Massively Parallel Monte Carlo Methods for Discrete Linear and Nonlinear Systems

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

March 24, 2013

Introduction

- 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

Physics-Based Motivation

Predictive nuclear reactor analysis enables... (DOE,2011)

- 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 (Evans,2010)(Pawlowski,2012)

Hardware-Based Motivation

- Modern hardware is moving in two directions (Kogge,2011):
  - Lightweight machines
  - Heterogeneous machines
  - Both characterized by low power and high concurrency
- Some issues:
  - Higher potential for both soft and hard failures (DOE,2012)
  - 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

Research Outline

- 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 for discrete systems leveraging Monte Carlo
  - Application to nonlinear problems of interest
  - Memory benefits
  - Performance benefits

Projection Methods

- Powerful class of iterative methods (Saad,2003)
- Provides theory that encapsulates most other iterative methods
- Leveraged in many modern physics codes at the petascale

Search Subspace $\mathcal{K}$

Extract the solution to $\mathbf{Ax} = \mathbf{b}$ from the search subspace:

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

Constraint Subspace $\mathcal{L}$

Constrain the extraction with the constraint subspace by asserting orthogonality with the residual ($\mathbf{r} = \mathbf{b} - \mathbf{Ax}$):

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

The Orthogonality Constraint

$$\tilde{\mathbf{r}} = \mathbf{r}_0 - \mathbf{A}\delta$$

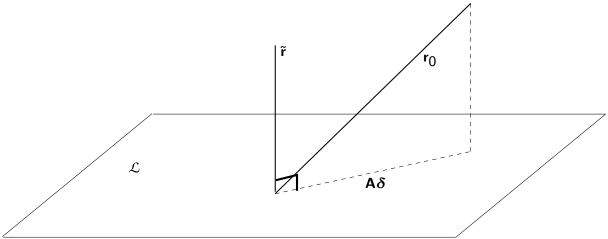


Figure: **Orthogonality constraint of the new residual with respect to $\mathcal{L}$.**

Minimization Property

The residual of the system is *minimized* with respect to the constraints

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

Krylov Subspace Methods

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

- For GMRES (Saad, 1986): $\mathcal{L} = \mathbf{A}\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0)$

- Yields the normal system $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$
- Require only the action of the operator
- Must generate an orthonormal basis for $\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0)$
  - Typically choose a Gram-Schmidt-like procedure such as Arnoldi or Lanzcos
  - Short and long recurrence relations available for orthogonalization

Parallel Projection Methods

- Parallel vector update

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

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

- Parallel dot product

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

- Parallel vector norm

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

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

Parallel Matrix-Vector Multiplication

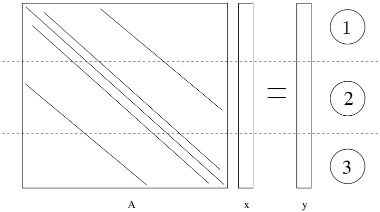


Figure: **Matrix-vector multiply Ax = y operation on 3 processors.**

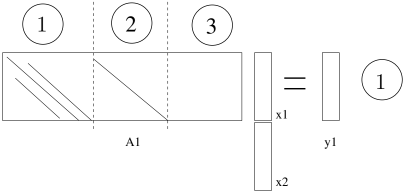


Figure: **Components of multiply operation owned by process 1.**

Projection Methods Scaling and Summary

- Global reduction operations observed not to impede scalability (Gropp,2001)
    - Dot product
    - Vector norms
- Nearest neighbor computations have poor algorithmic strong scaling
    - Matrix-vector multiply
    - Weak scaling is better
- Widely used in practice
- Krylov methods require only the action of the operator
- Parallelism achieved through a handful of operations
- Short and long recurrence relations available for orthogonalization

Monte Carlo Methods for Discrete Linear Systems

- 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

Monte Carlo Linear Solver Preliminaries

- Split the linear 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

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

Monte Carlo Linear Solver Preliminaries

- Expand the Neumann 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

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

- Define the *Neumann-Ulam decomposition*[1]

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

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

Direct Method

- Compute row-normalized transition probabilities and weights

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

- Generate an expectation value for the solution

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

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

Direct Method

- 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

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

Direct Method: Evolution of a Solution

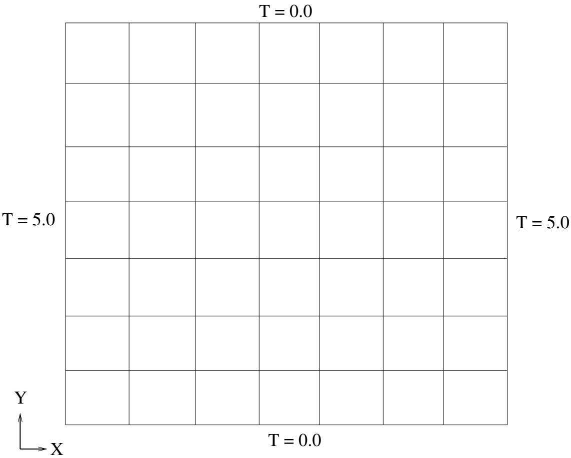


Figure: **Problem setup for 2D heat equation.** *Dirichlet conditions are set for the temperature on all 4 boundaries of the Cartesian grid. Background source of 1.0 present.* 50×50 *grid.*

