

Massively Parallel Monte Carlo Methods for Discrete Linear and Nonlinear Systems

Stuart R. Slattery
Engineering Physics Department
University of Wisconsin - Madison

October 31, 2012



- Predictive modeling and simulation enhances engineering capability
- Modern work focused on this task leverages multiple physics simulation (CASL, NEAMS)
- New hardware drives algorithm development (petascale and exascale)
- Monte Carlo methods have the potential to provide great improvements that permit finer simulations and better mapping to future hardware
- A set of massively parallel Monte Carlo methods is proposed to advance multiple physics simulation on contemporary and future leadership class machines





Predictive nuclear reactor analysis enables...

- Tighter design tolerance for improved thermal performance and efficiency
- Higher fuel burn-up
- High confidence in accident scenario models

Predictive nuclear reactor analysis enables...

- Tighter design tolerance for improved thermal performance and efficiency
- Higher fuel burn-up
- High confidence in accident scenario models

Multiple physics simulations are complicated...

- Neutronics, thermal hydraulics, computational fluid dynamics, structural mechanics, and many other physics
- Consistent models yield nonlinearities in the variables through feedback effects
- Tremendous computational resources are required with $O(1 \times 10^9)$ element meshes and $O(100,000)+$ cores used in today's simulations.

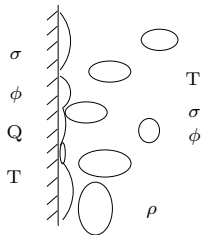
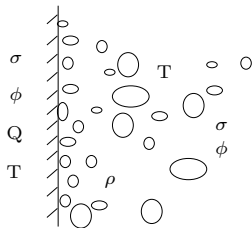


Figure: Departure from nucleate boiling scenario.

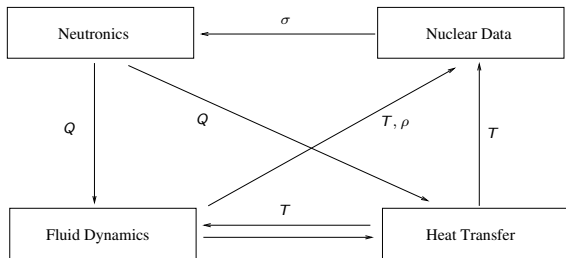
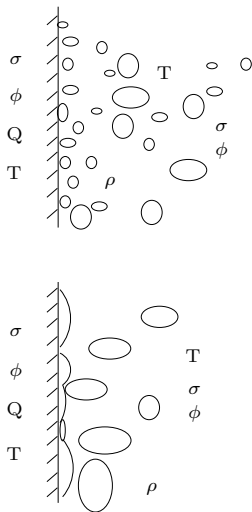


Figure: Multiphysics dependency analysis of departure from nucleate boiling.

Figure: Departure from nucleate boiling scenario.

- Modern hardware is moving in two directions:
 - Lightweight machines
 - Heterogeneous machines
 - Both characterized by low power and high concurrency
- Some issues:
 - Higher potential for both soft and hard failures
 - Memory restrictions are expected with a continued decrease in memory/FLOPS
- Potential resolution from Monte Carlo:
 - Soft failures buried within the tally variance
 - Hard failures are high variance events
 - Memory savings over conventional methods



- Parallelization of Monte Carlo methods for discrete systems
 - Parallel strategies taken from modern reactor physics methods
 - Research is required to explore varying parallel strategies
 - Scalability is of concern
- Development of a nonlinear solver leveraging Monte Carlo
 - Application to nonlinear problems of interest
 - Memory benefits
 - Performance benefits

- We seek solutions of the general linear operator equation

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} \in \mathbb{R}^{N \times N}, \mathbf{A} : \mathbb{R}^N \rightarrow \mathbb{R}^N, \mathbf{x} \in \mathbb{R}^N, \mathbf{b} \in \mathbb{R}^N$$

$$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$$

- $\mathbf{r} = \mathbf{0}$ when an exact solution is found.

- We seek solutions of the general linear operator equation

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} \in \mathbb{R}^{N \times N}, \mathbf{A} : \mathbb{R}^N \rightarrow \mathbb{R}^N, \mathbf{x} \in \mathbb{R}^N, \mathbf{b} \in \mathbb{R}^N$$

$$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$$

- $\mathbf{r} = \mathbf{0}$ when an exact solution is found.

A Requirement

Assert that \mathbf{A} is *nonsingular*. The solution is then:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

- General stationary methods are formed by splitting the linear operator

$$\mathbf{A} = \mathbf{M} - \mathbf{N} .$$

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x} + \mathbf{M}^{-1}\mathbf{b} .$$

- We identify $\mathbf{H} = \mathbf{M}^{-1}\mathbf{N}$ as the *iteration matrix*

$$\mathbf{x}^{k+1} = \mathbf{H}\mathbf{x}^k + \mathbf{c} .$$

- The qualities of the iteration matrix dictate convergence
- Define $\mathbf{e}^k = \mathbf{x}^k - \mathbf{x}$ as the error at the k^{th} iterate

$$\mathbf{e}^{k+1} = \mathbf{H}\mathbf{e}^k$$

- We diagonalize \mathbf{H} to extract its Eigenvalues

$$\|\mathbf{e}^k\|_2 = \rho(\mathbf{H})^k \|\mathbf{e}^0\|_2,$$

- We bound \mathbf{H} by $\rho(\mathbf{H}) < 1$ for convergence



- Powerful class of iterative methods
- Provides theory that encapsulates most other iterative methods
- Leveraged in many modern physics codes at the petascale

- Powerful class of iterative methods
- Provides theory that encapsulates most other iterative methods
- Leveraged in many modern physics codes at the petascale

Search Subspace \mathcal{K}

Extract the solution from the search subspace:

$$\tilde{\mathbf{x}} = \mathbf{x}_0 + \boldsymbol{\delta}, \boldsymbol{\delta} \in \mathcal{K}$$

- Powerful class of iterative methods
- Provides theory that encapsulates most other iterative methods
- Leveraged in many modern physics codes at the petascale

Search Subspace \mathcal{K}

Extract the solution from the search subspace:

$$\tilde{\mathbf{x}} = \mathbf{x}_0 + \boldsymbol{\delta}, \quad \boldsymbol{\delta} \in \mathcal{K}$$

Constraint Subspace \mathcal{L}

Constrain the extraction with the constraint subspace by asserting orthogonality with the residual:

$$\langle \tilde{\mathbf{r}}, \mathbf{w} \rangle = 0, \quad \forall \mathbf{w} \in \mathcal{L}$$

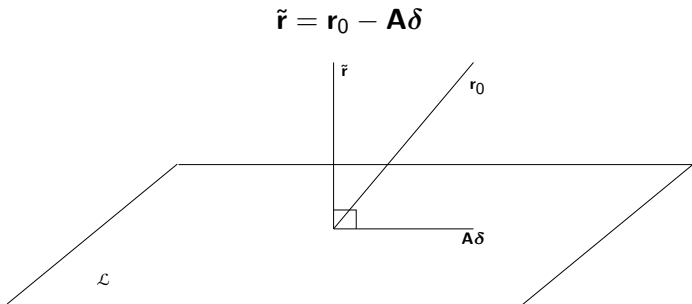


Figure: Orthogonality constraint of the new residual with respect to \mathcal{L} .

