Software Components for Solving PDEs

Anders Logg Ola Skavhaug

Simula Research Laboratory

June 6 2007

Acknowledgments: Martin Sandve Alnæs, Johan Hoffman, Johan Jansson, Claes Johnson, Robert C. Kirby, Matthew G. Knepley, Hans Petter Langtangen, Kent-Andre Mardal, Kristian Oelgaard, Marie Rognes, L. Ridgway Scott, Andy R. Terrel, Garth N. Wells, Åsmund Ødegård, Magnus Vikstrøm.

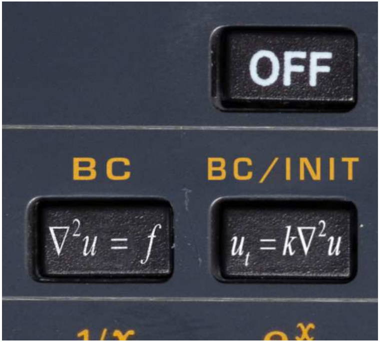
The FEniCS project

- Initiated in 2003
- Free software for the Automation of CMM
- Collaborators (in order of appearance):
  - University of Chicago
  - Argonne National Laboratory
  - Delft University of Technology
  - Royal Institute of Technology (KTH)
  - Simula Research Laboratory
  - Texas Tech
- Automate the solution of differential equations
- www.fenics.org

Automate the solution of differential equations

- Build a calculator
- One button for each equation?

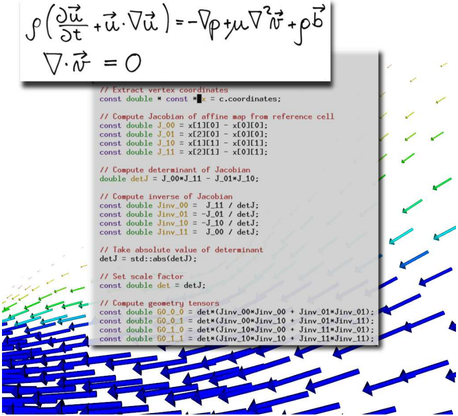
- Too many buttons!



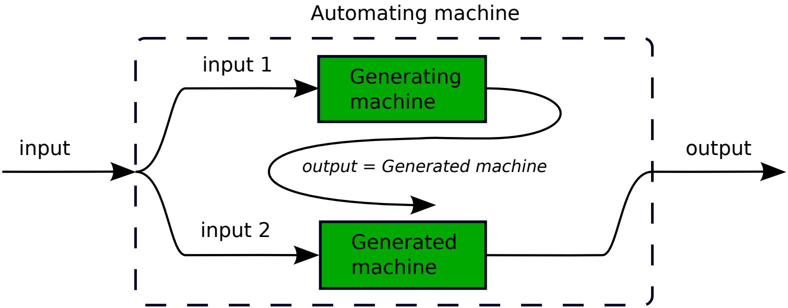
Automate the solution of differential equations

- Input: Differential equation
- Generate computer code
- Compute solution

- Build a calculator for each equation!
- Automate the generation of calculators!



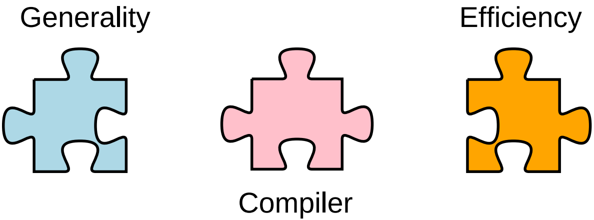
Automate the solution of differential equations



Generality *and* efficiency

- Any equation
- Any (finite element) method
- Maximum efficiency

Possible to combine generality with efficiency by generating code



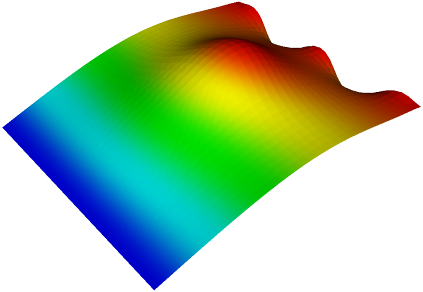
Poisson's equation

Differential equation

Differential equation:

$$-\Delta u = f$$

- Heat transfer
- Electrostatics
- Magnetostatics
- Fluid flow
- etc.



