# Simulation of high Reynolds number unsteady incompressible flow past a 3D NACA 0012 wing

Johan Jansson
Computational Technology Laboratory
KTH

May 23, 2012

# Table of contents

# Computational Technology Laboratory (CTL)

Adaptive FEM/software for continuum mechanics/high Re flow
Started in 2007 by me and Johan Hoffman, I'm co-supervisor of PhD students below

Johan Hoffman - Group leader
: Turbulent incompressible flow/geometry

Johan Jansson - Senior researcher
: Unicorn, turbulent flow/fluid-structure interaction

Aurélien Larcher - Postdoc
: Turbulent flow, RANS/LES

Jeanette Spühler - Phd student
: Heart simulation

Rodrigo Vilela de Abreu - Phd student
: Aeroacoustics

Niclas Jansson - Phd student
: Massively parallel adaptive FEM (DOLFIN-HPC)

Cem Degirmenci - Phd student
: Adaptive fluid-structure interaction

Kaspar Müller - Phd student
: Geophysical flows

Research and software development of Unicorn/DOLFIN/FEniCS

`http://ctl.csc.kth.se`

# Unicorn/FEniCS: Aims

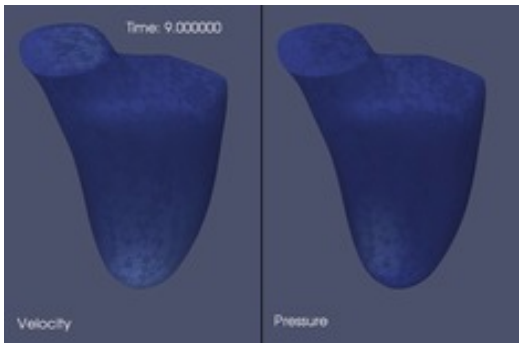
Automated high-performance computational mathematical modeling of industrial/realistic continuum mechanics

- Automated modeling:
  - Unified Continuum model (canonical model for continuum mechanics)
  - No explicit turbulence model
- Automated weak form evaluation/tensor assembly
- Automated duality-based error control/adaptivity
- Strong scaling on massively parallel hardware

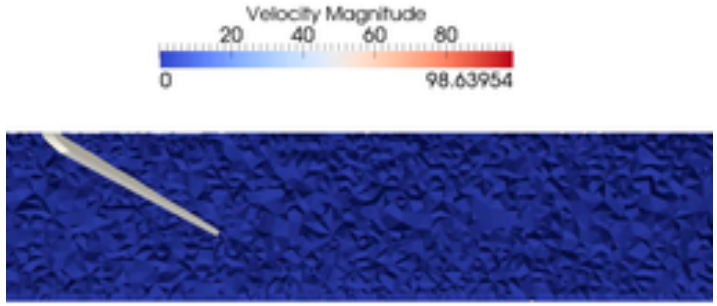
Flight simulation



Biomechanics




Aeroacoustics



# The Finite Element Method (notation)

Want to solve differential equation:

$R(u) = 0$ or $(R(u), v) = 0, \forall v \in V$ (weak/variational form) We seek a solution $U$ in finite element vector space $V^h$ of the form:

$$U(x) = \sum_{j=1}^{M} \xi_j \phi_j(x)$$

We require the residual to be orthogonal to $V^h$:

$$(R(U), v) = 0, \forall v \in V^h$$

Typical notation for linear problems: $(R(U), v) = a(U, v) - L(v) = 0$ Ex.

Poisson: $R(u) = \Delta u - f = 0, \quad (R(U), v) = (\nabla U, \nabla v) - (f, v) = 0$

# Unified Continuum (UC) formulation

Conservation equations (momentum, mass)
in Euler (laboratory) coordinates, stress $\sigma$ as data:

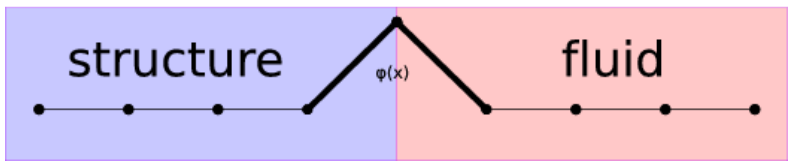
$$\begin{aligned} \rho(\partial_t u + (u \cdot \nabla)u) - \nabla \cdot \sigma &= 0 \quad \text{in } Q, \\ \nabla \cdot u &= 0 \quad \text{in } Q, \\ \partial_t \theta + (u \cdot \nabla)\theta &= 0 \quad \text{in } Q, \\ u(\cdot, 0) &= u^0 \quad \text{in } \Omega, \end{aligned}$$

Different constitutive equations for phases:

$$\sigma = \bar{\sigma} - pI \quad \bar{\sigma} = \theta\bar{\sigma}_f + (1-\theta)\bar{\sigma}_s$$

$$D_t\bar{\sigma}_s = 2\mu_s\epsilon + \nabla u\bar{\sigma}_s + \bar{\sigma}_s\nabla u^\top \qquad \bar{\sigma}_f = 2\mu_f\epsilon$$

