SAND2008-1743C

Analytic Sensitivities in Large-scale Production Applications via Automatic Differentiation with Sacado

Eric Phipps
Roscoe Bartlett, David Gay
Optimization & Uncertainty Quantification Department
Sandia National Laboratories
Albuquerque, NM USA

Parallel Processing for Scientific Computing
March 13, 2008

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.

Analytic Derivatives Enable Robust Simulation and Design Capabilities

- We need analytic first & higher derivatives for predictive simulations
  - Computational design, optimization and parameter estimation
  - Stability analysis
  - Uncertainty quantification
  - Verification and validation
- Analytic derivatives improve robustness and efficiency
  - Very hard to make finite differences accurate
- Infeasible to expect application developers to code analytic derivatives
  - Time consuming, error prone, and difficult to verify
  - Thousands of possible parameters in a large code
  - Developers must understand what derivatives are needed
- Automatic differentiation solves these problems

What is Automatic Differentiation (AD)?

- Technique to compute analytic derivatives without hand-coding the derivative computation
- How does it work -- freshman calculus
  - Computations are composition of simple operations (+, *, sin(), etc…) with known derivatives
  - Derivatives computed line-by-line, combined via chain rule
- Derivatives accurate as original computation
  - No finite-difference truncation errors
- Provides analytic derivatives without the time and effort of hand-coding them

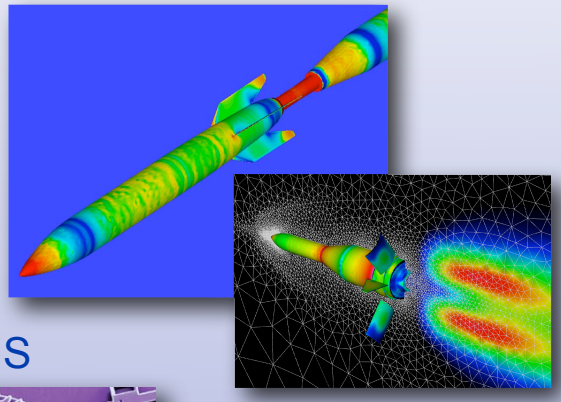
$$y = \sin(e^x + x \log x), \quad x = 2$$

		x	$\frac{d}{dx}$
$x \leftarrow 2$	$\frac{dx}{dx} \leftarrow 1$	2.000	1.000
$t \leftarrow e^x$	$\frac{dt}{dx} \leftarrow t\frac{dx}{dx}$	7.389	7.389
$u \leftarrow \log x$	$\frac{du}{dx} \leftarrow \frac{1}{x}\frac{dx}{dx}$	0.301	0.500
$v \leftarrow xu$	$\frac{dv}{dx} \leftarrow u\frac{dx}{dx} + x\frac{du}{dx}$	0.602	1.301
$w \leftarrow t + v$	$\frac{dw}{dx} \leftarrow \frac{dt}{dx} + \frac{dv}{dx}$	7.991	8.690
$y \leftarrow \sin w$	$\frac{dy}{dx} \leftarrow \cos(w)\frac{dw}{dx}$	0.991	-1.188

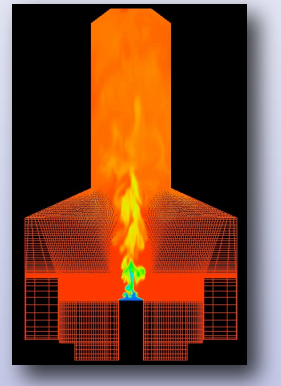
Sandia Physics Simulation Codes

- Element-based
  - Finite element, finite volume, finite difference, network, etc…
- Large-scale
  - Billions of unknowns
- Parallel
  - MPI-based SPMD
  - Distributed memory
- C++
  - Object oriented
  - Some coupling to legacy Fortran libraries
- We need AD techniques that fit these requirements

