Meshfree Methods for PDEs

PUMA: Partition of Unity Methods & Applications

Marc Alexander Schweitzer

Institut für Numerische Simulation
Rheinische Friedrich-Wilhelms-Universität Bonn

Fraunhofer-Institut für Algorithmen und Wissenschaftliches Rechnen (SCAI)

September 15, 2017

Abstract Theory - Real Challenges

PUMA Design

Keep method & code as versatile - no additional assumptions on components!

- The PU $\{\varphi_i\}$ does reproduce the constant only - but may be arbitrarily smooth.
- The basis functions ϑ_i^m of local space $V_i = \mathcal{P}^{p_i} + \mathcal{E}_i$ are not explicitely available.

- What if you chose the wrong basis - does it spoil the approximation?
- You do not know basis explicitly a priori.
- How to set essential boundary conditions?
- How to efficiently integrate the basis / assemble the stiffness matrix?
- You do not know stability of basis / conditioning of stiffness matrix.
- How to efficiently solve the linear system?
- How to do efficient and stable explicit time stepping?

schweitzer@scai.fraunhofer.de

Solutions within PUM

Our fundamental assumption

The PU satisfies the **flat-top property** and is **non-negative**.

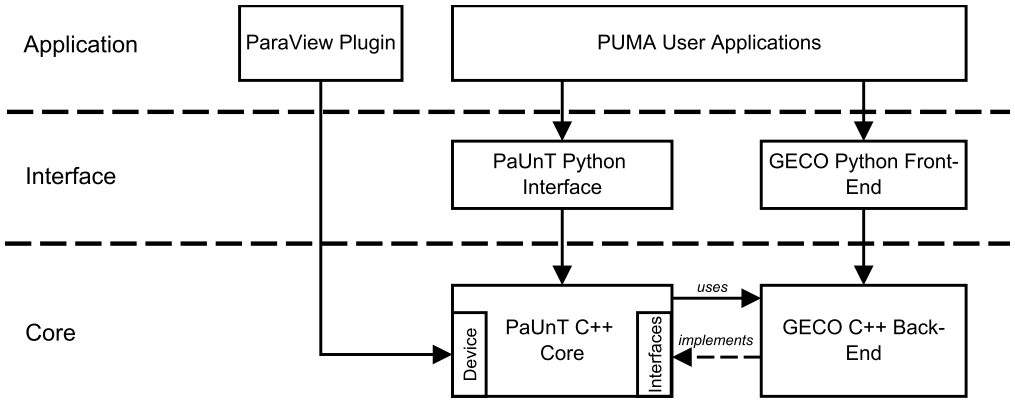
Integration	SIAM SISC **23** (2002)
Essential Boundary Conditions	Springer (2002), SIAM SISC **29** (2009)
Stability	Numer. Math. **118** (2011)
Multilevel solver	SIAM SISC **24** (2002), Springer (2005), (2013)
Variational lumping	SIAM SISC **35** (2013), Springer (2014)

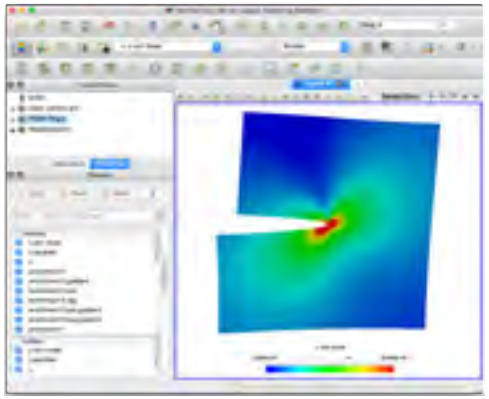
Framework components

- Fully automatic constructions, applicable to arbitrary enrichments.
- Completely transparent to user, (almost) no tunable parameters.

schweitzer@scai.fraunhofer.de

PUMA's Ingredients & Features





Components & Features

- General expression compiler (GECO)
- Particle-Partition of Unity Method (PaUnT)
- *h*-, *p*-, and *hp*-refinements
- Explicit and implicit time integrators
- Multilevel solvers, external solvers
- Solid mechanics toolbox
- ParaView plugin & filters

schweitzer@scai.fraunhofer.de

Poisson Problem

Governing Equations

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}), & \boldsymbol{x} \text{ in } \Omega, \\ u(\boldsymbol{x}) &= g(\boldsymbol{x}), & \boldsymbol{x} \text{ on } \Gamma_D \subseteq \partial\Omega, \\ \frac{\partial u(\boldsymbol{x})}{\partial n} &= 0, & \boldsymbol{x} \text{ on } \Gamma_N = \partial\Omega \setminus \Gamma_D \end{aligned}$$

where Δ is the Laplace operator, u the unknown function, f a prescribed right hand side function, Ω the spatial domain, $\partial\Omega$ its boundary, g an appropriate boundary condition and $\frac{\partial u(\boldsymbol{x})}{\partial n}$ the normal derivative of u with respect to $\partial\Omega$.

