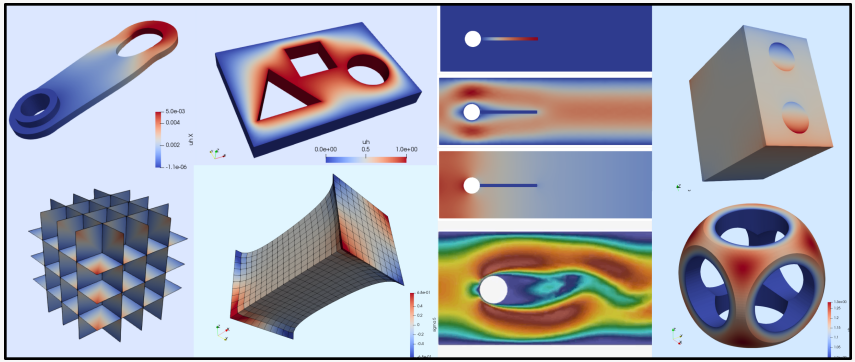
Why Gridap?



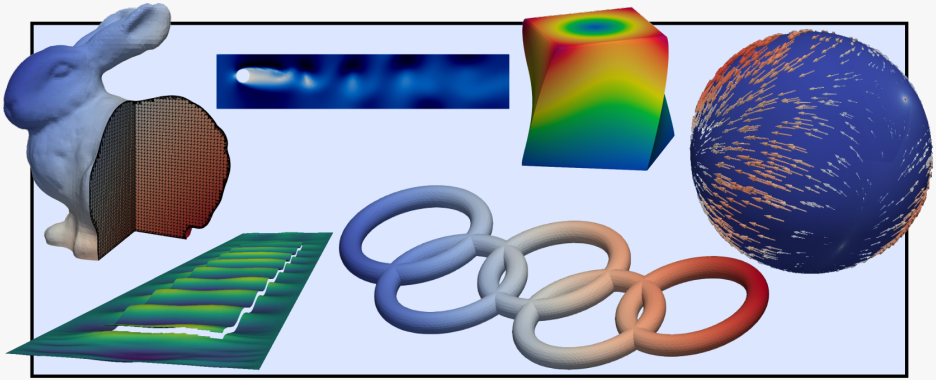
Santiago Badia, Monash University

Tutorial at NCI | Canberra 2023-11-24

MONASH University

What is Gridap?

- A general-purpose finite element library written in Julia

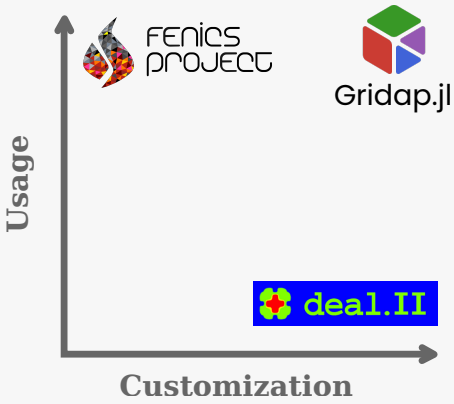


Main tools available

- Different flavours of FEM (GC, DG, embedded, ...)
- Simplices and n-cubes
- 1D, 2D, 3D, 4D...
- Lagrangian, Raviart-Thomas, Nedelec
- Complex number support
- Automatic differentiation (AD)
- Multi-field PDEs
- Time-dependent PDEs
- Serial and parallel implementation

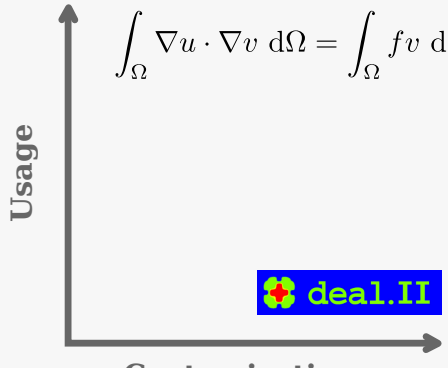
Why a new FEM library?

