Project Course in Scientific Computing

Intro meeting

Johan Jansson

jjan@csc.kth.se

CSC

KTH

Today's discussion

- Course overview
- Project plan
- Component structure of FEniCS
- FIAT
- FFC
- DOLFIN

Course overview

Project components

Projects typically has specific focus on one of the components:

Physical modeling
  Simplify equations, determine boundary conditions

Discretization
  Stability of method, error analysis

Solution algorithm
  Newton's method, adaptive algorithm, ...

Software implementation
  Implementation efficiency, generality, realization of complex algorithms

FEniCS

Goal of FEniCS

- One goal: automatic solution of PDE (variational formulation)
- Why?
  - Efficiency
    - Execution time
    - Development time
  - Safety - reduce chance of human error (bugs)

Component structure of FEniCS

- Automatic generation of finite elements (FIAT)

$$e = (K, P, \mathcal{N})$$

- Automatic evaluation of variational forms (FFC)

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u dx$$

- Automatic assembly of discrete systems (DOLFIN)

$A = 0$
`for all elements` $K \in \mathcal{T}_\Omega$
    A `+=` A^K

FIAT, the finite element tabulator

FIAT: FInite element Automatic Tabula

- Automates the generation of finite element basis functions
- Simplifies the specification of new elements
- Continuous and discontinuous Lagrange elements of arbitrary order
- Crouziex-Raviart (CR) elements
- Nedelec elements
- Raviart-Thomas (RT) elements
- Brezzi-Douglas-Marini (BDM) elements
- ...

FIAT: Implementation

- Use orthonormal basis on triangles and tets (Dubiner)
- Express basis functions in terms of the orthonormal basis
- Translate conditions of function space into linear algebraic relations
- Implemented in Python (as a Python module)

FIAT: Example

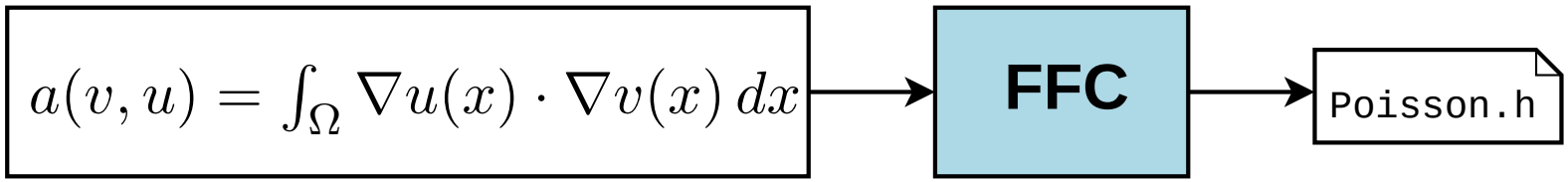
```
>>> from FIAT.Lagrange import *
>>> from FIAT.shapes import *
>>> element = Lagrange(TRIANGLE, 2)
>>> basis = element.function_space()
>>> v = basis[0]
>>> v([-1.0, -1.0])
1.0
```

(note reference triangle starts at (-1, -1) in FIAT)

FFC, the form compiler

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++

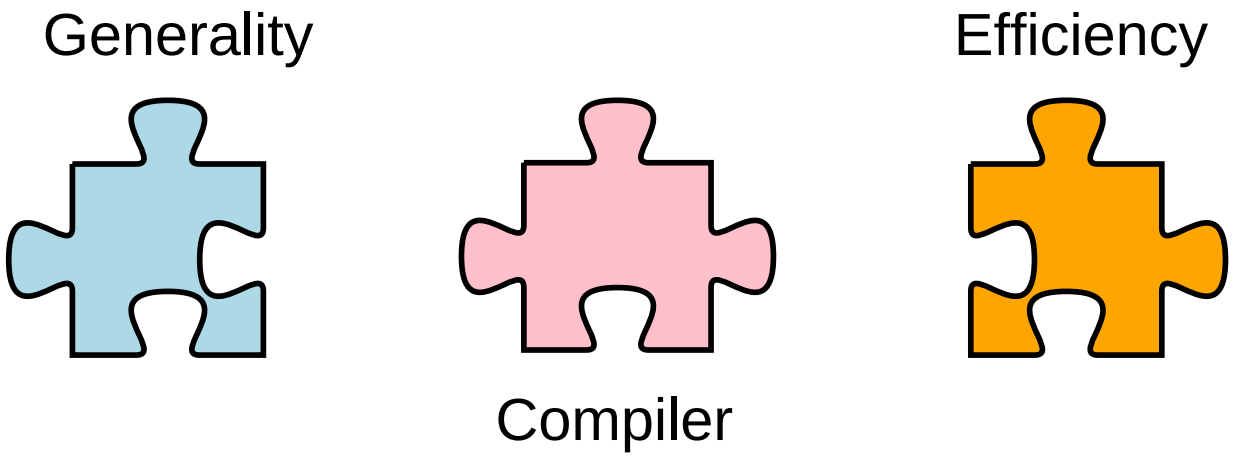



```
>> ffc [-l language] poisson.form
```

