Large-Eddy Simulation in Julia with Gridap

A Variational MultiScale approach

Carlo Brunelli

Royal Military Academy, Brussels, Belgium

Example: NACA0012 Airfoil at Reynolds Number 1000

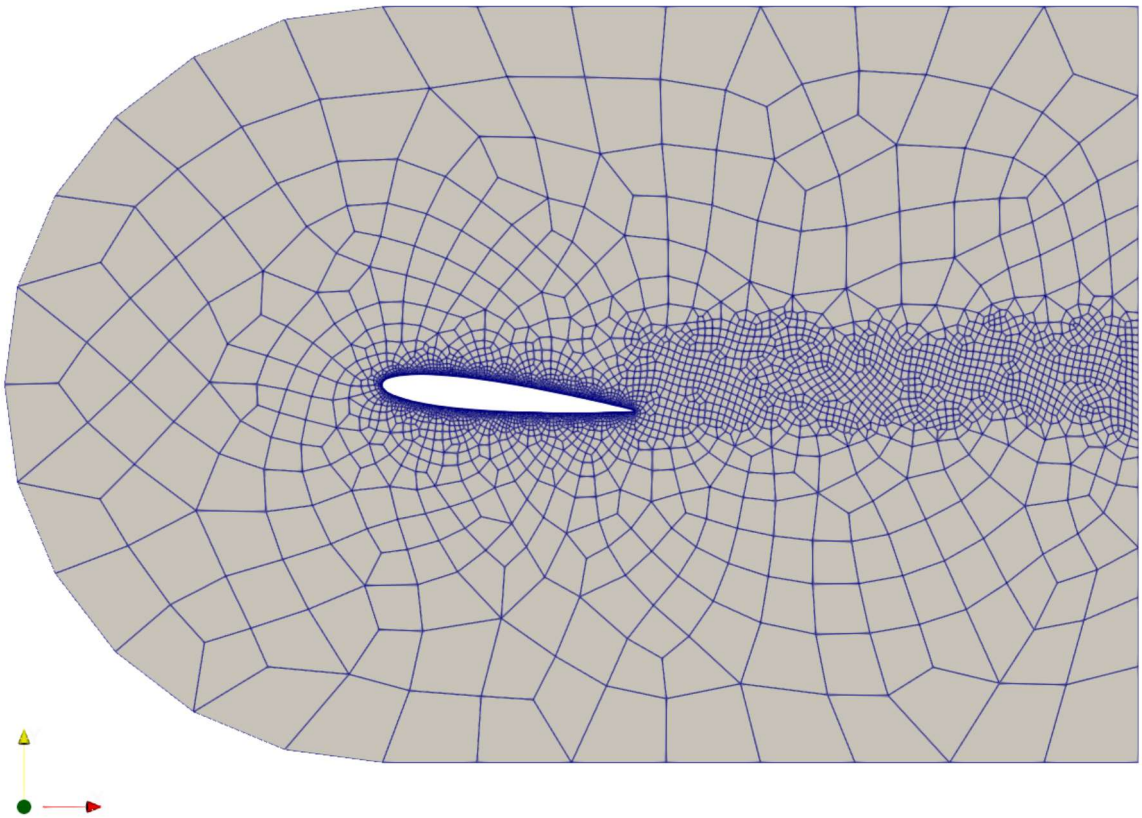
Keywords:

- Incompressible Flow (**u**, p)
- Finite Element Method (**v**, q)
- Order of Interpolation (k=1,2,...)
- **Use same order of interpolation for velocity and pressure** -> VMS

```
begin
    using Gridap, GridapGmsh
    using LineSearches: BackTracking
    using Gridap.ODEs: TransientFEOperator

end
```

```
model = GmshDiscreteModel("Mesh0.msh");
```

```
Info    : Reading 'Mesh0.msh'...
Info    : 143 entities
Info    : 12610 nodes
Info    : 12852 elements
Info    : Done reading 'Mesh0.msh'
```



```
begin
    order_u = 2
    order_p = 2
    D = 2 #2-dimensional problem
    order = maximum([order_u, order_p])
end;
```

```
begin
    Re = 1000.0
    ν = 1/Re #viscosity [1m chord, 1.0m/s inlet velocity]
    t0 = 0.0
    tF = 1.0
    dt = 0.1
    u_in = 1.0 #Inlet
    θ = 1.0 # ODE parameter
end;
```