Model Problem

$$\begin{aligned} \Omega &:= [0,1]^2, \\ \Gamma_D &:= \partial\Omega, \\ f(\boldsymbol{x}) &:= -6, \\ g(x,y) &:= 1 + x^2 + 2y^2 \end{aligned}$$

schweitzer@scai.fraunhofer.de

Poisson Problem

Weak Formulation

$$\int_\Omega \nabla \delta u \cdot \nabla v \, \mathrm{d}\boldsymbol{x} = \int_\Omega f v \, \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} \frac{\partial \delta u}{\partial n} v \, \mathrm{d}\boldsymbol{x}$$

where $u, v \in V$ are trial and test functions from the PUM space V. Which can be written as:

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

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

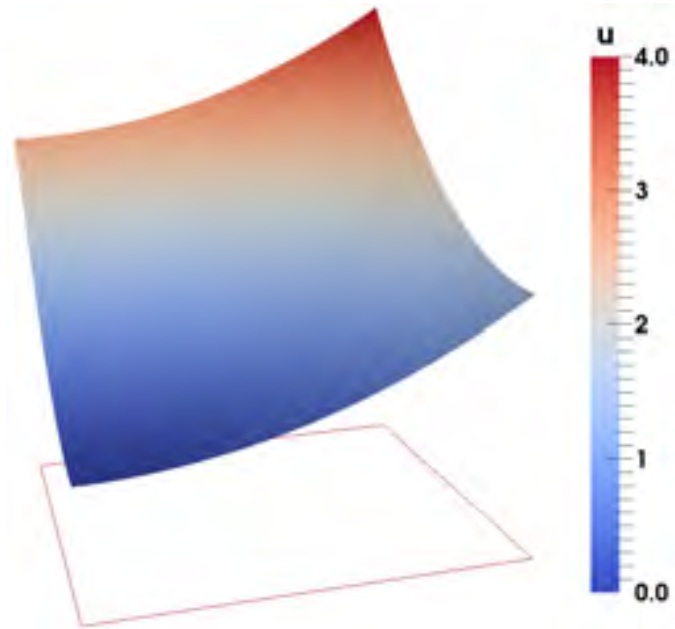
PUMA Representation

```
deltau = ScalarTrialFunction(V)
v = ScalarTestFunction(V)
f = -6
a = inner(grad(deltau), grad(v))*dx
L = f*v*dx
```

schweitzer@scai.fraunhofer.de

Poisson Problem

```
from puma import *

# Define domain and dicretization space
omega = UnitSquareDomain()
V = PumSpace(omega, level=3, polynomial_degree=1)

# Define Dirichlet boundary conditions
x = Position()
g = 1 + x[0]*x[0] + 2*x[1]*x[1]
bc = DirichletBoundaryCondition(V, None, g)

# Define scalar variational problem
deltau = ScalarTrialFunction(V)
v = ScalarTestFunction(V)
f = -6
a = inner(grad(deltau), grad(v))*dx
L = f*v*dx

# Solve for unknown function
u = ScalarFunction(V)
solve(a == L, u, bc)

# Write the solution to a file
write_to_pnt("ps_solution.pnt", (u, "u"))
```



Example: PUMA Python Script

```
# Import packages
from puma import *
# Create domain and boundary segments and crack
omega = RectangleDomain(Point(-1.,-1.), Point(1.,1.))
gamma_D = PhysicalGroupSubDomain(omega, "Right", "Bottom")
gamma_N = PhysicalGroupSubDomain(omega, "Top")
crack = CrackPolyline(omega, Point(-1.1,0.), Point(0.,0.))
# Set material properties
material = LinearElasticMaterialModel(omega, E=1, nu=0.3)
# Create function space
V = PumSpace(omega, level=5, stretch_factor=1.3, weight_type="cubic flat top spline", polynomial_degree=1, polynomial_type='Legendre Tensor Product')
# Create enrichments and enrich function space
x = Position()
T = Translation(-crack.second_tip()) & Rotation(crack.second_tip_direction())
near_crack_tip = distance_point_box(crack.second_tip(), PatchStretchedDomain()) < 0.5
V.enrich(sintheta2(T(x)), EnrichmentApplicationType.additive, near_crack_tip)
V.enrich(costheta2(T(x)), EnrichmentApplicationType.additive, near_crack_tip)
V.enrich(sintheta2sintheta(T(x)), EnrichmentApplicationType.additive, near_crack_tip)
V.enrich(costheta2sintheta(T(x)), EnrichmentApplicationType.additive, near_crack_tip)
at_crack = intersects_box_line(PatchStretchedDomain(), crack.first_tip(), crack.second_tip()) & Not(within_point_box(crack.second_tip(),
 ↪     PatchStretchedDomain()))
V.enrich(heaviside(T(x)), EnrichmentApplicationType.multiplicative, at_crack)
# Create transformation to stable basis
u = VectorTrialFunction(V)
v = VectorTestFunction(V)
st = create_stable_transformation((dot(u,v)+inner(grad(u),grad(v)))*dx)
# Define boundary conditions
g_D = (0,0)
bcs = [DirichletBoundaryCondition(V, gamma_D, g_D)]
# Define loading conditions
f = (0,-0.1)
n = Normal()
g_N = 0.1 * n
# Define variational problem
u = VectorTrialFunction(V)
v = VectorTestFunction(V)
a = material.second_variation_of_strain_energy(u,v)
L = dot(f,v)*dx + dot(g_N,v)*ds(gamma_N)
# Compute solution
u = VectorFunction(V)
solve(a == L, u, bcs, st)
# Output solution
write_to_pnt("crack_solution.pnt", (u, "u"), material)
```