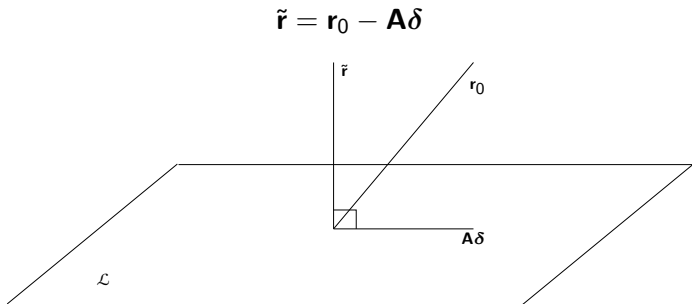


Figure: Orthogonality constraint of the new residual with respect to \mathcal{L} .

Minimization Property

The residual of the system will always be *minimized* with respect to the constraints

$$\|\tilde{\mathbf{r}}\|_2 \leq \|\mathbf{r}_0\|_2, \quad \forall \mathbf{r}_0 \in \mathbb{R}^N,$$



- Choose \mathbf{V} as a basis of \mathcal{K} and \mathbf{W} as a basis of \mathcal{L}

$$\delta = \mathbf{V}\mathbf{y}, \quad \forall \mathbf{y} \in \mathbb{R}^N$$

$$\mathbf{y} = (\mathbf{W}^T \mathbf{A} \mathbf{V})^{-1} \mathbf{W}^T \mathbf{r}_0$$

- Choose \mathbf{V} as a basis of \mathcal{K} and \mathbf{W} as a basis of \mathcal{L}

$$\delta = \mathbf{V}\mathbf{y}, \quad \forall \mathbf{y} \in \mathbb{R}^N$$

$$\mathbf{y} = (\mathbf{W}^T \mathbf{A} \mathbf{V})^{-1} \mathbf{W}^T \mathbf{r}_0$$

Projection Method Iteration

$$\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$$

$$\mathbf{y}^k = (\mathbf{W}^T \mathbf{A} \mathbf{V})^{-1} \mathbf{W}^T \mathbf{r}^k$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{V}\mathbf{y}^k$$

Update \mathbf{V} and \mathbf{W}

$$\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \dots, \mathbf{A}^{m-1}\mathbf{r}_0\}$$

$$\mathcal{L} = \mathbf{A}\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0)$$

- Yields the normal system $\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b}$
- Must generate an orthonormal basis $\mathbf{V}_m \in \mathbb{R}^{N \times m}$ for $\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0)$
- $\mathbf{W}_m = \mathbf{A} \mathbf{V}_m$
- Typically choose a Gram-Schmidt-like procedure such as Arnoldi or Lanczos

Algorithm 1 GMRES Iteration

```
1:  $\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$ 
2:  $\beta := \|\mathbf{r}_0\|_2$ 
3:  $\mathbf{v}_1 := \mathbf{r}_0/\beta$  {Create the orthonormal basis for the Krylov subspace}
4: for  $j = 1, 2, \dots, m$  do
5:    $\mathbf{w}_j := \mathbf{A}\mathbf{v}_j$ 
6:   for  $i = 1, 2, \dots, j$  do
7:      $h_{ij} \leftarrow \langle \mathbf{w}_j, \mathbf{v}_i \rangle$ 
8:      $\mathbf{w}_j \leftarrow \mathbf{w}_j - h_{ij}\mathbf{v}_i$ 
9:   end for
10:   $h_{j+1,j} \leftarrow \|\mathbf{w}_j\|_2$ 
11:   $\mathbf{v}_{j+1} \leftarrow \mathbf{w}_j/h_{j+1,j}$ 
12: end for{Apply the orthogonality constraints}
13:  $\mathbf{y}_m \leftarrow \operatorname{argmin}_{\mathbf{y}} \|\beta\mathbf{e}_1 - \mathbf{H}_m \mathbf{y}\|_2$ 
14:  $\mathbf{x}_m \leftarrow \mathbf{x}_0 + \mathbf{V}_m \mathbf{y}_m$ 
```

- Parallel vector update

$$\mathbf{y}[n] \leftarrow \mathbf{y}[n] + a * \mathbf{x}[n], \quad \forall n \in [1, N_g]$$

$$\mathbf{y}[n] \leftarrow \mathbf{y}[n] + a * \mathbf{x}[n], \quad \forall n \in [1, N_l]$$

- Parallel dot product

$$d_l = \mathbf{y}_l \cdot \mathbf{x}_l, \quad d_g = \sum_p d_l$$

- Parallel vector norm

$$\|x\|_{\infty, l} = \max_n \mathbf{y}[n], \quad \forall n \in [1, N_l]$$

$$\|x\|_{\infty, g} = \max_p \|x\|_{\infty, l}$$

Parallel Matrix-Vector Multiplication

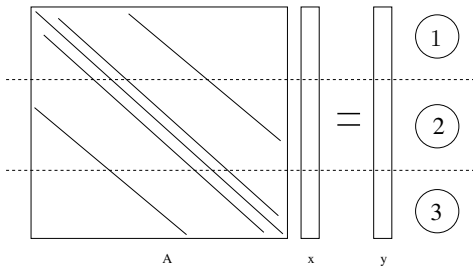


Figure: Matrix-vector multiply $Ax = y$ operation on 3 processors.

Parallel Matrix-Vector Multiplication

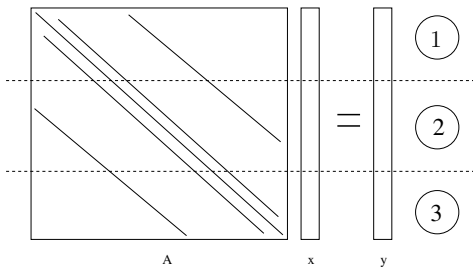


Figure: Matrix-vector multiply $Ax = y$ operation on 3 processors.

