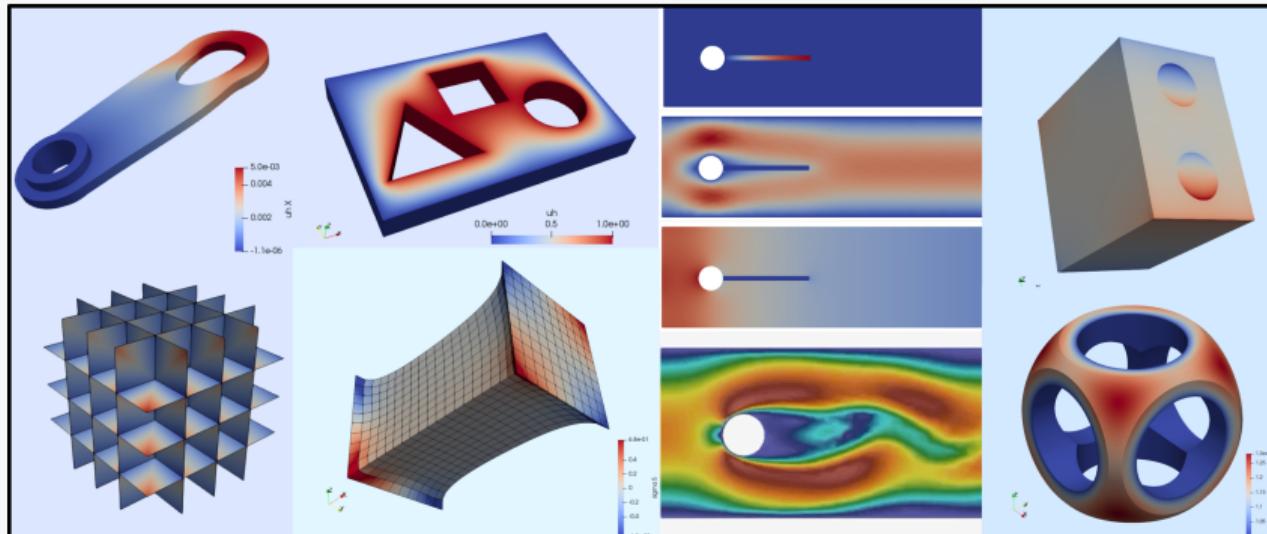


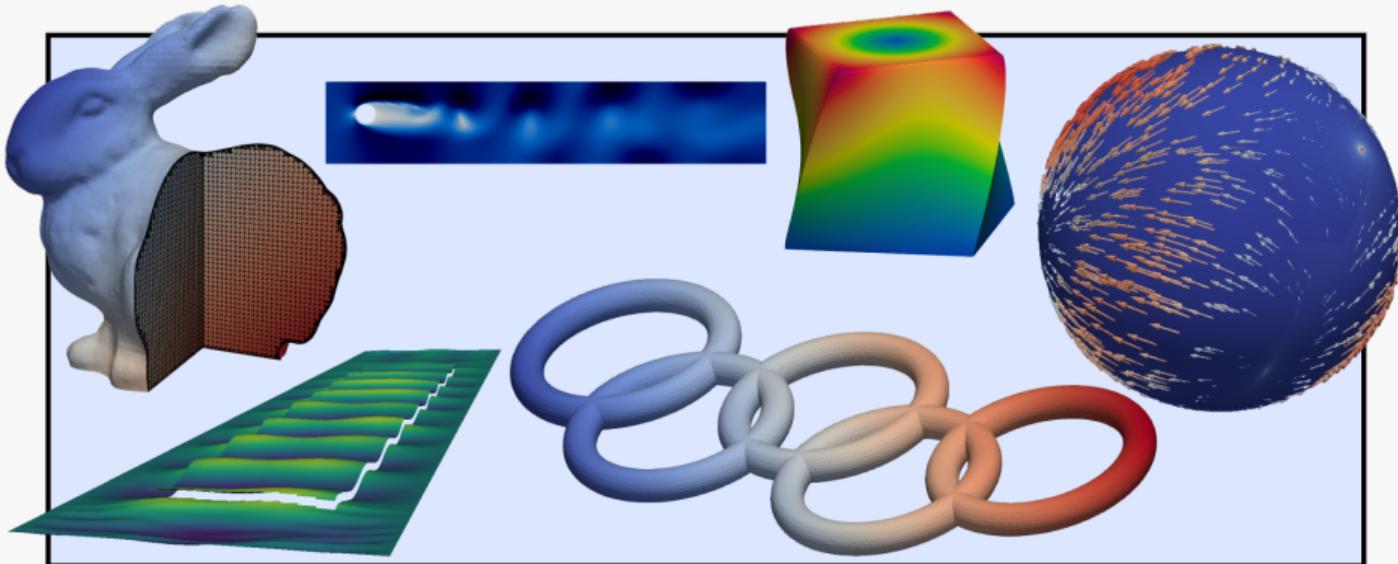
Why Gridap?



