Benchmark Results for the FEniCS Form Compiler

Anders Logg

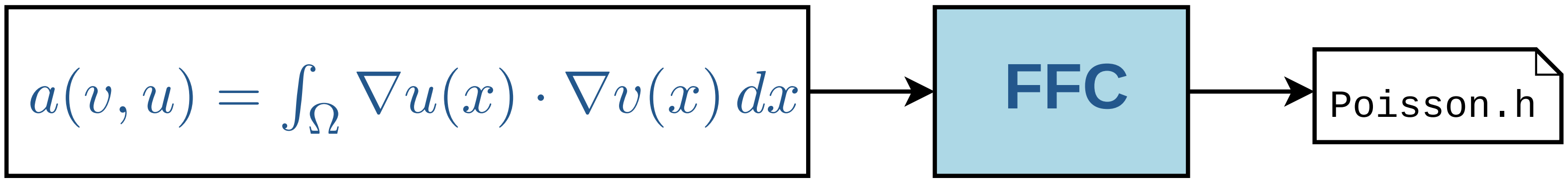
logg@tti-c.org

Toyota Technological Institute at Chicago

Acknowledgements: Johan Hoffman, Johan Jansson, Claes Johnson, Matthew Knepley, Robert C. Kirby, Ridgway Scott

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++ code

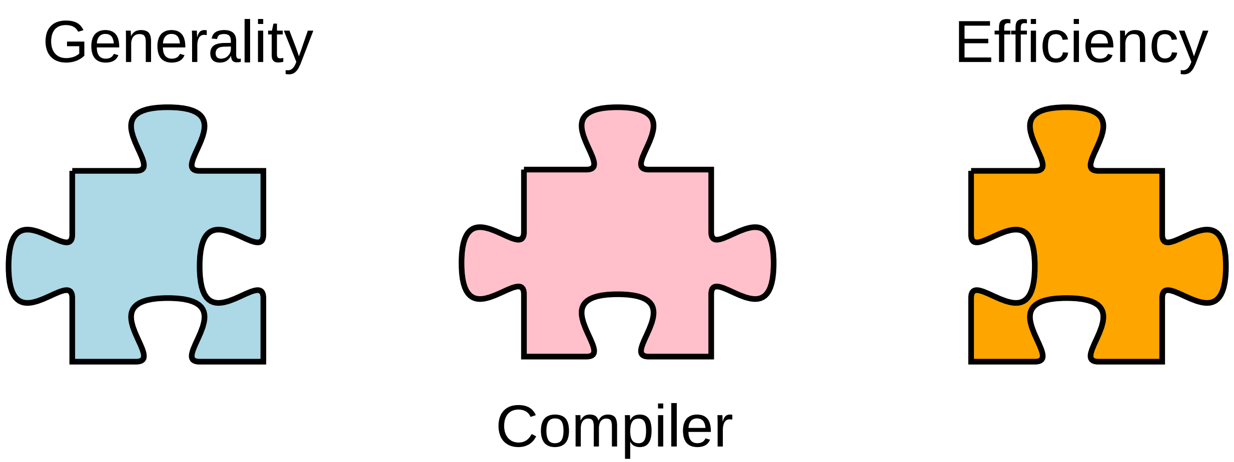



```
>> ffc [-l language] poisson.form
```

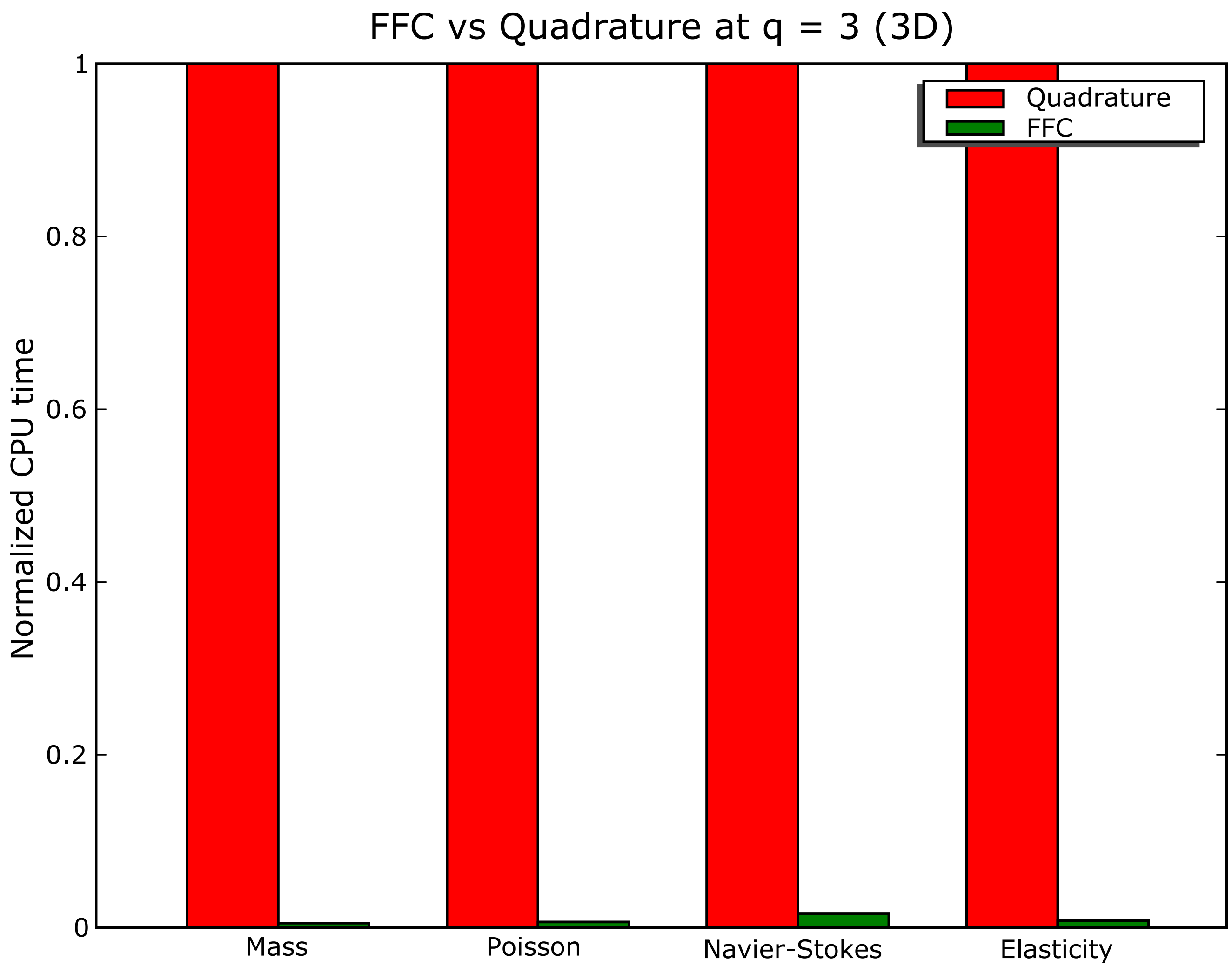
Design goals

- Any form
- Any element
- Maximum efficiency

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

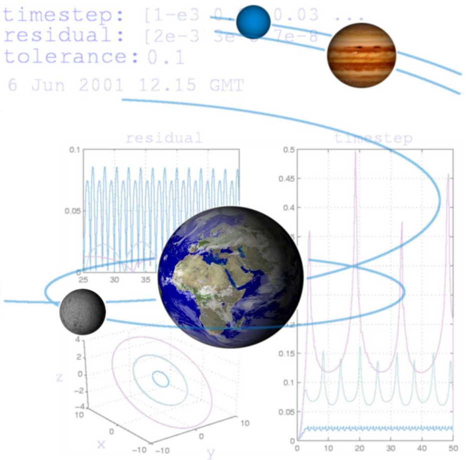


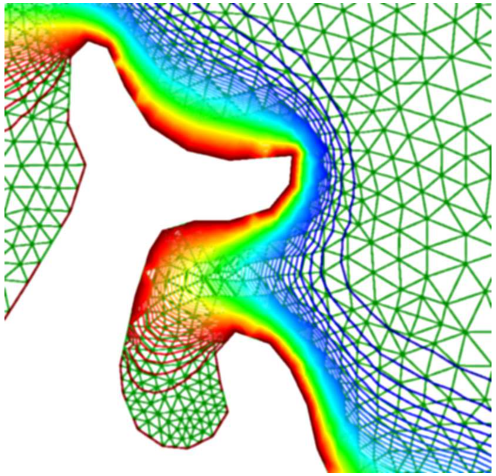
Impressive speedups (typically a factor 10–1000)



Outline

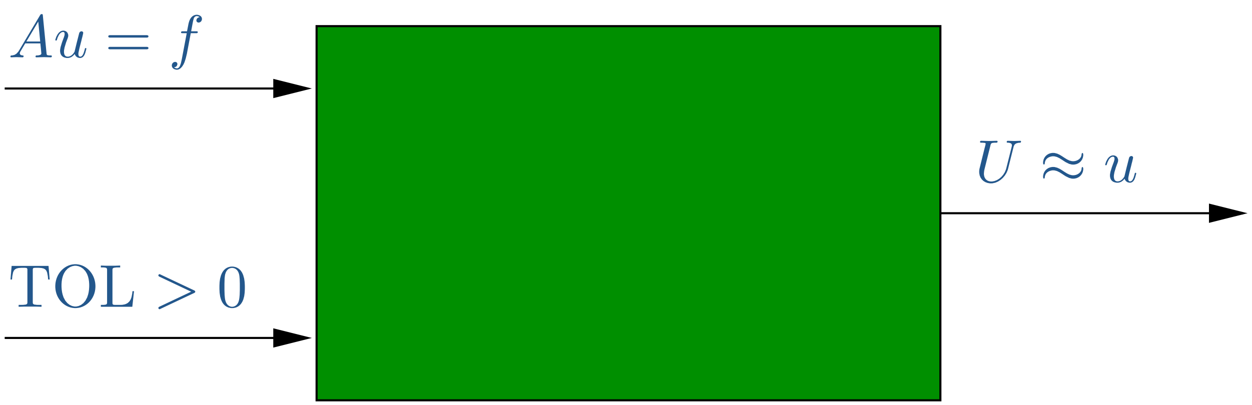
- Background
- FFC as a component of FEniCS
- Complexity of form evaluation
- Benchmark results