Arbitrary Enrichments

Analytic Enrichments

```
x = Position()
polar_transformation = PolarTransformation()
def sintheta2(x):
  radius = polar_transformation(x)[0]
  theta  = polar_transformation(x)[1]
  return sqrt(radius) * sin(theta / 2)
V.enrich(sintheta2(x), EnrichmentApplicationType.additive, near_origin)
```

Numerical Enrichments

Classical FE:

```
enrichment = EnrichmentSpaceFem2D(E, enrichment_data_file, enrichment_names, True)
should_enrich = within_box_box_touches(PatchStretchedDomain(), mesh_bounding_box)
compiled_feature = _compile_feature(should_enrich, V.domain_dimension(), V.parallel_partition().communicator())
V.enrich(enrichment, compiled_feature)
```

Particle Data e.g. Smoothed Particle Hydrodynamics:

```
enrichment = EnrichmentSpaceSPH2D(E, enrichment_data_file, enrichment_names, False)
should_enrich = within_box_box_touches(PatchStretchedDomain(), sph_domain_bounding_box)
compiled_feature = _compile_feature(should_enrich, V.domain_dimension(), V.parallel_partition().communicator())
V.enrich(enrichment, compiled_feature)
```

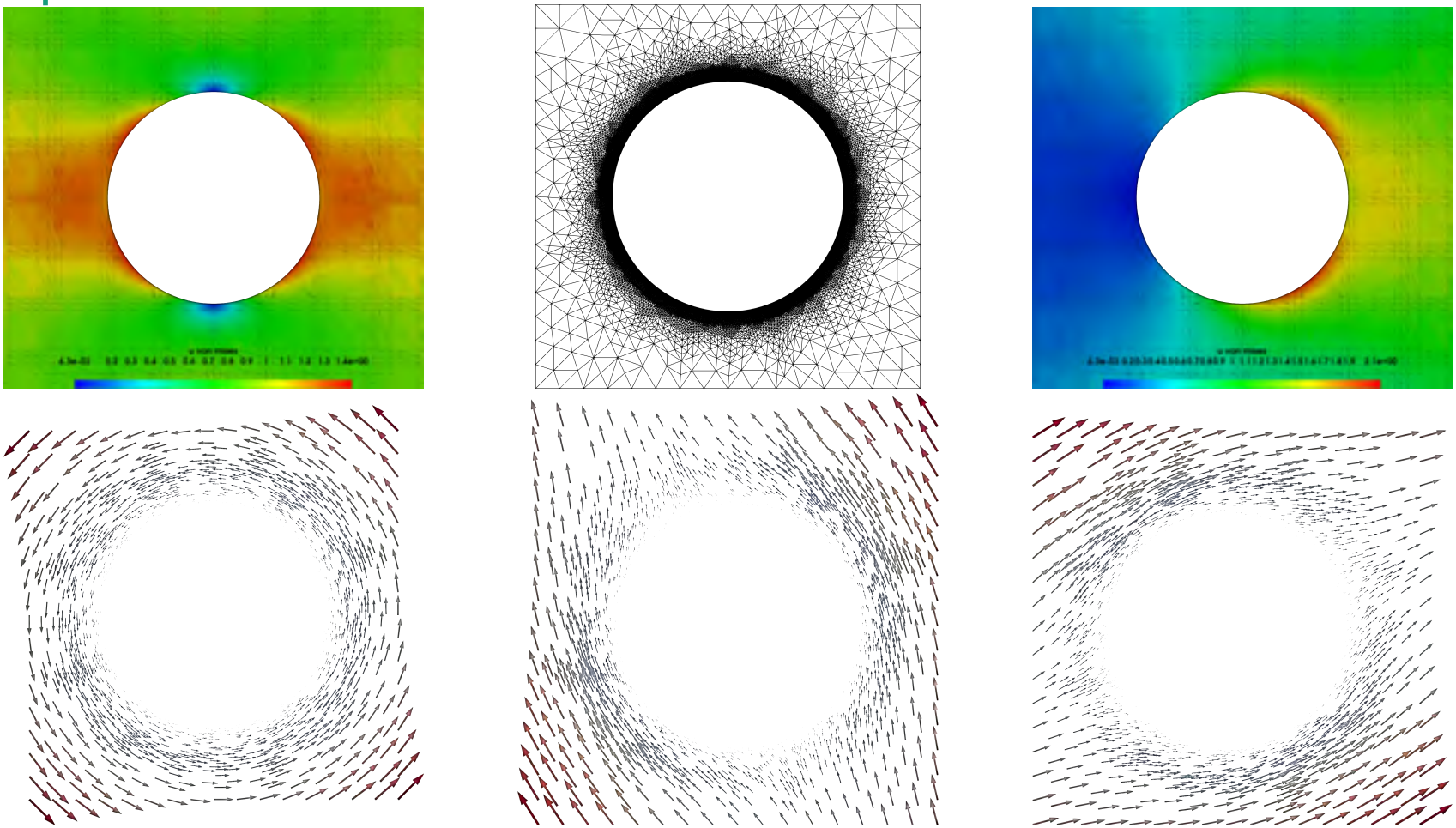
Stability

PUMA guarantees stability of basis of global approximation space via

```
st = create_stable_transformation((dot(u,v)+inner(grad(u),grad(v)))*dx)
```