- We needed a library both easy to use and easy to customize



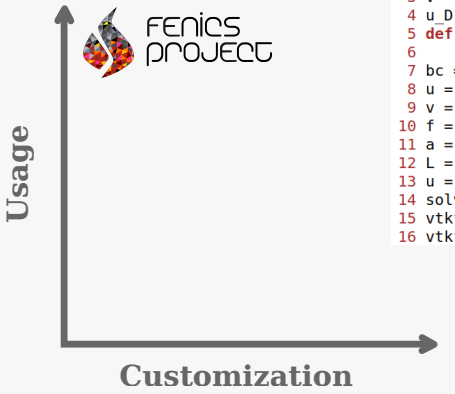
Why a new FEM library?

- We needed a library both easy to use and easy to customize



Why a new FEM library?

- We needed a library both easy to use and easy to customize

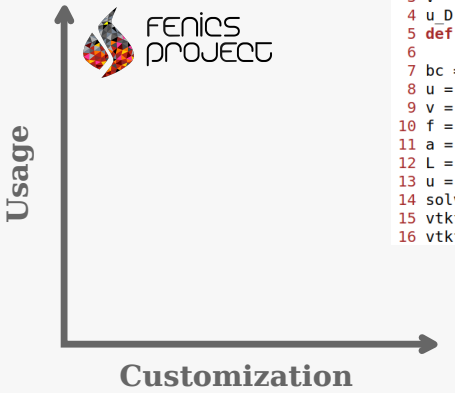



```
from fenics import *
mesh = UnitSquareMesh(8, 8)
V = FunctionSpace(mesh, 'P', 1)
u_D = Expression('1 + x[0]*x[0] + 2*x[1]*x[1]', degree=2)
def boundary(x, on_boundary):
    return on_boundary
bc = DirichletBC(V, u_D, boundary)
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(-6.0)
a = dot(grad(u), grad(v))*dx
L = f*v*dx
u = Function(V)
solve(a == L, u, bc)
vtkfile = File('poisson/solution.pvd')
vtkfile << u
```

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega$$

Why a new FEM library?

- We needed a library both easy to use and easy to customize



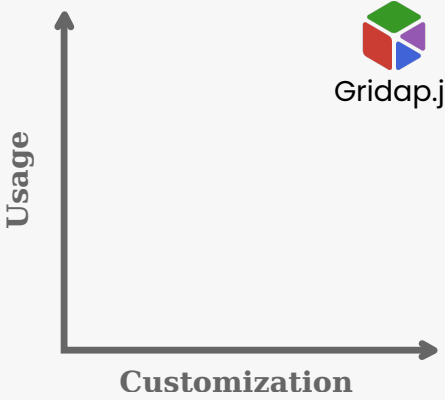

```
1 from fenics import *
2 mesh = UnitSquareMesh(8, 8)
3 V = FunctionSpace(mesh, 'P', 1)
4 u_D = Expression('1 + x[0]*x[0] + 2*x[1]*x[1]', degree=2)
5 def boundary(x, on_boundary):
6     return on_boundary
7 bc = DirichletBC(V, u_D, boundary)
8 u = TrialFunction(V)
9 v = TestFunction(V)
10 f = Constant(-6.0)
11 a = dot(grad(u), grad(v))*dx
12 L = f*v*dx
13 u = Function(V)
14 solve(a == L, u, bc)
15 vtkfile = File('poisson/solution.pvd')
16 vtkfile << u
```

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega$$

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

Why a new FEM library?

- We needed a library both easy to use and easy to customize



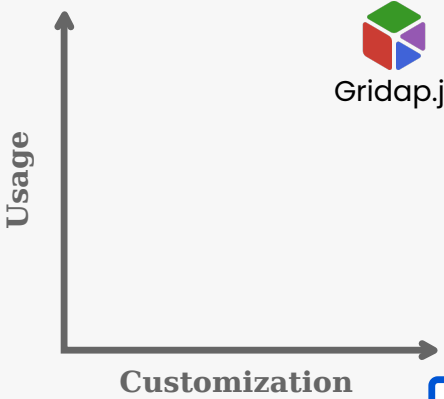
Gridap.jl

```
using Gridap
u(x) = x[1] + x[2]; f(x) = 0
domain = (0,1,0,1); cells = (4,4); order = 1
model = CartesianDiscreteModel(domain,cells)
Ω = Interior(model); dΩ = Measure(Ω,2*order)
reffe = ReferenceFE(lagrangian,Float64,order)
V = TestFESpace(Ω,reffe)
U = TrialFESpace(V,u)
a(u,v) = ∫( ∇(u)⋅∇(v) )dΩ
l(v) = ∫(v*f)dΩ
op = AffineFEOperator(a,l,U,V)
uh = solve(op)
writevtk(Ω,"results",cellfields=["uh"=>uh])
```

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega$$

Why a new FEM library?