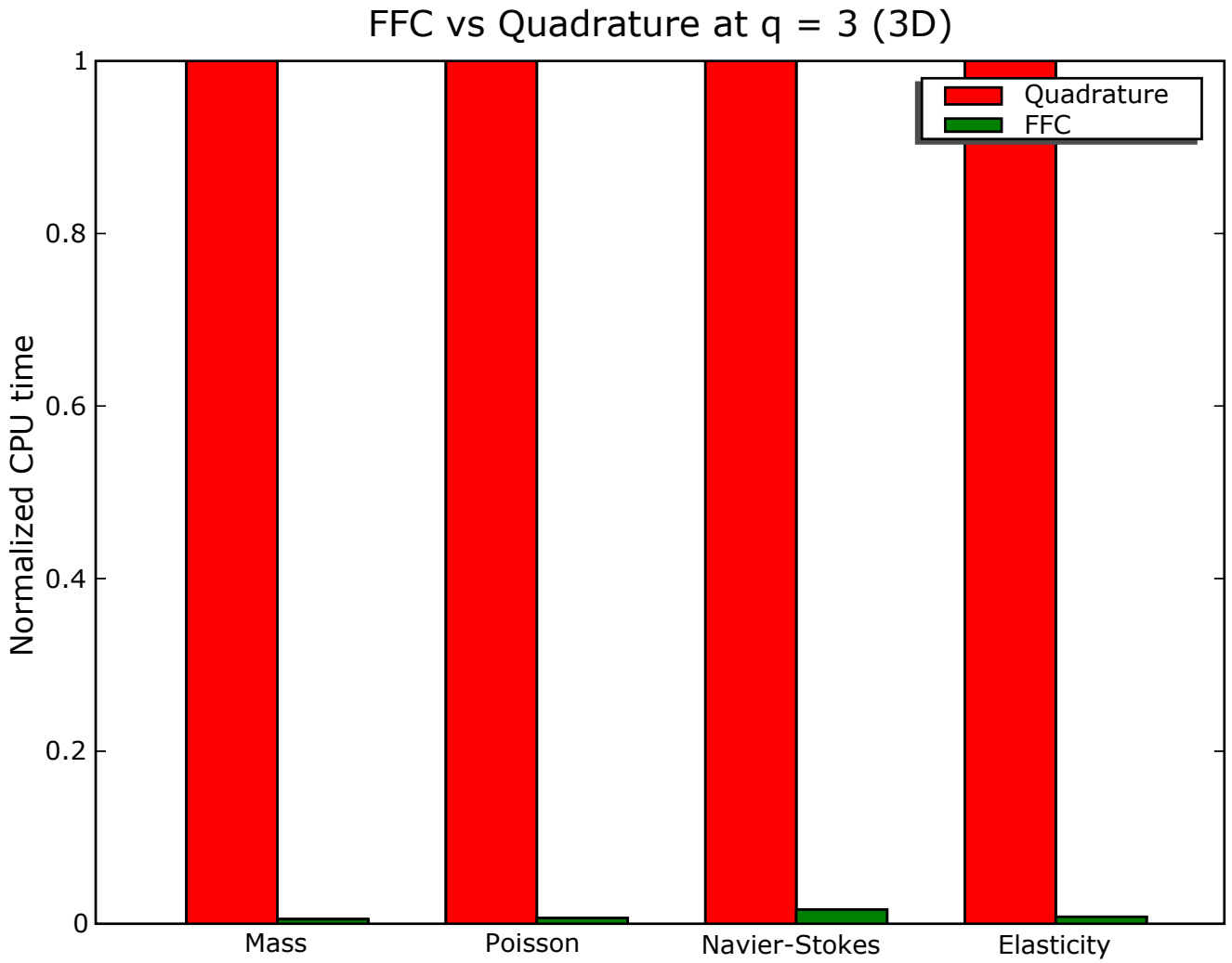
FFC: Design goals

- Any form
- Any element
- Maximum efficiency

Possible to combine generality with efficiency by using a compiler approach:



Impressive speedups



Basic example: Poisson's equation

- Strong form: Find $u \in \mathcal{C}^2(\overline{\Omega})$ with $u = 0$ on $\partial\Omega$ such that

$$-\Delta u = f \quad \text{in } \Omega$$

- Weak form: Find $u \in H_0^1(\Omega)$ such that

$$\int_\Omega \nabla u(x) \cdot \nabla v(x)\, dx = \int_\Omega f(x) v(x)\, dx \quad \text{for all } v \in H_0^1(\Omega)$$

- 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 *bilinear form* and $L : \hat{V} \to \mathbb{R}$ a *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