Background

Main goal: the Automation of CMM



- Input: Model $Au = f$ and tolerance $\mathrm{TOL} > 0$
- Output: Solution $U \approx u$ satisfying $\|U - u\| \leq \mathrm{TOL}$

- Produce a solution U satisfying a given accuracy requirement, using a minimal amount of work

Algorithm: the (Galerkin) finite element method

- Input: Variational problem $a(v,u) = L(v)$ for all v and discrete representation $(V, \hat{V})$
- Output: Discrete system $F(x) = 0$

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 $a : \hat{V} \times V \rightarrow \mathbb{R}$ a *bilinear form* and $L : \hat{V} \rightarrow \mathbb{R}$ 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 $V = \mathrm{span}\{\phi_i\}_{i=1}^M$ and $\hat{V} = \mathrm{span}\{\hat{\phi}_i\}_{i=1}^M$, we obtain the linear system

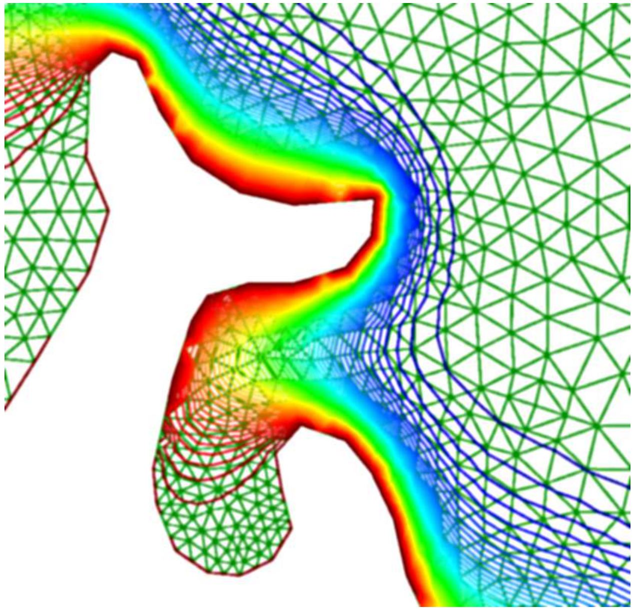
$$Ax = b$$

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

$$A_{ij}^K = a_K(\hat{\phi}_i, \phi_j)$$

for all local basis functions $\hat{\phi}_i$ and ϕ_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 Ω

- 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

$$A_i^K = a_K(\phi_{i_1}, \phi_{i_2}, \ldots, \phi_{i_r})$$

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:

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

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

Basic idea:

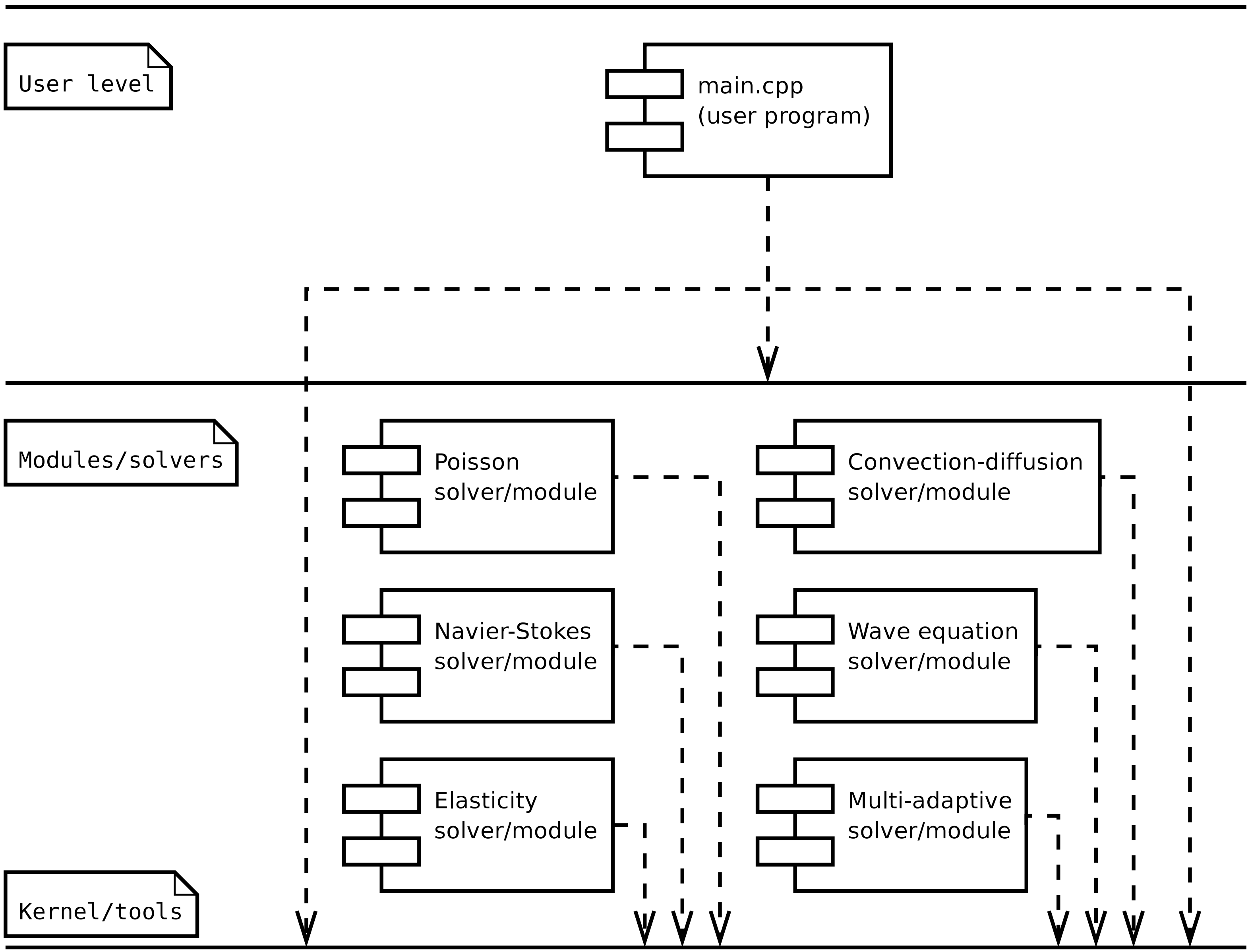
- Precompute A^0 at compile-time
- Generate optimal code for run-time evaluation of G_K and the product $A_{i\alpha}^0 G_K^\alpha$

FFC as a component of FEniCS

FEniCS components

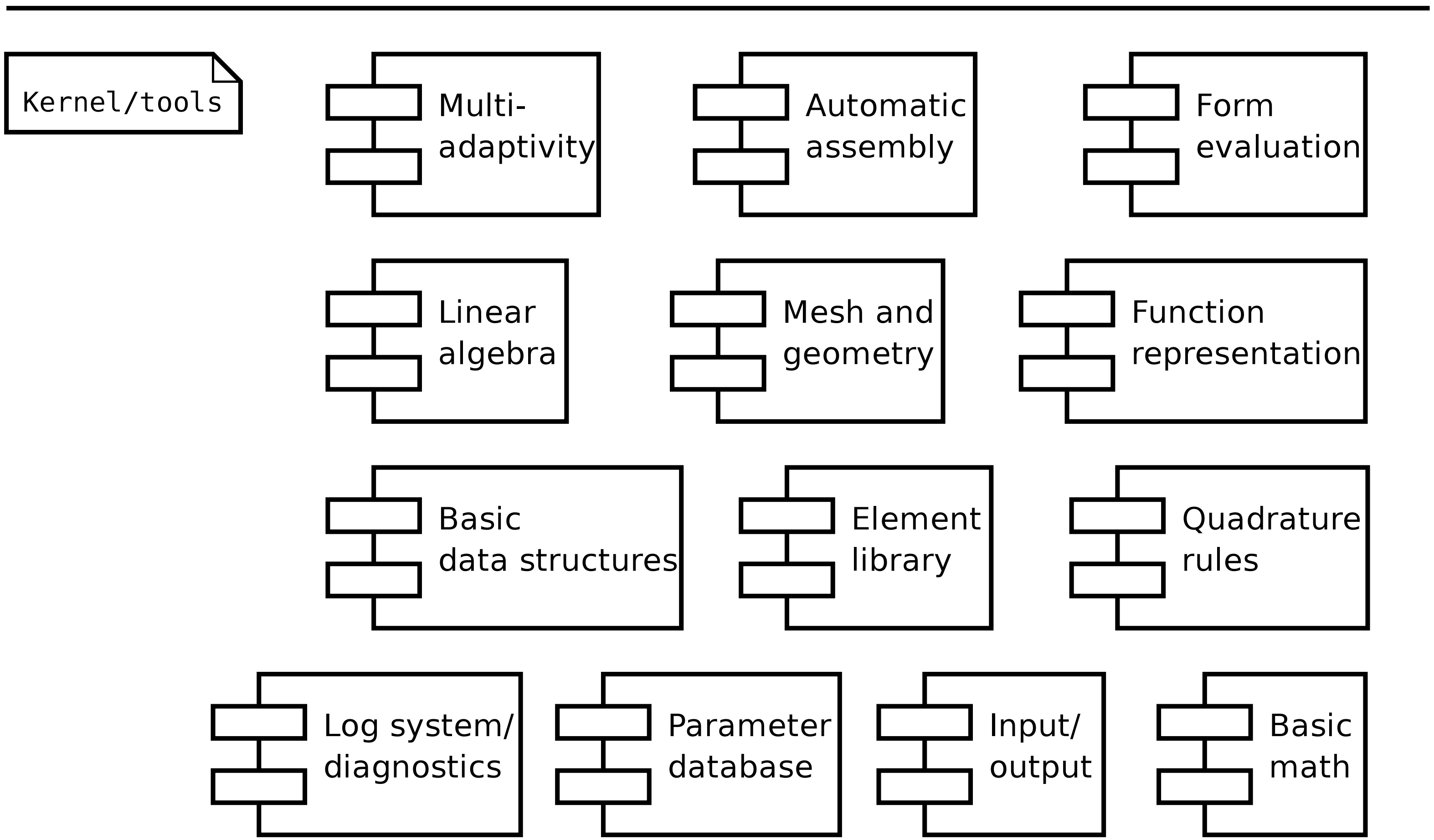
- **DOLFIN**, the C++ interface of FEniCS
  - Hoffman, Jansson, Logg, et al.
- **FErari**, optimized form evaluation
  - Kirby, Knepley, Scott
- **FFC**, the FEniCS Form Compiler
  - Logg
- **FIAT**, automatic generation of finite elements
  - Kirby, Knepley
- **Ko**, simulation of mechanical systems
  - Jansson
- **Puffin**, light-weight version for Octave/MATLAB
  - Hoffman, Logg
- (PETSc)