- We needed a library both easy to use and easy to customize



Gridap.jl

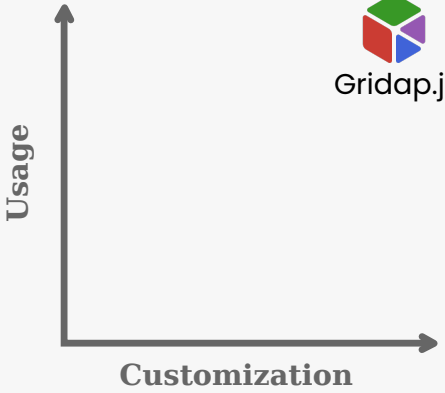
```
 1 using Gridap
 2 u(x) = x[1] + x[2]; f(x) = 0
 3 domain = (0,1,0,1); cells = (4,4); order = 1
 4 model = CartesianDiscreteModel(domain,cells)
 5 Ω = Interior(model); dΩ = Measure(Ω,2*order)
 6 reffe = ReferenceFE(lagrangian,Float64,order)
 7 V = TestFESpace(Ω,reffe)
 8 U = TrialFESpace(V,u)
 9 a(u,v) = ∫( ∇(u)⋅∇(v) )dΩ
10 l(v) = ∫(v*f)dΩ
11 op = AffineFEOperator(a,l,U,V)
12 uh = solve(op)
13 writevtk(Ω,"results",cellfields=["uh"=>uh])
```

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega$$

```
 9 a(u,v) = ∫( ∇(u)⋅∇(v) )dΩ
10 l(v) = ∫(v*f)dΩ
```

Why a new FEM library?

- We needed a library both easy to use and easy to customize

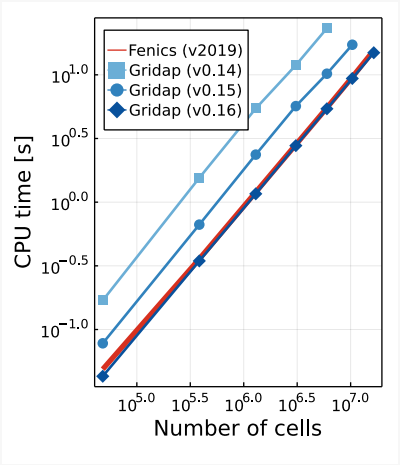


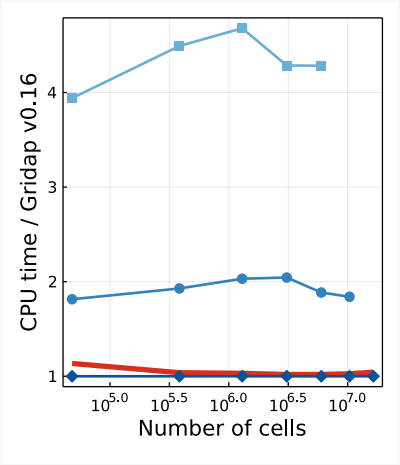
Gridap.jl

```julia
using Gridap
using Gridap.Fields
using Gridap.Polynomials
using Gridap.ReferenceFEs
using LinearAlgebra
using FillArrays

u(x) = x[1] + x[2]; f(x) = 0
domain = (0,1,0,1); cells = (4,4); order = 1
model = CartesianDiscreteModel(domain,cells)
reffe = ReferenceFE(QUAD,lagrangian,Float64,order)

# Reference quadrature rule
q =  Point{2,Float64}[(.5,.5)]
w = [1.]

# Cell-wise shape functions
ncells = length(conn)
cell_s = Fill(s,ncells)

# Cell-wise quadrature
cell_q = Fill(q,ncells)
cell_w = Fill(w,ncells)

# Geometrical map and Jacobian
cell_nodes = lazy_map(
  Broadcasting(ReIndex(nodes)),conn)
cell_φ = lazy_map(
  linear_combination,cell_nodes,cell_s)
cell_Jt = lazy_map(∇(),cell_φ)

# Cell-wise shape function derivatives
cell_∇ref_s = lazy_map(Broadcasting(∇),cell_s)
cell_invJt = lazy_map(Operation(inv),cell_Jt)
cell_∇s = lazy_map(
  Broadcasting(Operation(⋅)),cell_invJt,cell_∇ref_s)

# Cell-wise stifness matrix
cell_∇st = lazy_map(transpose,cell_∇s)
cell_∇s⋅∇st = lazy_map(
  Broadcasting(Operation(⋅)),cell_∇s,cell_∇st)
cell_mat = lazy_map(
  integrate,cell_∇s⋅∇st,cell_q,cell_w,cell_Jt)

# Assembly
V = TestFESpace(model,reffe)
U = TrialFESpace(V,u)
zh = zero(U)
assem = SparseMatrixAssembler(U,V)
cell_dofs = get_cell_dof_ids(V)
cell_vals = get_cell_dof_values(zh)
cell_vec = attach_dirichlet(cell_mat,cell_vals)
matdata = ([cell_mat],[cell_dofs],[cell_mat])
vecdata = ([cell_vec],[cell_dofs])
A = assemble_matrix(assem,matdata)
b = assemble_vector(assem,vecdata)
x = A\b
uh = FEFunction(U,x)
Ω = Interior(model)
writevtk(Ω,"results",cellfields=["uh"=>uh])
```