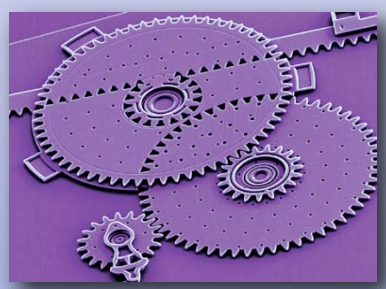
Fluids



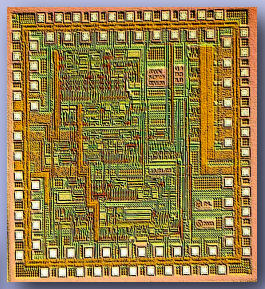
Combustion



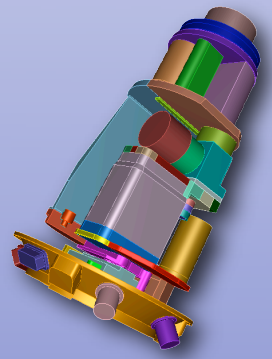
MEMS



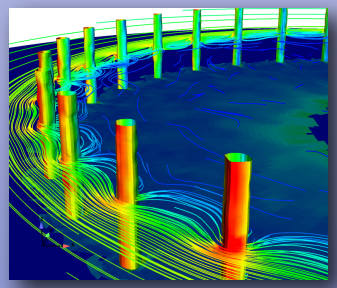
Circuits



Structures



Plasmas



Sandia National Laboratories

Automatic Differentiation Projects

- Many AD projects around the world, e.g.,
  - ADIFOR/ADIC (ANL, Rice) -- Fortran 77, simple C
  - OpenAD (ANL, Rice, Aachen) -- Fortran 77/95, C/C++
  - ADOL-C (TU-Desden) -- C/C++
  - TFAD<> -- C/C++
- Most source transformation tools limited to Fortran
- Most operator overloading based tools are slow
  - TFAD<> shows how to do this efficiently
- Many AD projects are geared towards “black-box” solutions
- We need efficient OO tools optimized for Sandia’s large-scale, parallel, C++ applications

Sacado: AD Tools for C++ Codes

- Trilinos package: *www.trilinos.sandia.gov*
- Sacado provides several modes of Automatic Differentiation (AD)
  - Forward (Jacobians, Jacobian-vector products, …)
  - Reverse (Gradients, Jacobian-transpose-vector products, …)
  - Taylor (High-order univariate Taylor series)
- Sacado implements AD via operator overloading and C++ templating
  - Expression templates for OO efficiency
  - Application code templating for easy incorporation
- Designed for use in large-scale C++ codes
  - Apply AD at "element-level"
  - Very successful in Charon application code
  - `Sacado::FEApp` example demonstrates approach
- Sacado provides other useful utilities
  - Scalar flop counting
  - Scalar parameter library
  - Template utilities

Simple Sacado Example

```
// The function to differentiate

double func(double a, double b, double c) {
  double r = c*std::log(b+1.)/std::sin(a);

  return r;
}

int main(int argc, char **argv) {
  double a = std::atan(1.0);                    // pi/4
  double b = 2.0;
  double c = 3.0;









  // Compute function
  double r = func(a, b, c);
```

Simple Sacado Example

```
#include "Sacado.hpp"

// The function to differentiate
template <typename ScalarT>
ScalarT func(const ScalarT& a, const ScalarT& b, const ScalarT& c) {
  ScalarT r = c*std::log(b+1.)/std::sin(a);

  return r;
}

int main(int argc, char **argv) {
  double a = std::atan(1.0);                        // pi/4
  double b = 2.0;
  double c = 3.0;

  // Fad objects
  int num_deriv = 2;                                // Number of independent variables
  Sacado::Fad::DFad<double> afad(num_deriv, 0, a); // First (0) indep. var
  Sacado::Fad::DFad<double> bfad(num_deriv, 1, b); // Second (1) indep. var
  Sacado::Fad::DFad<double> cfad(c);                // Passive variable

  // Compute function
  double r = func(a, b, c);

  // Compute function and derivative with AD
  Sacado::Fad::DFad<double> rfad = func(afad, bfad, cfad);

  // Extract value and derivatives
  double r_ad = rfad.val();     // r
  double drda_ad = rfad.dx(0);  // dr/da
  double drdb_ad = rfad.dx(1);  // dr/db
```