schweitzer@scai.fraunhofer.de

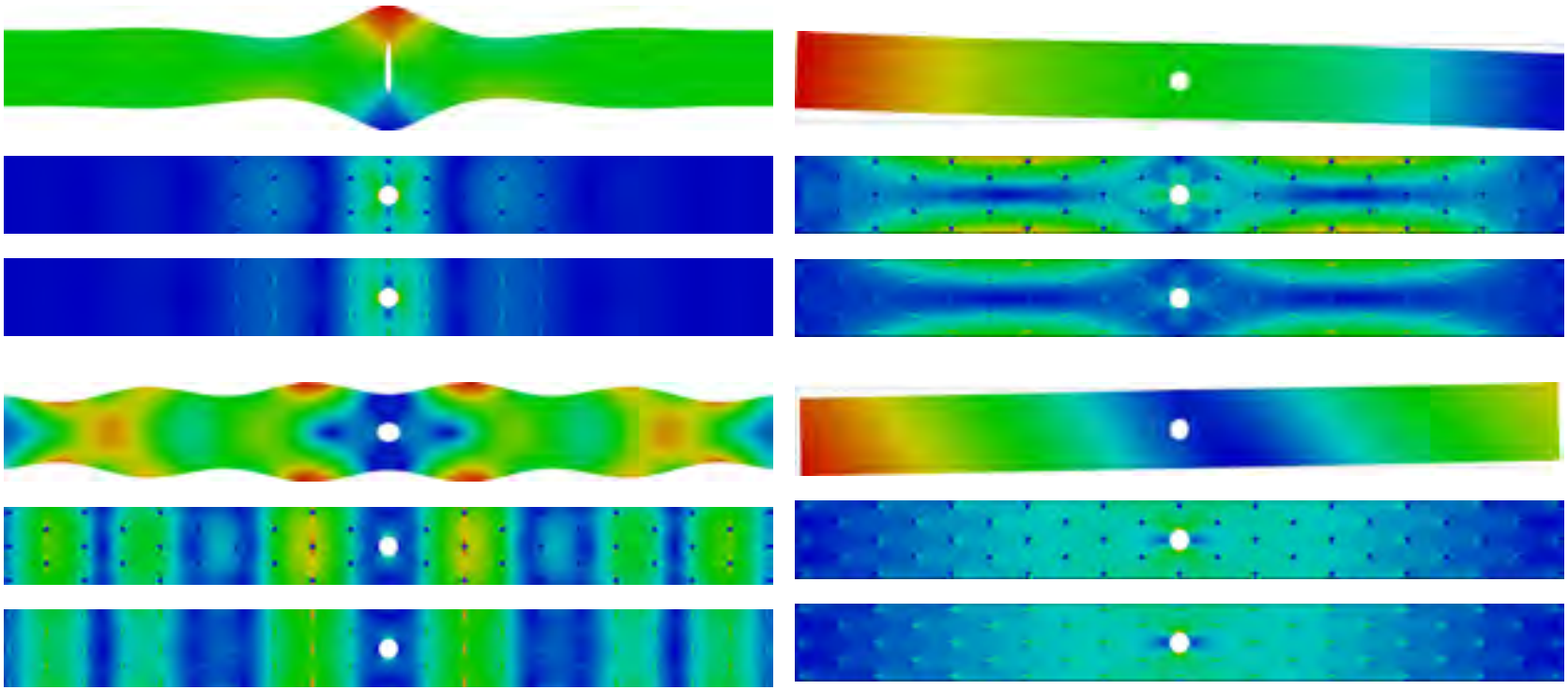
Example: FEM + PUMA + Numerical Enrichment



- PUMA can utilize pre-computed fine-scale basis functions e.g. from detailed FE analysis
- Fidelity of 10 such basis functions comparable to 50,000 locally refined finite elements

schweitzer@scai.fraunhofer.de

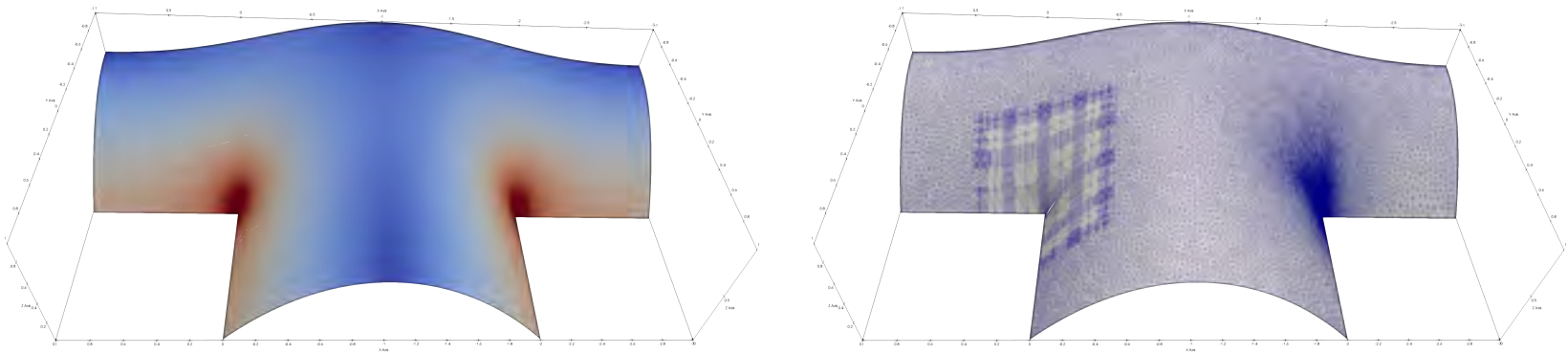
Failure in Composites



- Construction of optimal basis functions implictly resolving microstructure
- Exploration of design space and failure