Performance

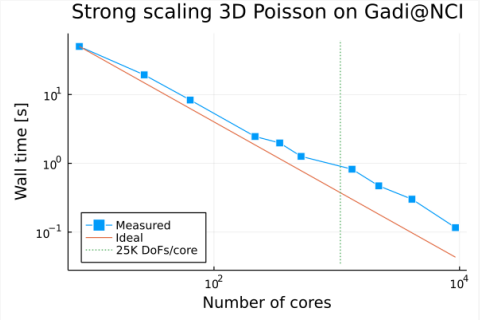
Wall time in FE assembly loop (Poisson eq.)

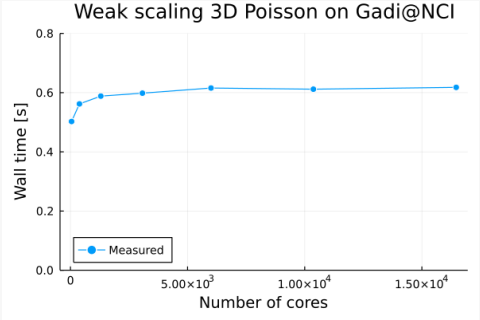




Parallel performance

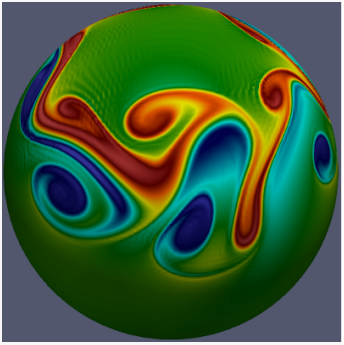
- 3D Poisson problem
- Wall time in FE assembly

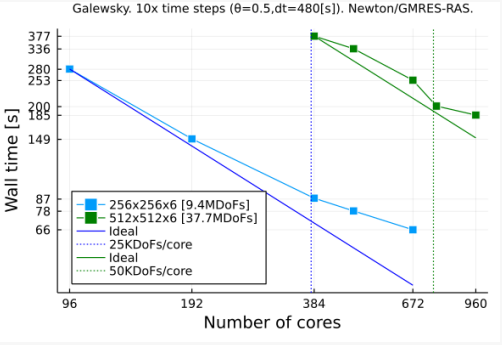




Parallel performance

Galewsky benchmark (shallow water eqs. on the sphere)





S Badia, AF Martin, FV (2022). GridapDistributed: a massively parallel finite element toolbox in Julia. Journal of Open Source Software

github.com/gridap

Gridap

Software ecosystem to solve PDEs in Julia

Follow

Overview | Repositories 30 | Projects | Packages | Teams | People 7 | Settings

Pinned

Customize pins

Gridap.jl Public
Grid-based approximation of partial differential equations in Julia
Julia ☆ 409 ⑂ 50