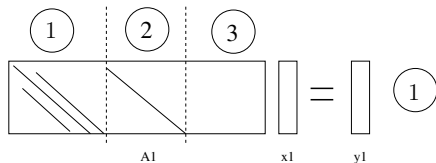


Figure: Components of multiply operation owned by process 1.



- Global reduction operations observed not to impede scalability
 - Dot product
 - Vector norms
- Nearest neighbor computations have poor algorithmic strong scaling
 - Matrix-vector multiply
 - Weak scaling is better



- First proposed by J. Von Neumann and S.M. Ulam in the 1940's
- Earliest published reference in 1950
- General lack of published work
- Modern work by Evans and others has yielded new applications

- Split the operator

$$\mathbf{H} = \mathbf{I} - \mathbf{A}$$

$$\mathbf{x} = \mathbf{H}\mathbf{x} + \mathbf{b}$$

- Generate the *Neumann series*

$$\mathbf{A}^{-1} = (\mathbf{I} - \mathbf{H})^{-1} = \sum_{k=0}^{\infty} \mathbf{H}^k$$

- Require $\rho(\mathbf{H}) < 1$ for convergence

$$\mathbf{A}^{-1}\mathbf{b} = \sum_{k=0}^{\infty} \mathbf{H}^k \mathbf{b} = \mathbf{x}$$

- Expand the Nuemann series

$$x_i = \sum_{k=0}^{\infty} \sum_{i_1}^N \sum_{i_2}^N \dots \sum_{i_k}^N h_{i,i_1} h_{i_1,i_2} \dots h_{i_{k-1},i_k} b_{i_k}$$

- Define a sequence of state transitions

$$\nu = i \rightarrow i_1 \rightarrow \dots \rightarrow i_{k-1} \rightarrow i_k$$

- Define the *Neumann-Ulam decomposition*¹

$$\mathbf{H} = \mathbf{P} \circ \mathbf{W}$$

¹The Hadamard product $\mathbf{A} = \mathbf{B} \circ \mathbf{C}$ is defined element-wise as $a_{ij} = b_{ij}c_{ij}$.

- Compute row-normalized transition probabilities and weights

$$p_{ij} = \frac{|h_{ij}|}{\sum_j |h_{ij}|}, \quad w_{ij} = \frac{h_{ij}}{p_{ij}}$$

- Generate an expectation value for the solution

$$W_m = \sum_{m=0}^k w_{i,i_1} w_{i_1,i_2} \cdots w_{i_{m-1},i_m}$$

$$X_\nu(i_0 = i) = \sum_{m=0}^k W_m b_{i_m}$$

- Compute the probability of a particular random walk permutation

$$P_\nu = p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k}$$

- Generate the estimator

$$E\{X(i_0 = i)\} = \sum_{\nu} P_\nu X_\nu$$

- Check that we recover the exact solution

$$\begin{aligned} E\{X(i_0 = i)\} &= \sum_{k=0}^{\infty} \sum_{i_1}^N \sum_{i_2}^N \cdots \sum_{i_k}^N p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k} w_{i,i_1} w_{i_1,i_2} \cdots w_{i_{k-1},i_k} b_{i_k} \\ &= x_i, \end{aligned}$$

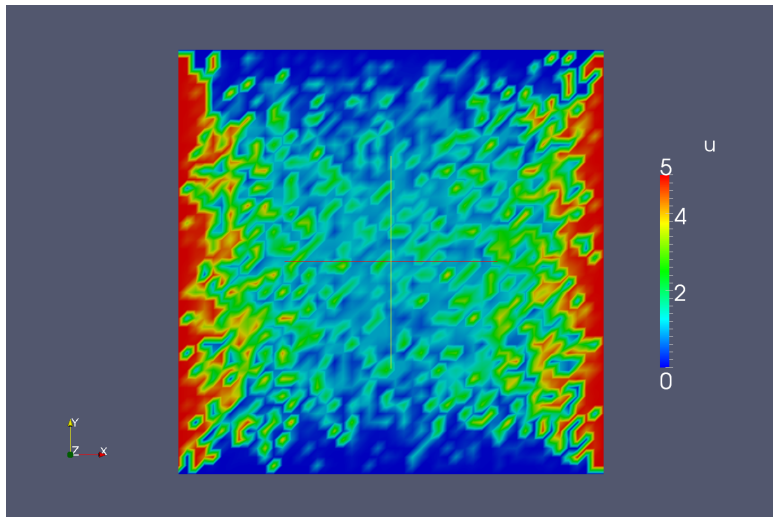


Figure: Direct solution to Poisson Equation. *1 history per state, 2.5×10^3 total histories. 0.785 seconds CPU time.*

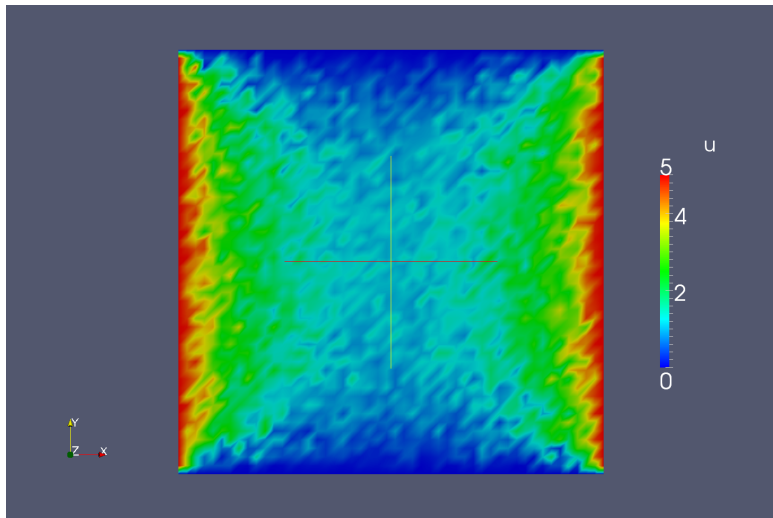


Figure: Direct solution to Poisson Equation. *10 histories per state, 2.5×10^4 total histories. 5.9 seconds CPU time.*