Differentiating Element-Based Codes

- Global residual computation (ignoring boundary computations):

$$f(\dot{x}, x, t, p) = \sum_{i=1}^{N} Q_i^T e_{k_i}(P_i\dot{x}, P_i x, t, p)$$

- Time-step Jacobian computation:

$$\alpha\frac{\partial f}{\partial \dot{x}} + \beta\frac{\partial f}{\partial x} = \sum_{i=1}^{N} Q_i^T \left(\alpha\frac{\partial e_{k_i}}{\partial \dot{x}_i} + \beta\frac{\partial e_{k_i}}{\partial x_i}\right) P_i, \quad \dot{x}_i = P_i\dot{x}, \;\; x_i = P_i x$$

- Parameter derivative computation:

$$\frac{\partial f}{\partial p} = \sum_{i=1}^{N} Q_i^T \frac{\partial e_{k_i}}{\partial p}$$

- Hybrid symbolic/AD procedure
  - Element-level derivatives computed via AD
  - Exactly the same as how you would do this “manually”
  - Avoids parallelization issues

Difficulties

- Template code introduces excessive compiler overhead
  - Explicit template instantiation
  - Preprocessor macros make this easy
- Real codes always call other libraries
  - BLAS/LAPACK
  - CHEMKIN
  - Linear/nonlinear solvers
  - Template interfaces (partial template specialization) are general solution
- Operator overloading overhead
  - Residual/derivative fills are not dominant cost

Scalability of AD in Charon

Set of N hypothetical chemical species:

$$2X_j \rightleftharpoons X_{j-1} + X_{j+1}, \quad j = 2, \ldots, N-1$$

Steady-state mass transfer equations:

$$\mathbf{u} \cdot \nabla Y_j + \nabla^2 Y_j = \dot{\omega}_j, \quad j = 1, \ldots, N-1$$

$$\sum_{j=1}^{N} Y_j = 1$$

Forward mode AD

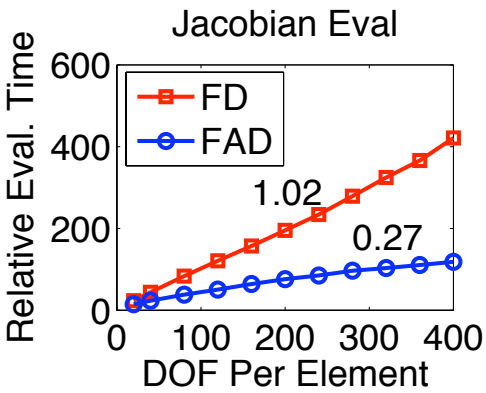
- ✓ Faster than FD
- ✓ Better scalability in number of PDEs
- ✓ Analytic Derivative

