# Automated Self-Assembly of Components for Multiphysics Simulations

K. Long

Department of Mathematics and Statistics
Texas Tech University

SIAM Conference on Parallel Computing
26 Feb 2010

# Outline

1. **High-level problem specifications**

2. **Automating simulation self-assembly from high-level interfaces**
   - Mathematical foundations
   - Software architecture
   - Performance and scalability

# Outline

# High-level specification of a PDE

## Fragment of user-level code for Poisson equation

```
/* Create unknown and test functions, discretized using first-order
 * Lagrange interpolants */
Expr u = new UnknownFunction(new Lagrange(2), "u");
Expr v = new TestFunction(new Lagrange(2), "v");

/* Create differential operator and coordinate function */
Expr dx = new Derivative(0);
Expr x = new CoordExpr(0);

/* Define the weak form */
Expr eqn = Integral(interior, -(dx*v)*(dx*u), quad)
     + Integral(interior, -2.0*v, quad);
/* Define the Dirichlet BC */
Expr bc = EssentialBC(leftPoint, v*u, quad);

/* Put together a linear problem */
LinearProblem prob(mesh, eqn, bc, v, u, vecType);
```

# Goals and approach

## From high level to low level

- High-level description encodes specification of low-level tasks
- The key to retaining performance is to distinguish high-level specification objects from low-level computational kernels
- High-level description encodes structural information we can exploit to improve performance
  - For example, identify and eliminate redundant computations
- Optimization of computational kernels then apply to any simulation built from them
- Our job is to "build the matrix" (or effect an MV multiply)

# The Sundance toolkit

## Ideas implemented in Sundance toolkit

- Sundance is a Trilinos package
- Available from `trilinos.sandia.gov`

# Outline

1. High-level problem specifications

2. Automating simulation self-assembly from high-level interfaces
   - Mathematical foundations
   - Software architecture
   - Performance and scalability

# Preliminaries

## Functionals

We work with functionals $F : (V, U) \to \mathbb{R}$ of the form

$$F[v, u,] = \int_\Omega \mathcal{F}(v, u)\, dx$$

where $\mathcal{F}$

- can be a **nonlinear** differential operator on $u$
- but must be a **linear, homogeneous** differential operator on $v$.

## Weak Equations

We are interested in equations of the form

$$F[v, u] = 0 \quad \forall v \in V$$

# Expository simplifications

## To keep notation compact during talk:

- Assume the same equations apply over whole domain
  - Trivially extended to multiple domains
- Ignore boundary conditions
  - Many BC methods (e.g., Nitsche) fit immediately into the framework as shown
  - Replacement BC methods require some annotation at user level and conditionals in matrix assembly, but otherwise fit into our framework

# Examples of some functionals

## Steady Navier-Stokes flow

$$F[\mathbf{v}, \mathbf{u}, q, p] = \int_\Omega [\nu \nabla \mathbf{v} : \nabla \mathbf{u} + p \nabla \cdot \mathbf{v} + \mathbf{v} \cdot (\mathbf{u} \cdot \nabla) \mathbf{u} + q \nabla \cdot \mathbf{u}]$$

## Lagrangian for Poisson source inversion w/ Tikhonov

$$\begin{aligned} F[v, \mu, \beta, u, \lambda, \alpha] = & \int_\Omega [(u - u^*) v + \nabla \lambda \cdot \nabla v] \, dx + \\ & + \int_\Omega [\nabla \mu \cdot \nabla u + \mu \alpha] \, dx + \\ & + \int_\Omega [\nabla \alpha \cdot \nabla \beta + \lambda \beta] \, dx \end{aligned}$$

# Discretization

- Introduce $N$-dimensional subspaces $V^h$ and $U^h$, with bases $\{\psi_i\}_{i=1}^N$ and $\{\phi_i\}_{i=1}^N$.
- Introduce $v^h = \sum_{i=1}^N \mathbf{v}_i \psi_i$ and $u^h = \sum_{i=1}^N \mathbf{u}_i \phi_i$ where $\mathbf{v}$ and $\mathbf{u}$ are vectors of expansion coefficients.