Poisson's equation

Variational formulation

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x$$

$$L(v) = \int_\Omega v f \, \mathrm{d}x$$

Poisson's equation

Implementation

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

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)

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

Linear elasticity

Differential equation

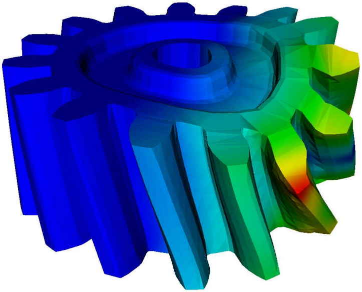
Differential equation:

$$-\nabla \cdot \sigma(u) = f$$

where

$$\sigma(v) = 2\mu\epsilon(v) + \lambda \operatorname{tr} \epsilon(v)\, I$$
$$\epsilon(v) = \frac{1}{2}(\nabla v + (\nabla v)^\top)$$

- Displacement $u = u(x)$
- Stress $\sigma = \sigma(x)$



Linear elasticity

Variational formulation

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$a(v, u) = \int_\Omega \nabla v : \sigma(u) \, \mathrm{d}x$$

$$L(v) = \int_\Omega v \cdot f \, \mathrm{d}x$$

Linear elasticity

Implementation

```
element = VectorElement("Lagrange", "tetrahedron", 1)

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)

def epsilon(v):
    return 0.5*(grad(v) + transp(grad(v)))

def sigma(v):
    return 2*mu*epsilon(v) + lmbda*mult(trace(epsilon(v)), I)

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

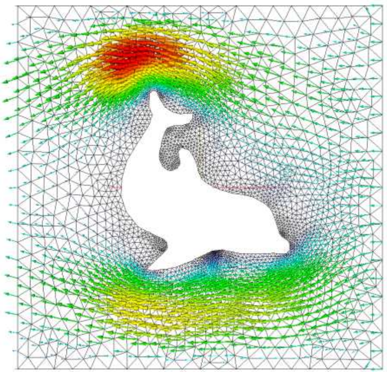
The Stokes equations

Differential equation

Differential equation:

$$-\Delta u + \nabla p = f$$
$$\nabla \cdot u = 0$$

- Fluid velocity $u = u(x)$
- Pressure $p = p(x)$



The Stokes equations

Variational formulation

Find $(u, p) \in V$ such that

$$a((v,q),(u,p)) = L((v,q)) \quad \forall (v,q) \in \hat{V}$$

where

$$a((v,q),(u,p)) = \int_\Omega \nabla v \cdot \nabla u - \nabla \cdot v\, p + q \nabla \cdot u \,\mathrm{d}x$$

$$L((v,q)) = \int_\Omega v \cdot f \,\mathrm{d}x$$

The Stokes equations

Implementation

```
P2 = VectorElement("Lagrange", "triangle", 2)
P1 = FiniteElement("Lagrange", "triangle", 1)
TH = P2 + P1

(v, q) = TestFunctions(TH)
(u, p) = TrialFunctions(TH)

f = Function(P2)

a = (dot(grad(v), grad(u)) - div(v)*p + q*div(u))*dx
L = dot(v, f)*dx
```

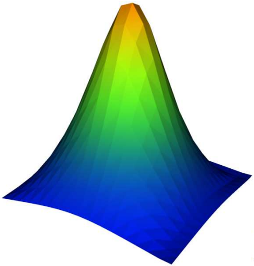
Mixed Poisson with $H(\mathrm{div})$ elements

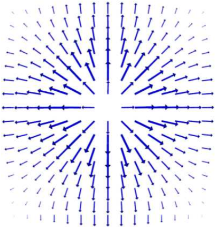
Differential equation

Differential equation:

$$\sigma + \nabla u = 0$$
$$\nabla \cdot \sigma = f$$

- $u \in L_2$
- $\sigma \in H(\mathrm{div})$





Mixed Poisson with $H(\mathrm{div})$ elements

Variational formulation

Find $(\sigma, u) \in V$ such that

$$a((\tau, w), (\sigma, u)) = L((\tau, w)) \quad \forall (\tau, w) \in \hat{V}$$

where

$$a((\tau, w), (\sigma, u)) = \int_\Omega \tau \cdot \sigma - \nabla \cdot \tau \, u + w \nabla \cdot \sigma \, \mathrm{d}x$$

$$L((\tau, w)) = \int_\Omega w \cdot f \, \mathrm{d}x$$

Mixed Poisson with $H(\mathrm{div})$ elements

Implementation