schweitzer@scai.fraunhofer.de

Example: FEM + PUMA + Analytic Enrichment



- Left re-entrant corner modelled with 100 PUMA higher order enrichment functions
- Right re-entrant corner modelled with 50,000 locally refined finite elements

schweitzer@scai.fraunhofer.de

Example: PUMA Python Script

```
# Import the python packages required for the application
from puma import *
# Create Domain
channel = ConstructiveSolidGeometry(Box(Point(0,0), Point(2.2, 0.41)))
channel.setCoDim1PhysicalGroupName(0, "Left")
channel.setCoDim1PhysicalGroupName(1, "Right")
channel.setCoDim1PhysicalGroupName(2, "Bottom")
channel.setCoDim1PhysicalGroupName(3, "Top")
center = Point(0.2, 0.2)
radius = 0.05
cylinder = ConstructiveSolidGeometry(Ball(center, radius), 32)
cylinder.setSurfacePhysicalGroupName("Cylinder")
omega = ConstructiveSolidGeometryDomain(channel - cylinder)

# Define boundaries
gamma_inflow = PhysicalGroupSubDomain(omega, "Left")
gamma_outflow = PhysicalGroupSubDomain(omega, "Right")
gamma_walls = PhysicalGroupSubDomain(omega, "Bottom", "Top")
gamma_cylinder = PhysicalGroupSubDomain(omega, "Cylinder")
```

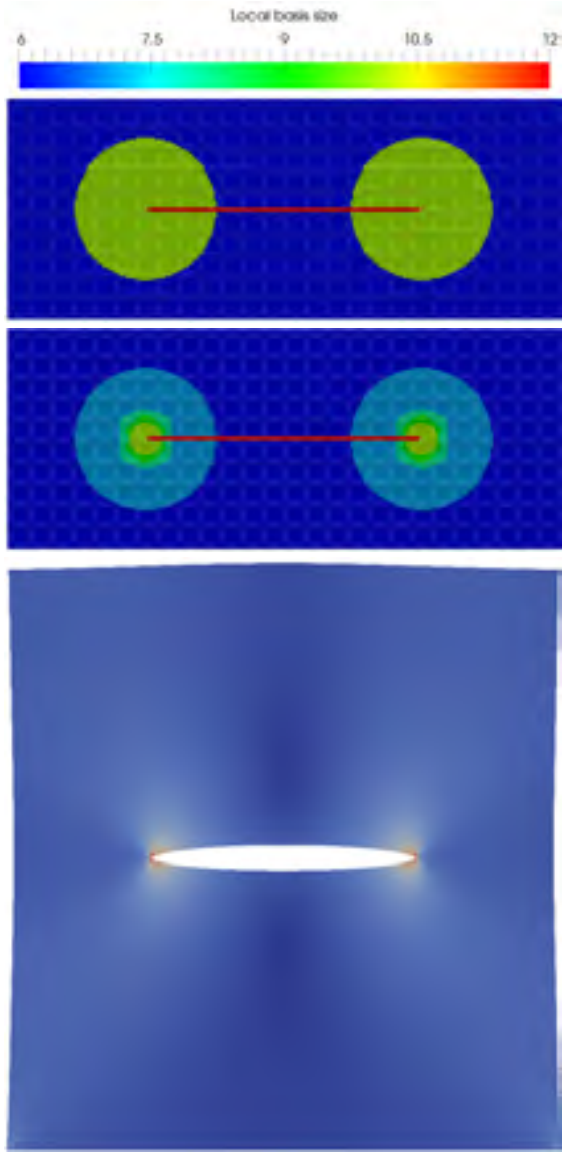
Example: PUMA Python Script

```
# total strain (infinitesimal)
def epsilon(v):
    return 0.5*(grad(v) + grad(v).T)
# thermal strain (isotropic)
def epsilon_t(T):
    return alpha*T*I_d
# mechanical strain
def epsilon_m(v, T):
    return epsilon(v) - epsilon_t(T)
# isotropic stress
def sigma(epsilon):
    return 2*mu*epsilon + lmbda*tr(epsilon)*I_d

V = PumSpace(omega, level=4, polynomial_degree=1)
W = V * V
V_u = W[0]
V_T = W[1]

deltau = VectorTrialFunction(V_u)
v = VectorTestFunction(V_u)
deltaT = ScalarTrialFunction(V_T)
s = ScalarTestFunction(V_T)

a_uu = inner(sigma(epsilon(deltau)), epsilon(v))*dx
a_TT = delta_t*k*inner(grad(deltaT), grad(s))*dx + rho*c*deltaT*s*dx
a_uT = -inner(sigma(epsilon_t(deltaT)), epsilon(v))*dx
a_Tu = 0*inner(deltau, n)*s*dx
a = a_uu + a_TT + a_uT + a_Tu
L_u = dot(f, v)*dx
L_T = delta_t*Q*s*dx + rho*c*T_last*s*dx
L = L_u + L_T

u = VectorFunction(V_u)
T = ScalarFunction(V_T)
uu = MixedFunction(W, [u, T])
solver = solve(a == L, uu, bcs, solver_parameters=solver_parameters)
```