Boundary Conditions

```
begin
    u_wall(x,t) = VectorValue(0.0, 0.0)
    u_wall(t::Real) = x -> u_wall(x,t)

    u_0(x,t) = VectorValue(u_in, 0.0)
    u_0(t::Real) = x -> u_0(x,t)
end;
```

```
begin
    reffeᵤ = ReferenceFE(lagrangian, VectorValue{D,Float64}, order_u)
    reffeₚ = ReferenceFE(lagrangian, Float64, order_p)
    V = TestFESpace(model, reffeᵤ, conformity=:H1, dirichlet_tags=
    ["inlet","limits","airfoil"])
    U = TransientTrialFESpace(V, [u_0,u_0,u_wall])
    Q = TestFESpace(model, reffeₚ, conformity=:H1, dirichlet_tags=["outlet"])
    P = TrialFESpace(Q, [0.0])
    Y = MultiFieldFESpace([V, Q])
    X = TransientMultiFieldFESpace([U, P])
end;

```

Intial Condition

```
uh0 = interpolate_everywhere(VectorValue(0.0,0.0), U(0));
```

```
ph0 = interpolate_everywhere(0.0, P(0));
```

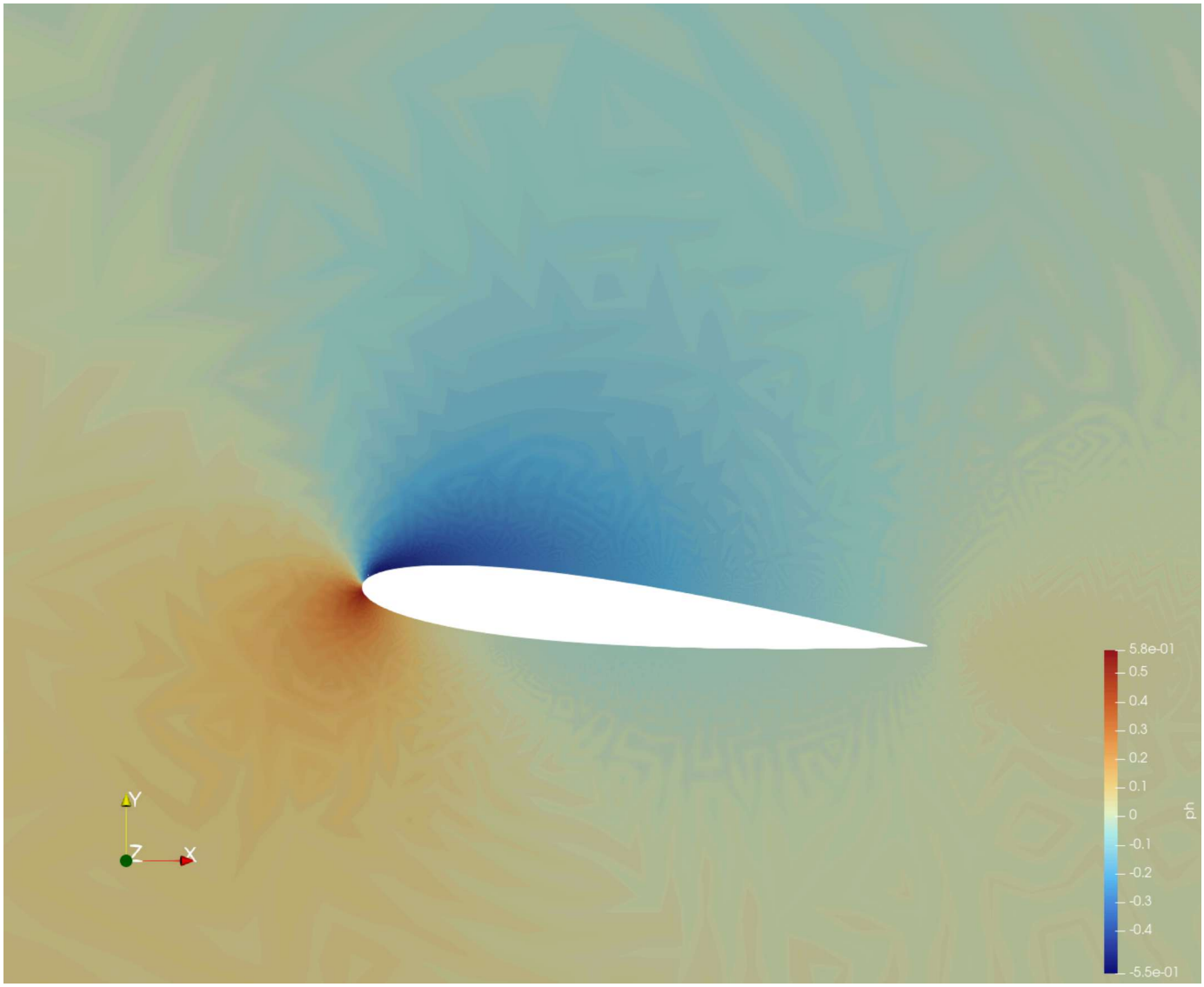
```
xh0 = interpolate_everywhere([uh0, ph0], X(0));
```