Direct Method: Evolution of a Solution

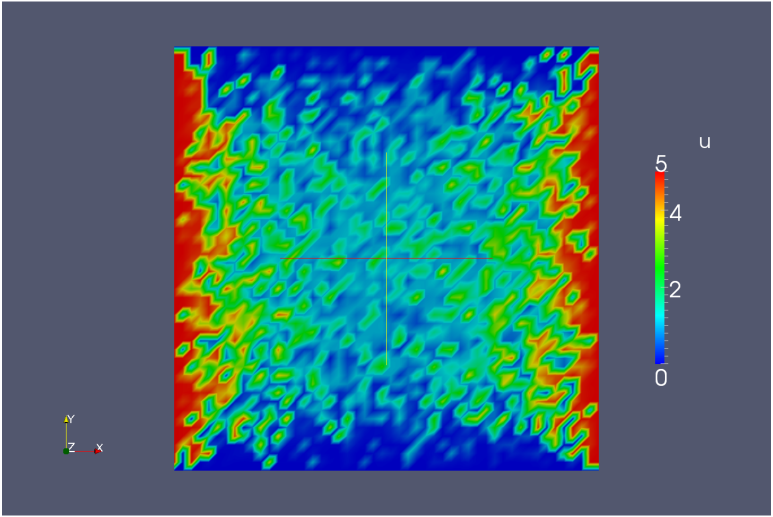


Figure: **Direct solution to Poisson Equation.** *1 history per state,* 2.5×10^3 *total histories. 0.785 seconds CPU time.*

Direct Method: Evolution of a Solution

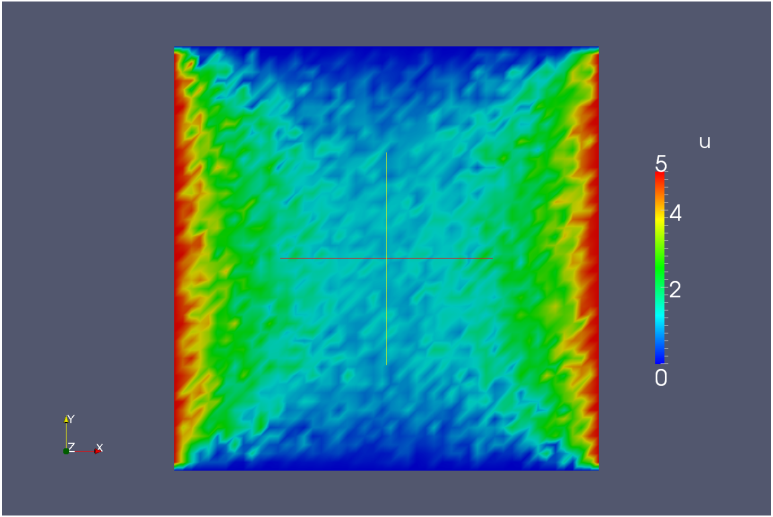


Figure: **Direct solution to Poisson Equation.** *10 histories per state,* 2.5×10^4 *total histories. 5.9 seconds CPU time.*

Direct Method: Evolution of a Solution

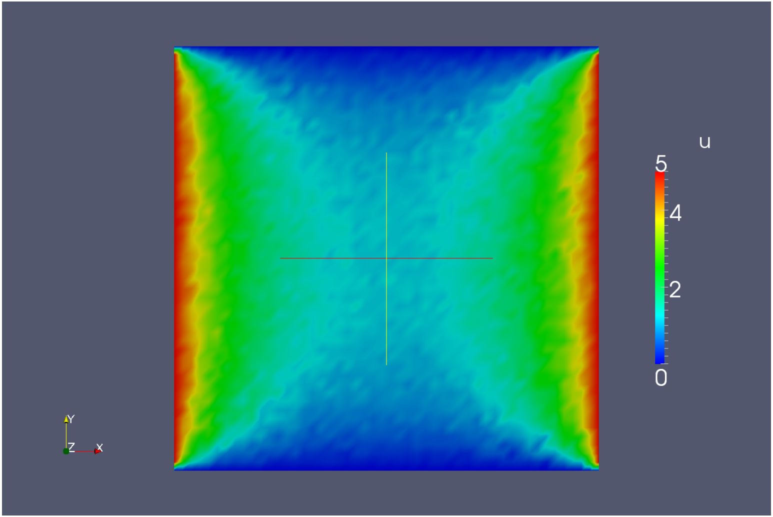


Figure: **Direct solution to Poisson Equation.** *100 histories per state,* 2.5×10^5 *total histories. 54.7 seconds CPU time.*