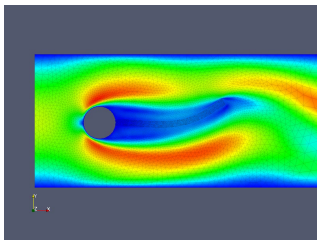
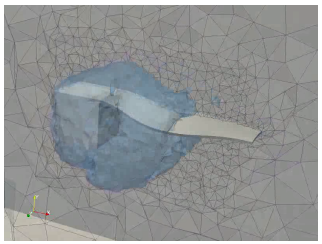
Discretize as one domain (exploited in error estimation):



# Unified Continuum (UC) G2/FEM discretization

Unicorn/FEniCS simulator with G2 (General Galerkin):

- Can use simple elements (piecewise linear)
- Adaptive error control/mesh refinement
- Least-squares/streamline diffusion type stabilization (necessary for convection dominated flow).
- Moving mesh methods (elastic smoothing, etc.) [Compere, Remacle, Jansson, Hoffman 2009])
- Model turbulent dissipation by adaptive residual-based DNS/LES.

2D benchmark within 2% of reference in mesh convergence study [Hoffman, Jansson, Stockli, M3AS 2010]

# Goal of FEniCS: Automation of solution of PDE by FEM

- Open source software - all software tools I show today you can inspect and modify. Scientific robustness: researchers control their own tools.
- Goal: Automatic solution of PDE
- Why?
  - Efficiency
    - ★ Execution time
    - ★ Development time (especially on parallel architectures)
    - ★ Complexity is high, so manual efficient implementation typically not possible.
    - ★ Implement automated kernel thoroughly (efficiency, parallel) - can solve all problems vs. one specialized solver for every problem: avoid having to repeat optimization/verification for every problem.
  - Safety - reduce chance of human error (bugs)

Some history: started in 2003, is developed and used as research platform for several research groups around the world (Cambridge, KTH, Simula, Texas Tech).

# FEniCS components

FEniCS: `www.fenicsproject.org` Open source software project for automated solution of PDE with many involved universities/individuals

- Automated generation of finite elements/basis functions (FIAT)

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

- Automated evaluation of variational forms (FFC)

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

- Automated assembly of discrete systems (DOLFIN)

$$\begin{aligned} &A = 0 \\ &\textit{for all elements } K \in \mathcal{T}_\Omega \\ &\quad A \mathrel{+}= A^K \end{aligned}$$

- Automated Unified Continuum modeling (Unicorn)

$$R(W, v) = (\rho(\partial_t u + (u \cdot \nabla)u) + \nabla \cdot \sigma - g, v)$$

# Component dependencies

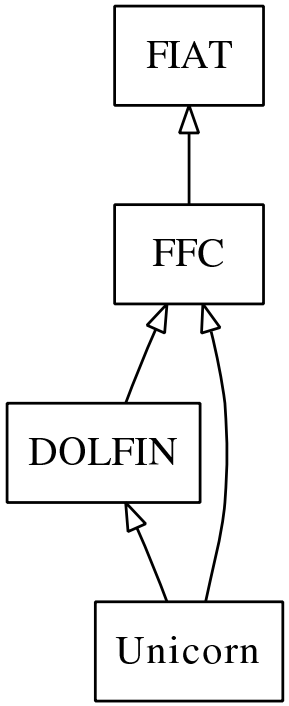
**Unicorn**
- UC forms
- `TimeDependentPDE`
- ErrorEstimate
- coefficients (eg. stabilization)
- mesh adaptation/smoothing

**DOLFIN**
- mesh representation
- global tensor assembly,
- Linear algebra interface
- load balancing

**FFC**
- form evaluation code generation

**FIAT**
- basis function representation

Dependencies



# Basic FEniCS Poisson demo

```
from dolfin import *

# Create mesh and define function space
mesh = UnitSquare(32, 32)
V = FunctionSpace(mesh, "CG", 1)

# Define Dirichlet boundary (x = 0 or x = 1)
def boundary(x):
    return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS

# Define boundary condition
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)

# Define variational problem
v = TestFunction(V)
u = TrialFunction(V)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)")
g = Expression("sin(5*x[0])")
a = inner(grad(v), grad(u))*dx
L = v*f*dx - v*g*ds

# Compute solution (assemble matrix/vector, solve linear system)
problem = VariationalProblem(a, L, bc)
u = problem.solve()

# Plot solution
plot(u, interactive=True)
```