```
BDM1 = FiniteElement("Brezzi-Douglas-Marini", "triangle", 1)
DG0 = FiniteElement("Discontinuous Lagrange", "triangle", 0)

element = BDM1 + DG0

(tau, w) = TestFunctions(element)
(sigma, u) = TrialFunctions(element)

f = Function(DG0)

a = (dot(tau, sigma) - div(tau)*u + w*div(sigma))*dx
L = w*f*dx
```

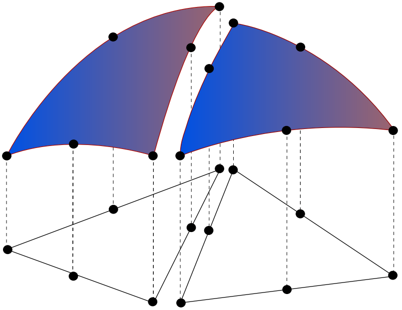
Poisson's equation with DG elements

Differential equation

Differential equation:

$$-\Delta u = f$$

- $u \in L_2$
- u discontinuous accross element boundaries



Poisson's equation with DG elements

Variational formulation (interior penalty method)

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$\begin{aligned} a(v, u) = & \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x \\ & + \sum_S \int_S -\langle \nabla v \rangle \cdot [\![u]\!]_n - [\![v]\!]_n \cdot \langle \nabla u \rangle + (\alpha/h) [\![v]\!]_n \cdot [\![u]\!]_n \, \mathrm{d}S \\ & + \int_{\partial\Omega} -\nabla v \cdot [\![u]\!]_n - [\![v]\!]_n \cdot \nabla u + (\gamma/h) v u \, \mathrm{d}s \\ L(v) = & \int_\Omega v f \, \mathrm{d}x + \int_{\partial\Omega} v g \, \mathrm{d}s \end{aligned}$$

Poisson's equation with DG elements

Implementation

```
DG1 = FiniteElement("Discontinuous Lagrange", "triangle", 1)

v = TestFunction(DG1)
u = TrialFunction(DG1)

f = Function(DG1)
g = Function(DG1)
n = FacetNormal("triangle")
h = MeshSize("triangle")