Direct Method: Evolution of a Solution

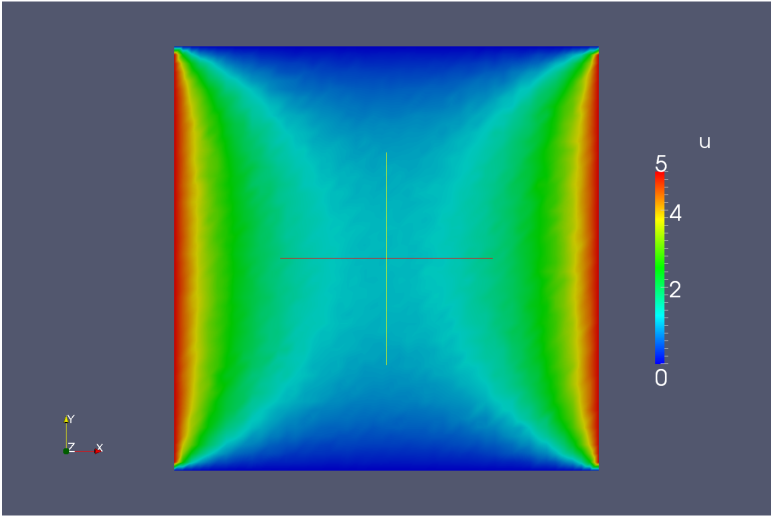


Figure: **Direct solution to Poisson Equation.** *1000 histories per state,* 2.5×10^6 *total histories. 644 seconds CPU time.*

Adjoint Method

- 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

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

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

Adjoint Method

- Generate the Neumann series for the adjoint operator

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

- Expand the series

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

- Pick another constraint to yield the original solution

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

Adjoint Method

- Use the adjoint Neumann-Ulam decomposition

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

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

- Build the estimator and expectation value

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

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

Adjoint Method: Evolution of a Solution

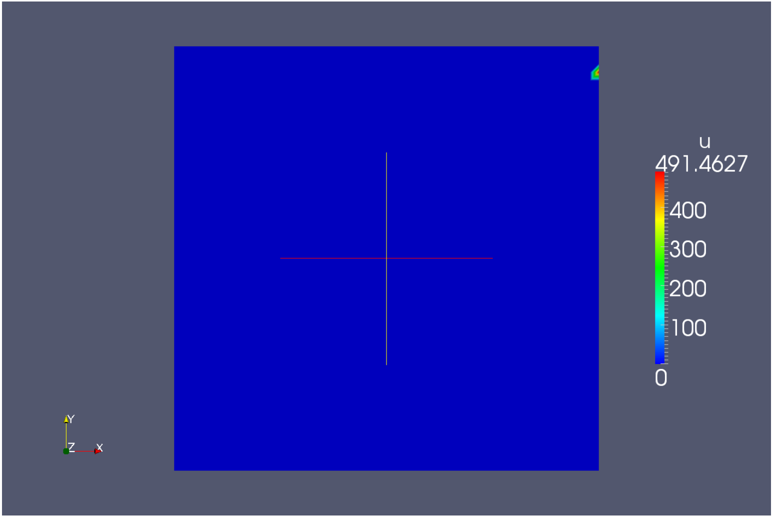


Figure: **Adjoint solution to Poisson Equation.** 1×10^0 *total histories, 0.286 seconds CPU time.*

Adjoint Method: Evolution of a Solution

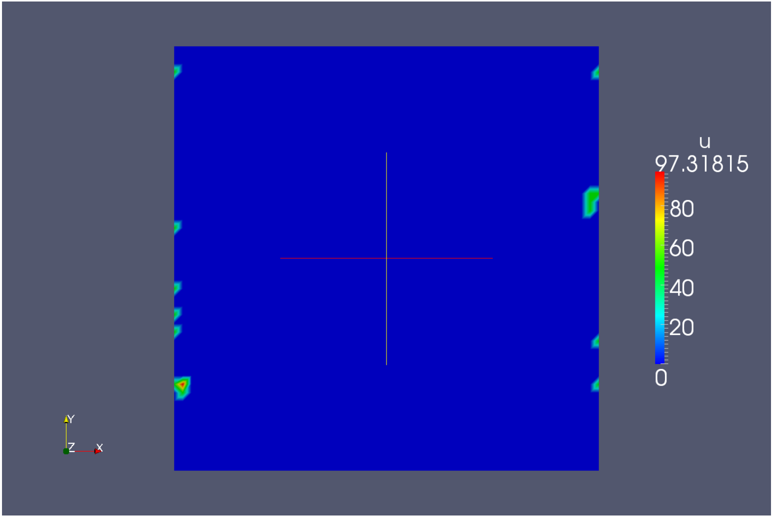


Figure: **Adjoint solution to Poisson Equation.** 1×10^1 *total histories, 0.278 seconds CPU time.*

Adjoint Method: Evolution of a Solution

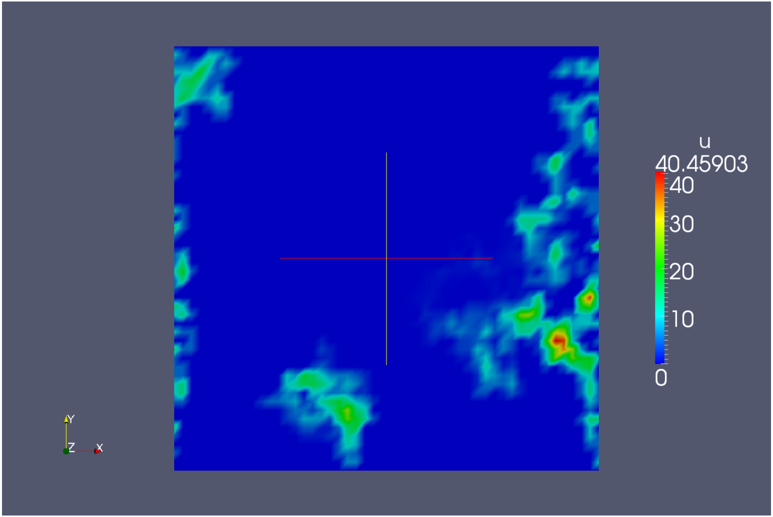


Figure: **Adjoint solution to Poisson Equation.** 1×10^2 *total histories, 0.275 seconds CPU time.*