schweitzer@scai.fraunhofer.de

Static non-linear elastic crack problem

```
# Define Dirichlet boundary conditions
g_D = (0, 0)
bcs = [DirichletBoundaryCondition(V, gamma_D, g_D)]

# Create stable transformation
du = VectorTrialFunction(V, with_phi=False)
v = VectorTestFunction(V, with_phi=False)
st = create_stable_transformation(inner(du,v)*dx + inner(grad(du),grad(v))*dx)

# Define variational problem
g_N = (0, 0.1)
f   = (0, 0)

du = VectorTrialFunction(V)
v = VectorTestFunction(V)
u = VectorFunction(V)

mat = StVenantKirchhoffMaterialModel(omega, E=1, nu=.3)

L = dot(f, v)*dx + dot(g_N, v)*ds(gamma_N)
R = mat.first_variation_of_strain_energy(v,u) - L
J = mat.second_variation_of_strain_energy(du,v,u)

# Solve for unknown function
solve(R == 0, u, bcs, st, J)
```



schweitzer@scai.fraunhofer.de

Hydraulic Fracturing

Governing Equations

Find (u, p) such that

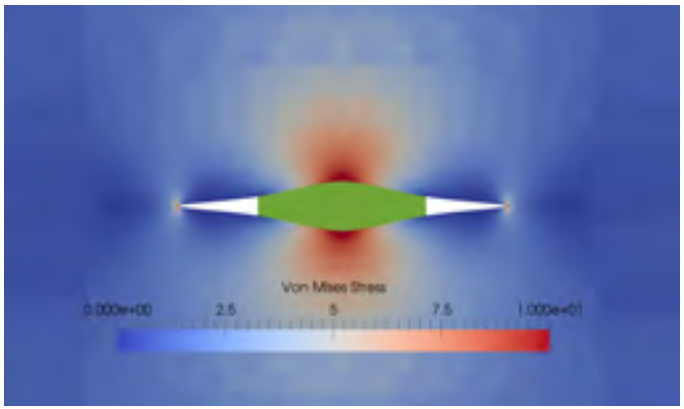
$$\begin{aligned} \langle \sigma(u), \epsilon(v) \rangle_\Omega - \langle \sigma_0 \cdot n, v \rangle_{\partial\Omega} - \langle p, v \rangle_{\Gamma_f} &= 0 \\ \langle \mathrm{cod}(\dot{u}), q \rangle_{\Gamma_f} + \frac{1}{\mu_f} \langle \mathrm{cod}(u)^3 \nabla p, \nabla q \rangle_{\Gamma_f} - \langle \frac{Q_0}{2} \delta_0, q \rangle_{\Gamma_f} &= 0 \end{aligned}$$

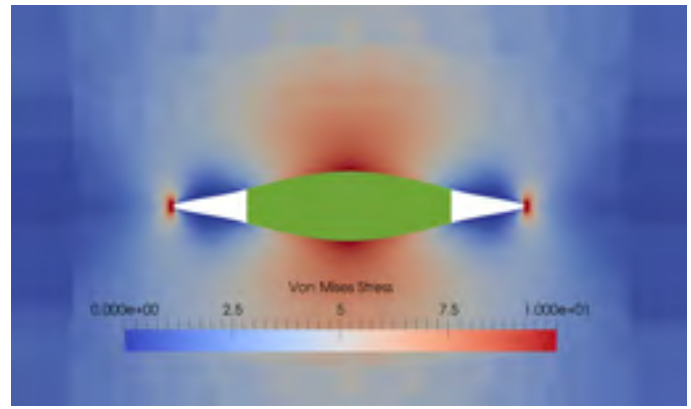
with

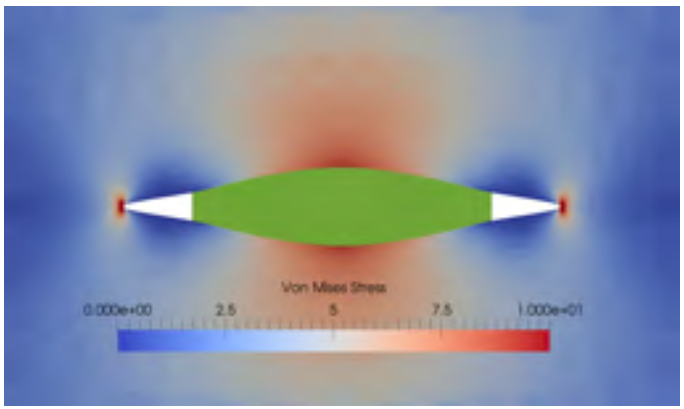
$$\begin{aligned} \sigma(u) &:= 2\mu\varepsilon(u) + \lambda\, \mathrm{trace}(\varepsilon(u))\mathbb{I} \\ \varepsilon(u) &:= \tfrac{1}{2}(\nabla u + (\nabla u)^T) \end{aligned}$$