a = dot(grad(v), grad(u))*dx - dot(avg(grad(v)), jump(u, n))*dS
  - dot(jump(v, n), avg(grad(u)))*dS
  + alpha/h('+')*dot(jump(v, n), jump(u, n))*dS
  - dot(grad(v), jump(u, n))*ds - dot(jump(v, n), grad(u))*ds
  + gamma/h*v*u*ds
L = v*f*dx + v*g*ds
```

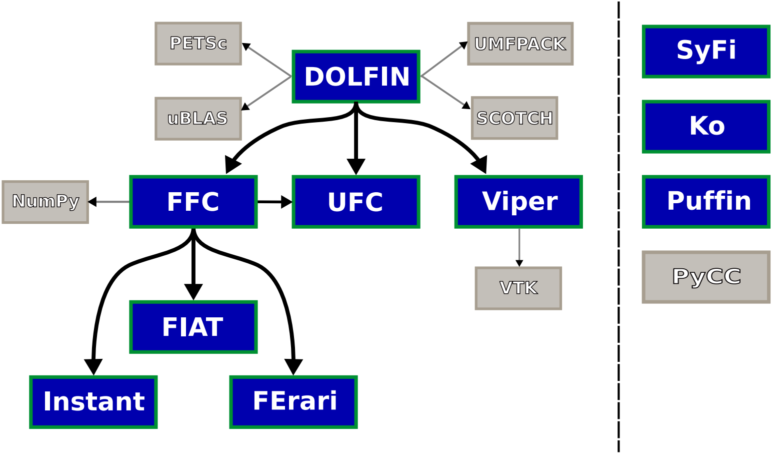
Journal papers (selection)

- *Automating the Finite Element Method*
  Logg, Arch. Comput. Methods Engrg. (2007)
- *Efficient Compilation of a Class of Variational Forms*
  Kirby, Logg, ACM Trans. Math. Software. (2007),
- *Geometric Optimization of the Evaluation of Finite Element Matrices*
  Kirby, Scott, SIAM J. Sci. Comput. (2007)
- *A Compiler for Variational Forms*
  Kirby, Logg, ACM Trans. Math. Software. (2006),
- *Optimizing FIAT with Level 3 BLAS*
  Kirby, ACM Trans. Math. Software. (2006)
- *Topological Optimization of the Evaluation of Finite Element Matrices*
  Kirby, Logg, Scott, Terrel, SIAM J. Sci. Comput. (2006)
- *Optimizing the Evaluation of Finite Element Matrices*
  Kirby, Knepley, Logg, Scott, SIAM J. Sci. Comput. (2005)
- *FIAT: A New Paradigm for Computing Finite Element Basis Functions*
  Kirby, ACM Trans. Math. Software. (2004)

Developer/user base

- 200–300 daily visits at `http://www.fenics.org/`
- 500 monthly downloads
- 10–15 active developers
- 50 registered users on the FEniCS wiki
- Workshops:
  - FEniCS'05 in Chicago
  - FEniCS'06 in Delft
  - FEniCS'07 (08) in Baton Rouge (LSU)
- Google page rank is 6/10 (Simula has 7/10)

Software components



Key Features

- Simple and intuitive object-oriented API, C++ or Python
- Automatic and efficient evaluation of variational forms
- Automatic and efficient assembly of linear systems
- General families of finite elements, including arbitrary order continuous and discontinuous Lagrange elements, BDM, RT
- Arbitrary mixed elements
- High-performance parallel linear algebra
- General meshes, adaptive mesh refinement
- mcG(q)/mdG(q) and cG(q)/dG(q) ODE solvers
- Support for a range of output formats for post-processing, including DOLFIN XML, ParaView/Mayavi/VTK, OpenDX, Octave, MATLAB, GiD
- Built-in plotting

Linear algebra

- Complete support for PETSc
  - High-performance parallel linear algebra
  - Krylov solvers, preconditioners
- Complete support for uBLAS
  - BLAS level 1, 2 and 3
  - Dense, packed and sparse matrices
  - C++ operator overloading and expression templates
  - Krylov solvers, preconditioners added by DOLFIN
- Uniform interface to both linear algebra backends
- LU factorization by UMFPACK for uBLAS matrix types
- Eigenvalue problems solved by SLEPc for PETSc matrix types
- Matrix-free solvers ("virtual matrices")

Recent updates

- UFC, a framework for finite element assembly
- Discontinuous Galerkin methods
- BDM and RT elements
- A new improved mesh library
- Mesh and graph partitioning
- Improved linear algebra supporting PETSc and uBLAS
- Optimized code generation (FErari)
- Improved ODE solvers
- PyDOLFIN, the Python interface of DOLFIN
- Bugzilla database, pkg-config
- Improved manual, compiler support, demos, matrix factory, file formats, . . .

PyCC

- Initiated in 2005
- In–house research tool for solving PDEs and ODEs
