A compiler for variational forms - practical results

USNCCM8

Johan Jansson
johanjan@math.chalmers.se

Chalmers University of Technology

Acknowledgements: Anders Logg and the FEniCS members
Overview

- **Part I - FEniCS Form Compiler (FFC):**
  - Motivation for FFC (generality of FEM)
  - Introduction to FFC
  - Benchmarks (FFC vs. quadrature)

- **Part II - Application (Elasto-Plasticity):**
  - Motivation for elasto-plastic model
  - Implementation of elasto-plastic model in FFC
  - Benchmarks (FFC vs. mass-spring)
  - Future work
Motivation for FFC

**FEniCS** project: Automation of Computational Mathematical Modeling (ACMM)

Finite Element Method: General method for automating discretization of differential equations

This generality is seldom reflected in software

Reasons: conceptual complexity, hand-written routines often outperform general routines

How can we overcome these difficulties?

Through a **Form Compiler** which automatically generates an optimal Finite Element routine (assembly)
Motivation for FFC

Advantages of compilation:

- A form compiler can be written in a high-level language and/or with high-level data structures which eases conceptual abstraction.
- A form compiler can pre-compute quantities which are known at compile time.

Disadvantages of compilation:

- Forms cannot easily be modified during run time.
FFC: the FEniCS Form Compiler

- Automates a key step in the implementation of finite element methods for partial differential equations
- Input: a variational form and a finite element
- Output: optimal C/C++

\[ a(v, u) = \int_{\Omega} \nabla u(x) \cdot \nabla v(x) \, dx \]

>> ffc [-l language] poisson.form
Basic example: Poisson’s equation

- Strong form: Find \( u \in C^2(\bar{\Omega}) \) with \( u = 0 \) on \( \partial \Omega \) such that
  \[-\Delta u = f \quad \text{in} \ \Omega\]

- Weak form: Find \( u \in H^1_0(\Omega) \) such that
  \[
  \int_{\Omega} \nabla u(x) \cdot \nabla v(x) \, dx = \int_{\Omega} f(x)v(x) \, dx \quad \text{for all} \ v \in H^1_0(\Omega)
  \]

- Standard notation: Find \( u \in V \) such that
  \[
  a(v, u) = L(v) \quad \text{for all} \ v \in \hat{V}
  \]
  with \( a : \hat{V} \times V \to \mathbb{R} \) a \textit{bilinear form} and \( L : \hat{V} \to \mathbb{R} \) a \textit{linear form} (functional)
Obtaining the discrete system

Let $V$ and $\hat{V}$ be discrete function spaces. Then

$$a(v, U) = L(v) \quad \text{for all } v \in \hat{V}$$

is a discrete linear system for the approximate solution $U \approx u$.

With $V = \text{span}\{\phi_i\}_{i=1}^M$ and $\hat{V} = \text{span}\{\hat{\phi}_i\}_{i=1}^M$, we obtain the linear system

$$Ax = b$$

for the degrees of freedom $x = (x_i)$ of $U = \sum_{i=1}^M x_i \phi_i$, where

$$A_{ij} = a(\hat{\phi}_i, \phi_j)$$

$$b_i = L(\hat{\phi}_i)$$
Computing the linear system: assembly

Noting that $a(v, u) = \sum_{K\in \mathcal{T}} a_K(v, u)$, the matrix $A$ can be assembled by

\[
A = 0 \\
\text{for all elements } K \in \mathcal{T} \\
A += A^K
\]

The element matrix $A^K$ is defined by

\[
A^K_{i,j} = a_K(\hat{\phi}_i, \phi_j)
\]

for all local basis functions $\hat{\phi}_i$ and $\phi_j$ on $K$
Multi-linear forms

Consider a multi-linear form

$$a : V_1 \times V_2 \times \cdots \times V_r \rightarrow \mathbb{R}$$

with $V_1, V_2, \ldots, V_r$ function spaces on the domain $\Omega$

- Typically, $r = 1$ (linear form) or $r = 2$ (bilinear form)
- Assume $V_1 = V_2 = \cdots = V_r = V$ for ease of notation