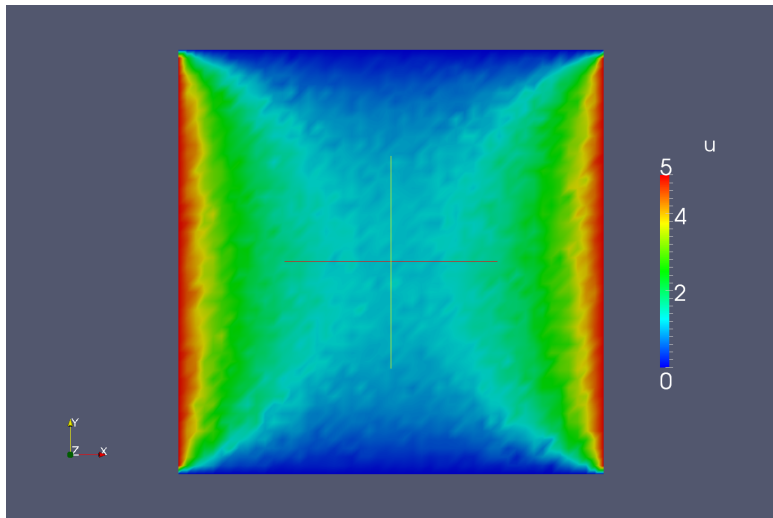


Figure: Direct solution to Poisson Equation. *100 histories per state, 2.5×10^5 total histories. 54.7 seconds CPU time.*

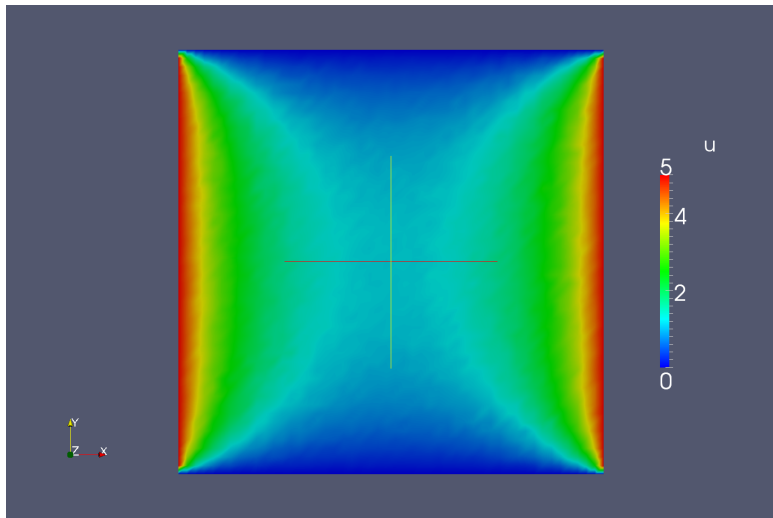


Figure: Direct solution to Poisson Equation. *100 histories per state, 2.5×10^6 total histories. 644 seconds CPU time.*

- Solve the adjoint linear system

$$\mathbf{A}^T \mathbf{y} = \mathbf{d}$$

$$\mathbf{y} = \mathbf{H}^T \mathbf{y} + \mathbf{d}$$

- Set the adjoint constraint

$$\langle \mathbf{A}^T \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{A} \mathbf{y} \rangle$$

$$\langle \mathbf{x}, \mathbf{d} \rangle = \langle \mathbf{y}, \mathbf{b} \rangle$$

- Generate the Neumann series for the adjoint operator

$$\mathbf{y} = (\mathbf{I} - \mathbf{H}^T)^{-1} \mathbf{d}$$

$$\mathbf{y} = \sum_{k=0}^{\infty} (\mathbf{H}^T)^k \mathbf{d}$$

- Expand the series

$$y_i = \sum_{k=0}^{\infty} \sum_{i_1}^N \sum_{i_2}^N \cdots \sum_{i_k}^N h_{i_k, i_{k-1}} \cdots h_{i_2, i_1} h_{i_1, i} d_{i_k}$$

- Pick another constraint to yield the original solution

$$\mathbf{d} = \boldsymbol{\delta}_i, \langle \mathbf{y}, \mathbf{b} \rangle = \langle \mathbf{x}, \boldsymbol{\delta}_i \rangle = x_i$$

- Use the adjoint Neumann-Ulam decomposition

$$\mathbf{H}^T = \mathbf{P} \circ \mathbf{W}$$

$$p_{ij} = \frac{|h_{ji}|}{\sum_j |h_{ji}|}, \quad w_{ij} = \frac{h_{ji}}{p_{ij}}$$

- Build the estimator and expectation value

$$X_\nu = \sum_{m=0}^k W_m b_{i_0} \delta_{i, i_m}$$

$$\begin{aligned} E\{X_j\} &= \sum_{k=0}^{\infty} \sum_{i_1}^N \sum_{i_2}^N \dots \sum_{i_k}^N b_{i_0} h_{i, i_1} h_{i_1, i_2} \dots h_{i_{k-1}, i_k} \delta_{i_k, j} \\ &= x_j, \end{aligned}$$

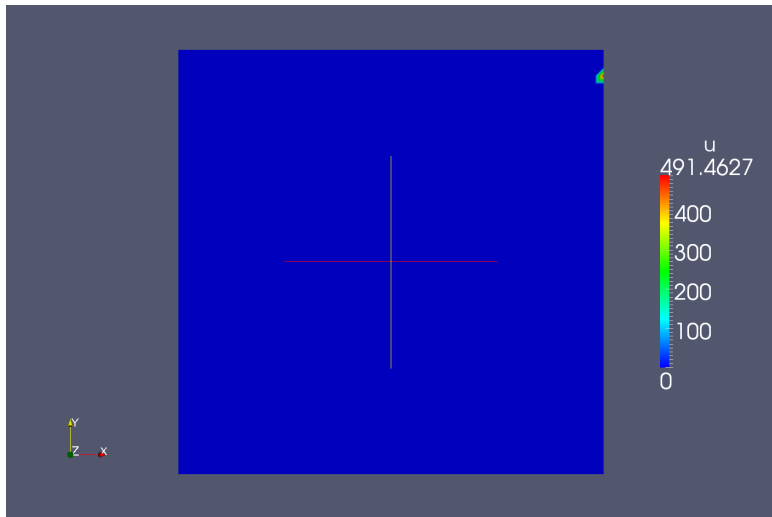


Figure: Adjoint solution to Poisson Equation. 1×10^0 total histories, 0.286 seconds CPU time.