DOLFIN

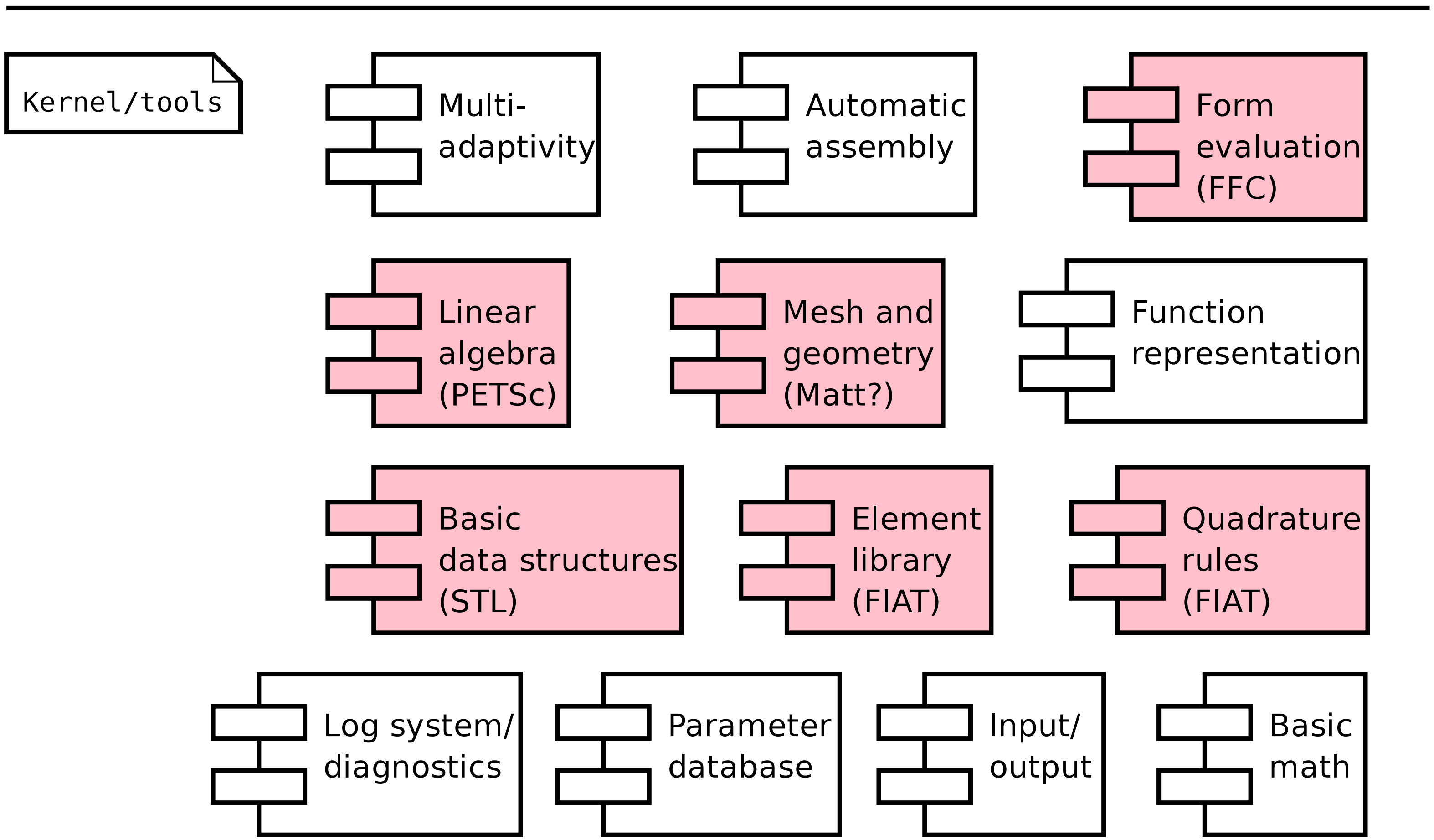


User level
main.cpp
(user program)
Modules/solvers
Poisson
solver/module
Convection-diffusion
solver/module
Navier-Stokes
solver/module
Wave equation
solver/module
Elasticity
solver/module
Multi-adaptive
solver/module
Kernel/tools

DOLFIN

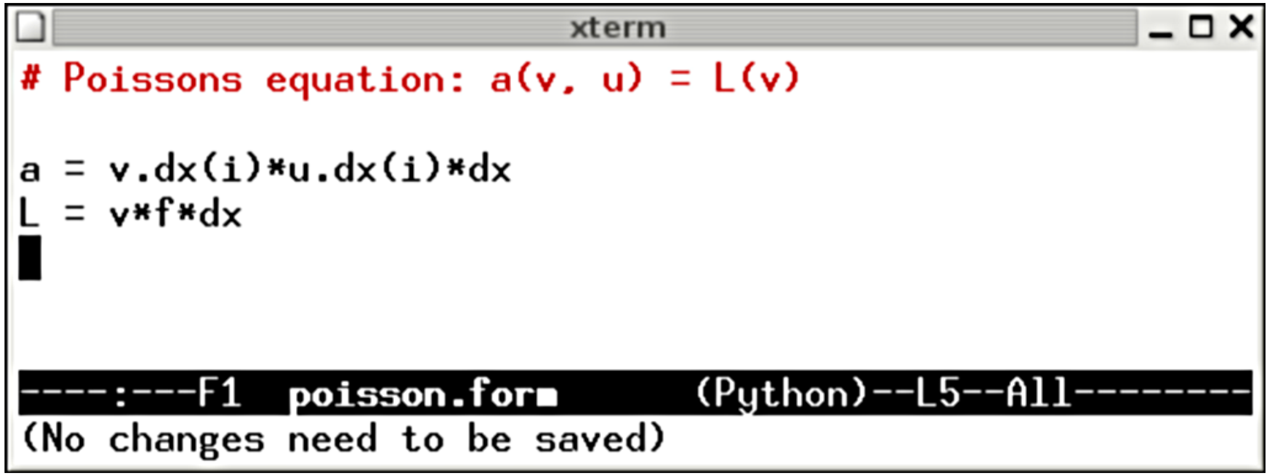

Kernel/tools
Multi-adaptivity
Automatic assembly
Form evaluation
Linear algebra
Mesh and geometry
Function representation
Basic data structures
Element library
Quadrature rules
Log system/ diagnostics
Parameter database
Input/ output
Basic math

DOLFIN becomes an interface



Basic usage: compiling a form

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



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

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

This will generate C++ code (`Poisson.h`) for **DOLFIN**

Basic usage: solving the PDE

```
#include <dolfin.h>
#include "Poisson.h"
using namespace dolfin;
int main()
{
  Poisson::BilinearForm a;
  Poisson::LinearForm L(f);

  Matrix A;
  Vector x, b;
  FEM::assemble(a, L, A, b, mesh, bc);
  GMRES::solve(A, x, b);

  Function u(x, mesh, a.trial());
  File file("poisson.m");
  file << u;

  return 0;
}
```

Complexity of form evaluation

Basic assumptions and notation

- Complexity of computing the element tensor A_i^K?
- Compare tensor-contraction (FFC) to quadrature
- Basic assumptions:
  - Bilinear form: $|i| = 2$
  - Exact integration of forms
- Notation:
  - q: polynomial order of basis functions
  - p: total polynomial order of form
  - d: dimension of Ω
  - n: dimension of function space ($n \sim q^d$)
  - N: number of quadrature points ($N \sim p^d$)
  - n_C: number of coefficients
  - n_D: number of derivatives

Complexity of tensor contraction

- Need to evaluate $A_i^K = A_{i\alpha}^0 G_K^\alpha$
- Rank of G_K^α is $n_C + n_D$
- Number of elements of A_i^K is n^2
- Number of elements of G_K^α is $n^{n_C} d^{n_D}$

- Total cost:

$$T_C \sim n^2 n^{n_C} d^{n_D} \sim (q^d)^2 (q^d)^{n_C} d^{n_D} \sim \underline{q^{2d+n_C d} d^{n_D}}$$

Complexity of quadrature

- Need to evaluate A_i^K at $N \sim p^d$ quadrature points
- Total order of integrand is $p = 2q + n_C q - n_D$
- Cost of evaluating integrand is $\sim n_C + n_D d + 1$