Want to compute the rank $r$ element tensor $A^K$ defined by

$$A^K_i = a_K(\phi_{i_1}, \phi_{i_2}, \ldots, \phi_{i_r})$$

with $\{\phi_i\}_{i=1}^n$ the local basis on $K$ and multi-index $i = (i_1, i_2, \ldots, i_r)$
Tensor representation

In general, the element tensor $A^K$ can be represented as the product of a reference tensor $A^0$ and a geometry tensor $G_K$:

$$A^K_i = A^0_{i\alpha} G_K^{\alpha}$$

- $A^0$: a tensor of rank $|i| + |\alpha| = r + |\alpha|$
- $G_K$: a tensor of rank $|\alpha|$

Basic idea:
- Precompute $A^0$ at compile-time
- Generate optimal code for run-time evaluation of $G_K$ and the product $A^0_{i\alpha} G_K^{\alpha}$
Example: Poisson

Form:

\[ a(v, u) = \int_{\Omega} \nabla v(x) \cdot \nabla u(x) \, dx \]

Evaluation:

\[ A_i^K = \int_K \nabla \phi_{i_1}(x) \cdot \nabla \phi_{i_2}(x) \, dx \]

\[ = \det F'_K \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \int_{K_0} \frac{\partial \Phi_{i_1}}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}}{\partial X_{\alpha_2}} \, dX = A_{i\alpha}^0 G_{K}^\alpha \]

with \( A_{i\alpha}^0 = \int_{K_0} \frac{\partial \Phi_{i_1}}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}}{\partial X_{\alpha_2}} \, dX \) and \( G_{K}^\alpha = \det F'_K \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \)
Basic usage: compiling a form

1. Implement the form using your favorite text editor (emacs):

   ```
   # Poissons equation: a(v, u) = L(v)
   
   a = v.dx(i)*u.dx(i)*dx
   L = v*f*dx
   
   F1 poisson.form  (Python)--L5--All-------
   (No changes need to be saved)
   ```

2. Compile the form using **FFC**:

   ```
   >> ffc poisson.form
   ```

   This will generate C++ code (*Poisson.h*) for **DOLFIN**
Example: Classical Elasticity

FFC representation (Elasticity.form):

```python
# The bilinear form for classical linear elasticity
# Compile this form with FFC: ffc Elasticity.form.

element = FiniteElement("Lagrange", "tetrahedron", 1)

c1 = Constant()  # Lame coefficient
c2 = Constant()  # Lame coefficient
f = Function(element)  # Source

v = BasisFunction(element)
u = BasisFunction(element)

a = (2.0 * c1 * u[i].dx(i) * v[j].dx(j) +
    c2 * (u[i].dx(j) + u[j].dx(i)) * (v[i].dx(j) + v[j].dx(i))) * dx
L = f[i] * v[i] * dx
```

A compiler for variational forms - practical results – p. 13
Example: Classical Elasticity

FFC output (**Elasticity.h**):

```cpp
BilinearForm(const real& c0, const real& c1) : ...

bool interior(real* block) const
{
    // Compute geometry tensors
    real G0_0_0_0_0 = det*c0*g00*g00;
    real G0_0_0_0_1 = det*c0*g00*g10;
    ...

    // Compute element tensor
    block[0] =
    3.333333333333329e-01*G0_0_0_0_0 + 3.333333333333329e-01*G0_0_0_0_1 +
    3.333333333333329e-01*G0_0_0_0_2 + 3.333333333333329e-01*G0_0_0_1_0 +
    3.333333333333329e-01*G0_0_0_1_1 + 3.333333333333329e-01*G0_0_0_1_2 +
    3.333333333333329e-01*G0_0_0_2_0 + 3.333333333333329e-01*G0_0_0_2_1 +
    3.333333333333329e-01*G0_0_0_2_2 + 1.666666666666664e-01*G1_0_0 +
    1.666666666666664e-01*G1_0_1 + 1.666666666666664e-01*G1_0_2 +
    1.666666666666664e-01*G1_1_0 + 1.666666666666664e-01*G1_1_1 +
    ...
```
Impressive speedups