# Automated matrix/vector assembly I

In general the matrix $A_h$, representing a bilinear form

$$a(u, v) = (A(u), v),$$

is given by

$$(A_h)_{ij} = a(\varphi_j, \hat{\varphi}_i).$$

and the vector $b_h$ representing a linear form

$$L(v) = (f, v),$$

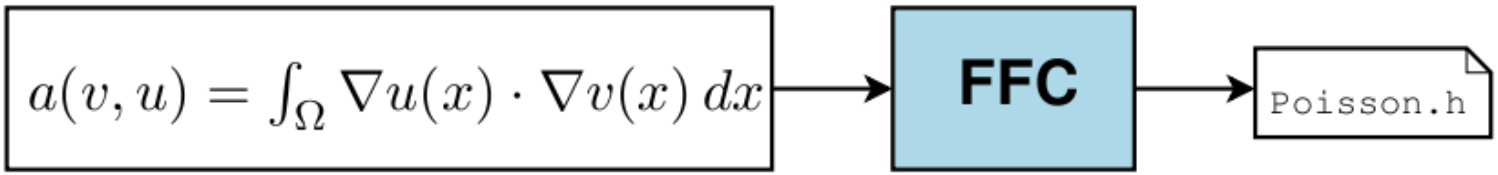
is given by

$$(b_h)_i = L(\hat{\varphi}_i).$$

Example (Poisson 1D):
$a(u, v) = (u', v') = \int_0^1 u'v'dx, \quad (A_h)_{ij} = a(\phi_j, \phi_i) = \int_0^1 \phi_j'\phi_i'dx$
$L(v) = (f, v) = \int_0^1 fvdx, \quad (b_h)_i = L(\phi_i) = \int_0^1 f\phi_i dx$

# Form compilation/code generation

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




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

# Generated quadrature code

```
virtual void tabulate_tensor(double* A, const double* const* w,
      const ufc::cell& c) const
{
...
  // Quadrature weight
  const static double W0 = 0.5;
  // Tabulated basis functions and arrays of non-zero columns
const static double Psi_w[1][3] =\
  {{0.33333333333, 0.33333333333, 0.33333333333}};
 const static double Psi_vu[1][2] = {{-1, 1}};
 static const unsigned int nzc0[2] = {0, 1};
 static const unsigned int nzc1[2] = {0, 2};
 // Geometry constants
 const double G0 = Jinv_00*Jinv_10*W0*det;
 const double G1 = Jinv_01*Jinv_11*W0*det;
 const double G2 = Jinv_00*Jinv_00*W0*det;
 const double G3 = Jinv_01*Jinv_01*W0*det;
 const double G4 = Jinv_10*Jinv_10*W0*det;
 const double G5 = Jinv_11*Jinv_11*W0*det;
 // Loop integration points
 for (unsigned int ip = 0; ip < 1; ip++)
 {
   // Compute function value
   double F0 = 0;
   for (unsigned int r = 0; r < 3; r++)
     F0 += Psi_w[ip][r]*w[0][r];
   const double Gip0 = (G0 + G1)*F0;
   const double Gip1 = (G2 + G3)*F0;
   const double Gip2 = (G4 + G5)*F0;
   for (unsigned int i = 0; i < 2; i++)
   {
     for (unsigned int j = 0; j < 2; j++)
     {
       A[nzc0[i]*3 + nzc0[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip1;
       A[nzc0[i]*3 + nzc1[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip0;
       A[nzc1[i]*3 + nzc0[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip0;
       A[nzc1[i]*3 + nzc1[j]] += Psi_vu[ip][i]*Psi_vu[ip][j]*Gip2;
     }
   }
 }
}
```

# UC Momentum form (FFC language)

```
def epsilon(u):
    return sym(grad(u))

def S(u, P):
    return P*Identity(d) - nu*grad(u)

def f(u, v):
    return -inner(grad(u)*u, v) + \
        inner(S(u, P), grad(v)) + \
        -d1*inner(grad(u)*u + grad(P), grad(v)*u) + \
        -d2*inner(div(u), div(v)) + \
        dot(fsource, v)

def dfdu(u, k, v):
    return -inner(grad(u)*UC, v) + \
        -inner(nu*grad(u), grad(v)) + \
        -d1*inner(grad(u)*UC, grad(v)*UC) + \
        -d2*inner(div(u), div(v))

# cG(1)cG(1) in time and space
def F(u, u0, k, v):
    uc = 0.5 * (u + u0)
    return (-dot(u, v) + dot(u0, v) + k*f(u, v))

def dFdu(u, u0, k, v):
    uc = 0.5 * u
    return (-dot(u, v) + k*dfdu(uc, k, v))

a = (dFdu(U1, U0, k, v)) * dx
L = (dFdu(UP, U0, k, v) - F(UP, U0, k, v)) * dx
```

# A posteriori error estimation

$$(Aw, v) = (w, A^* v) \quad \text{(adjoint definition)}$$
$$Au = f \quad \text{(primal equation)}$$
$$A^* \phi = \psi \quad \text{(dual equation)}$$

Boundary terms are assumed to vanish.

$$(e, \psi) = (e, A^* \phi) = (Ae, \phi) = (-R(U), \phi) \quad \text{(error representation)}$$