- PyCC aims at being a flexible and numerically efficient research tool for application scientists
- Fits together state–of–the–art libraries through Python:
  - DOLFIN (Mesh)
  - SyFi (Matrices)
  - Trilinos (AMG)
  - Hypre (AMG)

PyCC provides a Matlab-like environment for defining and solving PDEs in Python

- Focus on the (block) linear algebra level
- Matrices constructed through factory methods
- Support for generation of custom matrices
- Block preconditioning for optimal solution algorithms
- Numpy arrays used extensively for numerical efficiency in Python
- Live visualization through Viper (PyCC bi–product)
- Happy merge with DOLFIN/PyDOLFIN

Applications

PyCC development is application driven

- Cardiac computations
  - Mono- and bidomain equations
  - Cell modelling (ODEs/PDEs)
  - Inverse problems
- Flow problems
  - Stokes equation
  - Navier-Stokes equations
- Linear and non–linear elasticity

Publications (selection)

- *A linear system of partial differential equations modeling the resting potential of a heart with regional ischemia*
  MacLachlan, Sundnes, Skavhaug, Lysaker, Nielsen, Tveito, Math. Biosci. (2007)
- *Application of symbolic finite element tools to nonlinear hyperelasticity*
  Alnæs, Mardal, Sundnes, MekIT'07
- *SyFi - An Element Matrix Factory, with Emphasis on the Incompressible Navier-Stokes Equations*
  Mardal, Para'06
- *Using Python to Solve Partial Differential Equations*
  Mardal, Skavhaug, Lines, Staff, Ødegård, Computing in Science & Engineering (2007)

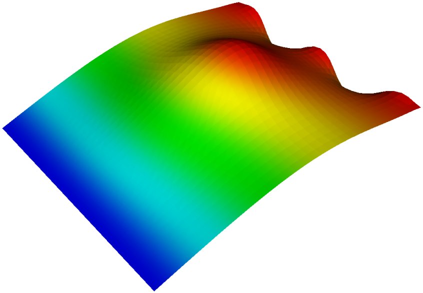
Poisson's equation

Differential equation

Differential equation:

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

- Heat transfer
- Electrostatics
- Magnetostatics
- Fluid flow
- etc.



Poisson's equation

Implementation

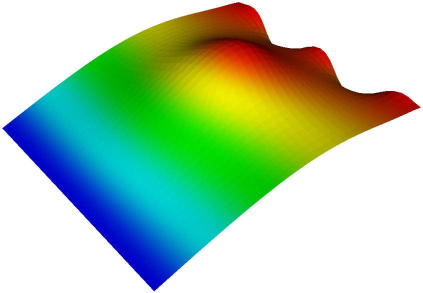
```
mesh = UnitSquare(100,100)
mfac = MatrixFactory(mesh)
L = mfac.computeStiffnessMatrix()
M = mfac.computeMassMatrix()
A = L
u = numpy.zeros(A.n)
b = M*f(mfac.x(), mfac.y())
ConjugateGradients.solve(A, u, b)
```

Heat equation

Differential equation

$$u_t = \Delta u + f \quad \text{in } \Omega, \quad t > 0$$
$$u(x,0) = u_0(x) \quad \text{in } \Omega \text{ at } t = 0$$

- Heat diffusion
- Particle diffusion
- Action potential propagation in nerve cells
- Financial processes
- etc.



Heat equation

Implementation

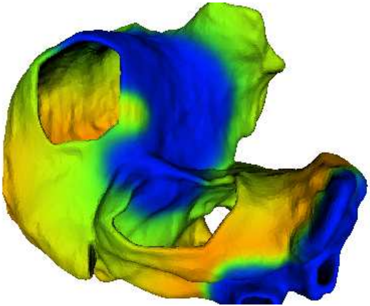
```
mesh = UnitSquare(100, 100)
mfac = MatrixFactory(mesh)
L = mfac.computeStiffnessMatrix()
M = mfac.computeMassMatrix()
A = M + dt*L
u = numpy.zeros(A.m)
for i in xrange(50):
    b = M*(u + f(mfac.x(),mfac.y(),t))
    ConjugateGradients.solve(A, u, b)
```

The Bidomain Equations

Differential Equation

$$\frac{\partial v}{\partial t} = \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u) - I_{\text{ion}}(v, s),$$
$$0 = \nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e) \nabla u)$$

- v: Transmembrane potential
- u: Extra cellular potential
- I_{ion}: ODE system describing the ion flow across the cell membrane
- M_i and M_e: conductivity tensors



Bidomain equations

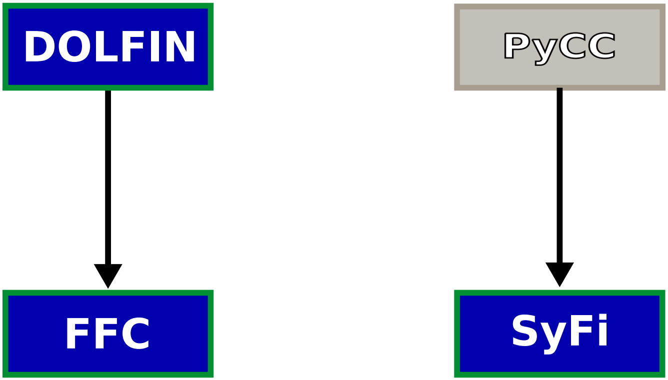
Implementation