FFC vs Quadrature at q = 3 (2D)
Results

Mass matrix 2D (1 million triangles)

Mass matrix 3D (1 million tetrahedrons)
Results

Poisson 2D (1 million triangles)

Poisson 3D (1 million tetrahedrons)
Results

Navier-Stokes 2D (1 million triangles)

Navier-Stokes 3D (1 million tetrahedrons)
Results

Elasticity 2D (1 million triangles)

Elasticity 3D (1 million tetrahedrons)
Motivation for elasto-plastic model

State of the art computer games use rigid body motion with joints (Half Life 2).

Motion pictures primarily use animation by hand, some cases of mass-spring simulation (hair, cloth).

Why don’t these applications use more advanced/general models?

Traditional elasticity models are difficult to understand ⇒ difficult to apply, use effectively.

Attempt to find a simple model, attempt to automate discretization of model.
Previous work - mass-spring model

Can we find an analogous PDE-model?
Simple derivation of model

Classical linear elasticity:

\[ u = x - X, \]
\[ \dot{u} - v = 0 \quad \text{in} \ \Omega^0, \]
\[ \dot{v} - \nabla \cdot \sigma = f \quad \text{in} \ \Omega^0, \]
\[ \sigma = E\varepsilon(u) = E(\nabla u^\top + \nabla u) \]
\[ E\varepsilon = \lambda \sum_k \varepsilon_{kk} I + 2\mu \varepsilon, \]
\[ v(0, \cdot) = v^0, \quad u(0, \cdot) = u^0 \quad \text{in} \ \Omega^0. \]

Only works for small displacements. Computations carried out on fixed geometry \( \Omega^0 \). Why not use the deformed geometry \( \Omega(t) \)?
The elastic model

Formulate the model in the deformed geometry $\Omega(t)$ (updated Lagrange):

\[
\begin{align*}
\dot{u} - v &= 0 \quad \text{in } \Omega(t), \\
\dot{v} - \nabla \cdot \sigma &= f \quad \text{in } \Omega(t), \\
\dot{\sigma} &= E\varepsilon(v) = E(\nabla v^T + \nabla v) \\
v(0, \cdot) &= v^0, \quad u(0, \cdot) = u^0 \quad \text{in } \Omega^0.
\end{align*}
\]

The model is a piecewise linear elastic model. Given some geometry $\Omega_i$ we compute using the linear model (small displacements) for a small time step/iteration and produce the geometry $\Omega_{i+1}$. The process is then repeated.
Examples

Elastic bar (Updated Lagrange)

Elastic bar (Mass-spring)

Elastic bar (Classical elasticity)
Viscosity

\[ \dot{\nu} - \nabla \cdot \sigma - \nu \nabla \cdot \epsilon(\nu) = f \quad \text{in } \Omega(t) \]

We add a simple viscous term to model viscosity in materials.
Plasticity

\[ \dot{\sigma} = E(\varepsilon(v) - \frac{1}{\nu_p}(\sigma - \pi\sigma)) \text{ in } \Omega(t), \]

\[ \pi\sigma = \frac{\sigma}{||\sigma||}, \quad ||\sigma|| > Y_s \]

\[ \pi\sigma = \sigma, \quad ||\sigma|| \leq Y_s \]

Visco-plastic model. \( \pi\sigma \) is the projection on to the set of admissible stresses. \( Y_s \) is the yield stress of the material.
Examples (Plasticity)