From this we can derive a canonical formula for the error estimate:

$$|(e, \psi)| \leq \sum_T \|hR(U)\|_K \|\nabla \phi\|_K \quad \text{(error bound)}$$

and an adaptive method to satisfy:

$$|(e, \psi)| \leq TOL$$

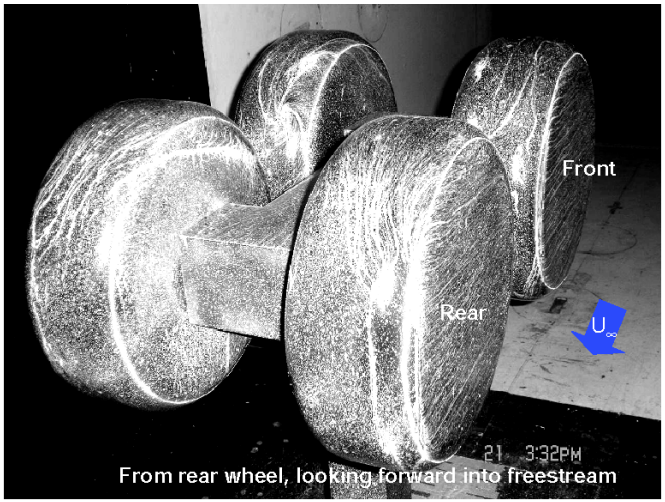
(iteratively refine cells with largest error contribution/indicators)

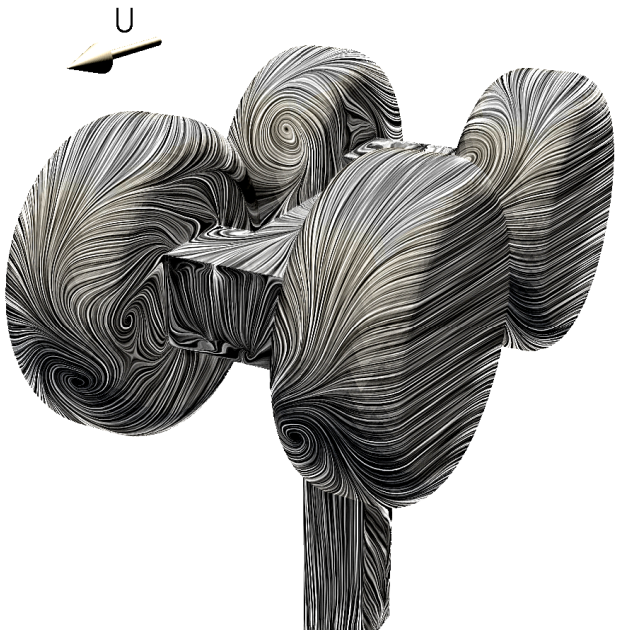
# Automated simulation: turbulence

Approximate turbulence as weak solutions by FEM with residual based stabilization (General Galerkin G2 method)

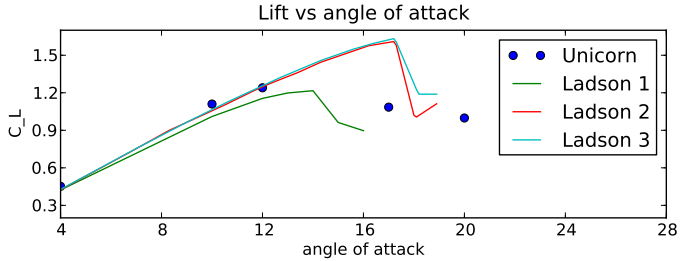
- No RANS/LES averaging/filtering
- No explicit turbulence/subgrid model
- Turbulent dissipation only from numerical stabilization (cf. Implicit LES, MILES [Fureby/Grinstein AIAA 99], VMM–LES [Bazilevs et.al. CMAME 07, Guasch/Codina 07])
- Modeling the effect of turbulent boundary layers by a skin friction stress boundary conditioon (cf. [Schumann JCP 75]), for high Re (for the NACA0012 wing for example) we use zero friction.

