Non-intrusive Uncertainty Quantification Using Random Fields in Parallel Finite Element Codes

SAND2010-1225C

Brian Carnes
John Red-Horse
bcarnes@sandia.gov

Sandia National Laboratories
Albuquerque, NM

Feb 24-26, 2010
SIAM Conference on Parallel Processing
for Scientific Computing

Sandia National Laboratories

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

National Nuclear Security Administration

Outline

- Non-intrusive Uncertainty Quantification
- KL Representations of Random Fields
- A Parallel KL Solver
- Application: Porous Flow & Geomechanics
- Future Work

Non-intrusive Uncertainty Quantification

- Non-intrusive UQ performs many simulations at different realizations of random inputs
- Useful for generating efficient surrogate models
- Inputs can be scalars or distributed fields (space, time)
- Code modification: support heterogeneous field data
- Exploits parallelism at two levels:
  - Parallel sampling for many UQ methods: MC, LHS, stochastic collocation, PCE
  - Application parallelism
- Random field discretization must also be parallel:
  - Compute realizations (easy)
  - Generate discretizations, e.g. KL eigenproblem (harder)

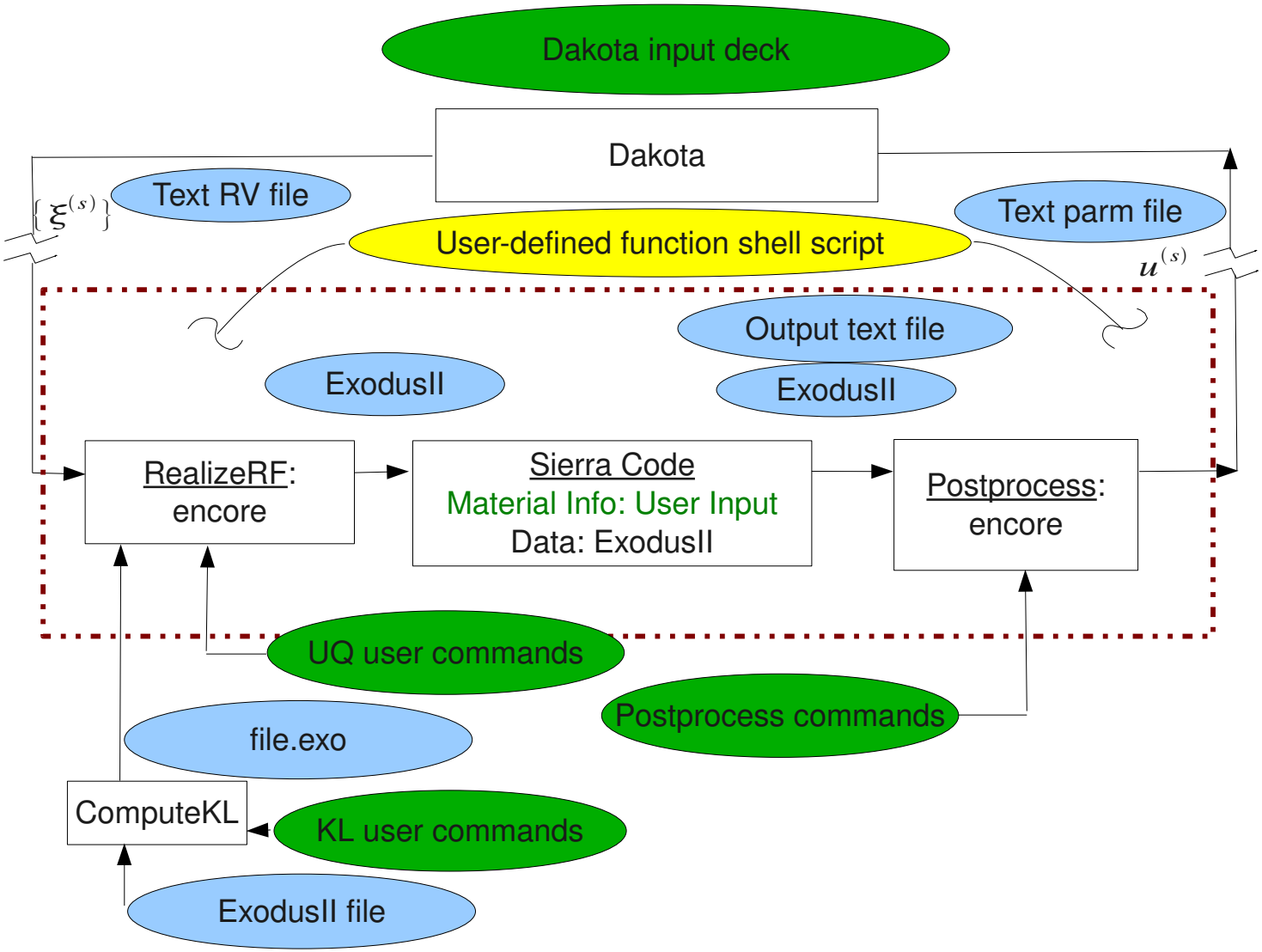
Sources of Uncertainty: RVs and RFs

- Random variables (RVs)
  - Based on probability space $(\Omega, \mathcal{S}, P)$
  - RV is (measurable) function from Ω into $\mathbb{R}$
  - Often used in simulations for scalar simulation model inputs
  - Scalar model outputs then become RVs
- Random fields (RFs)
  - Distributed in space and/or time as

$$a(x, \omega), \quad x \in D, \quad \omega \in \Omega$$

