Sundance: a Trilinos package for efficient development of efficient simulators

Kevin Long

Department of Mathematics and Statistics
Texas Tech University

Copper Mountain Trilinos Workshop
7 April 2008
Acknowledgements

Collaborators

- Ross Bartlett (Sandia)
- Catherine Beni (Caltech)
- Paul Boggs (Sandia)
- Roger Ghanem (USC)
- Vicki Howle (TTU)
- Rob Kirby (TTU)
- Jill Reese (Florida State)
- George Saad (USC)
- Bart van Bloemen Waanders (Sandia)
Kirby's Recursive Conundrum

Computers were invented to automate tedious, error-prone tasks. Computer programming is a tedious, error-prone task. (R. Kirby)
Kirby's Recursive Conundrum

Computers were invented to automate tedious, error-prone tasks. Computer programming is a tedious, error-prone task. (R. Kirby)

So why not program a computer to do it?
Our goal: simplify FEM simulation development without sacrificing performance

We want everything
Provide general multiphysics and intrusive capabilities, and a friendly user interface, and high performance

Our approach
- State in mathematical form the problems that arise “writing” an efficient intrusive code
- Write (by hand, once) a code to solve those meta-problems
Differentiation provides a path to automation

The mathematics of FEM system assembly is summarized as:

\[
\frac{\partial F}{\partial v_i} = \sum_\alpha \int_\Omega \frac{\partial F}{\partial D_\alpha v} D_\alpha \psi_i \\
\frac{\partial^2 F}{\partial v_i \partial u_j} = \sum_{\alpha,\beta} \int_\Omega \frac{\partial^2 F}{\partial D_\alpha v \partial D_\beta u} D_\alpha \psi_i D_\beta \phi_j
\]

(From KL, Howle, Kirby, and van Bloemen Waanders 2008)

The key idea

These equations bridge high-level problem specification and low-level computation. Fréchet differentiation connects:

- The abstract problem specification \( F \)
- The discretization specification: \( \psi, \phi \), and integration procedure
- The discrete matrix and vector elements \( \frac{\partial^2 F}{\partial v_i \partial u_j} \) and \( \frac{\partial F}{\partial v_i} \)
A plan for automation

<table>
<thead>
<tr>
<th>To make practical use of the “bridge theorem” we need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- A data structure for high-level symbolic description of functionals $F$</td>
</tr>
<tr>
<td>- Integrands $\mathcal{F}$ represented as DAG</td>
</tr>
<tr>
<td>- Automated selection of basis function combinations from element library, given signature of derivative</td>
</tr>
<tr>
<td>- Connection to finite element infrastructure for basis functions, mesh, quadrature, linear algebra, and solvers</td>
</tr>
<tr>
<td>- A top-level layer for problem specification</td>
</tr>
<tr>
<td>- A method to automate the organization of efficient in-place computations of numerical values of $\frac{\partial F}{\partial v}$, etc, given DAG for $\mathcal{F}$</td>
</tr>
</tbody>
</table>
Sundance: a Trilinos package for efficient development of efficient simulators

Sundance: a Trilinos package taking high-level abstractions to efficient code

Poisson-Boltzmann solver in a notebook

\[ \nabla^2 u = \sinh u, \quad \frac{\partial u}{\partial n} = 0 \text{ on } \Gamma_N, \quad u = u_D \text{ on } \Gamma_D \]

\[ \int_{\Omega} (\nabla \cdot \nabla u + \nabla \cdot \sinh u) \, ds - \int_{\Gamma_N} q \cdot n \, dl = 0 \quad \forall \, \nu \in \mathcal{H}^1 \]

with \( u = u_D \) on \( \Gamma_D \)

- use \( \mathbf{P} \) for \( u, v \)
- do integrals exactly when possible, with 4th-order Gauss otherwise
Sundance: a Trilinos package for efficient development of efficient simulators