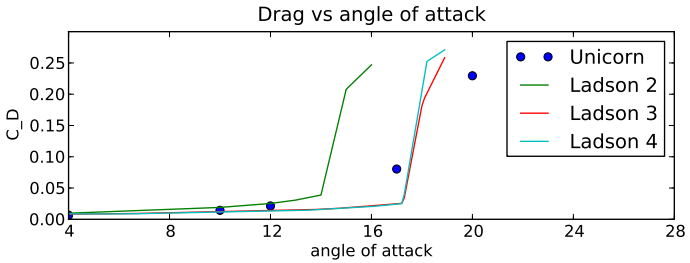
Oil film, experiment vs. simulation

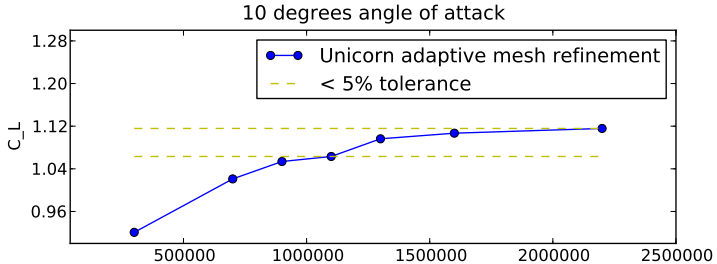


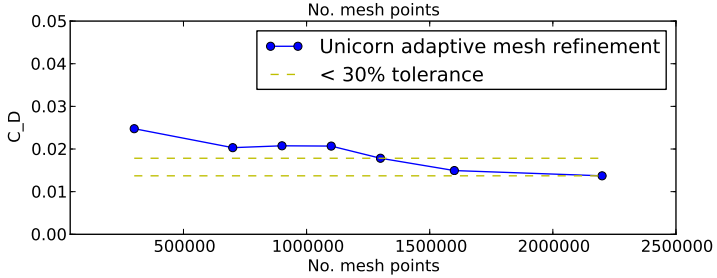



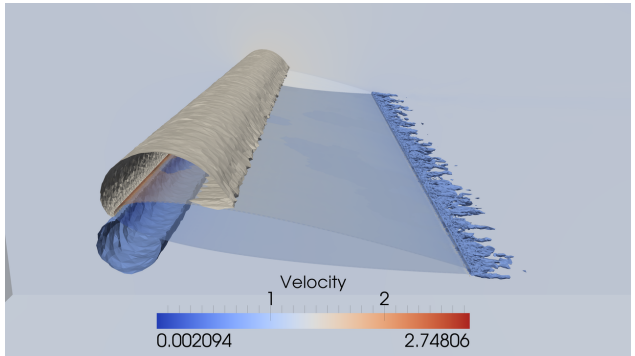
# NACA 0012 at realistic flight conditions, $\alpha = 10°$