  - At each point x the value $a(x, \cdot)$ is an RV
  - Discretization is needed to approximate an infinite number of RVs
  - Goal: make the RF case look like the case of a small vector of RVs
  - This applies to both intrusive and non-intrusive UQ

Non-intrusive UQ Framework (Sandia)



KL Representation of RFs

- We can view RFs as mean plus fluctuation

$$a(x,\omega) = m(x) + \alpha(x,\omega)$$

- Karhunen-Loeve (KL) series is a generalization of SVD to RFs
- KL series is computed from an orthogonal basis of eigenvectors $\{\phi_j\}$ in $L^2(D)$ (scaled by eigenvalues λ_j)

$$\alpha(x,\omega) = \sum_{j=1}^{\infty} \sqrt{\lambda_j}\, \eta_j(\omega)\, \phi_j(x)$$

- The coefficients $\{\eta_j\}$ are zero-mean, uncorrelated RVs.
- Computed from experimental data by projection:

$$\eta_j(\omega) \equiv \frac{1}{\sqrt{\lambda_j}} \int_D \alpha(x,\omega)\, \phi_j(x)\, dx$$

RF Discretization

- Approximations are obtained by truncating the series
- The number of RV coefficients (stochastic dimension) is truncated
- We have gone from infinite to finite number of RV coordinates
- Deterministic component $\phi_j(x)$ approximated using FE basis

$$\phi_j(x) \approx \sum_i \Phi_j^i v_i(x) \in V_h \subset L^2(D)$$

- Generating a parallel KL realization requires
  - Distributed data: eigenvector coefficients $\{\Phi_j^i\}$
  - Globally shared data: small vectors of correlated random variables $\{\eta_j\}$, eigenvalues $\{\lambda_j\}$
- Assembly of realizations requires no parallel communication

KL Realizations

- We plot realizations using different number of terms
- The RV coefficients η_j are assumed i.i.d. standard normal