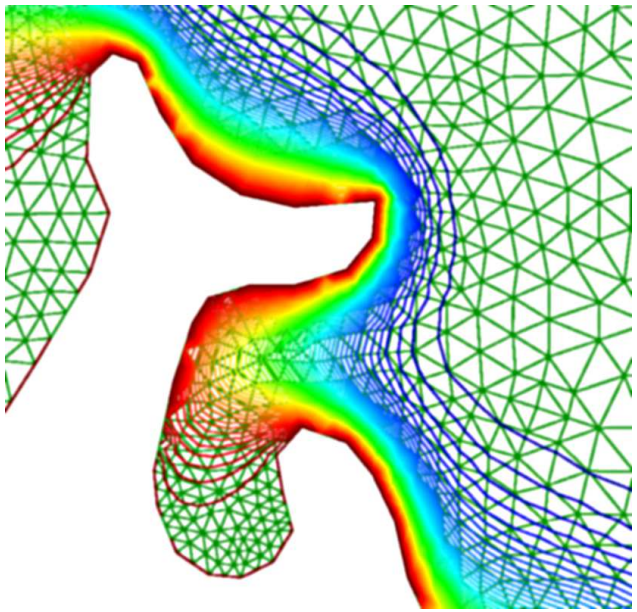
$$Ax = b$$

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

$A = 0$
for all elements $K \in \mathcal{T}$
    A += A^K

The *element matrix* A^K is defined by

$$A^K_{ij} = a_K(\hat{\phi}_i, \phi_j)$$

for all local basis functions $\hat{\phi}_i$ and ϕ_j on K

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:

$$A_i^K = A_{i\alpha}^0 G_K^\alpha$$

- 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 $A_{i\alpha}^0 G_K^\alpha$

Example 1: the mass matrix

- Form:

$$a(v,u) = \int_\Omega v(x)u(x)\,dx$$

- Evaluation:

$$A_i^K = \int_K \phi_{i_1}\phi_{i_2}\,dx$$

$$= \det F_K' \int_{K_0} \Phi_{i_1}(X)\Phi_{i_2}(X)\,dX = A_i^0 G_K$$

with $A_i^0 = \int_{K_0} \Phi_{i_1}(X)\Phi_{i_2}(X)\,dX$ and $G_K = \det F_K'$

Example 2: 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 $G_K^\alpha = \det F_K' \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}$

Example 3: Navier–Stokes

- Form:

$$a(v,u) = \int_\Omega v \cdot (w \cdot \nabla) u \, dx$$

- Evaluation:

$$A_i^K = \int_K \phi_{i_1} \cdot (w \cdot \nabla) \phi_{i_2} \, dx$$

$$= \det F_K' \frac{\partial X_{\alpha_3}}{\partial x_{\alpha_1}} w_{\alpha_2} \int_{K_0} \Phi_{i_1}[\beta] \Phi_{\alpha_2}[\alpha_1] \frac{\partial \Phi_{i_2}[\beta]}{\partial X_{\alpha_3}} \, dX = A_{i\alpha}^0 G_K^\alpha$$

with $A_{i\alpha}^0 = \int_{K_0} \Phi_{i_1}[\beta] \Phi_{\alpha_2}[\alpha_1] \frac{\partial \Phi_{i_2}[\beta]}{\partial X_{\alpha_3}} \, dX$ and

$G_K^\alpha = \det F_K' \frac{\partial X_{\alpha_3}}{\partial x_{\alpha_1}} w_{\alpha_2}$

FFC: Example form file (Poisson)

```
# The bilinear form a(v, u) and linear form L(v) for
# Poisson's equation, 2D version
#
# Compile this form with FFC: ffc Poisson2D.form

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

v = BasisFunction(element)
u = BasisFunction(element)
f = Function(element)

a = dot(grad(u), grad(v))*dx
L = f*v*dx
```

FFC: Example output (Poisson)

Note the factors 0.5 which are simply computed integrals of the integrand 1 on a triangle.

```
...

void eval(real block[], const AffineMap& map) const
{
  // Compute geometry tensors
  const real G0_0_0 = map.det*map.g00*map.g00 + map.det*map.g01*map.g01;
  const real G0_0_1 = map.det*map.g00*map.g10 + map.det*map.g01*map.g11;
  const real G0_1_0 = map.det*map.g10*map.g00 + map.det*map.g11*map.g01;
  const real G0_1_1 = map.det*map.g10*map.g10 + map.det*map.g11*map.g11;

  // Compute element tensor
  block[0] = 4.999999999999998e-01*G0_0_0 + 4.999999999999997e-01*G0_0_1 + ...
  block[1] = -4.999999999999998e-01*G0_0_0 - 4.999999999999997e-01*G0_0_1;
  block[2] = -4.999999999999997e-01*G0_1_0 - 4.999999999999996e-01*G0_1_1;
  ...
```

DOLFIN, C++ interface for solving PDEs

Assembly in DOLFIN

```
// Iterate over all cells in the mesh
for (CellIterator cell(mesh); !cell.end(); ++cell)
{
  // Update affine map
  map.update(*cell);

  // Update form
  a.update(map);

  // Compute maps from local to global degrees of freedom
  test_element.dofmap(test_dofs, *cell, mesh);
  trial_element.dofmap(trial_dofs, *cell, mesh);


  // Compute element matrix
  a.eval(block, map);

  // Add element matrix to global matrix
  A.add(block, test_dofs, m, trial_dofs, n);
}
```

Demonstration

- ...