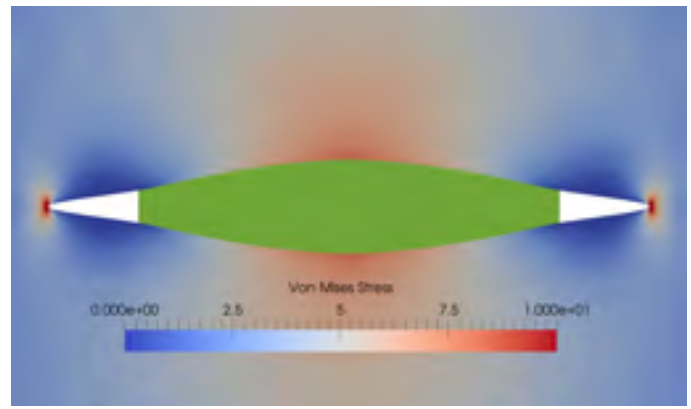
schweitzer@scai.fraunhofer.de

Hydraulic Fracturing









schweitzer@scai.fraunhofer.de

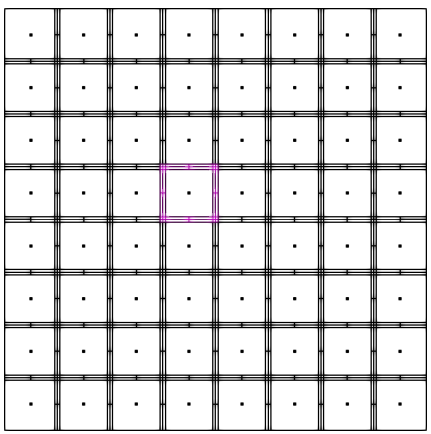
Fraunhofer SCAI

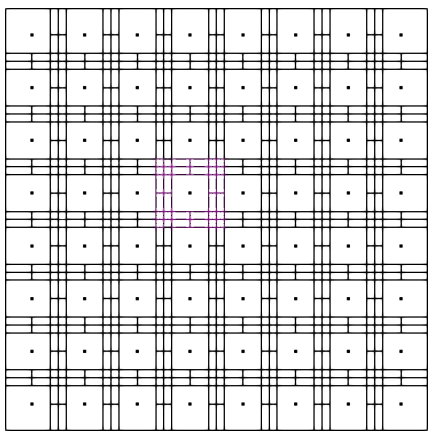
Arbitrary Smoothness of Approximation Space

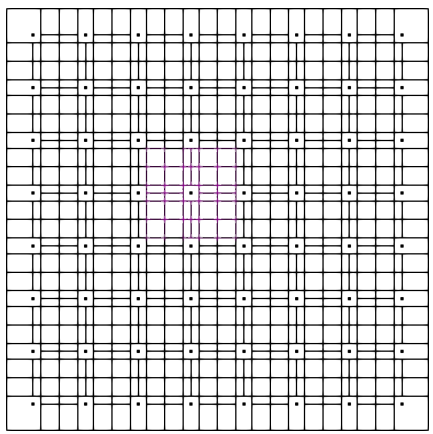
$$\varphi_i(x) := \frac{W_i(x)}{\sum_{j \in N_i} W_j(x)}$$

Weight Functions

- Weights W_j defined on stretched cells
- Choice of W_j determine regularity of PU functions φ_i
- All $W_j \in \mathcal{C}^s$ yields all $\varphi_i \in \mathcal{C}^s$
- Independent of number of neighboring patches $\#N_i$
- No increase in density of stiffness matrix