```
mesh = Mesh("atria.xml.gz")
mfac = MatrixFactory(mesh)
M = mfac.computeMassMatrix()
Ai = mfac.computeStiffnessMatrix(tf_i)
Aie = mfac.computeStiffnessMatrix(tf_ie)
A = M + dt*Ai
B = dt*Ai
Bt = dt*Ai
C = dt*Aie
A_b = BlockMatrix((A,B),(Bt,C))
Ainv_b = DiagBlockMatrix((prec(A),prec(C)))
```

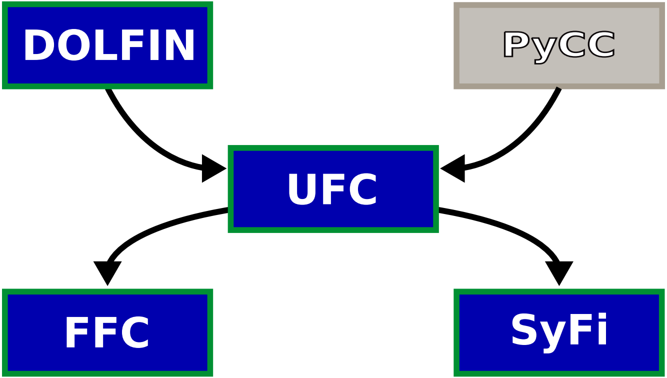
Future plans (highlights)

- UFL/UFC
- Automation of error control
  - Automatic generation of dual problems
  - Automatic generation of a posteriori error estimates
- Mesh algorithms
  - Adaptive mesh refinement
  - Mesh algorithms for ALE methods
- Improved geometry support
- Finite element exterior calculus
- Packaging, installation, testing
- Tighter integration (merge) between DOLFIN and PyCC

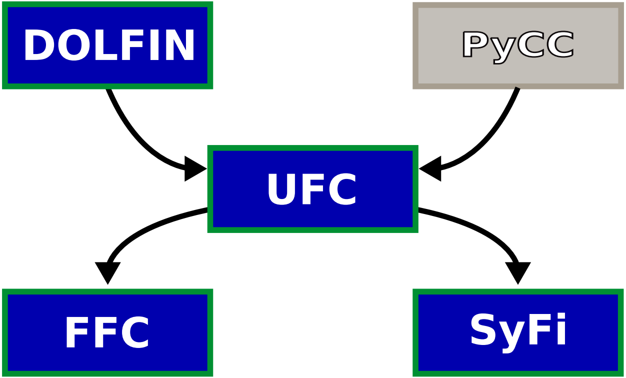
Happy merge



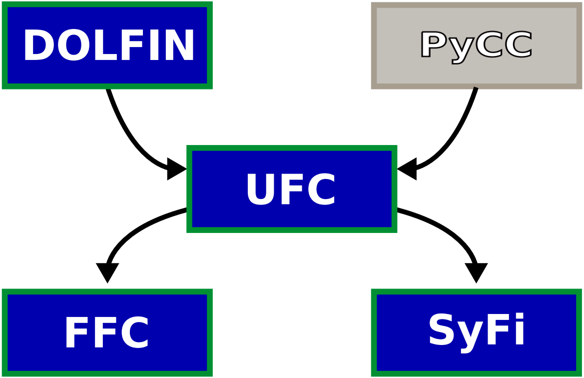
Happy merge



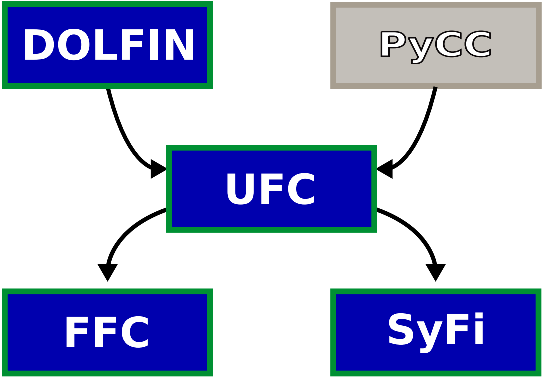
Happy merge



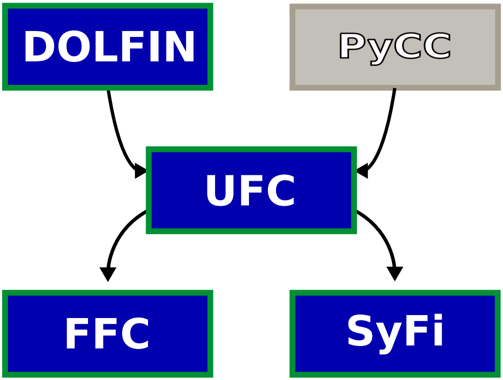
Happy merge



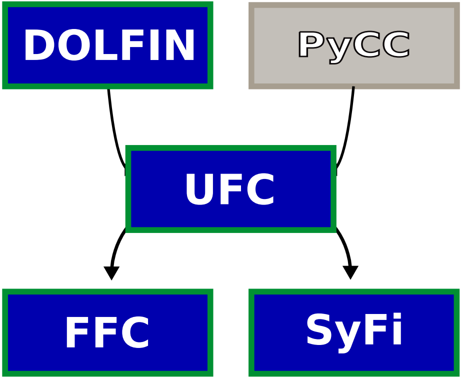
Happy merge



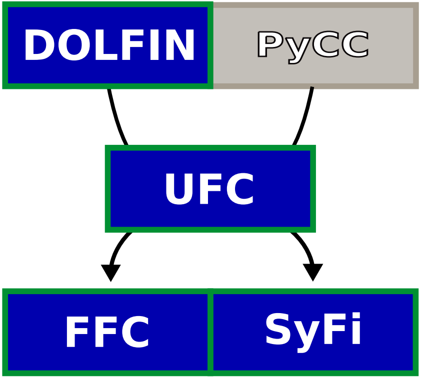
Happy merge



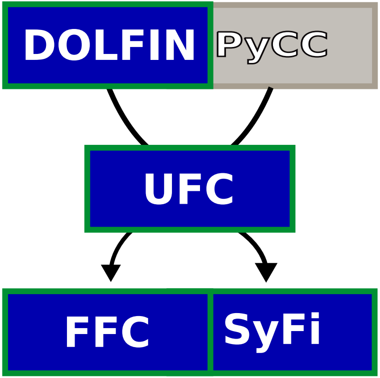
Happy merge



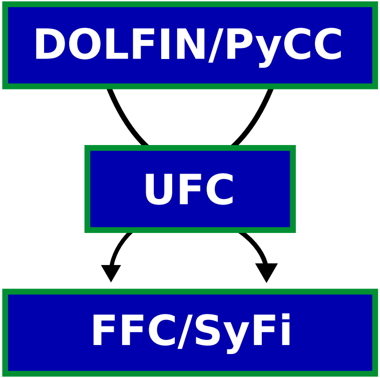