Adjoint Method: Evolution of a Solution

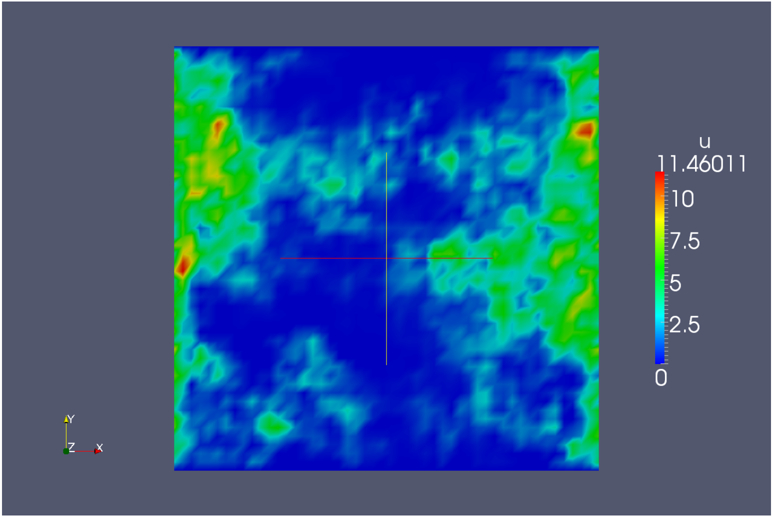


Figure: **Adjoint solution to Poisson Equation.** 1×10^3 *total histories, 0.291 seconds CPU time.*

Adjoint Method: Evolution of a Solution

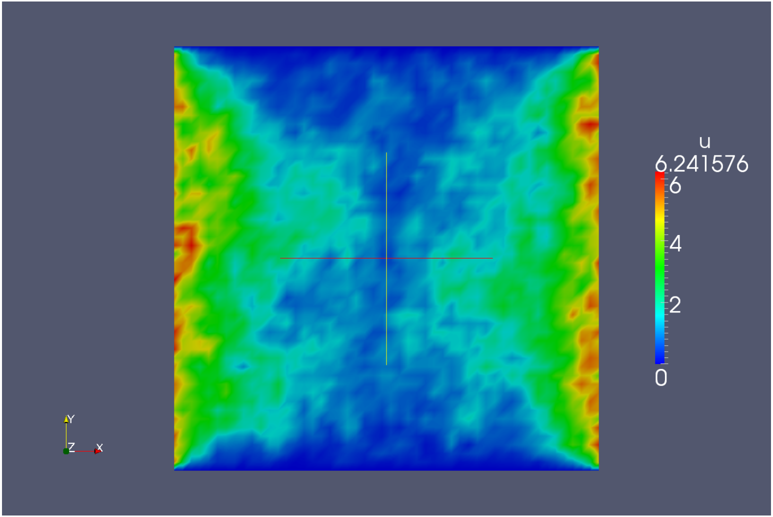


Figure: **Adjoint solution to Poisson Equation.** 1×10^4 *total histories, 0.428 seconds CPU time.*

Adjoint Method: Evolution of a Solution

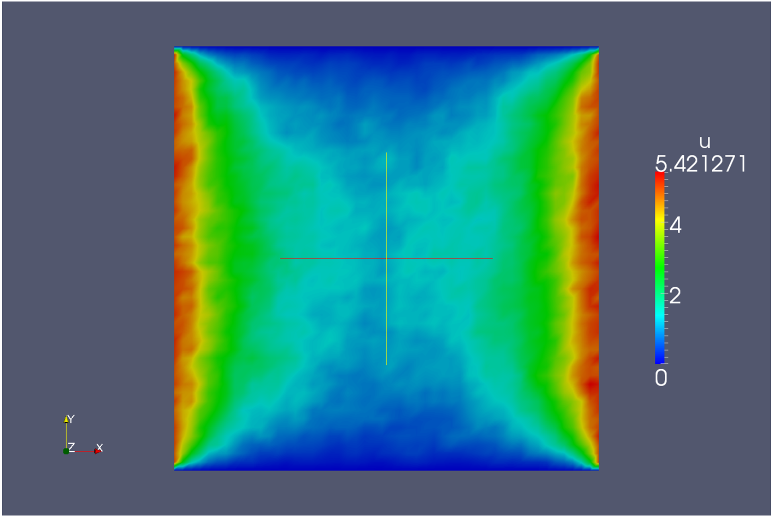


Figure: **Adjoint solution to Poisson Equation.** 1×10^5 *total histories, 1.76 seconds CPU time.*