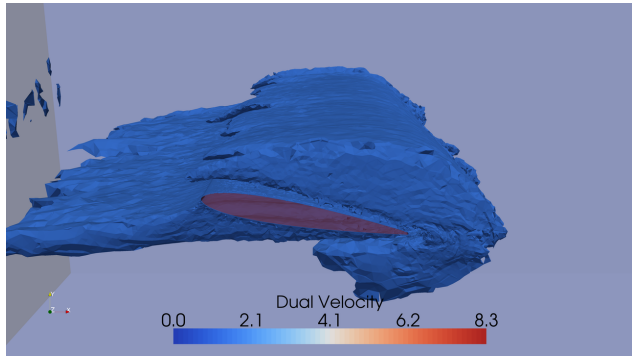


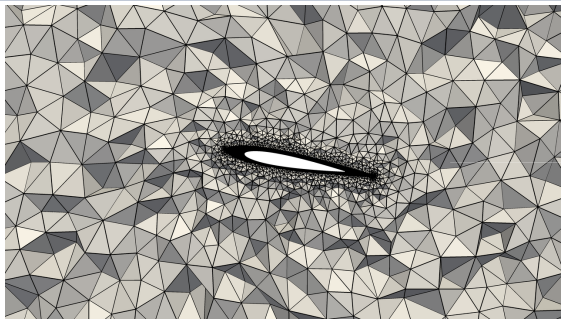



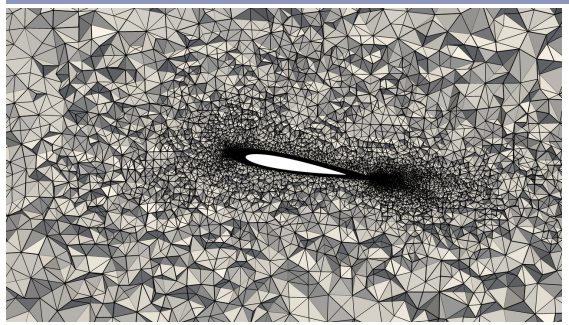



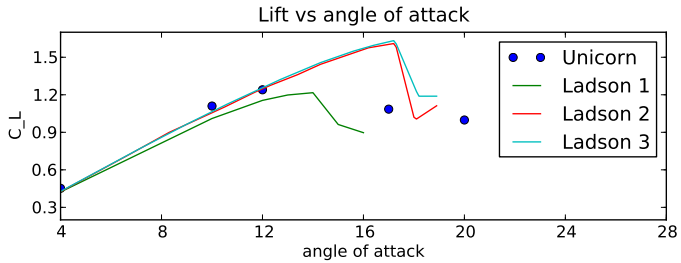



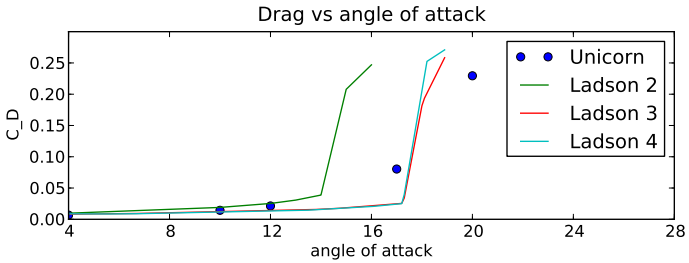



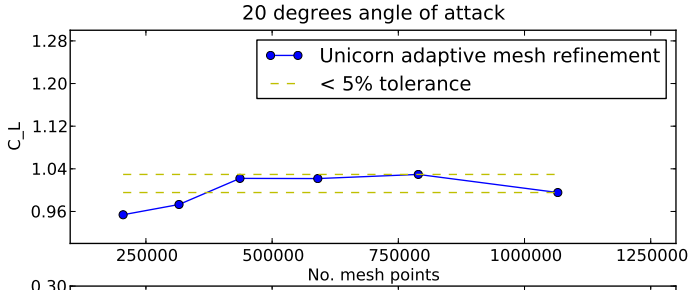



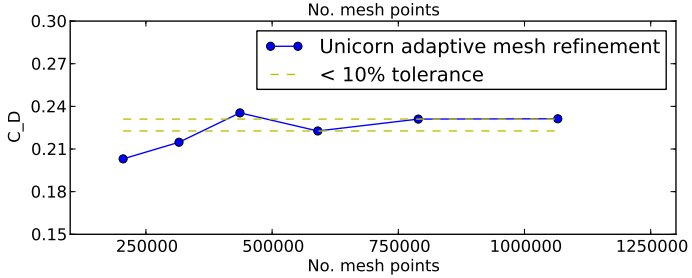


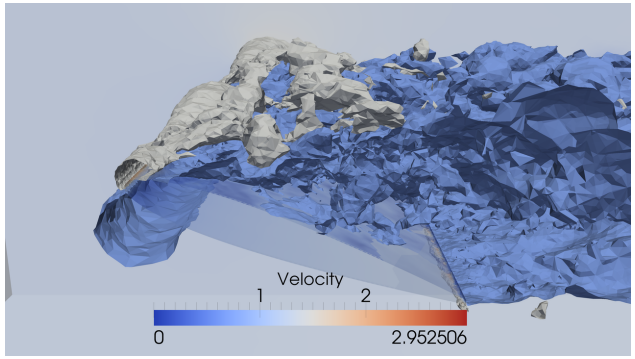
# NACA 0012 at realistic flight conditions, $\alpha = 20°$