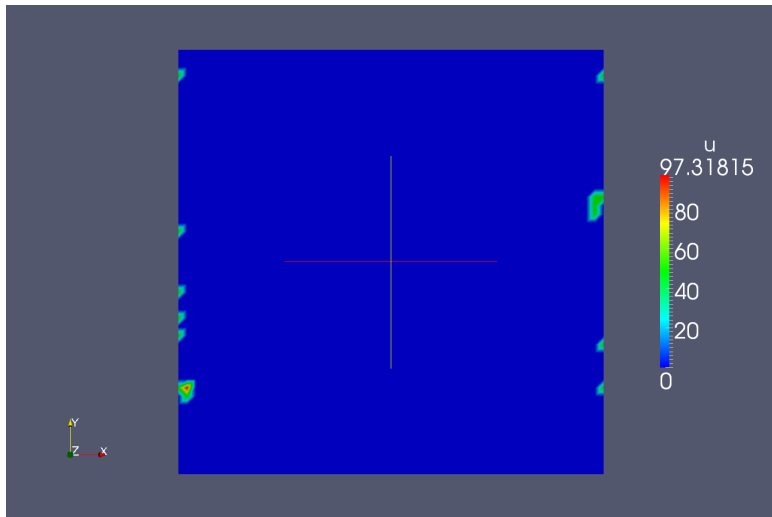


Figure: Adjoint solution to Poisson Equation. 1×10^1 total histories, 0.278 seconds CPU time.

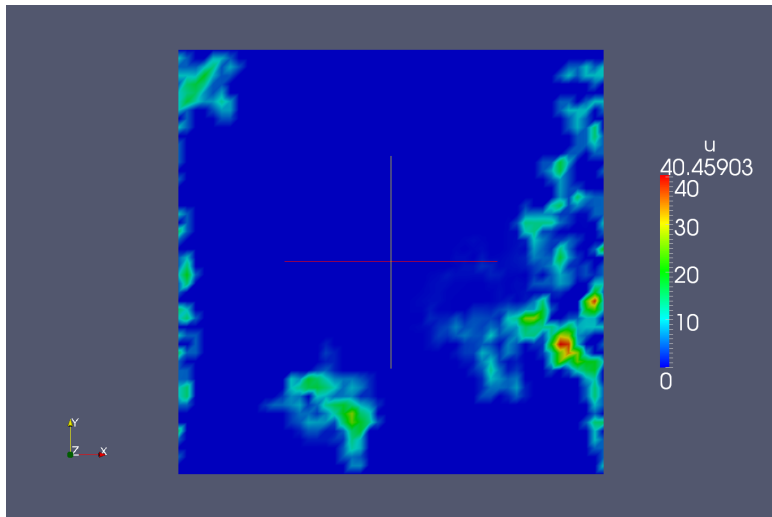


Figure: Adjoint solution to Poisson Equation. 1×10^2 total histories, 0.275 seconds CPU time.

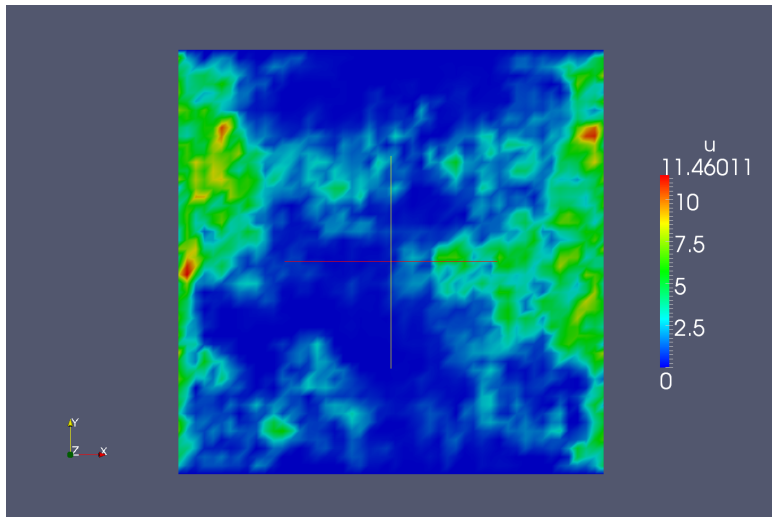


Figure: Adjoint solution to Poisson Equation. 1×10^3 total histories, 0.291 seconds CPU time.

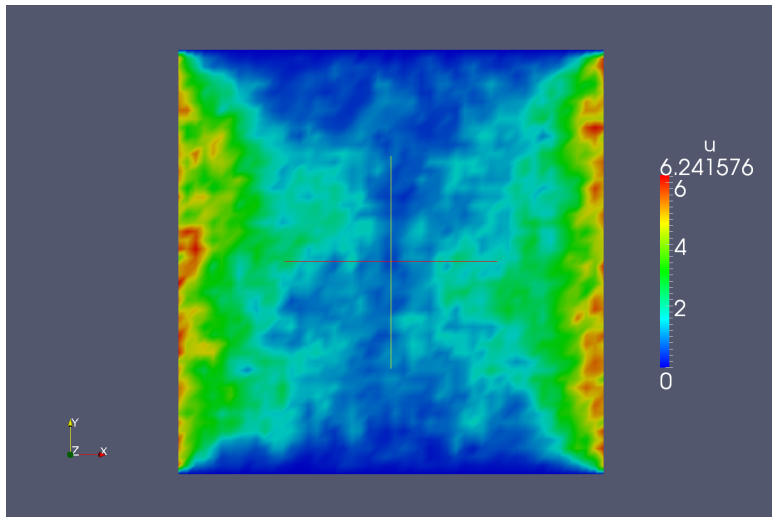


Figure: Adjoint solution to Poisson Equation. 1×10^4 total histories, 0.428 seconds CPU time.

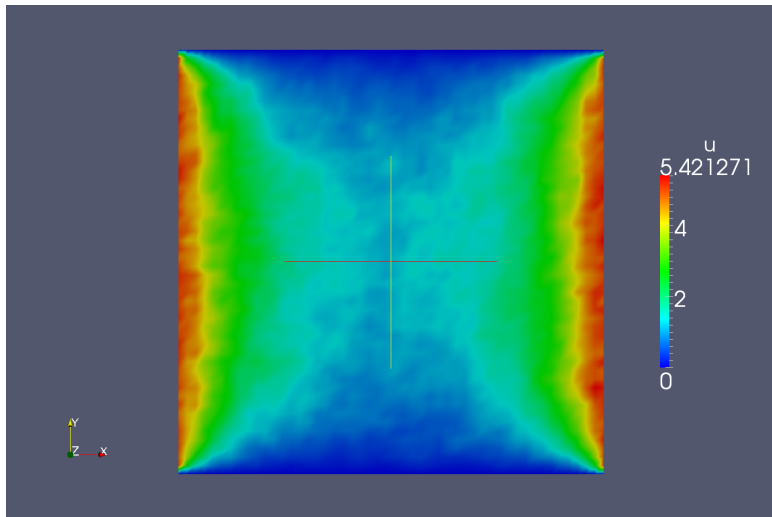


Figure: Adjoint solution to Poisson Equation. 1×10^5 total histories, 1.76 seconds CPU time.