Adjoint Method: Evolution of a Solution

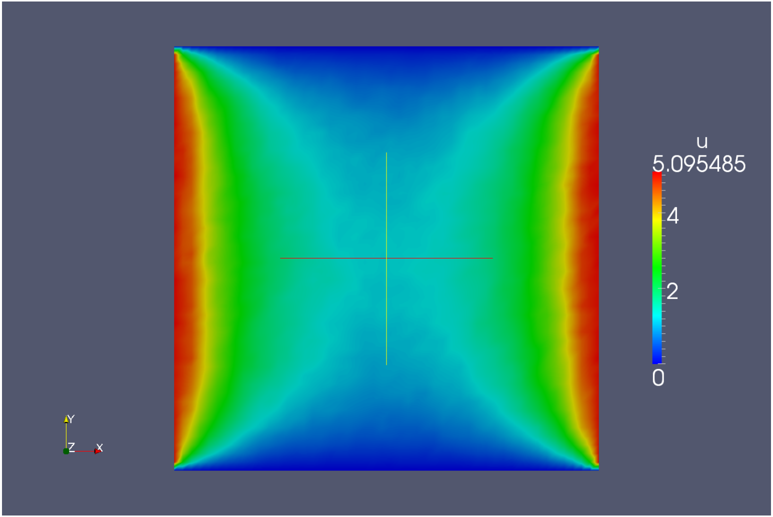


Figure: **Adjoint solution to Poisson Equation.** 1×10^6 *total histories, 15.1 seconds CPU time.*

Adjoint Method: Evolution of a Solution

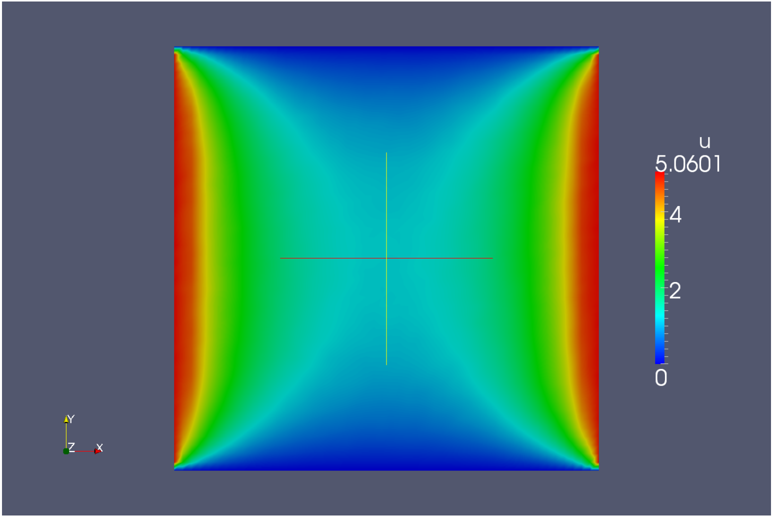


Figure: **Adjoint solution to Poisson Equation.** 1×10^7 *total histories, 149 seconds CPU time.*

Monte Carlo Synthetic-Acceleration

MCSA Iteration (Evans,2009)

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

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

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

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

- Neumann-Ulam methods bound by the Central Limit Theorem
- Build on Halton's 1962 Sequential Monte Carlo method
- Adjoint Neumann-Ulam solver computes the correction
- Decouples MC error from solution error, exponential convergence
- Demonstrated by Evans to be competitive with Krylov methods

Preconditioning Monte Carlo Methods

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

$$\mathbf{M} = diag(\mathbf{A})$$
$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$$

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

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

Direct vs. Adjoint Analysis

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

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

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

- Implicit Euler time differencing

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

Direct vs. Adjoint Analysis

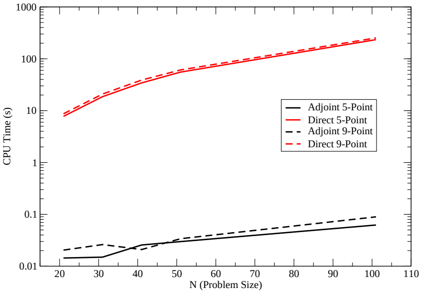


Figure: **CPU Time (s) to converge vs. Problem Size (N for an $N \times N$ square mesh).**

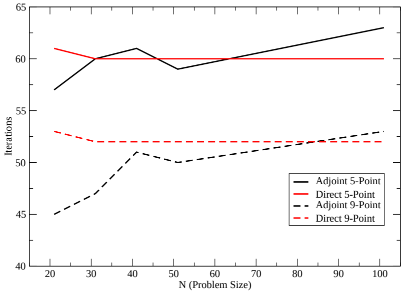


Figure: **Iterations to converge vs. Problem Size (N for an $N \times N$ square mesh).**

Direct vs. Adjoint Analysis

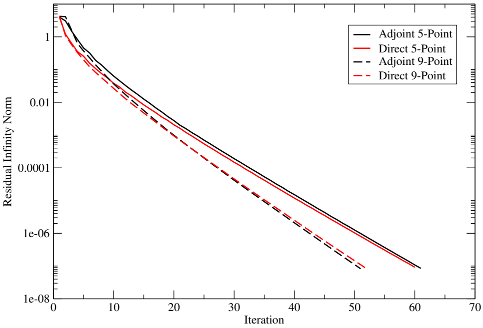


Figure: **Infinity norm of the solution residual vs. iteration number for a problem of fixed size.**