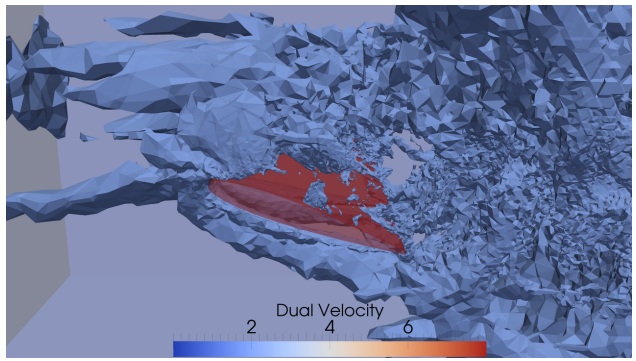


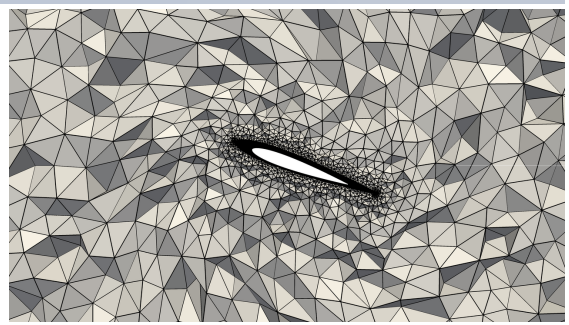



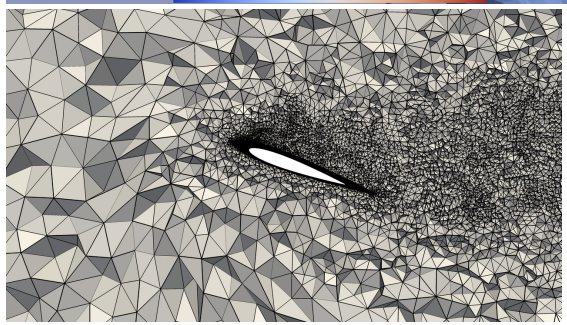



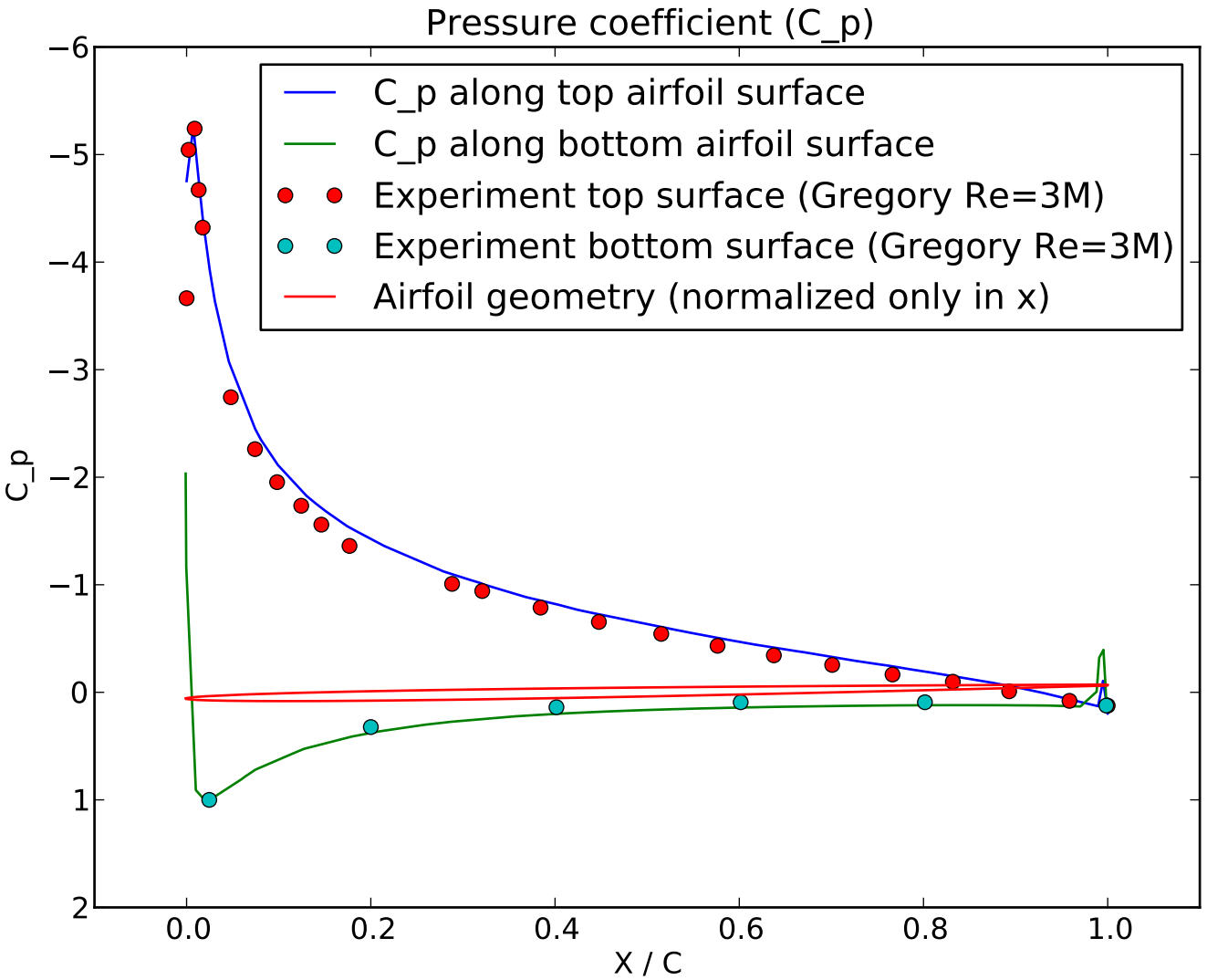



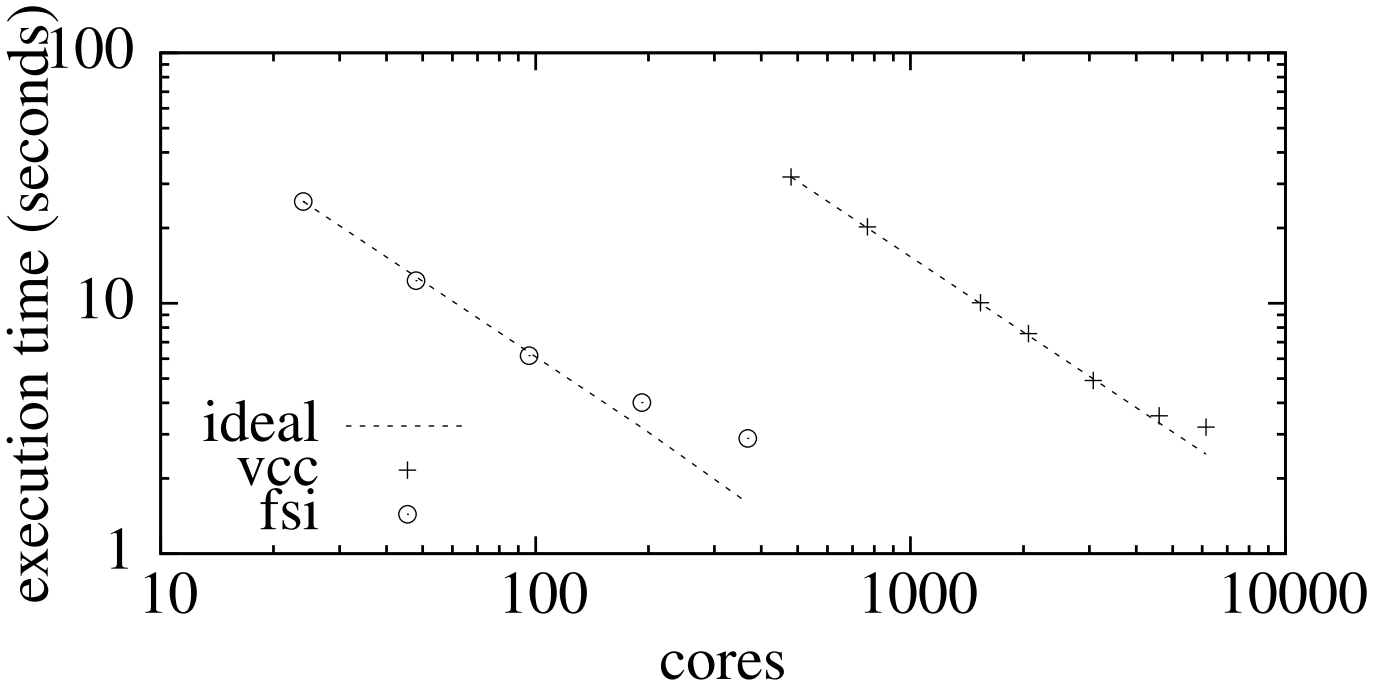


# NACA 0012, $\alpha = 10^\circ$, surface pressure



# Strong scaling verification

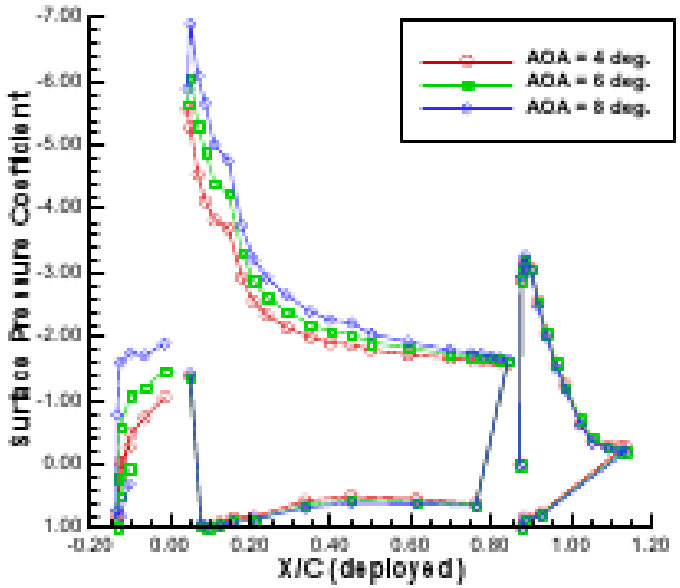
Strong linear scaling of full adaptive solve of incompressible flow up to ca. 5000 cores on Lindgren (PDC) supercomputer.

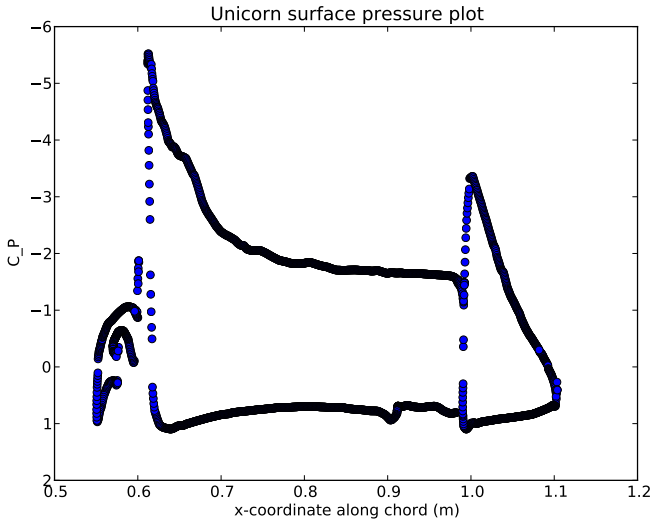



[Hoffman, Jansson, et. al., 2011 submitted], [Hoffman, Jansson, Jansson, 2011, SISC]

For the NACA0012 problem at $\alpha = 10^\circ$ on the finest mesh with 2.2 million mesh points on 768 cores, the simulation takes 16 hours, enabling overnight simulation for a wing at realistic flight conditions.

# Preliminary results for 30PN30P wing




Surface pressure plots: experimental reference left and Unicorn simulation results right for $AOA = 4°$.

# Conclusions

- Described the FEniCS framework for automated FEM for solving PDE
- Unicorn: Unified Continuum, automated modeling of continuum mechanics and turbulence
- Apply framework to flight simulation: convergence for pre- and post-stall to experiments
- Demonstrated strong linear scaling for massively parallel implementation
- (Demonstrated convergence of duality-based adaptive error control for FSI)
- NACA 0012 results available as preprint (see my home page).

`ctl.csc.kth.se`
`fenicsproject.org`