Plastic bar
Implementation in FFC

FFC representation (**ElasticityUpdated.form**):

```python
# Form for updated elasticity (velocity)

element1 = FiniteElement("Discontinuous vector Lagrange", "tetrahedron", 0)
element2 = FiniteElement("Vector Lagrange", "tetrahedron", 1)

nu = Constant()  # viscosity coefficient

w = BasisFunction(element2)
f = Function(element2)
sigma0 = Function(element1)
sigma0 = Function(element1)

element1 = FiniteElement("Discontinuous vector Lagrange", "tetrahedron", 0)
element2 = FiniteElement("Vector Lagrange", "tetrahedron", 1)

# Form for updated elasticity (velocity)

nu = Constant()  # viscosity coefficient

w = BasisFunction(element2)
f = Function(element2)
sigma0 = Function(element1)
sigma0 = Function(element1)

L = (f[i] * v[i] -
     (sigma0[i] * w[0].dx(i) +
      sigma1[i] * w[1].dx(i) +
      sigma2[i] * w[2].dx(i)) -
     nu * (
       epsilon0[i] * w[0].dx(i) +
       epsilon1[i] * w[1].dx(i) +
       epsilon2[i] * w[2].dx(i))) * dx
```

A compiler for variational forms - practical results – p. 28
Implementation in FFC

**FFC representation** *(ElasticityUpdatedSigma0.form):*

```
# Form for updated elasticity (stress component 0)

element1 = FiniteElement("Vector Lagrange", "tetrahedron", 1)
element2 = FiniteElement("Discontinuous vector Lagrange", "tetrahedron", 0)

c1 = Constant()  # Lame coefficient
c2 = Constant()  # Lame coefficient
nuplast = Constant()  # Plastic viscosity

q = BasisFunction(element2)
v = Function(element1)
sigma0 = Function(element2)
sigmanorm = Function(element2)  # Norm of sigma (stress)

Lplast = ((c1 * (sigma0[0] + sigma1[1] + sigma2[2]) * q[0]) +
          (c2 * sigma0[i] * q[i]))

Lelast = ((2 * c1 * v[i].dx(i) * q[0]) +
          (c2 * (v[i].dx(0) + v[0].dx(i))) * q[i])

L = (Lelast - nuplast * (1 - sigmanorm[0]) * Lplast) * dx
```
General examples

Visco-elastic cow

Plastic cow

Real time simulation
Updated elasticity vs. mass-spring
Time / dof

![Graph showing time vs. dof for FFC and mass-spring models.](image-url)
Profiling

- Spends 90% assembling, only 10% actually evaluating form, could likely be optimized further

<table>
<thead>
<tr>
<th>%time</th>
<th>calls</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.86</td>
<td>226382058</td>
<td>dolfin::Function::interpolate()</td>
</tr>
<tr>
<td>8.04</td>
<td>41162058</td>
<td>dolfin::AffineMap::updateTetrahedron()</td>
</tr>
<tr>
<td>3.23</td>
<td>10290000</td>
<td>dolfin::ElasticityUpdated::LinearForm::eval()</td>
</tr>
<tr>
<td>3.23</td>
<td>740882058</td>
<td>dolfin::Cell::id() const</td>
</tr>
<tr>
<td>2.57</td>
<td>617498784</td>
<td>dolfin::GenericCell::nodeID() const</td>
</tr>
<tr>
<td>2.46</td>
<td>10290000</td>
<td>dolfin::ElasticityUpdatedSigma2::LinearForm::eval()</td>
</tr>
<tr>
<td>2.30</td>
<td>10290000</td>
<td>dolfin::ElasticityUpdatedSigma0::LinearForm::eval()</td>
</tr>
<tr>
<td>2.15</td>
<td>10290000</td>
<td>dolfin::ElasticityUpdatedSigma1::LinearForm::eval()</td>
</tr>
</tbody>
</table>

Graph:

<table>
<thead>
<tr>
<th>%time</th>
<th>calls</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>89.7</td>
<td>20000</td>
<td>dolfin::FEM::assemble()</td>
</tr>
<tr>
<td>49.8</td>
<td>41162058</td>
<td>dolfin::Form::updateCoefficients()</td>
</tr>
</tbody>
</table>
Future work

FFC:

- Independent comparisons for FFC - benchmark against other PDE packages (also finite difference packages).
- Extend the elastic model: contact, friction (mass-spring model already does this).
- Space adaptivity
- Apply model in real applications (games for instance).
- Interface to fluid mechanics (Navier-Stokes).