- 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

Generalization of MCSA for Linear Problems

- Published work to date has used a physics-based formulation
    - Radiation transport equations used as model system
    - Transition probabilities built from problem-specific parameters
    - Probabilities and weights must be re-derived for each new equation set
- Desire a generalization for all linear operator equations
    - Requires a general parallel framework
    - Requires implementation with a general linear algebra framework
    - Operator, vector, and graph abstractions
- Neumann-Ulam solvers and MCSA implemented using the Trilinos Petra frameworks
- Can be leveraged in modern physics implementations

Parallelization of Monte Carlo Methods

- No literature observed for parallel Neumann-Ulam solvers
- Numerous references for modern parallel Monte Carlo methods in reactor physics
- Build a strategy for applying modern methods to the Neumann-Ulam method
- MCSA iteration-level parallelism comes from parallel matrix/vector operations

Domain Decomposition

- Each parallel process owns a piece of the domain
- Random walks must be transported across domains through communication

Brunner's Work (2006 and 2009)

- Looked at 4 communication patterns:
  - Fully-locking synchronous
  - Asynchronous-send/synchronous-receive
  - Master/slave
  - Binary tree master/slave
- Binary tree master/slave performed best but race conditions observed
- A later improvement to their work showed a fully asynchronous pattern to work best
- Poor scaling for unbalanced problems for all schemes

Multiple-Set Overlapping-Domain Decomposition

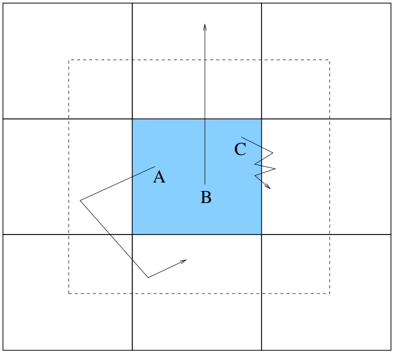


Figure: **Overlapping domain example illustrating how domain overlap can reduce communication costs.**

- Developed by Wagner and colleagues in 2010
- Each set contains the full domain
- Multiple sets replicate the domain
- Domains overlap within a set
- Reduces communication by a significant fraction
- Increases scalability through smaller processor sets
- Redundancy for resiliency (and useful work)

Domain-to-Domain Communication

- The amount of domain leakage dictates communication
- Per Siegel's 2012 work, define a leakage fraction

$$\lambda = \frac{average\ \#\ of\ particles\ leaving\ local\ domain}{total\ of\ \#\ of\ particles\ starting\ in\ local\ domain}$$

- Siegel observed λ to be bound empirically to the mean free path of the system
- Optimum ratio of local to global domain size exists
- Experiments show feasibility for large-scale calculations
  - Not bandwidth or latency limited in load-balanced case

Domain-to-Domain Communication

- The amount of domain leakage dictates communication
- Per Siegel's 2012 work, define a leakage fraction

$$\lambda = \frac{average \ \# \ of \ particles \ leaving \ local \ domain}{total \ of \ \# \ of \ particles \ starting \ in \ local \ domain}$$

- Siegel observed λ to be bound empirically to the mean free path of the system
- Optimum ratio of local to global domain size exists
- Experiments show feasibility for large-scale calculations
  - Not bandwidth or latency limited in load-balanced case
- How do we define λ empirically for linear operator equations?

Load Balancing

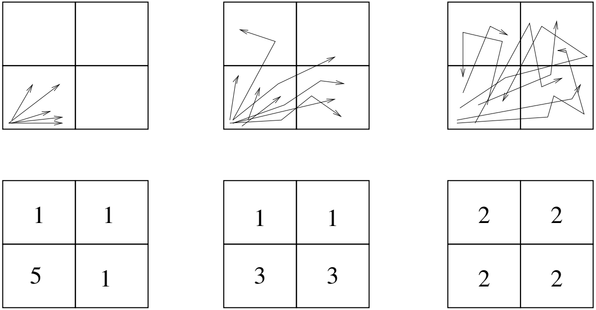


Figure: **Example illustrating how domain decomposition can create load balance issues in Monte Carlo.**

- Procassini's 2005 work addressed some concerns
- Dynamic balancing replicates domains independently
- Linear operator equations may benefit - fixed source problems

Parallel Adjoint Method

Strategy

- Direct analogs between particle transport and Neumann-Ulam solvers
- Aim for MSOD implementation and fully asynchronous communication patterns

Questions

- How much domain overlap is suitable for linear operator equations?
- Is memory a limitation for overlap and replication?
- Do the linear operator properties help select overlap?
- Will full-clip roulette perturb the MCSA solution?
- How much does MSOD facilitate scaling?