Happy merge



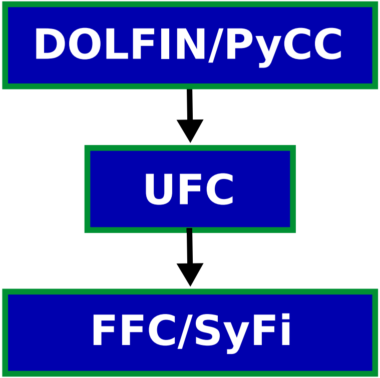
Happy merge



Happy merge

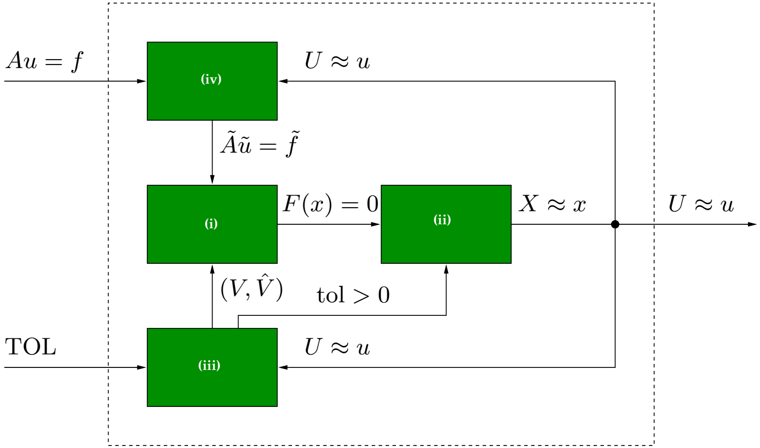


Happy merge



Additional slides

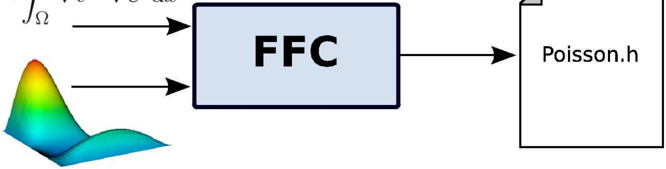
Automation of CMM



A common framework: UFL/UFC

- UFL - Unified Form Language
- UFC - Unified Form-assembly Code
- Unify, standardize, extend
- Working protototypes: FFC (Logg), SyFi (Mardal)

$$a(v, U) = \int_\Omega \nabla v \cdot \nabla U \, \mathrm{d}x$$



Benchmarks

- Measure CPU time for the evaluation of the element tensor (the "element stiffness matrix")
- Code automatically generated by the form compiler FFC
- Compute speedup compared to a standard quadrature-based approach with loops over quadrature points

Form	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$	$q = 6$	$q = 7$	$q = 8$
Mass 2D	12	31	50	78	108	147	183	232
Mass 3D	21	81	189	355	616	881	1442	1475
Poisson 2D	8	29	56	86	129	144	189	236
Poisson 3D	9	56	143	259	427	341	285	356
Navier–Stokes 2D	32	33	53	37	—	—	—	—
Navier–Stokes 3D	77	100	61	42	—	—	—	—
Elasticity 2D	10	43	67	97	—	—	—	—
Elasticity 3D	14	87	103	134	—	—	—	—

Compiling Poisson's equation: non-optimized, 16 ops