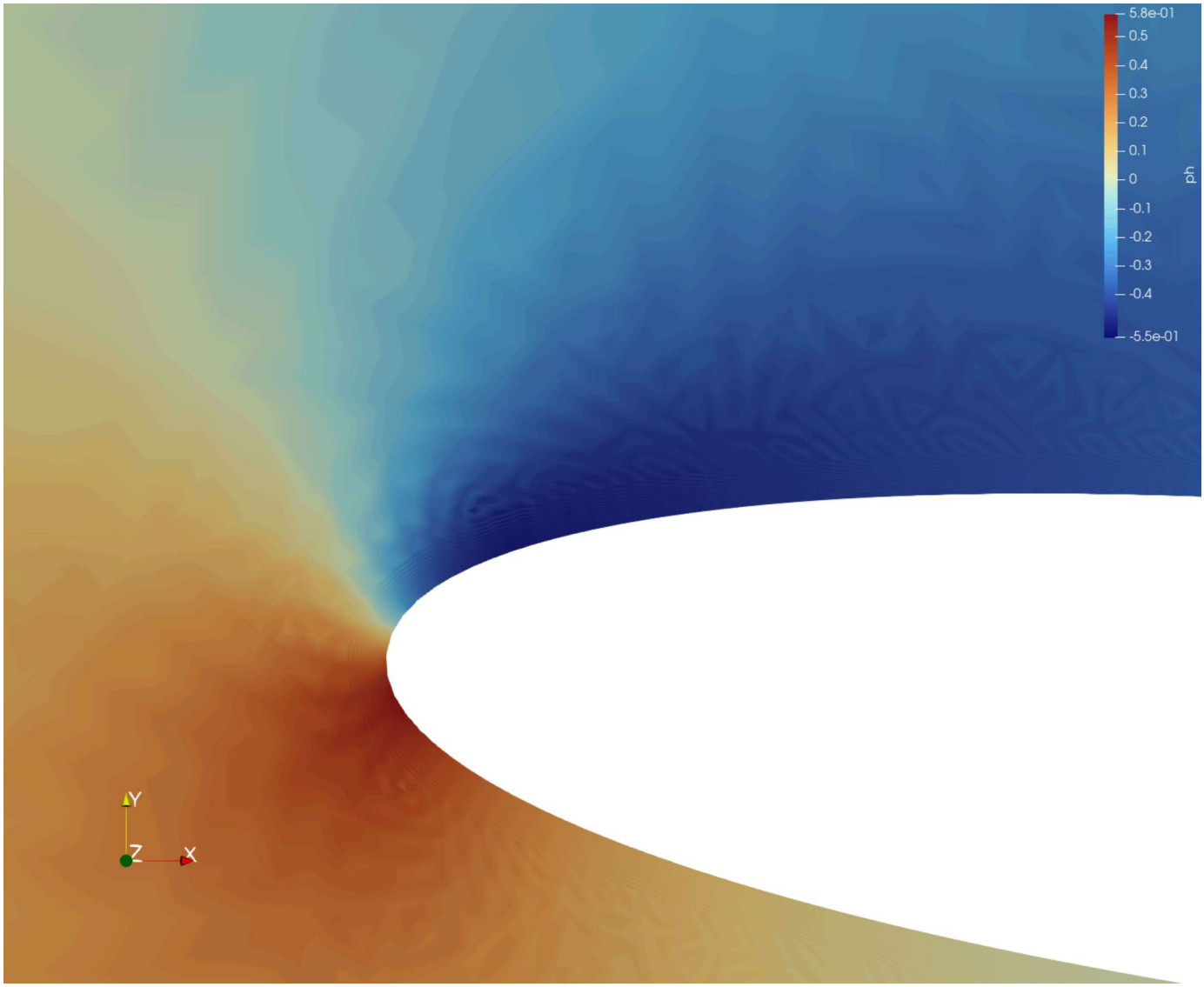
```
begin
    degree = order*2
    Ω = Triangulation(model)
    dΩ = Measure(Ω,degree)
end;
```