Parallel MCSA

MCSA Iteration

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

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

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

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

- Richardson iteration and residual computation require parallel matrix-vector multiply and parallel vector update
- This work will generate a parallel adjoint Neumann-Ulam solver
- Application of correction requires parallel vector update
- Convergence checks through parallel norm computation

Monte Carlo Solution Methods for Nonlinear Problems

- Many multiphysics problems of interest are nonlinear
- Segregated methods lack the consistency of fully implicit methods
- Newton methods often leverage Krylov solvers
  - Robust implementations
  - No operator required
  - Memory intensive
- Monte Carlo methods need the full operator
- Automatic construction of the linear operator available
  - Ideal for Monte Carlo
  - Relaxes memory requirements
  - Potential scaling improvements
  - Resiliency benefits

Nonlinear Preliminaries

- We seek solutions of the general nonlinear problem

$$\mathbf{F}(\mathbf{u}) = \mathbf{0}$$
$$\mathbf{u} \in \mathbb{R}^n,\ \mathbf{F} : \mathbb{R}^N \to \mathbb{R}^N$$

- We interpret the exact solution $\mathbf{u}$ to be the roots of $\mathbf{F}(\mathbf{u})$

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

- Form Newton's method

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

Newton-Krylov Methods

- Choose a Krylov method to solve for the Newton correction
- GMRES with a long recurrence relation observed as more robust (Knoll,2004)
- Generates a monotonically decreasing residual from maintaining the optimization and orthogonality conditions

Newton-Krylov Methods

- Choose a Krylov method to solve for the Newton correction
- GMRES with a long recurrence relation observed as more robust (Knoll,2004)
- 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

Jacobian Approximations

Matrix-Free Approximation

- Krylov methods only need the action of the linear operator

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

- Forms the basis of Jacobian-Free Newton-Krylov (JFNK) methods
  - Sensitive to scaling and discretization error (Kelly,1995)
  - Eventually break even with generating and storing the full Jacobian (Knoll,1994)

Automatic Differentiation

- Automatically generate Jacobians from nonlinear function evaluations
  - Overload math operators and apply the chain rule (FAD)
  - Yields evaluations as accurate as function discretization
- Performance studies give acceptable results for use in large-scale, production physics codes (Bartlett,2006)

Jacobian Storage vs. Subspace Storage

- Jacobian will be in a compressed row storage format

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

$$\begin{array}{lccccccccccccccc} \mathbf{A}\ \textit{values}: & 2 & 8 & 4 & 5 & 1 & 2 & 1 & 1 & 3 & 7 & 2 & 4 & 9 & 9 & 1 \\ \textit{column}: & 1 & 3 & 1 & 2 & 4 & 2 & 3 & 5 & 3 & 4 & 6 & 4 & 5 & 5 & 6 \\ \textit{row start}: & 1 & 3 & 6 & 9 & 12 & 14 & 16 \end{array}$$

- CRS matrix with q bands storage: $\left[(2q+1)N\right]$ for $\mathbf{x} \in \mathbb{R}^{N\times N}$

Jacobian Storage vs. Subspace Storage

- Jacobian will be in a compressed row storage format

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

$\mathbf{A}$ *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								

- CRS matrix with q bands storage: $\left[(2q+1)N\right]$ for $\mathbf{x} \in \mathbb{R}^{N\times N}$
- m Krylov iterations require $(m+1)$ subspace vectors
- Subspace storage: $\left[(m+1)N\right]$ for $\mathbf{x} \in \mathbb{R}^{N\times N}$

Jacobian Storage vs. Subspace Storage

- Jacobian will be in a compressed row storage format

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

$$\begin{array}{lccccccccccccccc} \mathbf{A}\ \textit{values}: & 2 & 8 & 4 & 5 & 1 & 2 & 1 & 1 & 3 & 7 & 2 & 4 & 9 & 9 & 1 \\ \textit{column}: & 1 & 3 & 1 & 2 & 4 & 2 & 3 & 5 & 3 & 4 & 6 & 4 & 5 & 5 & 6 \\ \textit{row start}: & 1 & 3 & 6 & 9 & 12 & 14 & 16 \end{array}$$

- CRS matrix with q bands storage: $\left[(2q+1)N\right]$ for $\mathbf{x} \in \mathbb{R}^{N\times N}$
- m Krylov iterations require $(m+1)$ subspace vectors
- Subspace storage: $\left[(m+1)N\right]$ for $\mathbf{x} \in \mathbb{R}^{N\times N}$

If we need 25 Krylov iterations to converge...

- Break-even scenario: $4q+1 = m$
- Jacobian and probability matrix storage with 6 bands or less is cheaper

The FANM Method

Forward-Automated Newton-MCSA

Algorithm 1 FANM

$k := 0$
while $||\mathbf{F}(\mathbf{u}^k)|| > \epsilon ||\mathbf{F}(\mathbf{u}^0)||$ **do**
    $\mathbf{J}(\mathbf{u}^k) \leftarrow AD(\mathbf{F}(\mathbf{u}^k))$ {Automatic differentiation}
    $\mathbf{J}(\mathbf{u}^k)\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k)$ {Solve for the Newton correction with MCSA}
    $\mathbf{u}^{k+1} \leftarrow \mathbf{u}^k + \delta\mathbf{u}^k$
    $k \leftarrow k + 1$
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