- Total cost:

$$\begin{aligned} T_Q &\sim n^2 N(n_C + n_D d + 1) \sim (q^d)^2 p^d (n_C + n_D d + 1) \\ &\sim \underline{q^{2d}(2q + n_C q - n_D)^d (n_C + n_D d + 1)} \end{aligned}$$

Tensor contraction vs quadrature

$$\begin{aligned} T_C &\sim q^{2d+n_C d} d^{n_D} \\ T_Q &\sim q^{2d}(2q + n_C q - n_D)^d (n_C + n_D d + 1) \end{aligned}$$

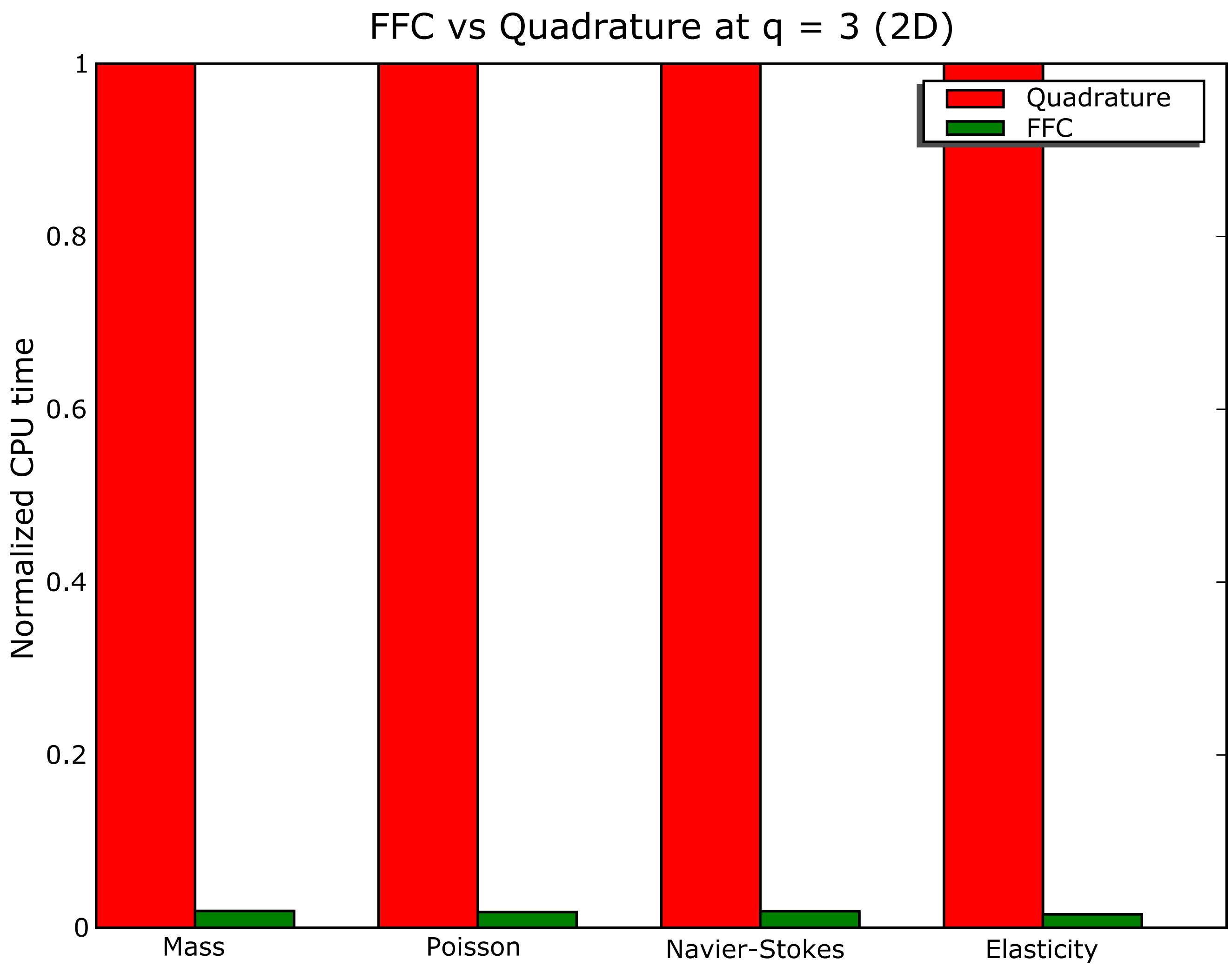
Speedup:

$$T_Q/T_C \sim \frac{(2q + n_C q - n_D)^d (n_C + n_D d + 1)}{q^{n_C d} d^{n_D}}$$

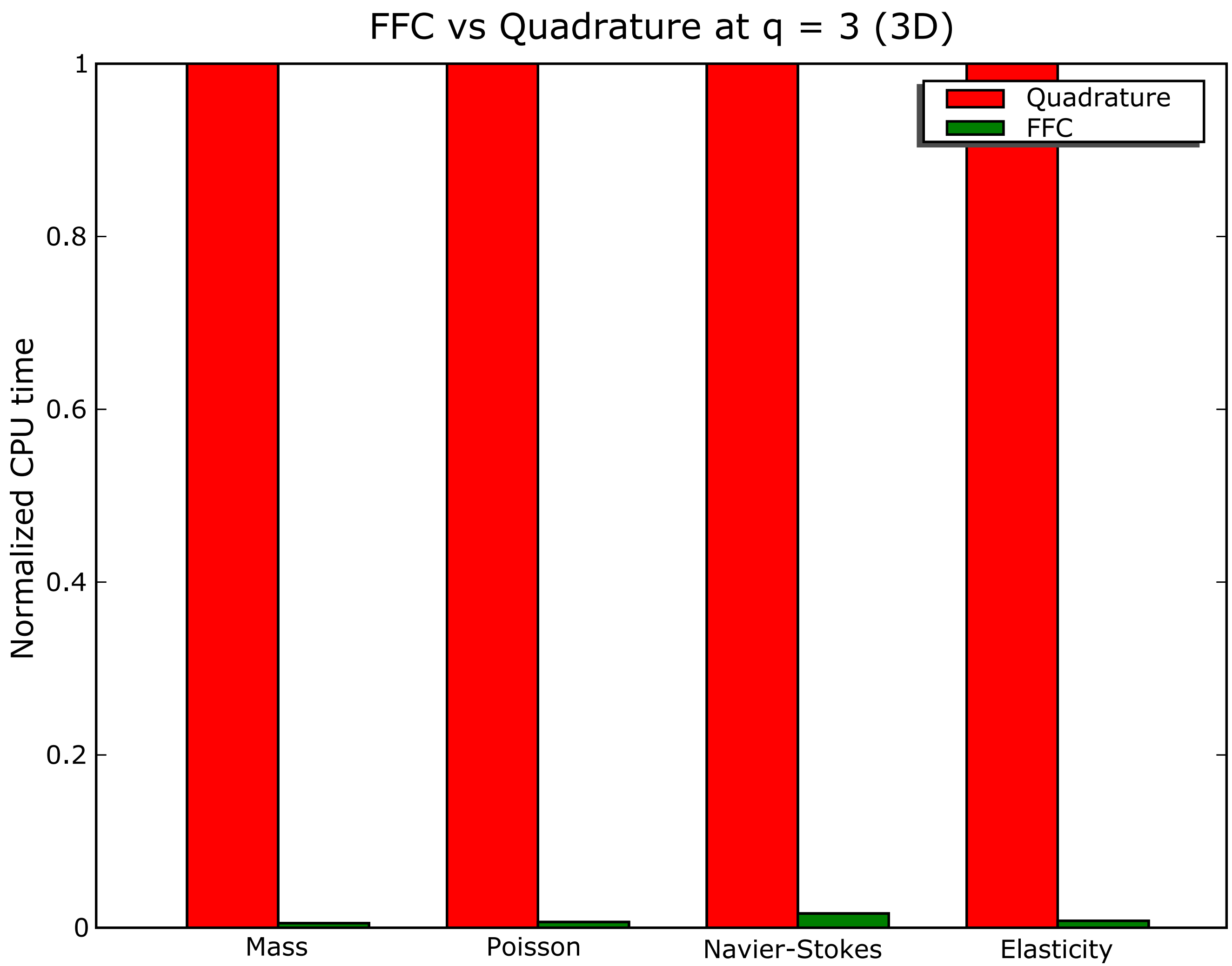
- Rule of thumb: tensor contraction wins for $n_C = 0, 1$
- Mass matrix ($n_C = n_D = 0$): $T_Q/T_C \sim (2q)^d$
- Poisson ($n_C = 0, n_D = 2$): $T_Q/T_C \sim (2q - 2)^d (2d + 1)/d^2$
- Not clear that tensor contraction wins for the stabilization term of Navier–Stokes: $(w \cdot \nabla) u \, (w \cdot \nabla) v$
- Need an intelligent system that can do both!