Santiago Badia, Monash University
Tutorial at NCI | Canberra 2023-11-24

What is Gridap?

- ▶ A general-purpose finite element library written in Julia

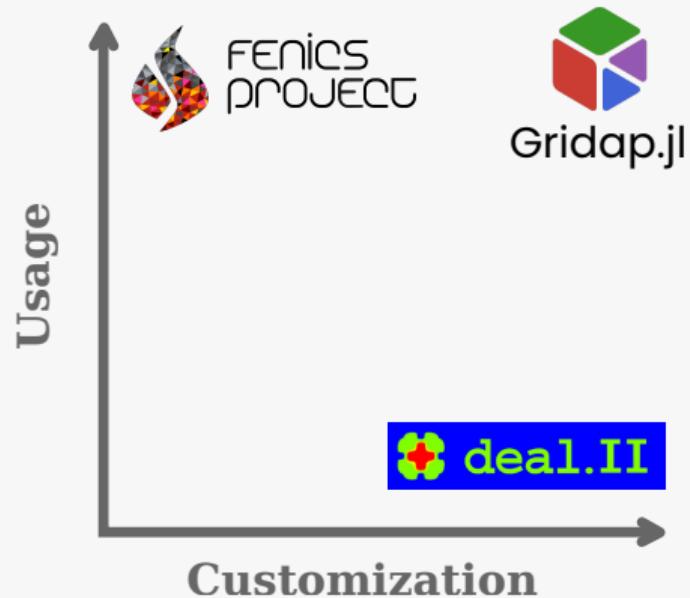


Main tools available

- ▶ 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

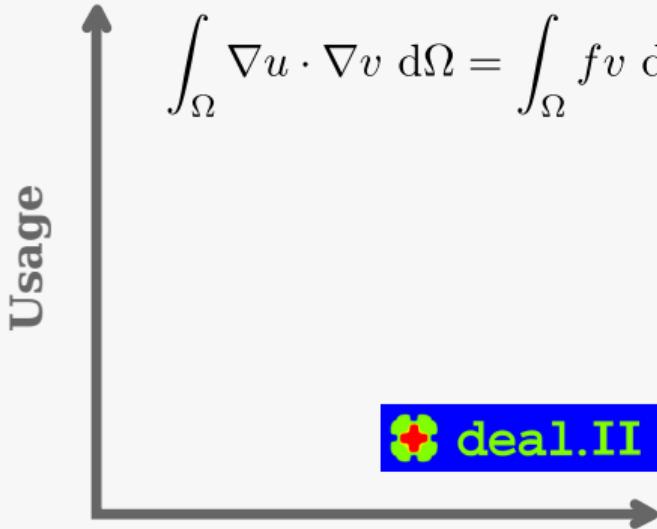
Why a new FEM library?