Variational Formulation

```
var_eq(t, (u, p), (v, q)) = ∫(∂t(u) ⋅ v)dΩ + ∫((u ⋅ ∇(u)) ⋅ v)dΩ - ∫((∇ ⋅ v) * p)dΩ +
∫((q * (∇ ⋅ u)))dΩ + ν * ∫(∇(v) ⊙ ∇(u))dΩ;
```

$$\int_\Omega \frac{\partial u}{\partial t} \cdot v \, \mathrm{d}\Omega + \int_\Omega \nu \nabla v \cdot \nabla u \, \mathrm{d}\Omega - \int_\Omega (\nabla \cdot v) \, p \, \mathrm{d}\Omega + \int_\Omega q \, (\nabla \cdot u) \, \mathrm{d}\Omega + \int_\Omega (u \cdot \nabla(u)) \cdot v \, \mathrm{d}\Omega$$



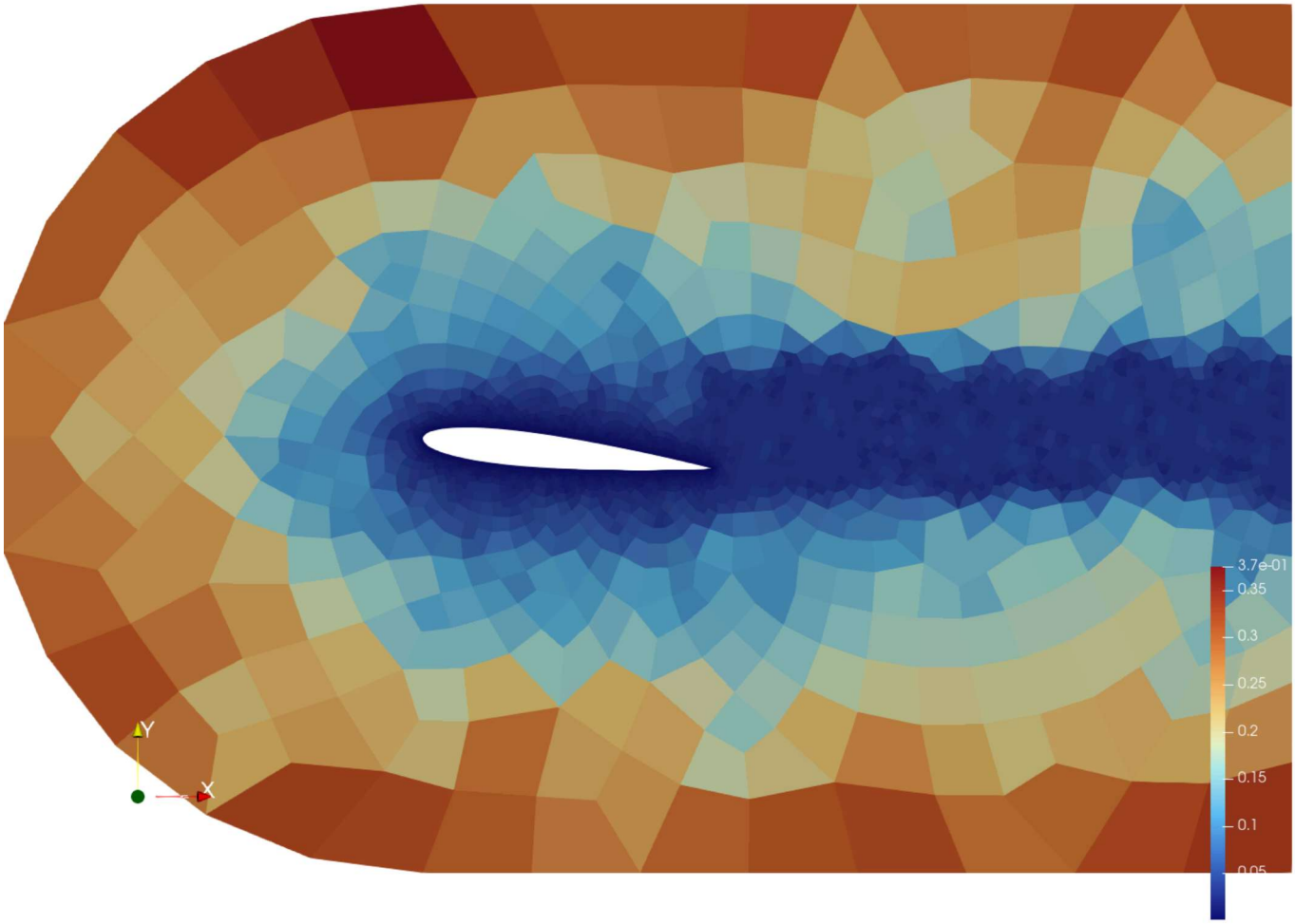


Stabilization (SUPG)

Enable Stabilization: ☑

Define cell size

```
1 h = lazy_map(h -> h^(1 / D), get_cell_measure(Ω));
```



Residual of Momentum

```
Rm(t, (u, p)) = ∂t(u) + u ⋅ ∇(u) + ∇(p);  #- ν*Δ(u)
```

$$\frac{\partial u}{\partial t} + u \cdot \nabla(u) + \nabla(p) - \nu\Delta(u)$$

Residual of Continuity

```
Rc(u) = ∇ ⋅ u;
```

$$\nabla \cdot u$$