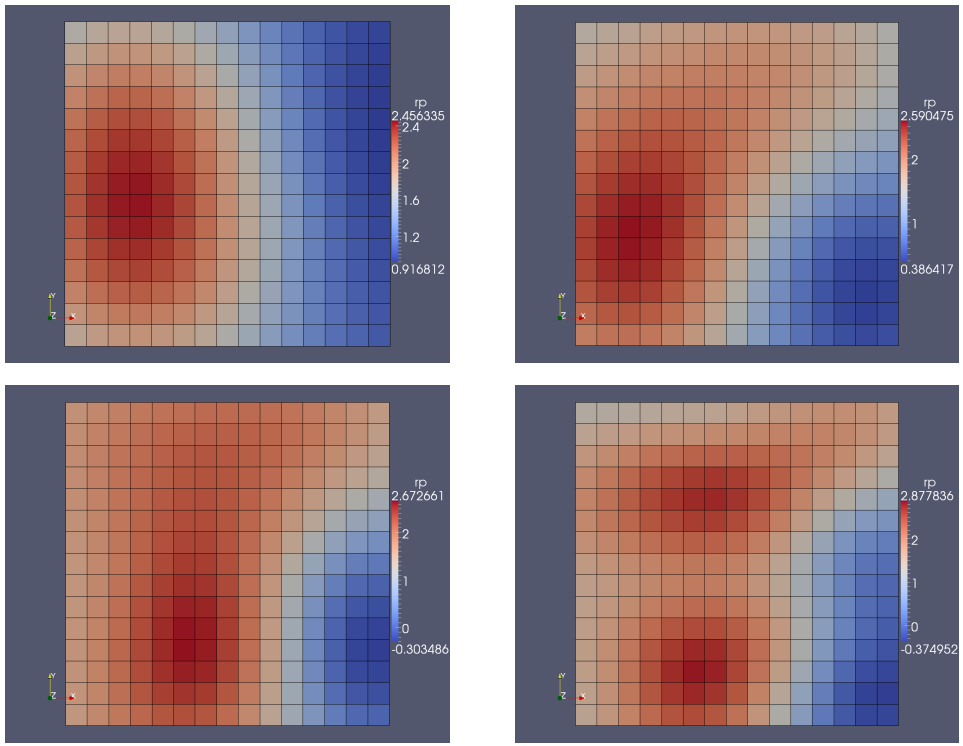


Figure: Realizations with 2, 4, 8, 16 terms (in xy-plane)

KL Eigenproblem

- Covariance kernel (final connection to data):

$$C(x,y) \equiv E[\alpha(x,\omega)\,\alpha(y,\omega)]$$

- KL expansion requires solution of eigenproblem:

$$\int_D \int_D C(x,y)\,\phi(y)\,v(x)\,dy\,dx = \lambda \int_D \phi(x)\,v(x)\,dx, \quad v \in L^2(D)$$

- C is symmetric positive definite, so eigenvalues are real, positive, and decrease to zero (Mercer's theorem).
- We are interested only in largest eigenvalues

Discrete KL Solver

- FE approximation using $V_h \subset L^2(D)$ leads to generalized eigenproblem

$$A\,\Phi = \lambda\, B\Phi$$

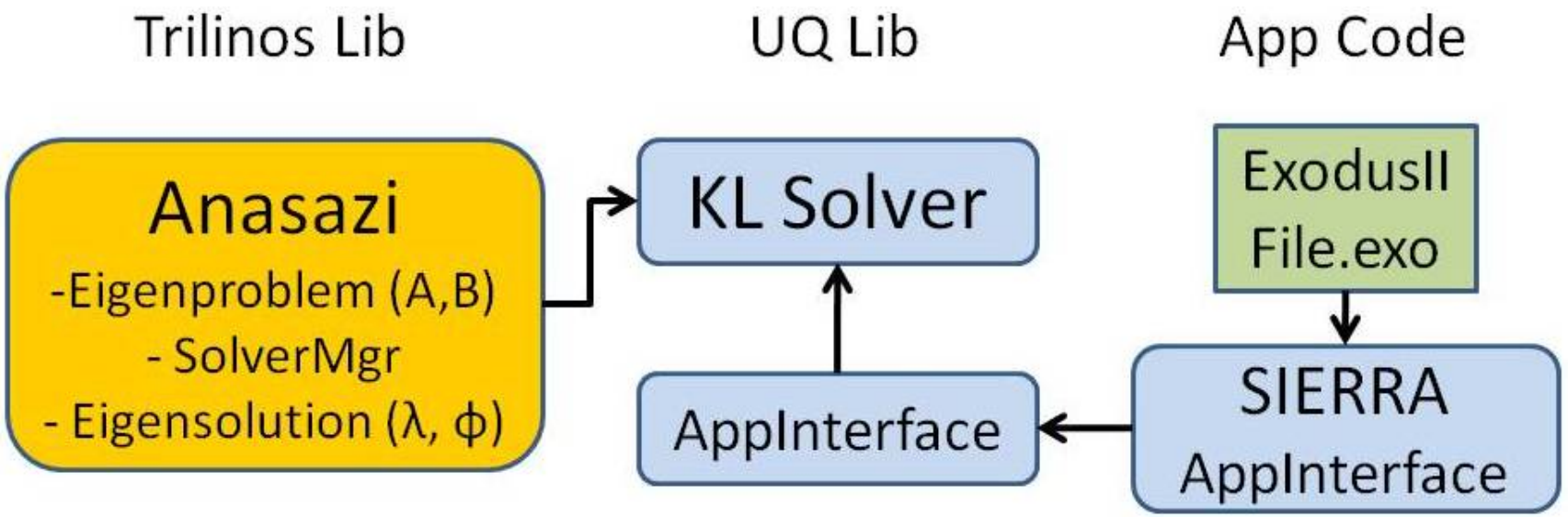
- Dense matrix A, sparse matrix B (both SPD)

$$A_{ij} \equiv \int_D \int_D C(x,y)\, v_i(y)\, v_j(x)\, dy\, dx, \quad B_{ij} \equiv \int_D v_i(x)\, v_j(x)\, dx$$