- Approximation on spaces $V^h, U^h$ takes $F[v, u]$ to

$$\hat{F}(\mathbf{v}, \mathbf{u}) : (\mathbb{R}^N, \mathbb{R}^N) \to \mathbb{R}.$$

- The equation $F[v, u] = 0 \; \forall \, v \in V$ becomes

$$\hat{F}(\mathbf{v}, \mathbf{u}) = 0 \; \forall \, \mathbf{v} \in \mathbb{R}^N.$$

# A simple but consequential lemma

## Lemma

$\mathbf{u}^*$ *is a solution of* $\hat{F}(\mathbf{v}, \mathbf{u}) = 0 \quad \forall\, \mathbf{v} \in \mathbb{R}^N$ *iff*

$$\left.\frac{\partial \hat{F}}{\partial v_i}\right|_{\mathbf{v}=0,\mathbf{u}=\mathbf{u}^*} = 0, \quad i = 1 : N$$

## Proof.

Follows immediately from linearity & homogeneity of $F$ as it acts on $\mathbf{v}$. □

# A simple but consequential lemma

## Lemma

$\mathbf{u}^*$ *is a solution of* $\hat{F}(\mathbf{v}, \mathbf{u}) = 0 \quad \forall\, \mathbf{v} \in \mathbb{R}^N$ *iff*

$$\left.\frac{\partial \hat{F}}{\partial v_i}\right|_{\mathbf{v}=0, \mathbf{u}=\mathbf{u}^*} = 0, \quad i = 1 : N$$

## Derivatives are with respect to *test functions*

Not to be confused with first-order optimality conditions for a variational problem

# Differentiation provides the path to automation

## A simple theorem establishes these relations

$$\frac{\partial F}{\partial v_i} = \sum_{\alpha} \int_{\Omega} \frac{\partial \mathcal{F}}{\partial D_\alpha v} D_\alpha \psi_i$$

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

## The key idea

This theorem bridges high-level problem specification and low-level computation! Fréchet differentiation connects:

- The abstract problem specification $\mathcal{F}$
- 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

## To make practical use of the bridge theorem we need:

- A data structure for high-level symbolic description of functionals $F$
- Integrands $\mathcal{F}$ represented as DAG
- Automated selection of basis function combinations from element library, given signature of derivative
- Connection to finite element infrastructure for basis functions, mesh, quadrature, linear algebra, and solvers
- A top-level layer for problem specification
- A method to automate the organization of efficient in-place computations of *numerical values* of $\frac{\partial \mathcal{F}}{\partial v}$, etc, given DAG for $\mathcal{F}$
- Low-level AD tools not suitable

# Example: Burgers' Equation

## Weak form of Burgers' equation on $[0, 1]$

$$\int_0^1 \left[ v u D_x u + c D_x v D_x u \right] \, dx = 0 \quad \forall v \in H^1_\Omega .$$

(Ignoring BCs)

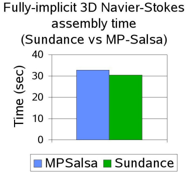
# Burgers' Example: Mapping from Derivative Signature to Coefficient and Basis Combination

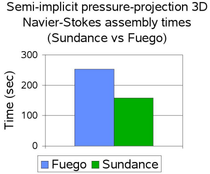
| Derivative | Multiset | Value | Basis combination | Integral |
|---|---|---|---|---|
| $\frac{\partial \mathcal{G}}{\partial v}$ | $\{v\}$ | $u_0 D_x u_0$ | $\phi_i$ | $\int u_0 D_x u_0 \phi_i$ |
| $\frac{\partial \mathcal{G}}{\partial D_x v}$ | $\{D_x v\}$ | $c D_x u_0$ | $D_x \phi_i$ | $\int c D_x u_0 D_x \phi_i$ |
| $\frac{\partial^2 \mathcal{G}}{\partial v \partial u}$ | $\{v, u\}$ | $D_x u_0$ | $\phi_i \phi_j$ | $\int D_x u_0 \phi_i \phi_j$ |
| $\frac{\partial^2 \mathcal{G}}{\partial v \partial D_x u}$ | $\{v, D_x u\}$ | $u_0$ | $\phi_i D_x \phi_j$ | $\int u_0 \phi_i D_x \phi_j$ |
| $\frac{\partial^2 \mathcal{G}}{\partial D_x v \partial D_x u}$ | $\{D_x v, D_x u\}$ | $c$ | $D_x \phi_i D_x \phi_j$ | $\int c D_x \phi_i D_x \phi_j$ |

# Performance and Scalability Results and Methods

# 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 instrusion
  - Can only compare forward problem performance
  - Comparisons do *not* include additional gains enabled by Sundance’s intrusive capabilities



# Comparison to generated code (Dolfin)

| Stokes assembly timings, 3D Taylor-Hood | | | |
|---|---|---|---|
| verts | tets | $p = 2; 1$ | |
| | | Sundance | Dolfin |
| 142 | 495 | 0.07216 | 0.3362 |
| 874 | 3960 | 0.6677 | 2.793 |
| 6091 | 31680 | 5.521 | 22.57 |
| 45397 | 253440 | 45.97 | crash |

# Parallel scalability of assembly process

| Processors | Assembly time |
| ---: | ---: |
| 4 | 54.5 |
| 16 | 54.7 |
| 32 | 54.3 |
| 128 | 54.4 |
| 256 | 54.4 |

- 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

# A key to high level ease-of-use without low performance: division of labor

### Decouple user-level representiation 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