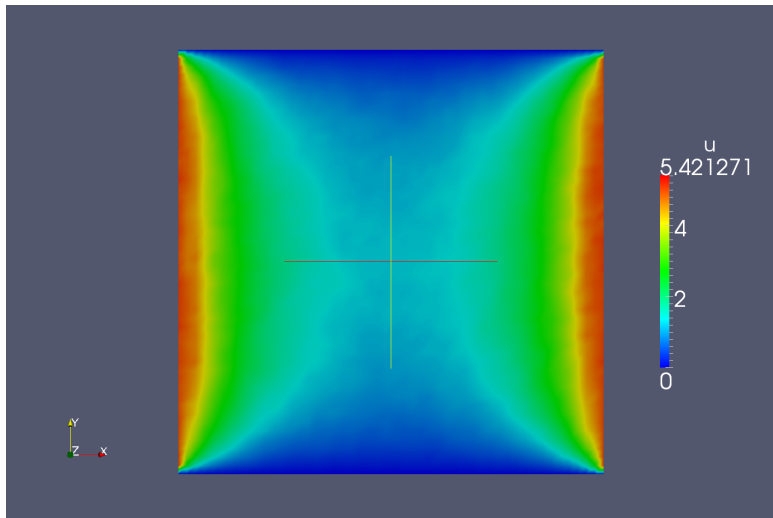


Figure: Adjoint solution to Poisson Equation. 1×10^6 total histories, 15.1 seconds CPU time.

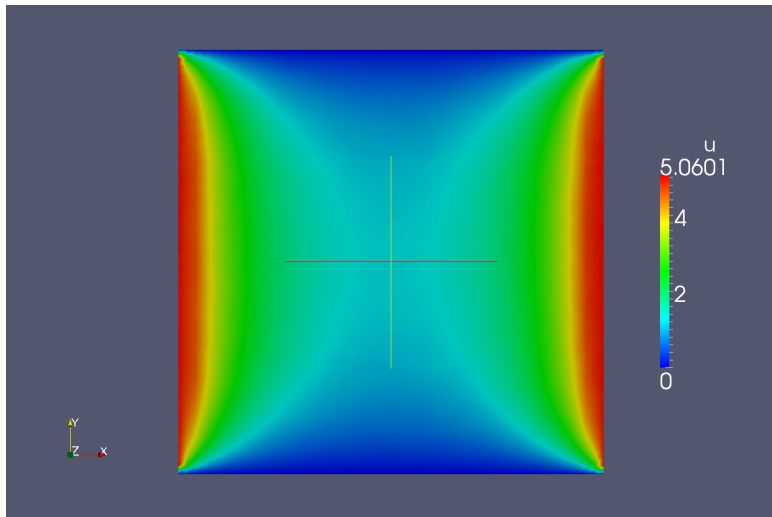


Figure: Adjoint solution to Poisson Equation. 1×10^7 total histories, 149 seconds CPU time.

- Neumann-Ulam methods bound by the Central Limit Theorem
- Halton proposed an iterative residual method
- Iteration error decoupled from Monte Carlo error
- Exponential convergence

$$\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$$

$$\mathbf{A}\delta^k = \mathbf{r}^k$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \delta^k$$

- Split the operator to yield Richardson's iteration

$$\mathbf{x} = (\mathbf{I} - \mathbf{A})\mathbf{x} + \mathbf{b}$$
$$\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\mathbf{x}^k + \mathbf{b}$$

- Define the iteration error

$$\delta\mathbf{x}^k = \mathbf{x} - \mathbf{x}^k$$
$$\delta\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\delta\mathbf{x}^k$$

- Subtract $(\mathbf{I} - \mathbf{A})\delta\mathbf{x}^{k+1}$

$$\begin{aligned}\mathbf{A}\delta\mathbf{x}^{k+1} &= (\mathbf{I} - \mathbf{A})(\mathbf{x}^{k+1} - \mathbf{x}^k) \\ &= \mathbf{r}^{k+1}\end{aligned}$$

- The following converges in one iteration with exact inversion of \mathbf{A} :

$$\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\mathbf{x}^k + \mathbf{b}$$

$$\mathbf{A}\delta\mathbf{x}^{k+1} = \mathbf{r}^{k+1}$$

$$\mathbf{x} = \mathbf{x}^{k+1} + \delta\mathbf{x}^{k+1}$$

MCSA Iteration

$$\mathbf{x}^{k+1/2} = (\mathbf{I} - \mathbf{A})\mathbf{x}^k + \mathbf{b}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1/2}$$

$$\hat{\mathbf{A}}\delta\mathbf{x}^{k+1/2} = \mathbf{r}^{k+1/2}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2}$$

- Adjoint Neumann-Ulam solver computes the correction
- Decouples Monte Carlo error from solution error
- Exponential convergence
- Demonstrated by Evans and colleagues to be competitive with Krylov methods

- No symmetry requirements
- Require $\rho(\mathbf{H}) < 1$
- Choose Jacobi preconditioning at a minimum

$$\mathbf{M} = \text{diag}(\mathbf{A})$$

$$\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$$

- Yields a preconditioned MCSA iteration with no in-state transitions

$$\mathbf{x}^{k+1/2} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{x}^k + \mathbf{b}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{M}^{-1}\mathbf{Ax}^{k+1/2}$$

$$\mathbf{M}^{-1}\mathbf{A}\delta\mathbf{x}^{k+1/2} = \mathbf{r}^{k+1/2}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2}$$

- Analysis needed to select Monte Carlo method
- Time-dependent 2-dimensional Poisson equation
- Spectral radius fixed
- Sparsity varied with 2 Laplacian stencils

$$\nabla_5^2 = \frac{1}{\Delta^2} [u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}]$$

$$\nabla_9^2 = \frac{1}{6\Delta^2} [4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} + u_{i-1,j-1} \\ + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} - 20u_{i,j}]$$

- Implicit Euler time differencing

$$\mathbf{A}\mathbf{u}^{n+1} = \mathbf{u}^n$$

Direct vs. Adjoint Analysis

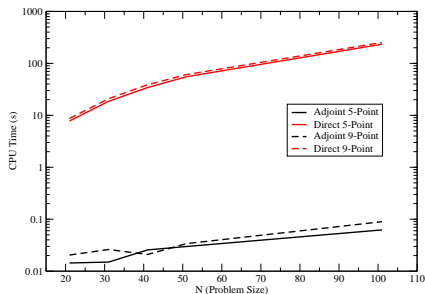


Figure: CPU Time (s) to converge vs. Problem Size (N for an $N \times N$ square mesh).