- Assembly of B matrix is standard (mass matrix)
- Assembly of dense A matrix requires double element loop

Lib/App Implementation of KL Solver

- Chose to parallelize matrix assembly independent of application
- Each processor provides MPI communicator and arrays of
  - local integration points for all local elements
  - volume weights (Jacobian times quadrature weights)
  - FE basis function values
- App does not have to do any parallel communication



Parallel Computing Considerations

- Main goals: apply many processors to
  - achieve speedup for a problem of fixed size
  - solve very large problems efficiently
- Local storage is limited - data is distributed among procs
- Cost of communication much higher than computation
- Code must run on many platforms
- Work must be load balanced
- Choice of parallel communication method
  - We will use MPI for all parallel operations

Challenges for Parallelization of KL Solver

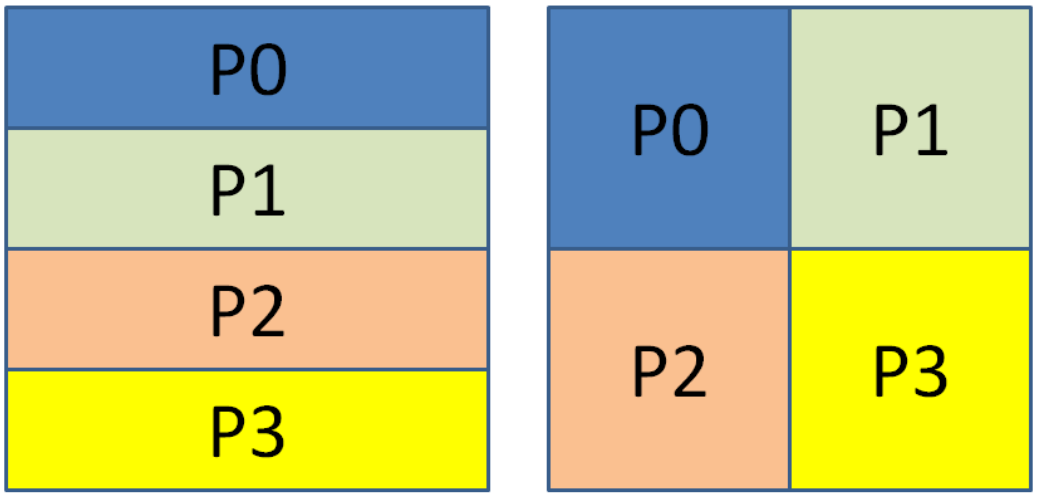
- RF may not be evenly distributed (load imbalance)
  - mesh typically uniformly distributed on nodes
  - RFs on surfaces and subdomains
- Each node can hold only a fraction of the mesh
- Matrix storage must be scalable
  - for very large problems, storage may be impractical
- Assembly must be efficient and scalable
- Preconditioning needed for iterative eigensolver
- Eigensolver must be parallel.

Matrix Storage Limitations

- Let N be num elements, P be num procs
- Assume each proc has N/P mesh elements (balanced load)
  - Range for N: $O(10^4)$-$O(10^9)$
  - Range for P: $O(10^1)$-$O(10^5)$
- Need to store N^2 matrix entries, N^2/P per proc
  - Compare to $O(N/P)$ per proc for sparse matrices
- Range for N^2/P: $O(10^3)$-$O(10^{13})$
- Matrix storage will be impractical for very large problems
- This suggests ultimately a matrix-free approach