Stabilization parameters

```
function τ(u, h)
        τ₂ = h^2 / (4 * ν) #diffusion scale
        r = 2
        τ₃ = dt / 2 #time-limit scale

        val(x) = x
        val(x::Gridap.Fields.ForwardDiff.Dual) = x.value

        u = val(norm(u))

        if iszero(u)
            return (1 / τ₂^r + 1 / τ₃^r)^(-1 / r)
        end

        τ₁ = h / (2 * u) #convective scale
        return (1 / τ₁^r + 1 / τ₂^r + 1 / τ₃^r)^(-1 / r)

end;
```

$$\tau = \left(\frac{2u}{h} + \frac{4\nu}{h^2} + \frac{2}{dt} \right)^{-1}$$

```
τb(u, h) = (u ⋅ u) * τ(u, h);
```

$$\tau_b = (u \cdot u)\tau$$

```
stab_eq(t, (u, p), (v, q)) = ∫((τ ∘ (u, h) * (u ⋅ ∇(v) + ∇(q))) ⊙ Rm(t, (u, p))    +
 τb ∘ (u, h) * (∇ ⋅ v) ⊙ Rc(u) )dΩ;

```

$$\int_\Omega \tau \cdot (u \cdot \nabla(v) + \nabla(q)) R_m \, \mathrm{d}\Omega + \int_\Omega \tau_b \cdot (\nabla \cdot v) R_c \, \mathrm{d}\Omega$$