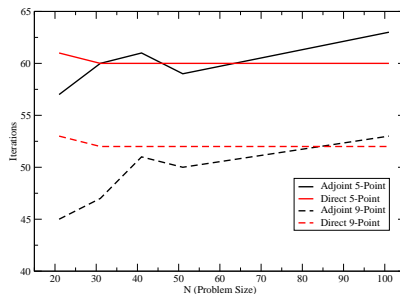


Figure: Iterations to converge vs. Problem Size (N for an $N \times N$ square mesh).

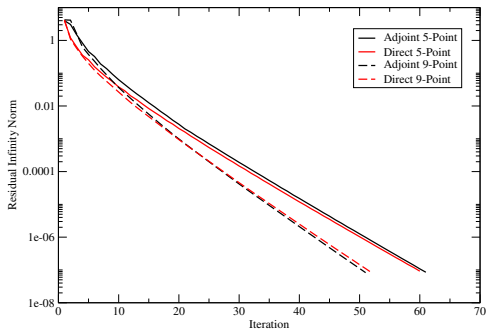


Figure: Infinity norm of the solution residual vs. iteration number for a problem of fixed size.

- CPU time dominating factor in method selection
- Significant speedup with adjoint method
- Does not affect convergence behavior
- Use adjoint with MCSA and Sequential Monte Carlo



- No literature observed for parallel Neumann-Ulam solvers
- Numerous references for modern parallel Monte Carlo methods in reactor physics
- MCSA parallelism comes from parallel matrix/vector operations



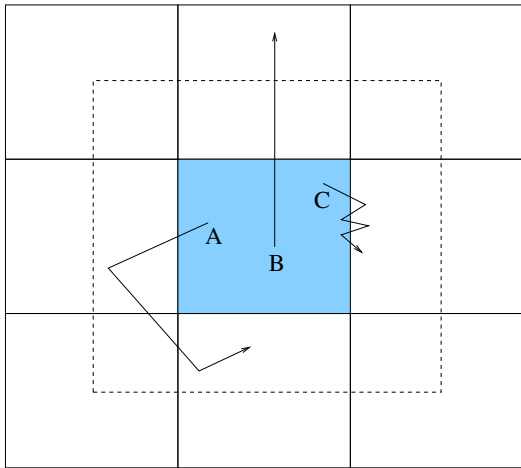


Figure: Overlapping domain example illustrating how domain overlap can reduce communication costs.



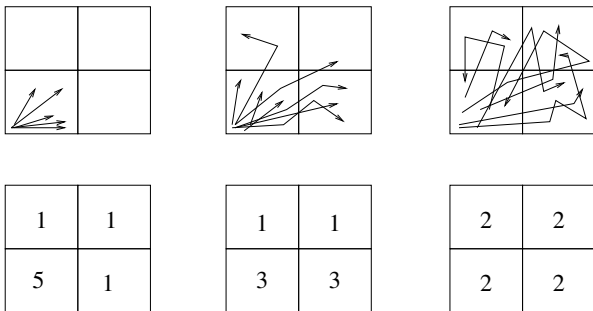


Figure: Example illustrating how domain decomposition can create load balance issues in Monte Carlo.

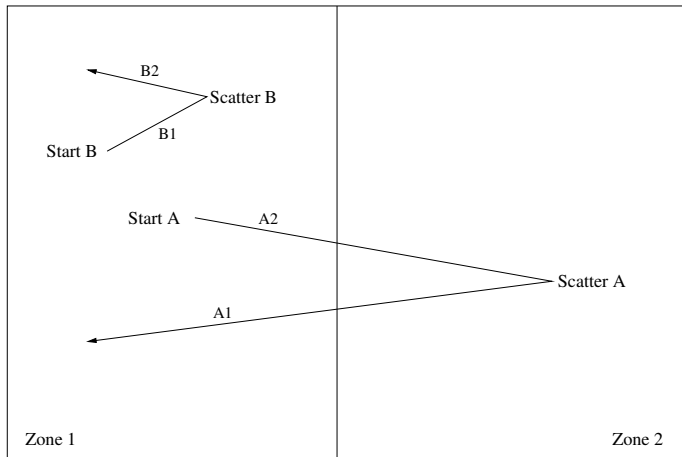


Figure: Gentile's example illustrating how domain decomposition can create reproducibility issues in Monte Carlo.





Monte Carlo Solution Methods for Nonlinear Problems



- We seek solutions of the general nonlinear problem

$$\mathbf{F}(\mathbf{u}) = \mathbf{0}$$

$$\mathbf{u} \in \mathbb{R}^n, \mathbf{F} : \mathbb{R}^N \rightarrow \mathbb{R}^N$$

- We interpret the exact solution \mathbf{u} to be the roots of $\mathbf{F}(\mathbf{u})$
- Taylor expand the residuals at the $k + 1$ iterate about the k iterate

$$\mathbf{F}(\mathbf{u}^{k+1}) = \mathbf{F}(\mathbf{u}^k) + \mathbf{F}'(\mathbf{u}^k)(\mathbf{u}^{k+1} - \mathbf{u}^k) + \frac{\mathbf{F}''(\mathbf{u}^k)}{2}(\mathbf{u}^{k+1} - \mathbf{u}^k)^2 + \dots$$

- Assert \mathbf{u}^{k+1} is the exact solution

$$-\mathbf{F}(\mathbf{u}^k) = \mathbf{F}'(\mathbf{u}^k)(\mathbf{u}^{k+1} - \mathbf{u}^k)$$

- $\mathbf{F}'(\mathbf{u}^k)$ is the *Jacobian* $\mathbf{J}(\mathbf{u})$

$$J_{ij} = \frac{\partial F_i(\mathbf{u})}{\partial u_j}$$

- $(\mathbf{u}^{k+1} - \mathbf{u}^k)$ is the solution update from the k iterate to the $k + 1$ iterate

$$\delta \mathbf{u}^k = \mathbf{u}^{k+1} - \mathbf{u}^k$$

- Form Newton's method

$$\mathbf{J}(\mathbf{u})\delta \mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \delta \mathbf{u}^k$$

- A form of inexact Newton methods

$$\|\mathbf{J}(\mathbf{u}^k)\delta\mathbf{u}^k + \mathbf{F}(\mathbf{u}^k)\| \leq \eta^k \|\mathbf{F}(\mathbf{u}^k)\|, \quad \eta^k \in [0, 1)$$

- Choose a Krylov method to solve for the Newton correction
- GMRES with a long recurrence relation observed as more robust
- Generates a monotonically decreasing residual from maintaining the optimization and orthogonality conditions

- A form of inexact Newton methods

$$\| \mathbf{J}(\mathbf{u}^k) \delta \mathbf{u}^k + \mathbf{F}(\mathbf{u}^k) \| \leq \eta^k \| \mathbf{F}(\mathbf{u}^k) \|, \quad \eta^k \in [0, 1)$$

- Choose a Krylov method to solve for the Newton correction
- GMRES with a long recurrence relation observed as more robust
- Generates a monotonically decreasing residual from maintaining the optimization and orthogonality conditions

Where does the Jacobian come from?