Parallel Matrix Storage

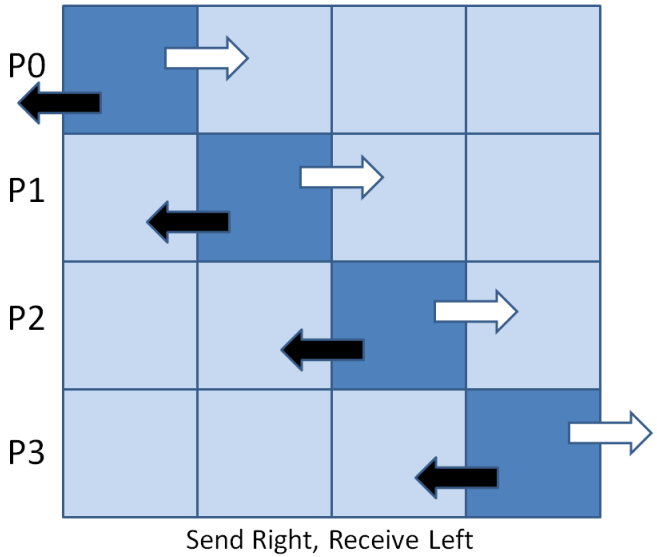
- Row storage scheme (current implementation)
  - $N/P \times N$ dense row block on each proc
  - Requires element data from P procs to assemble
- Square submatrix storage
  - Each proc has $N/\sqrt{P} \times N/\sqrt{P}$ dense submatrix
  - Block width is $\sqrt{P}\,(N/P)$
  - Requires element data from $2\sqrt{P}$ procs to assemble
- Matrix free (storage as needed for matvec)



Row Storage Block Storage

Parallel Assembly

- Each proc starts with N/P elements
- Local diagonal blocks computed in serial
- Row storage: loop over $(P-1)$ nonlocal columns
  - Send/receive data between pairs of procs
  - Assemble using localized data
- Square submatrix storage
  - Each proc needs data from $2\sqrt{P}$ procs
  - Fewer communication calls



Eigensolver and Preconditioner

- The eigensolver is an iterative block Davidson solver (Arbenz et. al.,"A Comparison of Eigensolvers for Large-scale 3D Modal Analysis Using AMG-Preconditioned Iterative Methods", IJNME, 64 (2005))
- Implemented in the Trilinos/Anasazi library (trilinos.sandia.gov)
- Requires only matvec operation
- Computes only a subset of eigenvalues (largest)
- Can leverage preconditioner of dense matrix A
  - Plan is to implement block Jacobi
  - Storage of diagonal N^2/P^2 submatrix per proc
  - One time generation cost

Application: KL Problem

- Example of kernel with correlation length scale $L = 7.92e{+}3$

$$C(x, y) = \exp(-|x - y|/L)$$

- Domain is brick with $L_x = L_y = 2.64e{+}4$ and $L_z = 2.4e{+}3$
- Coarse mesh size is about $2.64e{+}3$
- Hex meshes with $N = 10^3, 15^3, 20^3, 30^3, 40^3, 60^3, 80^3$ elements