Resolution

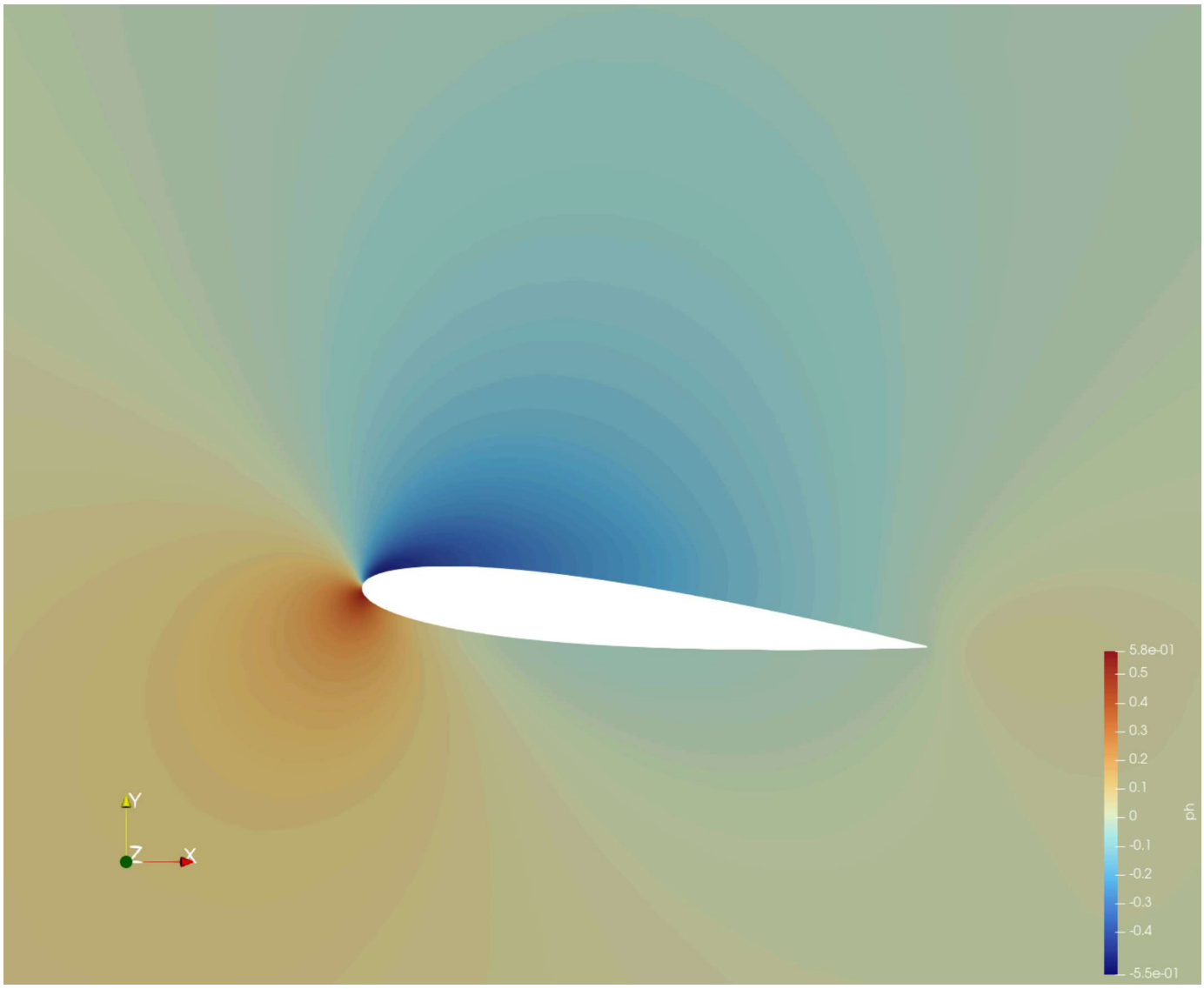
```
if STAB
    res(t,(u,p),(v,q))= var_eq(t, (u, p), (v, q))  + stab_eq(t, (u, p), (v, q))
else
    res(t,(u,p),(v,q))= var_eq(t, (u, p), (v, q))
end;