Sundance: a Trilinos package taking high-level abstractions to efficient code

Poisson-Boltzmann solver in Sundance

```cpp
Mesh mesh = mesh.getMesh();

/* Create a cell filter that will identify the maximal cells */
CellFilter interior = new MaximalCellFilter();
CellFilter edges = new DimensionalCellFilter(1);
CellFilter left = edges.labeledSubset(1);

/* Create unknown and test functions, discretized using first-order */
LagrangeInterpolants *
int order = 1;
Expr u = new UnknownFunction(new Lagrange(order), "u");
Expr v = new TestFunction(new Lagrange(order), "v");

/* Create differential operator and coordinate functions */
Expr dx = new Derivative(0);
Expr dy = new Derivative(1);
Expr grad = List(dx, dy);

/* We need a quadrature rule for doing the integrations */
QuadratureFamily quad2 = new GaussianQuadrature(2);
QuadratureFamily quad4 = new GaussianQuadrature(4);

/* Define the weak form */
Expr eqn = IntegralInterior, (grad*U)*(grad*V) + V*exp(-U), quad2;

/* Define the Dirichlet BC */
Expr bc = EssentialBC(left, v*(u-1.0), quad4);

/* Create a discrete space, and discretize the function 1.0 on it */
DiscreteSpace discSpace(mesh, new Lagrange(order), vecType);
Expr u0 = new DiscreteFunction(discSpace, 1.0, "u0");

/* Create a TSF NonlinearOperator object */
NonlinearOperator<Double> F = new NonlinearProblem(mesh, eqn, bc, V, u, u0, vecType);
```
Math-based automated assembly is at least as efficient as matrix assembly in hand-coded, problem-tuned “gold standard” codes.

- Comparison of assembly times for 3D forward problems
  - Sundance uses same solvers (Trilinos) as gold-standard codes
- MP-Salsa and Fuego don’t allow intrusion
  - Can only compare forward problem performance
  - Comparisons do not include additional gains enabled by Sundance’s intrusive capabilities
Parallel scalability of assembly process

<table>
<thead>
<tr>
<th>Processors</th>
<th>Assembly time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>54.5</td>
</tr>
<tr>
<td>16</td>
<td>54.7</td>
</tr>
<tr>
<td>32</td>
<td>54.3</td>
</tr>
<tr>
<td>128</td>
<td>54.4</td>
</tr>
<tr>
<td>256</td>
<td>54.4</td>
</tr>
</tbody>
</table>

- Assembly times for a model CDR problem on ASC Red Storm
- Weak scalability means: assembly time remains constant as number of processors increases in proportion to problem size
- Results demonstrate Sundance is weakly scalable
How can user-friendly, intrusion-friendly code be fast?

High performance is a result of:

- Amortization of overhead
- Careful memory management
- Effective use of BLAS
- Work reduction through data flow analysis

With our unified formulation, effort spent tuning computational kernels applies immediately to diverse problem types and arbitrary PDE
Amortization, memory management, and BLAS

**Amortize DAG traversal**
- Process batches of elements and quadrature points

**Minimize allocations and copies**
- Maintain a stack of work vectors
- Automatically identify vectors that can be operated into
  - *e.g.* identify opportunities for $x+=y$ instead of $x=x+y$

**BLAS**
- Aggregate integral transformations, hit with level 3 BLAS
Pre-computation data flow analysis lets us avoid unnecessary work

**Sparsity determination**
- Don’t store or compute derivatives known to be zero or unused
- Distinguish spatially-constant from spatially-variable derivatives

**We must track data flow through these operations:**
- Multivariable, multiargument chain rule (special cases: +,-,×,/)  
- Spatial differentiation

**Set theoretical data flow analysis**
- Tracks changes in sets of nonzero, constant, and variable derivatives through evaluation process
- Implemented using STL set/multiset classes
A key to high level simplicity without low performance: division of labor

Decouple user-level representation from low-level evaluation

Reduces human factors / performance tradeoffs
- User-level objects optimized for human factors
- Low-level objects optimized for performance

Allows interchangeable evaluators under a common interface
- Easy to upgrade, tune, and experiment with evaluators w/o impact on user
- Future: different evaluators for different architectures
Summary

- Mathematical formulation of discrete system assembly process enables automated transition from “blackboard” mathematics to efficient PDE simulation
- Automated assembly code compares favorably in both efficiency and scalability to “gold standard” simulators
- Our method transparently enables intrusive algorithms, realizing even greater performance gains for optimization, sensitivity, and UQ
- By developing a unifying mathematical framework for PDE simulation, our results can be applied to a wide range of types of discretization methods and physical problems

The one-sentence description

We use high-level symbolic components to automate the organization of low-level high-performance numerical computations