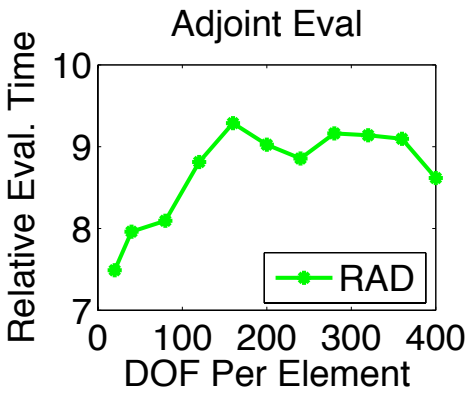
Reverse mode AD

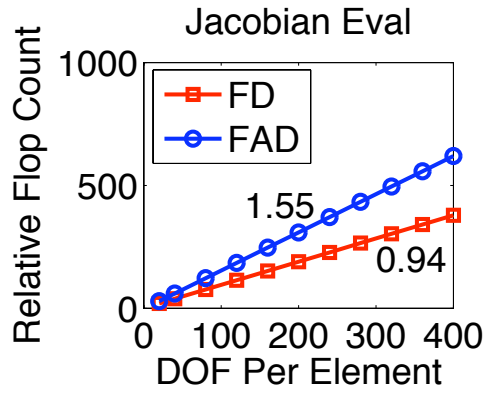
- ✓ Scalable adjoint/gradient

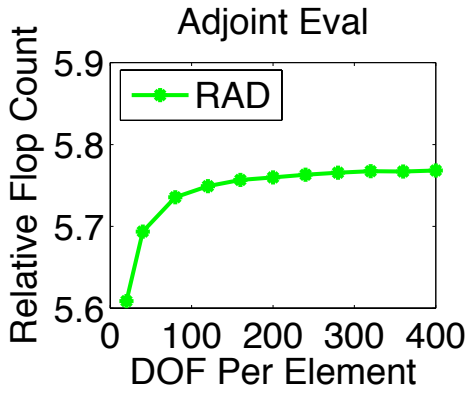
$$J^T w = \nabla(w^T f(x))$$

Scalability of the element-level derivative computation







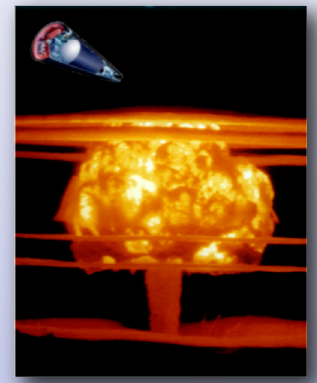


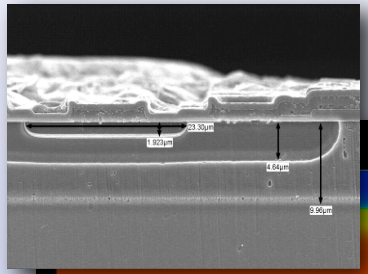
DOF per element = 4*N

Sandia National Laboratories

QASPR

Qualification of electronic devices in hostile environments



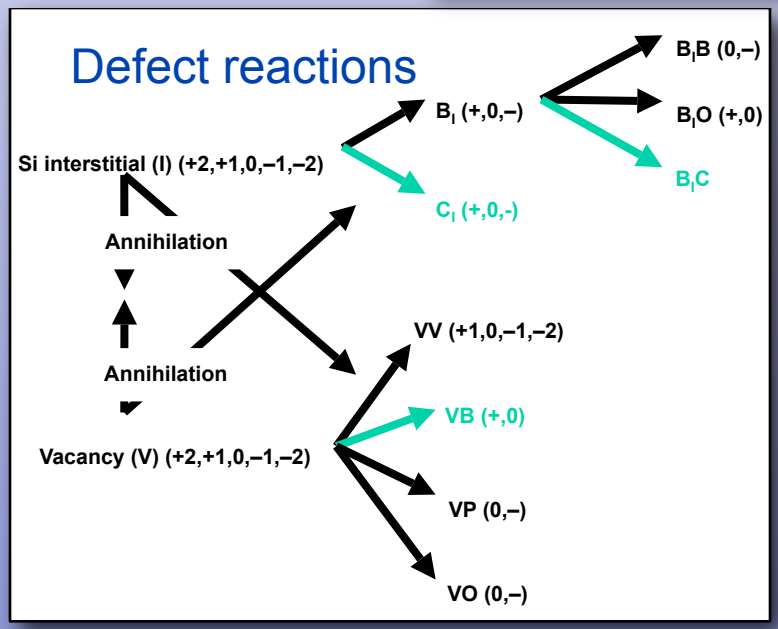


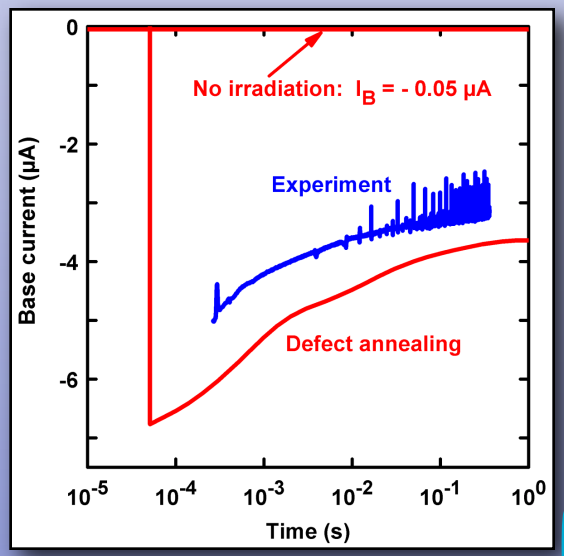
Stockpile BJT

CHARON

PDE semiconductor device simulation

Defect reactions





Sandia National Laboratories

Charon Drift-Diffusion Formulation with Defects

Current Conservation for e- and h+

$$\frac{\partial n}{\partial t} - \nabla \cdot J_n = -R_n(\psi, n, p, Y_1, \ldots, Y_N), \quad J_n = -n\mu_n \nabla\psi + D_n \nabla n$$