```

```
op = TransientFEOperator(res,X,Y);
```

```
1 nls_solver = NLSolver(show_trace=true, iterations=10,method=:newton);
```

```
1 ode_solver = ThetaMethod(nls_solver,dt,0);
```

```
1 sol_t = solve(ode_solver, op, t0, tF, xh0);
```



```
begin
        if STAB
            pathname = "results_stab/"
        else
            pathname = "results/"
        end


        mkpath(pathname)

        createpvd(pathname) do pvd
          pvd[0] = createvtk(Ω, "$(pathname)/NACA_0" * ".vtu", cellfields=["uh" => uh0, "ph"=>ph0], nsubcells=order)
          for (tn, xh) in sol_t
              uh,ph=xh
            pvd[tn] = createvtk(Ω, "$(pathname)/NACA_$tn" * ".vtu", cellfields=["uh" => uh, "ph"=>ph],nsubcells=order)
          end
        end
    end;
```

Variational MultiScale

- **Gridap** ecosystem wrapper
- Parallel exection on CPUs
- PETSc
- Different types of stabilization (tensor, scalar, VMS, SUPG)
- Turbulence intensity at the inlet

VMS vs SUPG

```
"""
    compute_d(trian::Gridap.Geometry.BodyFittedTriangulation, D::Int64) #trian == Ω

The inverse of the cell-map-field. It is evaluated in the middle of the reference domain."""
function compute_d(trian::Gridap.Geometry.BodyFittedTriangulation, D::Int64) #trian == Ω

  ξₖ = get_cell_map(trian)
  Jt = lazy_map(Broadcasting(∇), ξₖ)
  inv_Jt = lazy_map(Operation(inv), Jt)

  eval_point = Point(0.5 .* ones(D)) #(0.5,0.5) or (0.5,0.5,0.5)
  d = lazy_map(Gridap.Arrays.evaluate, inv_Jt, Fill(eval_point, num_cells(trian)))
  return d
end;

```

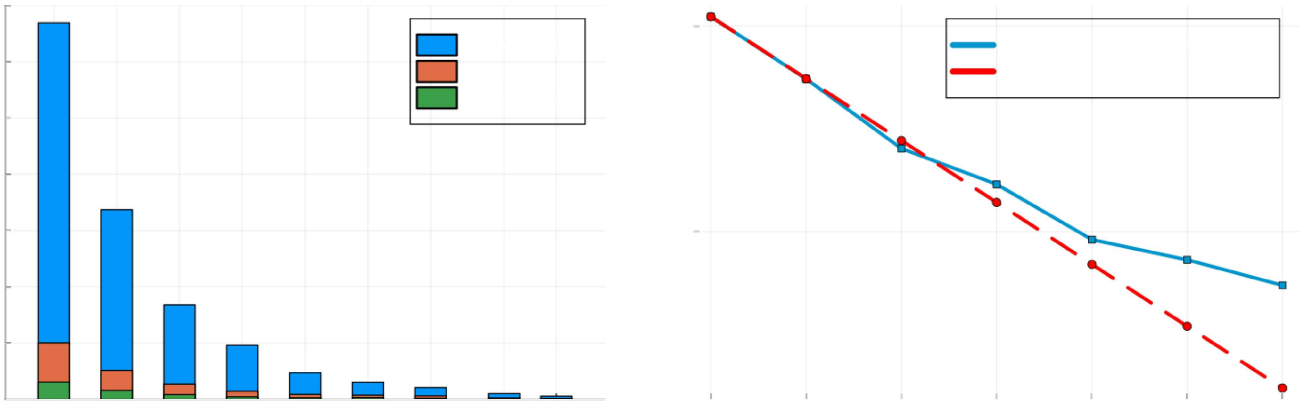
Extra terms, not only stabilization but sug-grid scale modelling

```
begin
        TRm(t, (u, p)) = τm ∘ (u) ∗ Rm(t, (u, p))

        #Variational equations
        B^G(t, (u, p), (v, q)) = ∫(∂t(u) ⋅ v)dΩ + ∫((u ⋅ ∇(u)) ⋅ v)dΩ - ∫((∇ ⋅ v) ∗ p)dΩ + ∫((q ∗ (∇ ⋅ u)))dΩ + ν ∗ ∫(∇(v) ⊙ ∇(u))dΩ

        #SUPG terms
        B_SUPG(t, (u, p), (v, q)) = ∫((u ⋅ ∇(v) + ∇(q)) ⊙ TRm(t, (u, p)))dΩ + ∫((∇ ⋅ v) ⊙ (τc ∘ (u)  ∗ Rc(u)))dΩ

        #First VMS term
        B_VMS1(t, (u, p), (v, q)) = ∫((u ⋅ (∇(v))') ⊙ TRm(t, (u, p)))dΩ

        #Second VMS term
        B_VMS2(t, (u, p), (v, q)) = -1 ∗ ∫((∇(v)) ⊙ (outer(TRm(t, (u, p)), TRm(t, (u, p)))))dΩ
end;
```