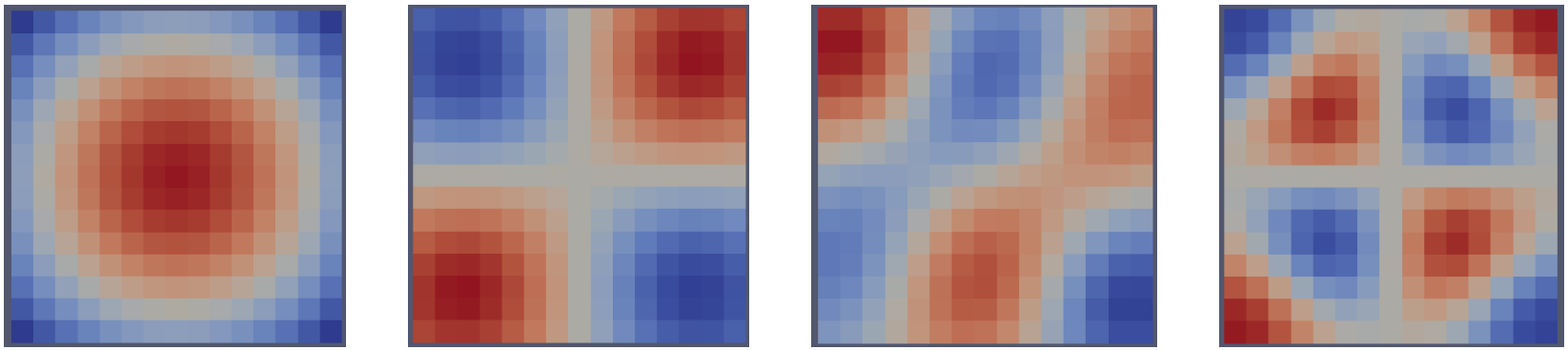
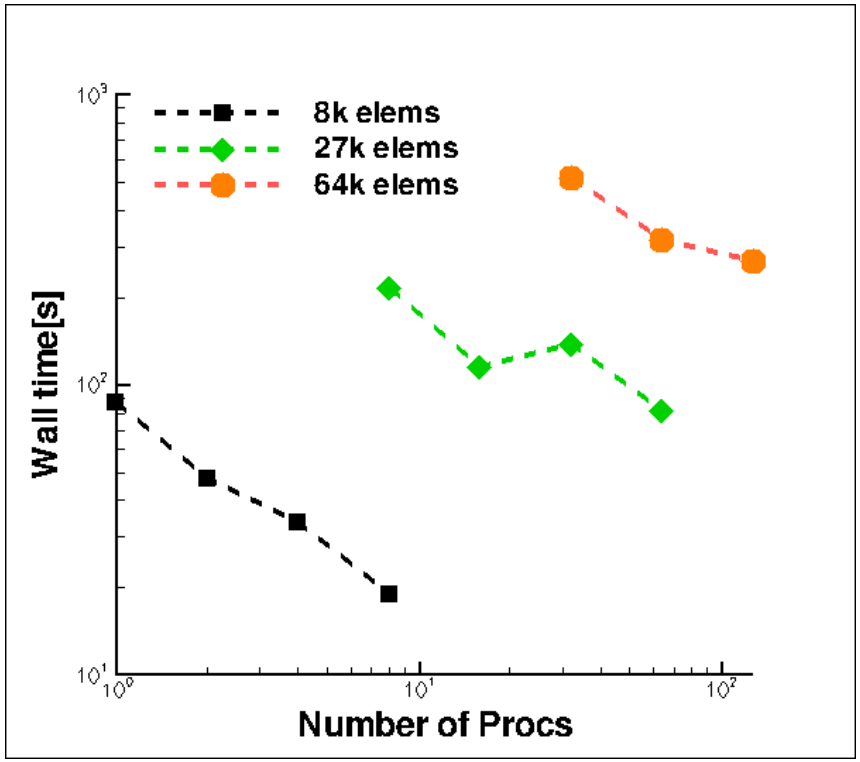
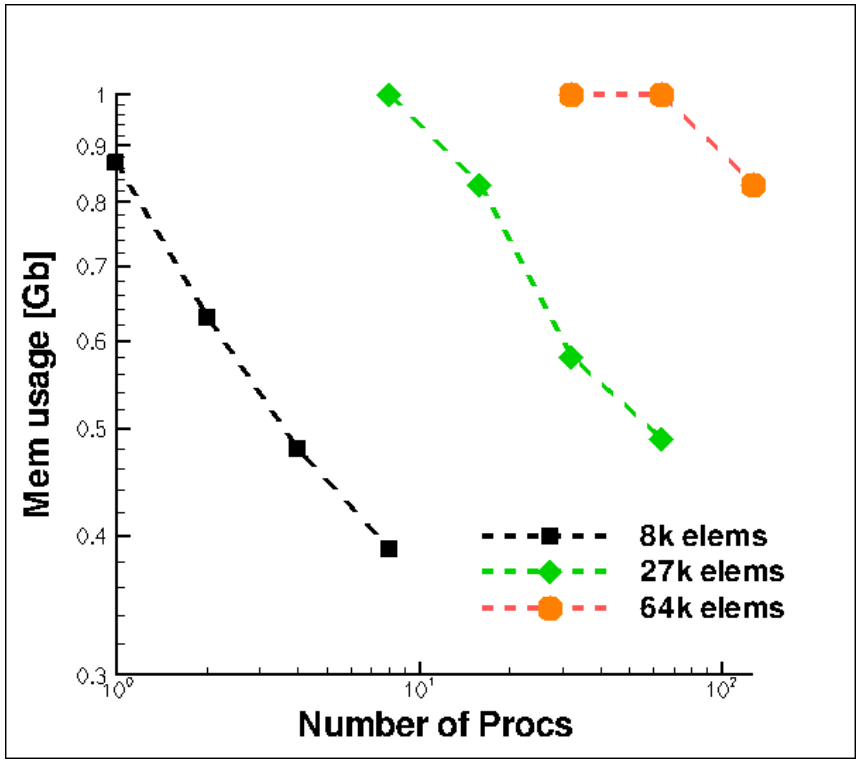


Figure: Eigenfunctions 1, 4, 8, 12 (in xy-plane) on 15^3 mesh

KL solver performance (cluster)

- Thunderbird: 2 procs/node, 6 GB/node, Infiniband, Open MPI
- Ran four different meshes from 8k to 216k elements