$$\frac{\partial p}{\partial t} + \nabla \cdot J_p = -R_p(\psi, n, p, Y_1, \ldots, Y_N), \quad J_p = -p\mu_p \nabla\psi - D_p \nabla p$$

Defect Continuity

$$\frac{\partial Y_i}{\partial t} + \nabla \cdot J_{Y_i} = -R_{Y_i}(\psi, n, p, Y_1, \ldots, Y_N), \quad J_{Y_i} = -\mu_i Y_i \nabla\psi - D_i \nabla Y_i$$

Electric potential

$$-\nabla(\varepsilon \nabla\psi(x)) = -q\left(p(x) - n(x) + N_D^+(x) - N_A^-(x)\right) - \sum_{i=1}^{N} q_i Y_i(x)$$

Recombination/ generation source terms

R_X

Include electron capture and hole capture by defect species and reactions between various defect species

Electron emission/capture

$Z^i \leftrightarrow Z^{i+1} + e^-$

$$R_{[Z^i \to Z^{i+1} + e^-]} \propto \sigma_{[Z^i \to Z^{i+1} + e^-]} Z^i \exp\left(\frac{\Delta E_{[Z^i \to Z^{i+1} + e^-]}}{kT}\right)$$

Activation Energy

Cross section

Sandia National Laboratories

Forward Sensitivity Analysis with Rythmos

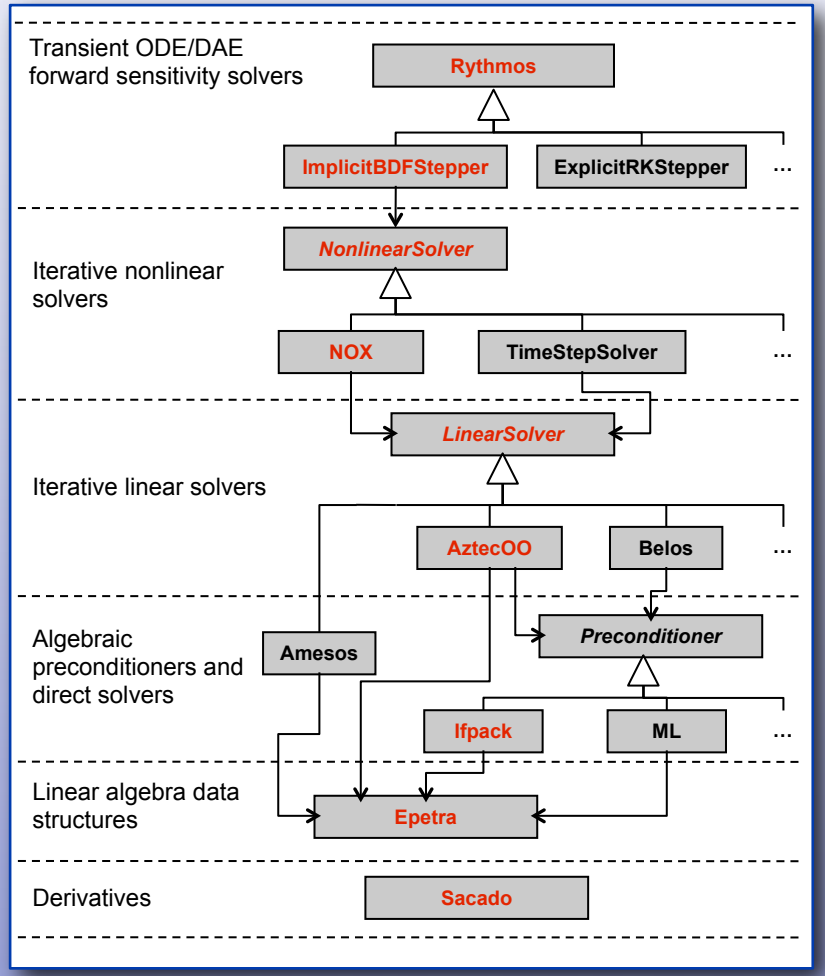
- Discretized PDE system:

$$f(\dot{x}, x, p, t) = 0$$
$$\hat{g}(p, t) = g(\dot{x}(t), x(t), p, t)$$

- Forward sensitivity problem

$$\frac{\partial f}{\partial \dot{x}}\left(\frac{\partial \dot{x}}{\partial p}\right) + \frac{\partial f}{\partial x}\left(\frac{\partial x}{\partial p}\right) + \frac{\partial f}{\partial p} = 0$$
$$\frac{\partial \hat{g}}{\partial p} = \frac{\partial g}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial g}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial g}{\partial p}$$

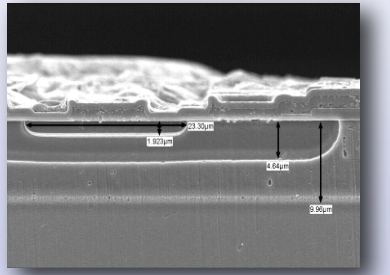
- Rythmos time integration package
  - Todd Coffey, Ross Bartlett (SNL)
  - Implicit BDF time integration method
  - Variable order, step size
  - Staggered corrector forward sensitivity method

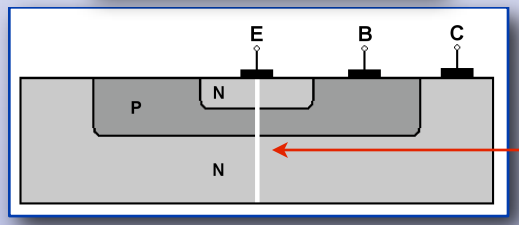


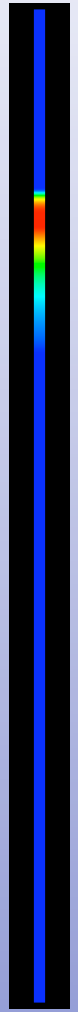
Sandia National Laboratories

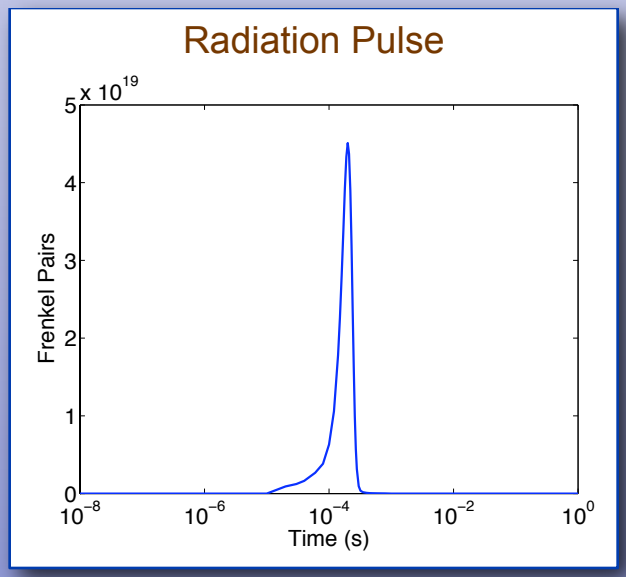
Sensitivity Analysis of a Pseudo-1D BJT

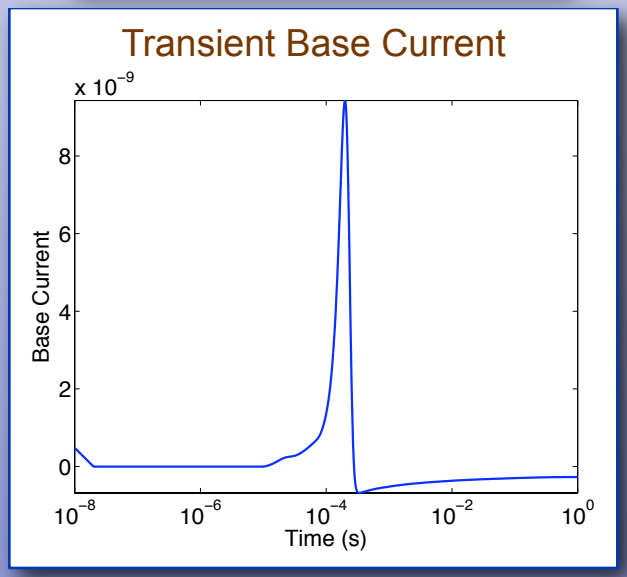
- 9x0.1 micron pseudo-1D simulation
- 1046x1 quad cells
- Linear finite elements + SUPG
- 2 carriers + 35 defect species
- 108,030 unknowns on 32 processors
- 84 carrier-defect reactions
- 126 parameters for sensitivity analysis
- Base current provides observation function