Computational Efficiency

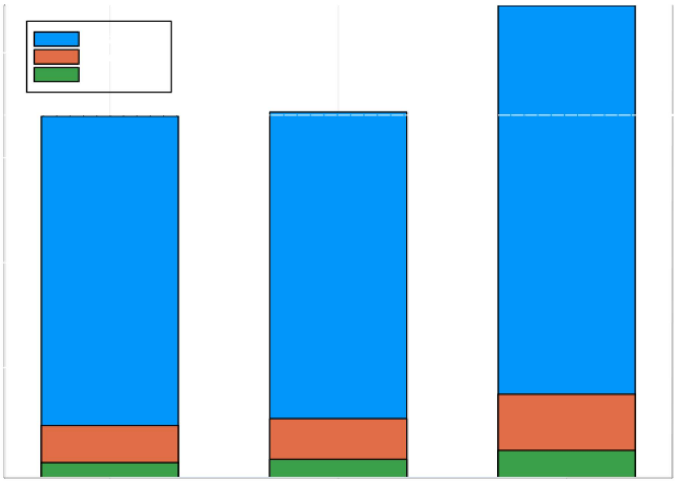
Strong scalability

Strong scaling for TGV2D using N=1200 and second order elements



Weak Scalability

Weak scaling for TGV2D



Examples

Wing-Tail interaction

Taylor-Green Vortices 3D

Contacts

Carlo Brunelli, PURE:

carlo.brunelli@mil.be

Segregated Resolution

Splitting the contributions

$$\dot{\boldsymbol{p}} = \frac{\Delta \boldsymbol{p}}{\Delta t} = \frac{\boldsymbol{p}^{n+1} - \boldsymbol{p}^{n}}{\Delta t}$$

$$\dot{\boldsymbol{u}} = \frac{\Delta \boldsymbol{u}}{\Delta t} = \frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^{n}}{\Delta t} = \boldsymbol{a}$$

$$\begin{bmatrix} 0 & T_{qu} \\ 0 & T_{vu} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{u}} \end{bmatrix} + \begin{bmatrix} A_{qp} & A_{qu} \\ A_{vp} & A_{vu} \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{u} \end{bmatrix} = 0$$

$$(T_{vu} + \theta \Delta t A_{vu}) \Delta \boldsymbol{a}^* = -A_{vu} \boldsymbol{u}^m - A_{vp} \boldsymbol{p}^m - (T_{vu} + \theta \Delta t A_{vu}) \boldsymbol{a}^m + \\ + A_{vu} \Delta t \boldsymbol{a}^m + (1 - \theta) A_{vp} \sum_{i=0}^{m} \Delta \boldsymbol{p}^i$$

$$(T_{qu} + \Delta t A_{qu}) \boldsymbol{\Delta a} + A_{qp} \sum_{i=0}^{m+1} \Delta \boldsymbol{p}^i = -A_{qu} \boldsymbol{u}^n - A_{qp} \boldsymbol{p}^n - (T_{qu} + \Delta t A_{qu}) \boldsymbol{a}^m$$