```
void eval(real block[], const AffineMap& map) const
{
  [...]

  block[0] = 0.5*G0_0_0 + 0.5*G0_0_1 +
             0.5*G0_1_0 + 0.5*G0_1_1;
  block[1] = -0.5*G0_0_0 - 0.5*G0_1_0;
  block[2] = -0.5*G0_0_1 - 0.5*G0_1_1;
  block[3] = -0.5*G0_0_0 - 0.5*G0_0_1;
  block[4] = 0.5*G0_0_0;
  block[5] = 0.5*G0_0_1;
  block[6] = -0.5*G0_1_0 - 0.5*G0_1_1;
  block[7] = 0.5*G0_1_0;
  block[8] = 0.5*G0_1_1;
}
```

Compiling Poisson's equation: `ffc -O`, 11 ops

```
void eval(real block[], const AffineMap& map) const
{
  [...]

  block[1] = -0.5*G0_0_0 + -0.5*G0_1_0;
  block[0] = -block[1] + 0.5*G0_0_1 + 0.5*G0_1_1;
  block[7] = -block[1] + -0.5*G0_0_0;
  block[6] = -block[7] + -0.5*G0_1_1;
  block[8] = -block[6] + -0.5*G0_1_0;
  block[2] = -block[8] + -0.5*G0_0_1;
  block[5] = -block[2] + -0.5*G0_1_1;
  block[3] = -block[5] + -0.5*G0_0_0;
  block[4] = -block[1] + -0.5*G0_1_0;
}
```

Compiling Poisson's equation: `ffc -f blas`, 36 ops

```
void eval(real block[], const AffineMap& map) const
{
  [...]

  cblas_dgemv(CblasRowMajor, CblasNoTrans,
              blas.mi, blas.ni, 1.0,
              blas.Ai, blas.ni, blas.Gi,
              1, 0.0, block, 1);
}
```