{\mathbf{v}}$

$\bar{P}$

$$\mathbf{T} = P\mathbf{I} + \frac{2}{3}\mu(\nabla\cdot\mathbf{v})\mathbf{I} - \boxed{\mu_{\text{eff}}}(\nabla\mathbf{v} + \nabla\mathbf{v}^T)$$

$$\boxed{\begin{aligned}
&\mu_{eff} = \mu + \mu_t \qquad \nu_t = (C_w\Delta)^2\frac{\left(\bar{\mathbf{S}}^d_{ij}\bar{\mathbf{S}}^d_{ij}\right)^{3/2}}{\left(\bar{\mathbf{S}}_{ij}\bar{\mathbf{S}}_{ij}\right)^{5/2} + \left(\bar{\mathbf{S}}^d_{ij}\bar{\mathbf{S}}^d_{ij}\right)^{5/4}}\\
&\mu_t = \bar{\rho}\nu_t\\
&\bar{\mathbf{S}}_{ij} = \frac{1}{2}\left(\frac{\partial\bar{\mathbf{v}}_i}{\partial\mathbf{x}_j} + \frac{\partial\bar{\mathbf{v}}_j}{\partial\mathbf{x}_i}\right) \qquad \bar{\Omega}_{ij} = \frac{1}{2}\left(\frac{\partial\bar{\mathbf{v}}_i}{\partial\mathbf{x}_j} - \frac{\partial\bar{\mathbf{v}}_j}{\partial\mathbf{x}_i}\right)\\
&\bar{\mathbf{S}}^d_{ij} = \bar{\mathbf{S}}_{ik}\bar{\mathbf{S}}_{kj} + \bar{\Omega}_{ik}\bar{\Omega}_{kj} - \frac{1}{3}\delta_{ij}\left[\bar{\mathbf{S}}_{mn}\bar{\mathbf{S}}_{mn} + \bar{\Omega}_{mn}\bar{\Omega}_{mn}\right]
\end{aligned}}$$

LES WALE Model

Complexity in Multiphysics Simulation Environments

Physics Model Complexity

- Solving multiphysics PDE systems generates complexity:
  - Complex interdependent coupled physics
  - Multiple proposed mathematical models
  - Different numerical formulations (e.g. space-time discretizations)
  - Exploring complex solution spaces (steady-state, transient, stability, bifurcation, design optimization, uncertainty quantification)
- Supporting multiplicity in models and solution techniques often leads to complex code with **complicated logic** and **fragile software designs**

Solution Algorithm Complexity

- From implicit forward solves to analyzing complex solution spaces:
  - Simultaneous analysis and design adds requirements
  - Do not burden analysts/physics experts with analysis algorithm requirements: i.e. programming sensitivities for implicit solvers, optimization, stability, bifurcation analysis and UQ

Engine must be flexible, extensible, maintainable and EFFICIENT!

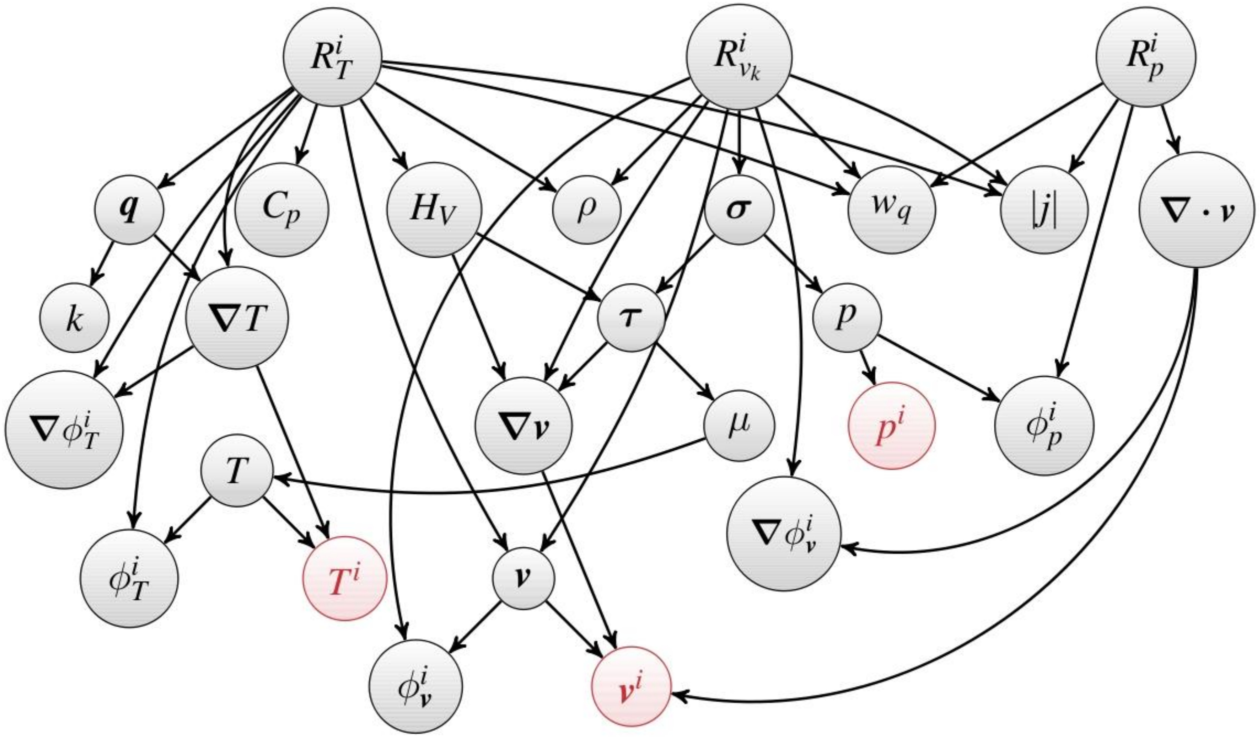
Sandia National Laboratories

Multi-level Parallelism

- Early evidence suggests mpi+threads

- **Option 1: Each core has full graph**
  Each core/thread evaluates the full graph over a subset of cells/elements

- **Option 2: Single graph with multiple threads (Task Scheduling)**
  - Use a priority queue to start critical/expensive nodes first
  - Better cache utilization


R^i_T
$R^i_{v_k}$
R^i_p
q
C_p
H_V
ρ
σ
w_q
$|j|$
$\nabla \cdot v$
k
∇T
τ
p
$\nabla \phi^i_T$
∇v
μ
p^i
ϕ^i_p
T
$\nabla \phi^i_v$
ϕ^i_T
T^i
v
ϕ^i_v
v^i