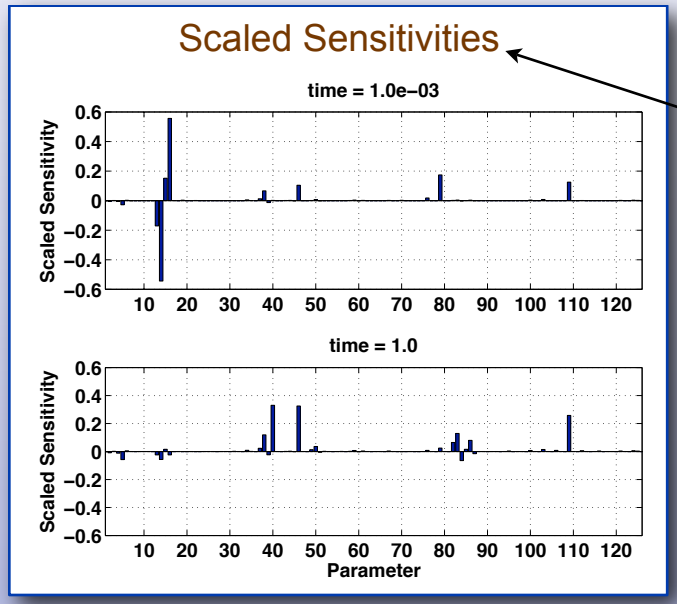




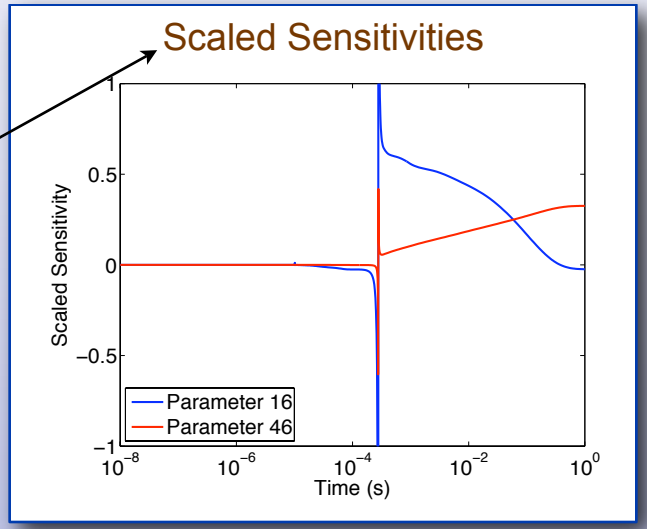


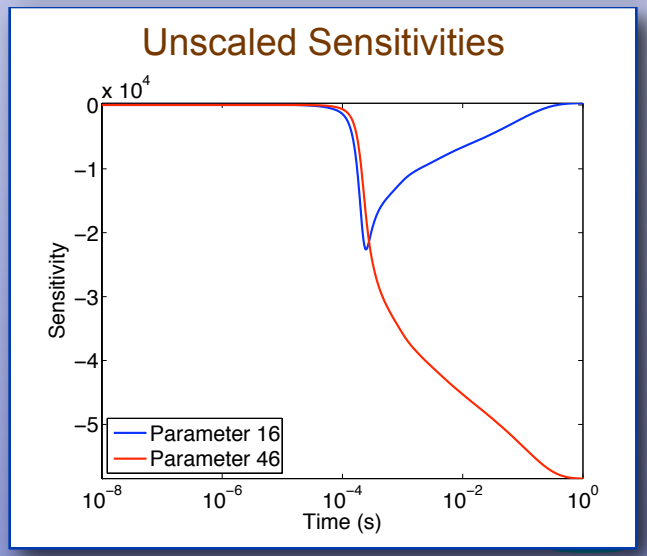
Sandia National Laboratories

Transient Base Current Sensitivities



$$\frac{p}{I}\frac{dI}{dp}$$



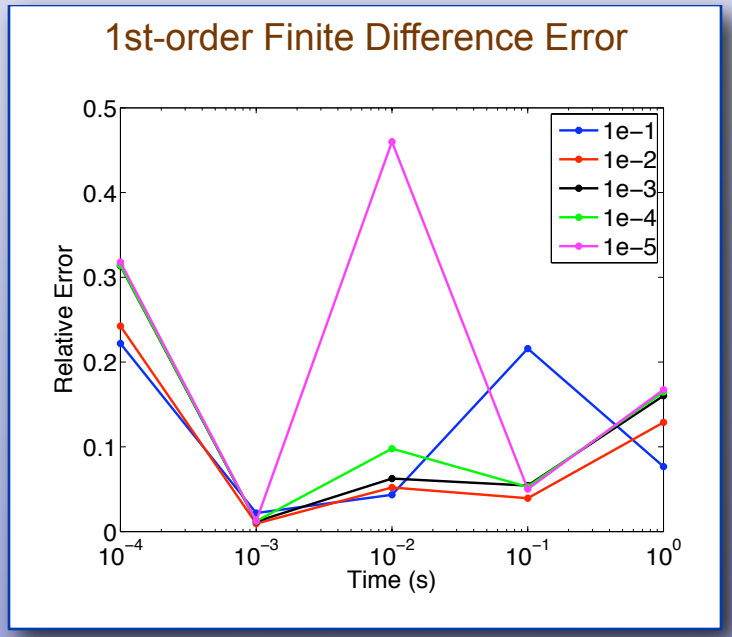


#	Reaction	Parameter	Value
14	$V^{--} \rightarrow e^- + V^-$	activation energy	0.09
16	$e^- + V^0 \rightarrow V^-$	cross-section	2.40E-14
46	$e^- + PV^0 \rightarrow PV^0$	cross-section	1.50E-15

Sandia National Laboratories

Comparison to Black-Box Finite Differences

- Run-times:
  - Forward simulation: 105 min.
  - Direct sensitivities: 931 min.
  - Black-box, first-order FD: ~13,000 min.
- Direct approach more efficient
  - 14x speed-up
- Direct approach more accurate
  - 1-2 correct digits w/FD
  - FD requires tighter tolerances to achieve higher accuracy
- Direct approach more robust
  - Accuracy solely dictated by time-integration error



Sandia National Laboratories

Summary

- Simple AD approach
  - Apply AD at element-level
  - Simple, fast, Sacado AD tools
  - Templating
- AD + Rythmos provides remarkable improvements over black-box FD for transient sensitivities
  - Efficiency, accuracy, robustness
- Transformational enabling technology
  - Allows app developers to focus on physics, not derivatives
  - Enables advanced embedded analysis provided by Trilinos
  - Application templating provides algorithmic hooks