Application: Porous Flow & Geomechanics

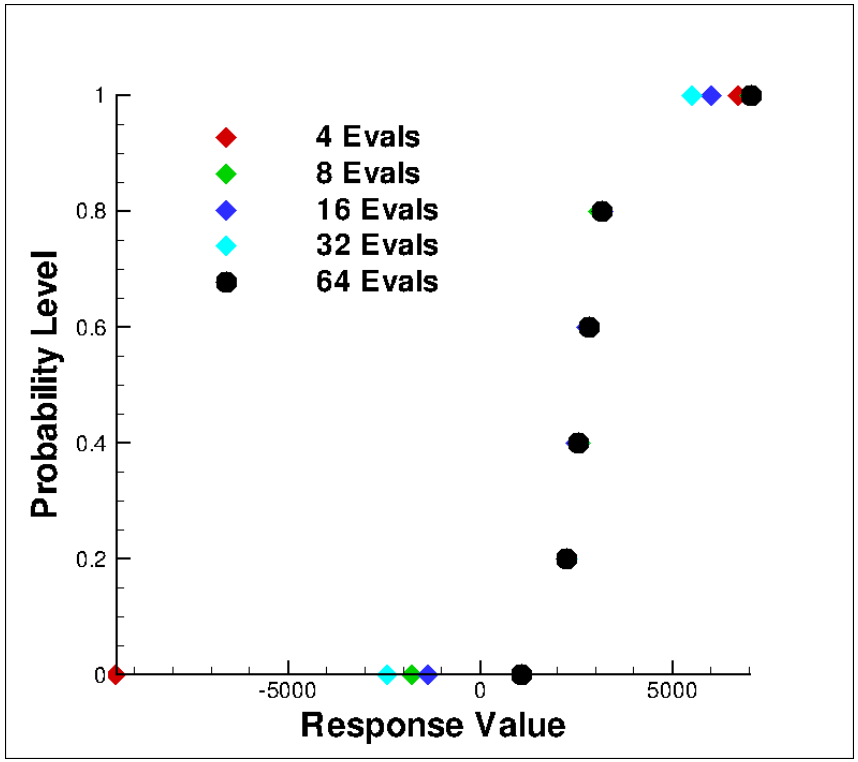
- Problem from Dean et. al., “A Comparison of Techniques for Coupling Porous Flow and Geomechanics”, SPE J. (2006), 11(1), 132-140.
- Single phase fluid depletion from central well (Aria)
- Poroelastic deformation of solid matrix (Adagio)
- Coupling from fluid pore pressure and displacements
- Random fields have constant mean and variance scaling
  - Initial pore pressure distribution (Aria and Adagio)
  - Relative permeability scaling (Aria)
  - Young’s modulus (Adagio)
- Response function is the pressure at a point in the well at final time