- A form of inexact Newton methods

$$\| \mathbf{J}(\mathbf{u}^k) \delta \mathbf{u}^k + \mathbf{F}(\mathbf{u}^k) \| \leq \eta^k \| \mathbf{F}(\mathbf{u}^k) \|, \quad \eta^k \in [0, 1)$$

- Choose a Krylov method to solve for the Newton correction
- GMRES with a long recurrence relation observed as more robust
- Generates a monotonically decreasing residual from maintaining the optimization and orthogonality conditions

Where does the Jacobian come from?

- The Jacobian can come from hand-coded derivatives
 - Tedious and error prone
 - Repeated for each equation set and hard to do for multiphysics

- Krylov methods only need the action of the linear operator
- The action of the Jacobian can be approximated using a forward difference

$$\mathbf{J}(\mathbf{u})\mathbf{v} = \frac{\mathbf{F}(\mathbf{u} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})}{\epsilon}$$

- Forms the basis of Jacobian-Free Newton-Krylov (JFNK) methods
 - Sensitive to scaling and discretization error
 - Still have to form part of Jacobian periodically for preconditioning
 - Eventually break even with generating and storing the full Jacobian

- Automatically generate Jacobians from nonlinear function evaluations
 - Overload math operators and apply the chain rule (FAD)
 - Yields evaluations as accurate as function discretization
- Modern packages take an element-level assembly approach
 - Function evaluations are local, communication builds global data

$$\mathbf{J}(\mathbf{u}) = \sum_{i=1}^N \mathbf{Q}_i^T \mathbf{J}_k \mathbf{P}_i$$

$$e_k : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{m_k}, \mathbf{J}_{k_i} = \partial e_{k_i} / \partial P_i u, \mathbf{P} \in \mathbb{R}^{m_{k_i} \times N}$$

- Performance studies give acceptable results for use in large-scale, production physics codes



Forward-Automated Newton-MCSA

Forward-Automated Newton-MCSA

Algorithm 3 FANM

```
1:  $k := 0$ 
2: while  $\|\mathbf{F}(\mathbf{u}^k)\| > \epsilon \|\mathbf{F}(\mathbf{u}^0)\|$  do
3:    $\mathbf{J}(\mathbf{u}^k) \leftarrow AD(\mathbf{F}(\mathbf{u}^k))$  {Automatic differentiation}
4:    $\mathbf{J}(\mathbf{u}^k)\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$  {Solve for the Newton correction with MCSA}
5:    $\mathbf{u}^{k+1} \leftarrow \mathbf{u}^k + \delta\mathbf{u}^k$ 
6:    $k \leftarrow k + 1$ 
7: end while
```

Forward-Automated Newton-MCSA

Algorithm 4 FANM

```
1:  $k := 0$ 
2: while  $\|\mathbf{F}(\mathbf{u}^k)\| > \epsilon \|\mathbf{F}(\mathbf{u}^0)\|$  do
3:    $\mathbf{J}(\mathbf{u}^k) \leftarrow AD(\mathbf{F}(\mathbf{u}^k))$  {Automatic differentiation}
4:    $\mathbf{J}(\mathbf{u}^k)\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$  {Solve for the Newton correction with MCSA}
5:    $\mathbf{u}^{k+1} \leftarrow \mathbf{u}^k + \delta\mathbf{u}^k$ 
6:    $k \leftarrow k + 1$ 
7: end while
```

- Robustness of Newton's method (inexact)
- Accuracy and convenience of FAD
- Parallelism, memory, and resiliency benefits of MCSA
- Requires only nonlinear function evaluations



- Jacobian will be in a compressed row storage format

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 8 & 0 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 3 & 7 & 0 & 2 \\ 0 & 0 & 0 & 4 & 9 & 0 \\ 0 & 0 & 0 & 0 & 9 & 1 \end{bmatrix}$$

- Jacobian will be in a compressed row storage format

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 8 & 0 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 3 & 7 & 0 & 2 \\ 0 & 0 & 0 & 4 & 9 & 0 \\ 0 & 0 & 0 & 0 & 9 & 1 \end{bmatrix}$$

A values : 2 8 4 5 1 2 1 1 3 7 2 4 9 9 1
column : 1 3 1 2 4 2 3 5 3 4 6 4 5 5 6
row start : 1 3 6 9 12 14 16

- Jacobian will be in a compressed row storage format

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 8 & 0 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 3 & 7 & 0 & 2 \\ 0 & 0 & 0 & 4 & 9 & 0 \\ 0 & 0 & 0 & 0 & 9 & 1 \end{bmatrix}$$

A values : 2 8 4 5 1 2 1 1 3 7 2 4 9 9 1
column : 1 3 1 2 4 2 3 5 3 4 6 4 5 5 6
row start : 1 3 6 9 12 14 16

- m Krylov iterations require $(m + 1)$ subspace vectors
- Storage requirement is $[(m + 1)N]$ for $\mathbf{x} \in \mathbb{R}^{N \times N}$

- Jacobian will be in a compressed row storage format

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 8 & 0 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 3 & 7 & 0 & 2 \\ 0 & 0 & 0 & 4 & 9 & 0 \\ 0 & 0 & 0 & 0 & 9 & 1 \end{bmatrix}$$

A values : 2 8 4 5 1 2 1 1 3 7 2 4 9 9 1
column : 1 3 1 2 4 2 3 5 3 4 6 4 5 5 6
row start : 1 3 6 9 12 14 16

- m Krylov iterations require $(m + 1)$ subspace vectors
- Storage requirement is $[(m + 1)N]$ for $\mathbf{x} \in \mathbb{R}^{N \times N}$

If we need 10 Krylov iterations to converge...

- 66 elements required for subspace vectors
- 72 total elements required for Jacobian and probability matrix

















Proposed Challenge Problem