Benchmark results

Impressive speedups



Impressive speedups



Test case 1: the mass matrix

- Mathematical notation:

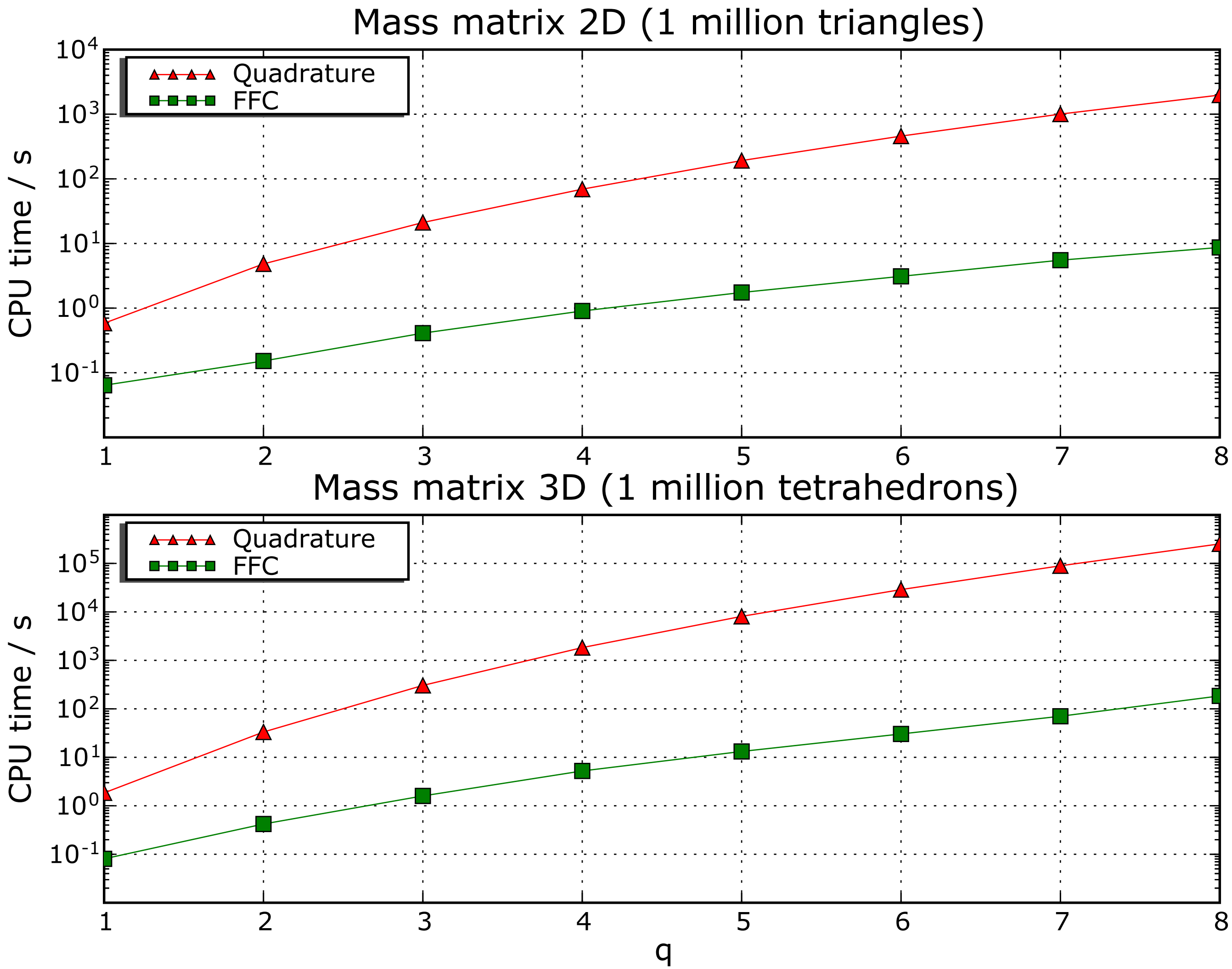
$$a(v, u) = \int_\Omega uv \, dx$$

- FFC implementation:

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

a = u*v*dx
```

Results


Mass matrix 2D (1 million triangles)
Quadrature
FFC
CPU time / s
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
1 2 3 4 5 6 7 8
Mass matrix 3D (1 million tetrahedrons)
Quadrature
FFC
CPU time / s
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
1 2 3 4 5 6 7 8
q

Test case 2: Poisson

- Mathematical notation:

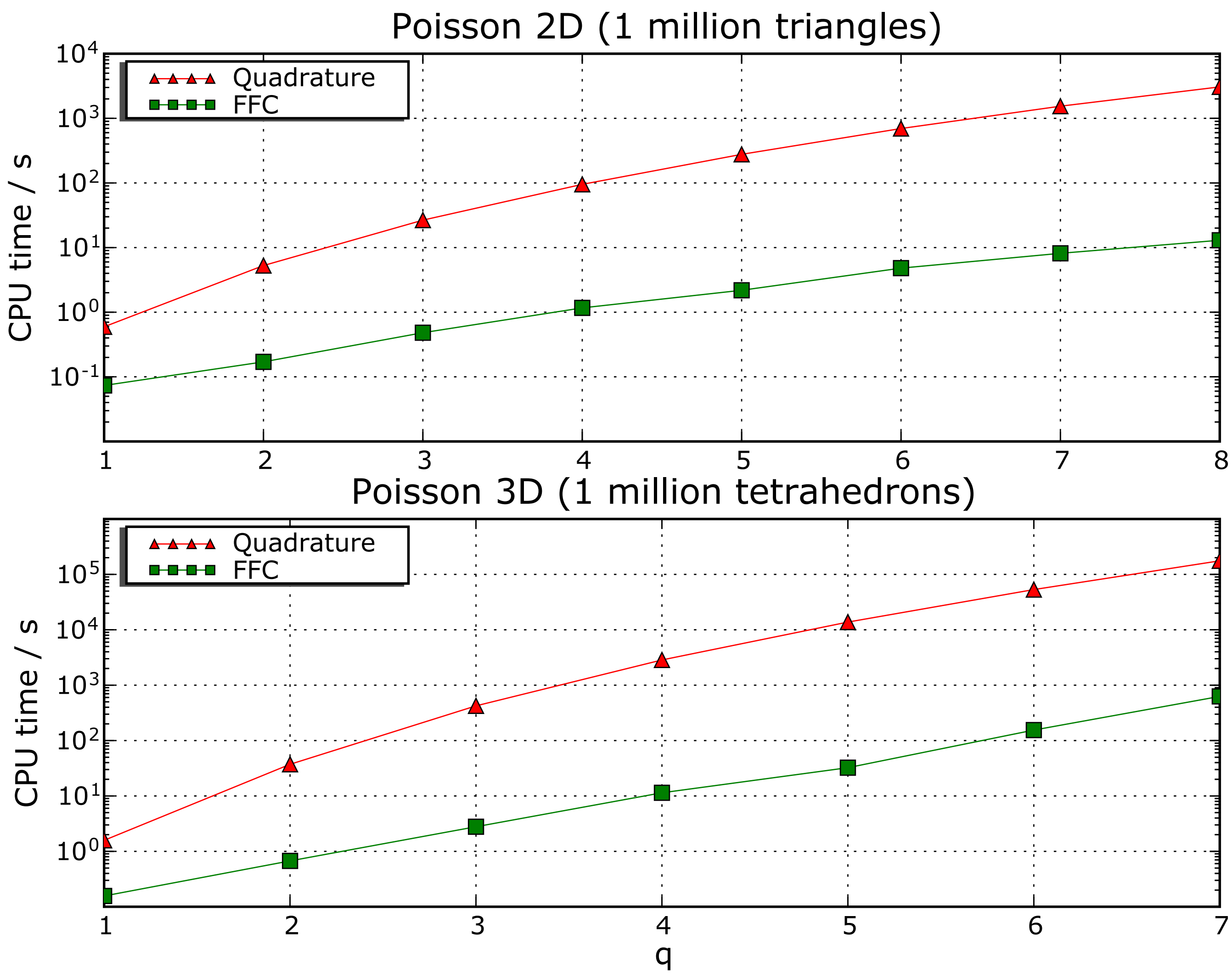
$$a(v, u) = \int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega \sum_{i=1}^{d} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \, dx$$

- FFC implementation:

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

a = u.dx(i)*v.dx(i)*dx
```

Results



Poisson 2D (1 million triangles)
Quadrature
FFC
CPU time / s
10^4
10^3
10^2
10^1
10^0
10^{-1}
1 2 3 4 5 6 7 8
Poisson 3D (1 million tetrahedrons)
Quadrature
FFC
CPU time / s
10^5
10^4
10^3
10^2
10^1
10^0
1 2 3 4 5 6 7
q

Test case 3: Navier–Stokes

- Mathematical notation:

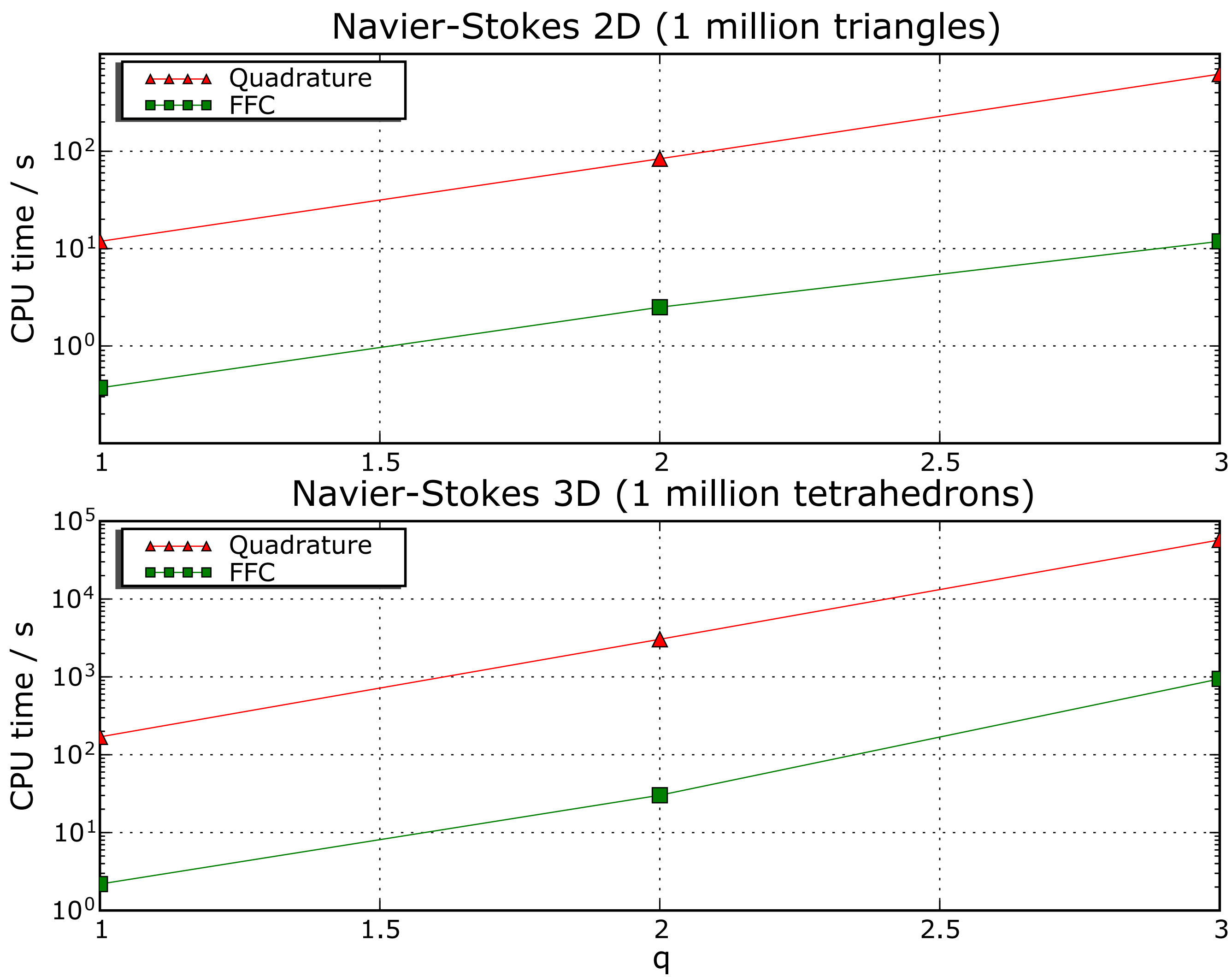
$$a(v, u) = \int_\Omega (w \cdot \nabla u) v \, dx = \int_\Omega \sum_{i=1}^{d} \sum_{j=1}^{d} w_j \frac{\partial u_i}{\partial x_j} v_i \, dx$$

- FFC implementation:

```
v = BasisFunction(element)
u = BasisFunction(element)
w = Function(element)

a = w[j]*u[i].dx(j)*v[i]*dx
```

Results


Navier-Stokes 2D (1 million triangles)
Quadrature
FFC
CPU time / s
10^2
10^1
10^0
1
1.5
2
2.5
3
Navier-Stokes 3D (1 million tetrahedrons)
Quadrature
FFC
CPU time / s
10^5
10^4
10^3
10^2
10^1
10^0
1
1.5
2
2.5
3
q

Test case 4: Linear elasticity

- Mathematical notation:

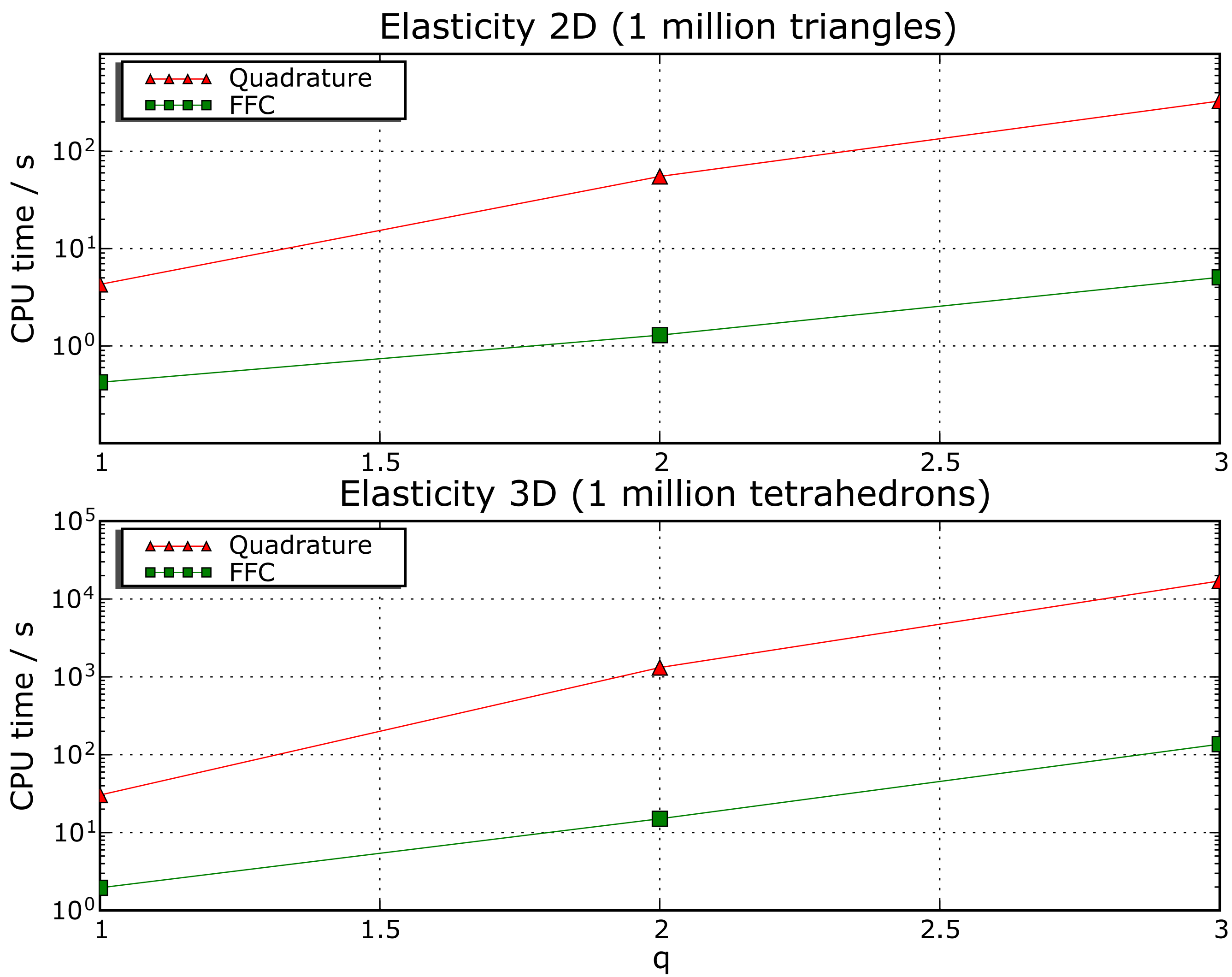
$$
\begin{aligned}
a(v,u) &= \int_\Omega \tfrac{1}{4}(\nabla u + (\nabla u)^\top) : (\nabla v + (\nabla v)^\top)\, dx \\
&= \int_\Omega \sum_{i=1}^d \sum_{j=1}^d \tfrac{1}{4}(\tfrac{\partial u_i}{\partial x_j} + \tfrac{\partial u_j}{\partial x_i})(\tfrac{\partial v_i}{\partial x_j} + \tfrac{\partial v_j}{\partial x_i})\, dx
\end{aligned}
$$

- FFC implementation:

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

a = 0.25*(u[i].dx(j) + u[j].dx(i)) * \
         (v[i].dx(j) + v[j].dx(i)) * dx
```

Results



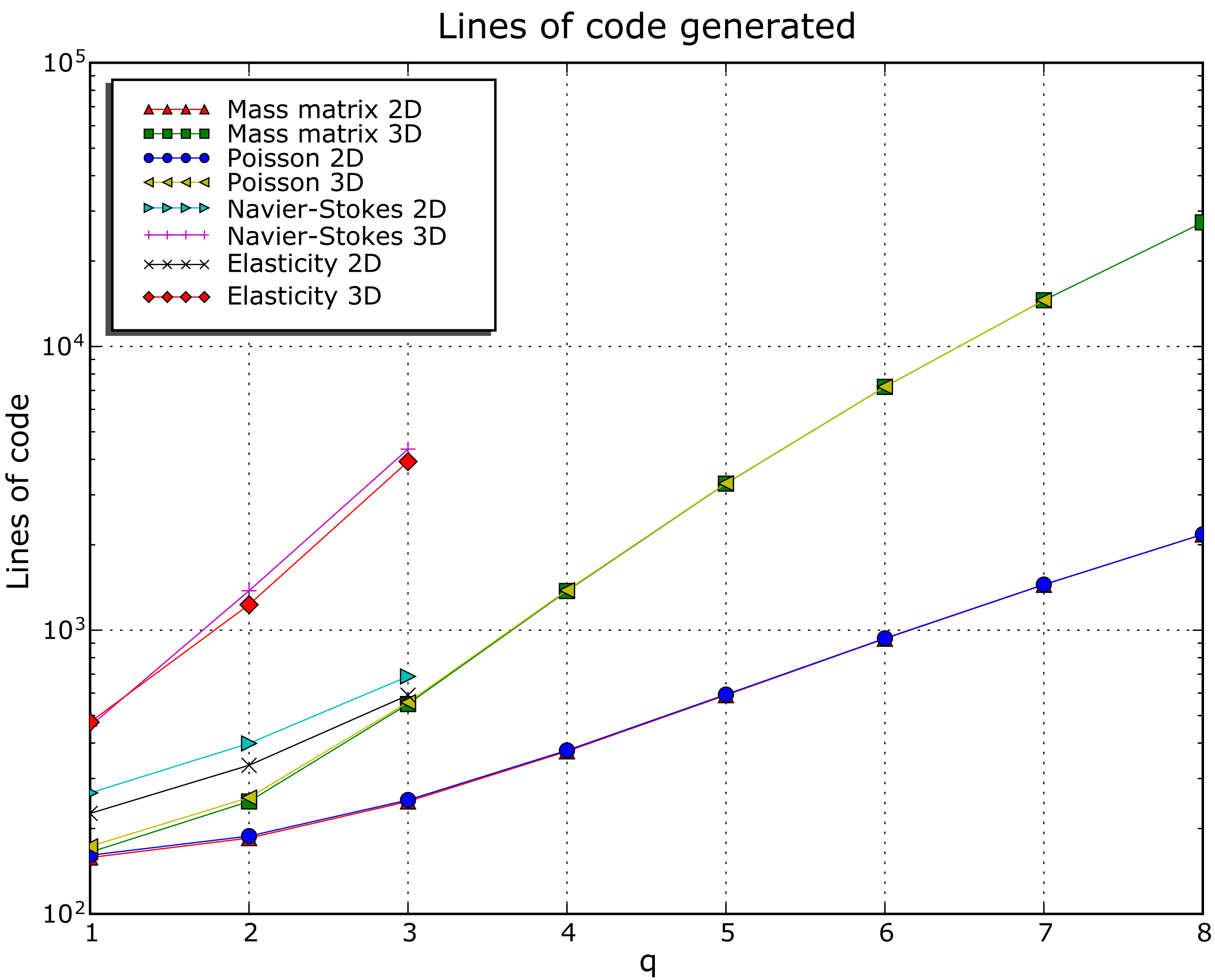
Elasticity 2D (1 million triangles)
Quadrature
FFC
CPU time / s
10^2
10^1
10^0
1
1.5
2
2.5
3
Elasticity 3D (1 million tetrahedrons)
Quadrature
FFC
CPU time / s
10^5
10^4
10^3
10^2
10^1
10^0
1
1.5
2
2.5
3
q

Speedup

Form	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$	$q = 6$	$q = 7$	$q = 8$
Mass 2D	9.1	31.8	51.5	76.7	109.9	147.8	182.2	227.9
Mass 3D	23.0	79.0	190.5	350.6	612.1	951.0	1270.9	1368.5
Poisson 2D	8.1	30.9	55.2	81.6	126.9	144.6	189.0	236.1
Poisson 3D	10.1	55.4	152.1	249.9	425.2	343.8	280.6	—
Navier–Stokes 2D	32.0	33.5	52.3	—	—	—	—	—
Navier–Stokes 3D	77.7	100.7	60.9	—	—	—	—	—
Elasticity 2D	10.1	42.7	64.8	—	—	—	—	—
Elasticity 3D	15.5	87.5	125.0	—	—	—	—	—

- Impressive speedups but far from optimal
- Data access costs more than flops
- Solution: build arrays and call BLAS (Level 2 or 3)

Code bloat



Lines of code generated
Mass matrix 2D
Mass matrix 3D
Poisson 2D
Poisson 3D
Navier-Stokes 2D
Navier-Stokes 3D
Elasticity 2D
Elasticity 3D
10^5
10^4
10^3
10^2
Lines of code
1
2
3
4
5
6
7
8
q

Future directions for FFC

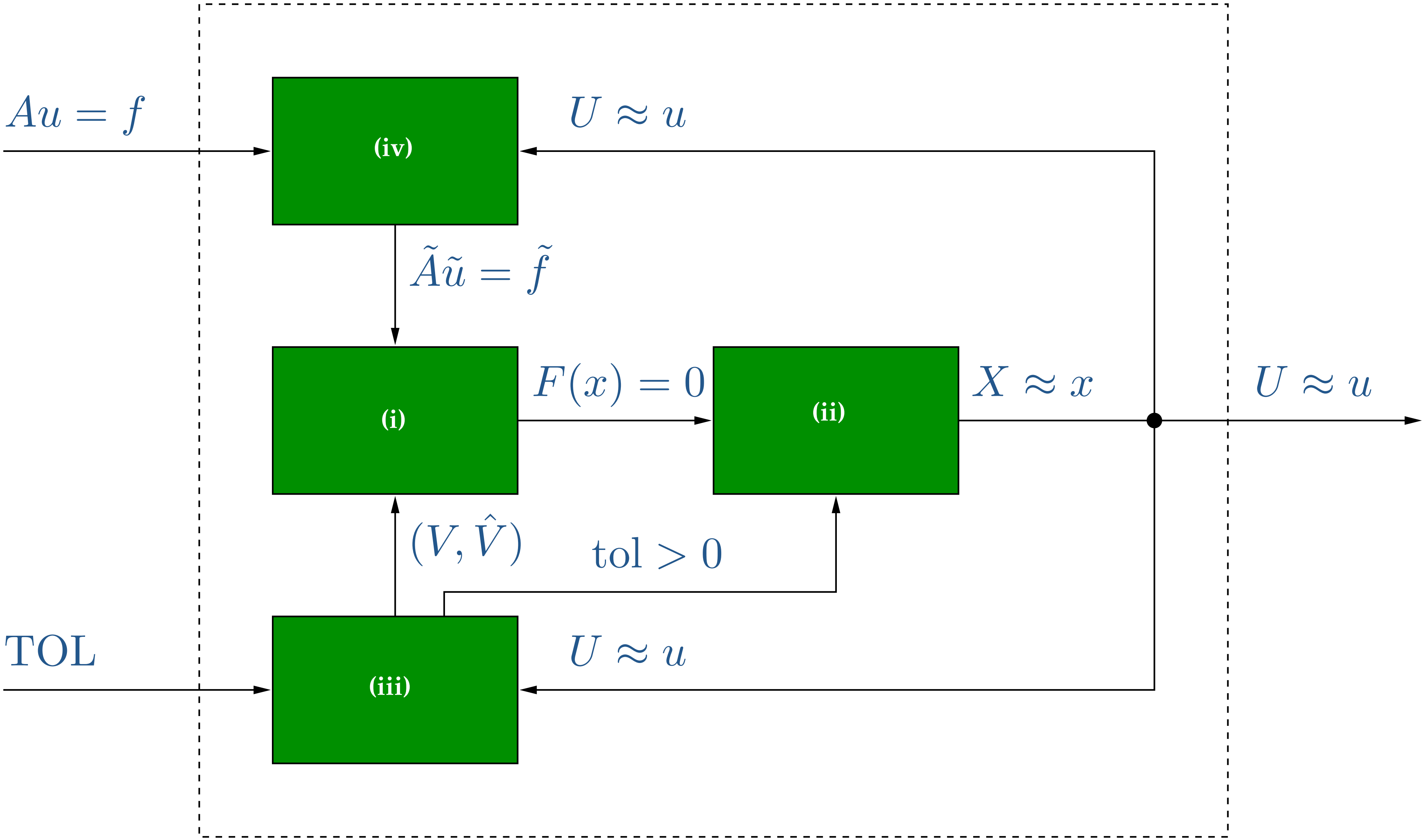
- Improve speed of compiler
- Add remaining features for full support of general Lagrange elements (dof map still missing for $q > 1$)
- Build arrays and call BLAS (Level 2 or 3) when appropriate
- Add support for new elements to FIAT/FFC: *Crouzeix–Raviart*, *Raviart–Thomas*, *Nedelec*, *Brezzi–Douglas–Marini*, *Brezzi–Douglas–Fortin–Marini*, *Arnold–Winther*, *Taylor–Hood*, . . .
- And yes, I need to write a manual

- Further information:

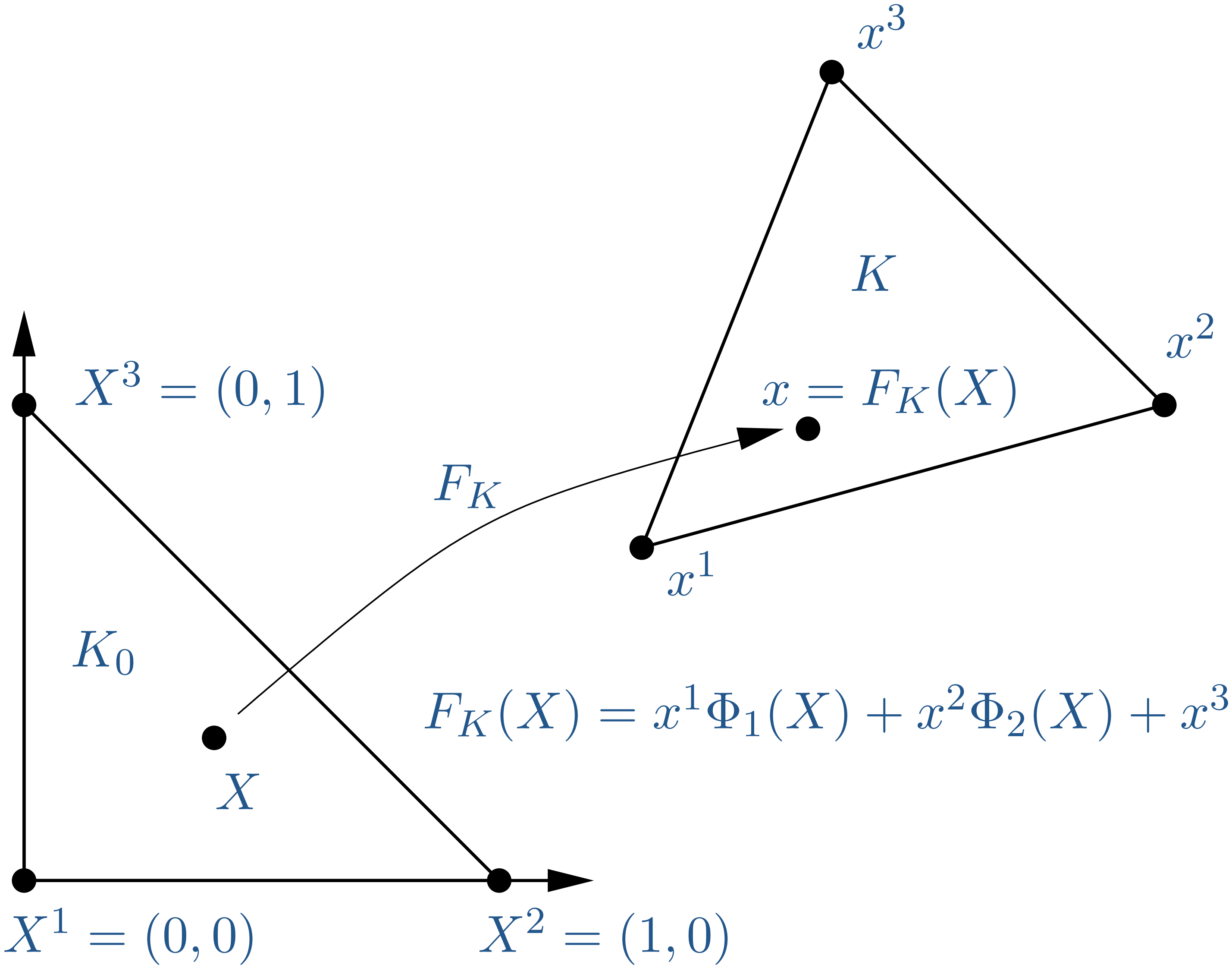
  `http://www.fenics.org/ffc/`

Additional slides

The Automation of CMM



The (affine) map $F_K : K_0 \to K$



$$F_K(X) = x^1\Phi_1(X) + x^2\Phi_2(X) + x^3\Phi_3(X)$$

Example 1: the mass matrix

- Form:

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

- Evaluation:

$$\begin{aligned} A_i^K &= \int_K \phi_{i_1}\phi_{i_2}\,dx \\ &= \det F_K' \int_{K_0} \Phi_{i_1}(X)\Phi_{i_2}(X)\,dX = A_i^0 G_K \end{aligned}$$

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:

$$\begin{aligned} A_i^K &= \int_K \nabla \phi_{i_1}(x) \cdot \nabla \phi_{i_2}(x)\, dx \\ &= \det F_K' \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta} \int_{K_0} \frac{\partial \Phi_{i_1}}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}}{\partial X_{\alpha_2}}\, dX = A^0_{i\alpha} G_K^\alpha \end{aligned}$$

with $A^0_{i\alpha} = \int_{K_0} \frac{\partial \Phi_{i_1}}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}}{\partial X_{\alpha_2}}\, dX$ 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:

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

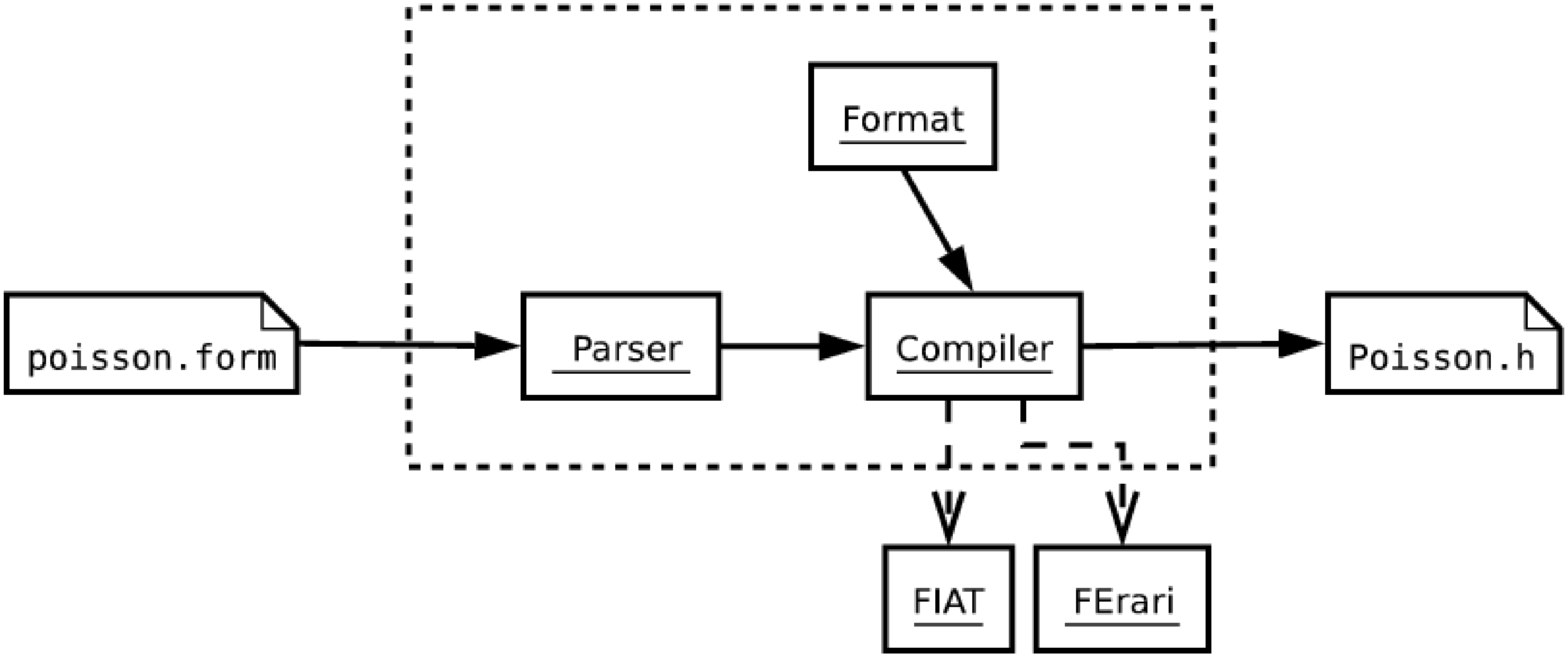
$$= \det F_K' \frac{\partial X_{\alpha_3}}{\partial x_{\alpha_1}} w_{\alpha_2} \int_{K_0} \Phi_{i_1}[\beta] \Phi_{\alpha_2}[\alpha_1] \frac{\partial \Phi_{i_2}[\beta]}{\partial X_{\alpha_3}} \, dX = A_{i\alpha}^0 G_K^\alpha$$

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

Example: Poisson with $\mathcal{P}^2$ elements in 2D

3 3 3 3	1 0 1 0	0 1 0 1	0 0 0 0	0 -4 0 -4	-4 0 -4 0
1 1 0 0	3 0 0 0	0 -1 0 0	0 4 0 0	0 0 0 0	-4 -4 0 0
0 0 1 1	0 0 -1 0	0 0 0 3	0 0 4 0	0 0 -4 -4	0 0 0 0
0 0 0 0	0 0 4 0	0 4 0 0	8 4 4 8	-8 -4 -4 0	0 -4 -4 -8
0 0 -4 -4	0 0 0 0	0 -4 0 -4	-8 -4 -4 0	8 4 4 8	0 4 4 0
-4 -4 0 0	-4 0 -4 0	0 0 0 0	0 -4 -4 -8	0 4 4 0	8 4 4 8

Components



Representation of forms

- Need to build a data structure to represent forms:

$$V_B = \{\frac{\partial^{|\cdot|}\phi_{(\cdot)}}{\partial x_{(\cdot)}} : \quad \phi : \mathbb{N} \rightarrow V\}$$

$$V_P = \{c \prod v : \quad v \in V_B, \quad c \in \mathbb{R}\}$$

$$V_S = \{\sum v : \quad v \in V_P\}$$

Note: $V_B \subset V_P \subset V_S \subset \{v : \mathbb{N}^r \times \Omega \rightarrow \mathbb{R}\}$

- V_S is an algebra (a vector space with multiplication):

$$v, w \in V_S \Rightarrow v + w \in V_S$$

$$v, w \in V_S \Rightarrow vw \in V_S$$

Evaluation of forms

For any $v = \sum c \prod \partial^{|\cdot|} \phi_{(\cdot)} / \partial x_{(\cdot)}$, we have

$$
\begin{aligned}
A_i^K &= a_K(\phi_{i_1}, \phi_{i_2}, \ldots, \phi_{i_n}) = \int_K v_i \, dx \\
&= \sum \left(\int_K c \prod \partial^{|\cdot|} \phi_{(\cdot)} / \partial x_{(\cdot)} \, dx \right)_i \\
&= \sum c'_\alpha \left(\int_{K_0} \prod \partial^{|\cdot|} \Phi_{(\cdot)} / \partial X_{(\cdot)} \, dX \right)_{i\alpha} \\
&= \sum A^0_{i\alpha} G_K^\alpha,
\end{aligned}
$$

where $A^0_{i\alpha} = \left(\int_{K_0} \prod \partial^{|\cdot|} \Phi_{(\cdot)} / \partial X_{(\cdot)} \, dX \right)_{i\alpha}$ and $G_K^\alpha = c'_\alpha$

Implementation (click to return)

- Build a class hierarchy: `BasisFunction`, `Product`, `Sum` corresponding to the spaces $V_B \subset V_P \subset V_S$
- Overload operators `+`, `-`, `*`, `[`$\cdot$`]`, `.dx(`$\cdot$`)` :

$$\begin{cases} + : V_B \times V_B \longrightarrow V_S \\ + : V_B \times V_P \longrightarrow V_S \\ \quad \dots \\ + : V_S \times V_S \longrightarrow V_S \end{cases} \qquad \begin{cases} * : V_B \times V_B \longrightarrow V_P \\ * : V_B \times V_P \longrightarrow V_P \\ \quad \dots \\ * : V_S \times V_S \longrightarrow V_S \end{cases}$$

- Examples:

```
a = v*u*dx
a = v.dx(i)*u.dx(i)*dx
a = v[i]*w[j]*u[i].dx(j)*dx
L = v*f*dx + v*10*ds
```