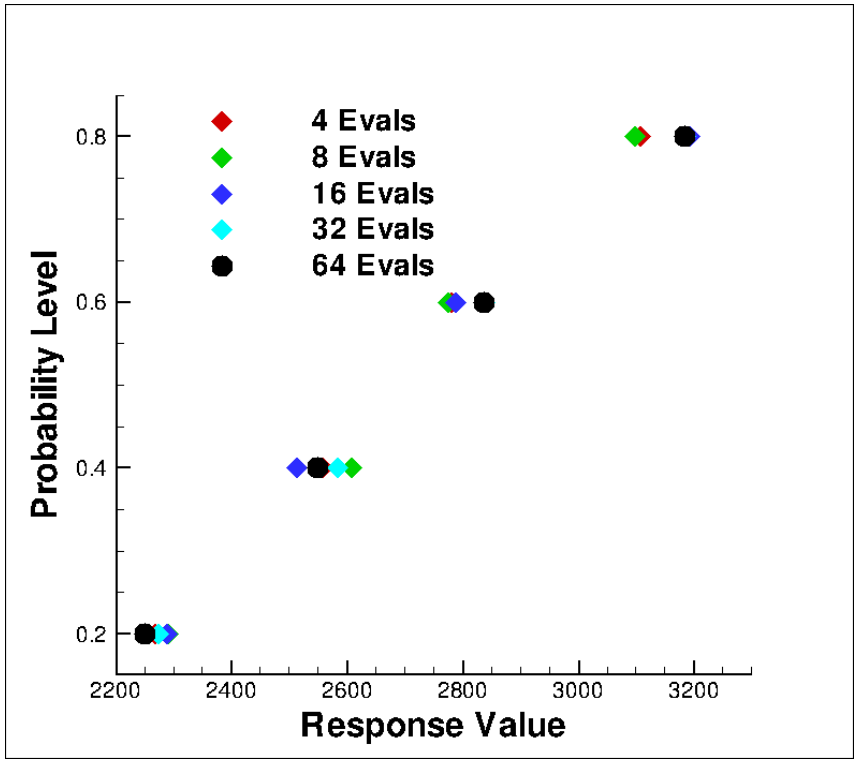
DAKOTA Inputs

- DAKOTA is a toolkit including many non-intrusive UQ algorithms
- A PCE surrogate model of the response function is used
- PCE coefficients are estimated by evaluating the coupled Aria/Adagio model a fixed number of times
- Can also use advanced sampling/quadrature methods
- Each RV can use the same RV coefficients or independent ones
  - We assume i.i.d. standard normal for now
- The surrogate model is evaluated 100k times to generate statistics
- We compute response levels for six probability levels (including tails)

Effect of number of model evaluations

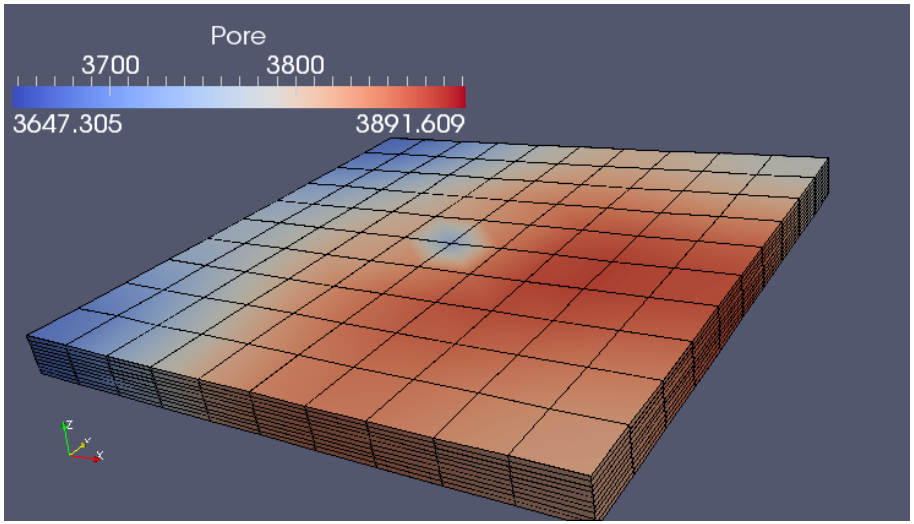
- Two RVs used (two term RF expansions), same for all RFs
- Model evaluations increased from 4-64.

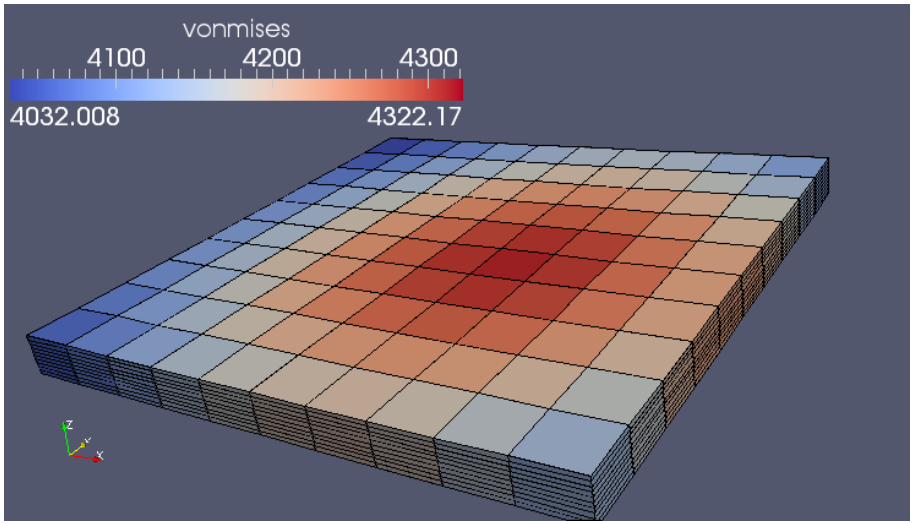




Solution States

- We plot fluid pore pressure and Von Mises stress for a realization





Future Work

- PCE expansions of RF
- Parallel scaling (MPI profiling)
- Preconditioning
- Library interface to experimental data
- Matrix free KL solver
- Move lib code to Trilinos/STK-Encore
- Simple ExodusII-based exec based on STK-Encore/STK-Mesh