Parallel FANM Method

Algorithm 2 FANM

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

- Modern FAD packages are parallelized using element-level assembly
- This work will generate a parallel MCSA solver
- Application of the Newton correction requires a parallel vector update
- Convergence checks through parallel norm computation

Research Proposal

- Methods verification
- Numerical experiments
- Challenge problem

Monte Carlo Methods Verification

- Analytic solution to the heat equation for the linear methods
- Sequence of Navier-Stokes benchmarks for the nonlinear methods
  - Thermal convection cavity problem (De Vahl Davis, 1983)
  - Lid driven cavity problem (Ghia et al., 1982)
  - Backward-Facing step problem (Gartling, 1990)
- Tuning benchmark parameters varies the strength of nonlinearities

$$\rho \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot \mathbf{T} - \rho \mathbf{g} = \mathbf{0}$$

$$\nabla \cdot \mathbf{u} = 0$$

$$\rho C_p \mathbf{u} \cdot \nabla T + \nabla \cdot \mathbf{q} = 0$$

$$\mathbf{T} = -P\mathbf{I} + \mu[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$$

$$\mathbf{q} = -k \nabla T$$

Proposed Numerical Experiments

Proposed Numerical Experiments

Domain Overlap Studies for Parallel Neumann-Ulam Method

- Correlate domain overlap behavior to linear system properties
- Analyze random walk transport with respect to the operator:
  - Eigenvalues
  - Sparsity
  - Asymmetry
- Evaluate communication cost
- Evaluate memory cost

Proposed Numerical Experiments

Domain Overlap Studies for Parallel Neumann-Ulam Method

- Correlate domain overlap behavior to linear system properties
- Analyze random walk transport with respect to the operator:
  - Eigenvalues
  - Sparsity
  - Asymmetry
- Evaluate communication cost
- Evaluate memory cost

Domain Replication Studies for Parallel Neumann-Ulam Method

- Feasibility from a memory perspective
- Impact on scalability
- Impact on load balancing

Proposed Numerical Experiments

Parallel Performance and Numerical Accuracy Studies for MCSA Method

- Characterize Neumann-Ulam parameters required for good correction
- Feasibility of MCSA with full-clip Neumann-Ulam
- Performance for asymmetric systems
- Memory usage compared to Krylov methods

Proposed Numerical Experiments

Parallel Performance and Numerical Accuracy Studies for MCSA Method

- Characterize Neumann-Ulam parameters required for good correction
- Feasibility of MCSA with full-clip Neumann-Ulam
- Performance for asymmetric systems
- Memory usage compared to Krylov methods

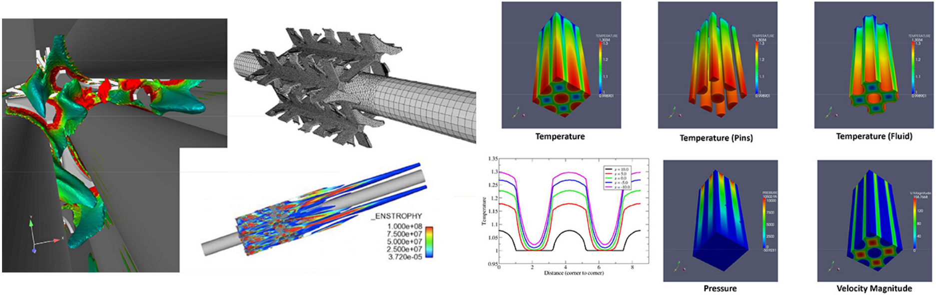
Parallel Performance and Numerical Accuracy Studies for FANM Method

- Feasibility for problems of interest
- Memory usage vs. Newton-Krylov methods
- Scalability vs. Newton-Krylov methods

Proposed Challenge Problem

- Problems of interest are coupled large scale problems in reactor physics
- The Consortium for Advanced Simulation of LWRs (CASL) is a modeling and simulation program aimed at:
  - Higher power uprates and efficiency
  - Higher burn-up
  - Predictive accident scenario analysis
- CASL and industry partners identified challenge problems:
  - Departure from nucleate boiling
  - Grid-to-rod-fretting
- Our challenge problem should reflect how this work aids the solution of these problems

Proposed Challenge Problem



- CASL has utilized the Drekar multiphysics code (SNL)
- Coupled fluid flow and heat transfer helps characterize many phenomena
- Drekar is massively parallel and leverages Newton-Krylov methods
- Propose using the largest Drekar problem to date as a challenge problem for the new Monte Carlo methods

Images source: www.casl.gov

Conclusion

- Proposed research and development of new Monte Carlo methods
  - Parallelization of Monte Carlo methods for linear systems
  - FANM for nonlinear systems
- Verification through benchmarks
- Numerical experiments for understanding
- Challenge problem for application
- Directed towards hard problems in nuclear reactor analysis:
  - Application to multiphysics
  - Potential improvements for scalability
  - Potential improvements for memory consumption
  - Looking forward to exascale

Acknowledgments

Thanks to...

- Paul Wilson
- Tom Evans
- ORNL and SNL

Support

This work was performed under appointment to the Nuclear Regulatory Commission Fellowship program at the University of Wisconsin - Madison Department of Engineering Physics.