schweitzer@scai.fraunhofer.de

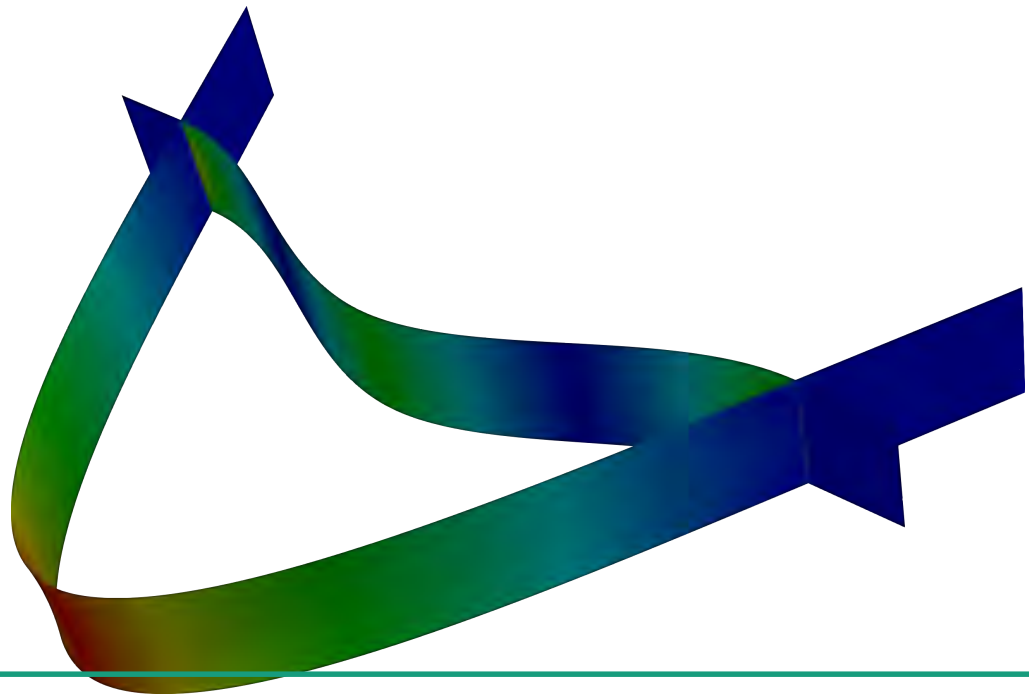
Fraunhofer SCAI

Arbitrary Systems of PDEs

Differential Operators

```
a = dot(grad(u), grad(v))*dx
a = laplace(u)*v*dx
a = inner(hessian(u), hessian(v))*dx
```

- Strong and weak forms of differential operators
- Second order or fourth order PDEs



schweitzer@scai.fraunhofer.de

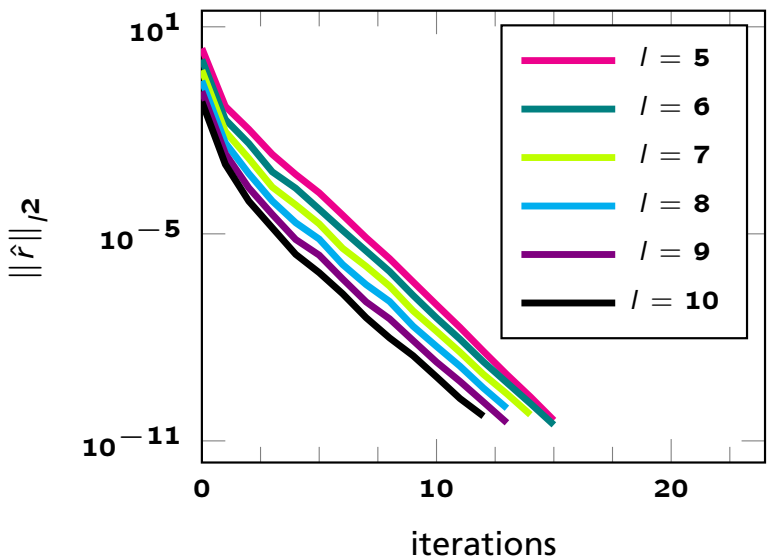
Example: PUMA Python Script

```
# Define variational problem
unit_normal = Normal()
unit_tangent = Tangent()
u = ScalarTrialFunction(V)
v = ScalarTestFunction(V)
bending_stiffness = E * plate_thickness**3 / (12 * (1-nu*nu))
flexural_rigidity  = bending_stiffness

def curvature(u):
    return -hessian(u)

def bending_strain(u):
    return z*curvature(u)

def bending_moments(u):
    return ( (bending_stiffness * (curvature(u)[0,0] + nu * curvature(u)[1,1]),
 ↪   bending_stiffness * (1-nu) * curvature(u)[0,1]),
            (bending_stiffness * (1-nu) * curvature(u)[1,0], bending_stiffness * (nu *
 ↪   curvature(u)[0,0] + curvature(u)[1,1])) )

a = inner(bending_moments(u), curvature(v))*dx
```

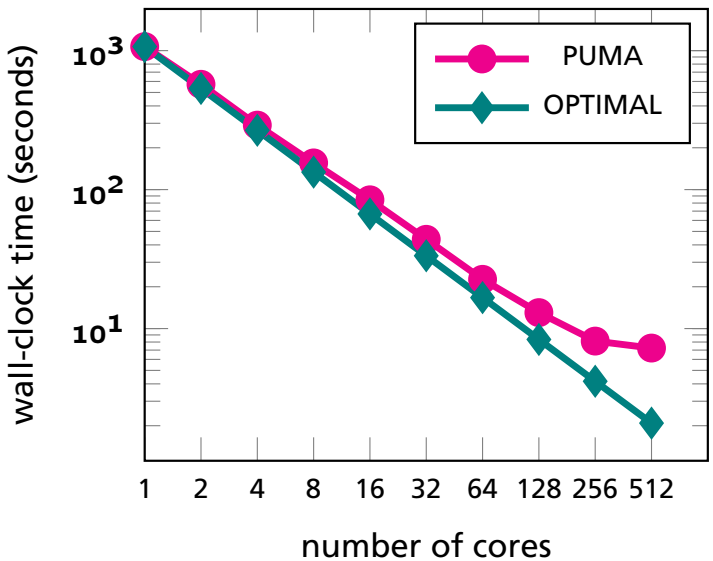
Efficient Solvers & Parallel Performance

- Guaranteed stability for arbitrary enrichments
- Construction of multilevel solvers agnostic of employed enrichments

V(2,2)-PCG convergence history



Strong scaling (Dof=3,145,728)



- PUMA's multilevel solver is rather robust wrt enrichments
- PUMA scales to thousands of cores and billions of Dof

schweitzer@scai.fraunhofer.de