Tutorials Public
Start solving PDEs in Julia with Gridap.jl
Julia ☆ 73 ⑂ 28

GridapGmsh.jl Public
Gmsh generated meshes for Gridap
Julia ☆ 23 ⑂ 11

GridapEmbedded.jl Public
Embedded finite element methods in Julia
Julia ☆ 19 ⑂ 6

GridapODEs.jl Public archive
Time stepping for Gridap
Julia ☆ 32 ⑂ 6

GridapDistributed.jl Public
Parallel distributed-memory version of Gridap
Julia ☆ 47 ⑂ 6

Repositories

Find a repository… | Type | Language | Sort | New

GridapP4est.jl Public
Julia ☆ 5 MIT ⑂ 0 ⊙ 1 ⇄ 0 Updated 8 hours ago

View as: **Public**
You are viewing this page as a public user.
You can create a README file visible to anyone.

People

Invite someone

Top languages

Julia TeX

Most used topics

Manage

julia | finite-elements | numerical-methods | partial-differential-equations | pdes

Gridap tutorials

Gridap tutorials

Search docs

Version dev

Contents

How to start

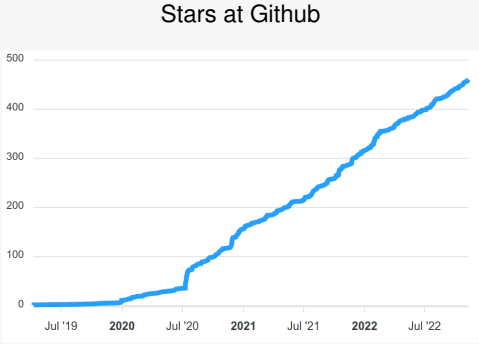
There are different ways to use the tutorials:

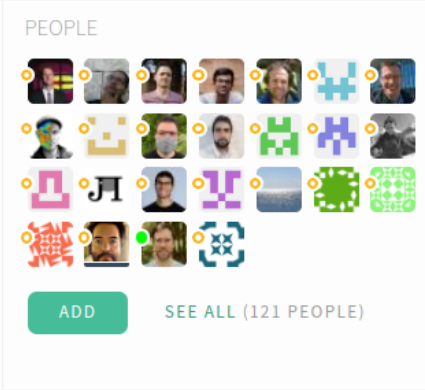
- **[Recommended]** Reading the html version of th
to the material with no setup steps. Simply click
- **[Recommended]** Running the Jupyter notebook
See instructions in the How to run the notebook
tutorials if you want to run the code and inspect the generated results with Paraview.

- Introduction
- Tutorial 1: Poisson equation
- Tutorial 2: Code validation
- Tutorial 3: Linear elasticity
- Tutorial 4: p-Laplacian
- Tutorial 5: Hyper-elasticity
- Tutorial 6: Poisson equation (with DG)
- Tutorial 7: Darcy equation (with RT)
- Tutorial 8: Incompressible Navier-Stokes
- Tutorial 9: Stokes equation
- Tutorial 10: Isotropic damage model
- Tutorial 11: Fluid-Structure Interaction
- Tutorial 12: Electromagnetic scattering in 2D
- Tutorial 13: Low-level API Poisson equation
- Tutorial 14: On using DrWatson.jl
- Tutorial 15: Interpolation of CellFields
- Tutorial 16: Poisson equation on parallel distr
- Tutorial 17: Transient Poisson equation
- Tutorial 18: Topology optimization

Community of users

https://gitter.im/Gridap-jl/community





How to get started

Pre-requisites

- Basic knowledge of Julia and FEM

Learning resources

- Video at JuliaCon 2020 `https://www.youtube.com/watch?v=txcb3ROQBS4`
- Video at JuliaCon 2021 `https://www.youtube.com/watch?v=hsQiFP4S5RY`
- Video at JuliaCon 2022 `https://www.youtube.com/watch?v=heeiSoKnlUk`
- Tutorials `https://gridap.github.io/Tutorials/dev/`
- F. Verdugo, S. Badia (2022). The software design of Gridap: a Finite Element package based on the Julia JIT compiler. Computer Physics Communications.

How to get help

- Ask in our chat `https://gitter.im/Gridap-jl/community`
- If you found a bug: Ask in the chat then open an issue with a reproducer
- If you want to contribute: Notify us via the chat before substantial work