- 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

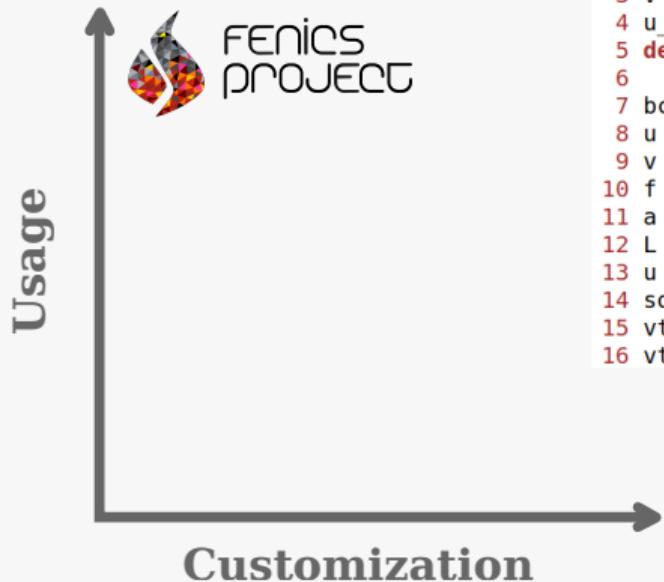


Customization

```
1 #include <deal.II/grid/tria.h>
2 #include <deal.II/grid/dof_handler.h>
3 #include <deal.II/grid/grid_generator.h>
4 #include <deal.II/grid/tria_accessor.h>
5 #include <deal.II/base/quadrature_lib.h>
6 #include <deal.II/base/function.h>
7 #include <deal.II/base/tensor.h>
8 #include <deal.II/base/quadrature_point.h>
9 #include <deal.II/base/values.h>
10 #include <deal.II/base/quadrature_lib.h>
11 #include <deal.II/base/tensor.h>
12 #include <deal.II/base/tensor_product.h>
13 #include <deal.II/base/matrix_tools.h>
14 #include <deal.II/base/vector.h>
15 #include <deal.II/base/matrix.h>
16 #include <deal.II/base/sparse_matrix.h>
17 #include <deal.II/base/dynamic_sparsity_pattern.h>
18 #include <deal.II/base/numerics.h>
19 #include <deal.II/base/preconditioner.h>
20 #include <deal.II/base/numerical_data.h>
21 #include <deal.II/base/numerical_data.h>
22 #include <deal.II/base/numerical_data.h>
23 using namespace dealii;
24 class Step3
25 {
26 public:
27     Step3();
28     ~Step3();
29 private:
30     void make_grid();
31     void setup_system();
32     void assemble_system();
33     void solve();
34     void output_results() const;
35     Triangulation<2> triangulation;
36     FE_Q<2> fe;
37     DoFHandler<2> dof_handler;
38     DynamicSparsityPattern system_matrix;
39     SparseMatrix<double> system_matrix;
40     Vector<double> system_rhs;
41     SolverCG<double> solver;
42 };
43 Step3::Step3()
44 : dof_handler(triangulation)
45 {
46     // ...
47     dof_handler.distribute_dofs(fe);
48 }
49 void Step3::make_grid()
50 {
51     GridGenerator::hyper_cube(triangulation, -1, 1);
52     triangulation.refine_global(5);
53     std::cout << "Number of active cells: " << triangulation.n_active_cells();
54     std::endl;
55 }
56 void Step3::setup_system()
57 {
58     dof_handler.distribute_dofs(fe);
59     std::cout << "Number of degrees of freedom: " << dof_handler.n_dofs();
60     std::endl;
61     DynamicSparsityPattern dsp(dof_handler.n_dofs());
62     DoFTools<2> dof_tools(dof_handler, dsp);
63     dof_tools.create_sparsity_pattern(system_matrix, solution.reinit(dof_handler.n_dofs()));
64     system_matrix.reinit(dof_handler.n_dofs());
65     system_rhs.reinit(dof_handler.n_dofs());
66 }
67 void Step3::assemble_system()
68 {
69     QGauss<2> quadrature_formula(degree + 1);
70     FEValues<2> fe_values(fe,
71         quadrature_formula,
72         update_gradients | update_JxW_values);
73     const unsigned int dofs_per_cell = fe.dofs_per_cell;
74     FullMatrix<double> cell_matrix(dofs_per_cell, dofs_per_cell);
75     Vector<double> cell_rhs(dofs_per_cell);
76     std::vector<types::global_dof_index> local_dof_indices(dofs_per_cell);
77     for (const auto cell : dof_handler.active_cell_iterators())
78     {
79         fe_values.reinit(cell);
80         cell_matrix = 0;
81         cell_rhs = 0;
82         for (const unsigned int q_index : fe_values.quadrature_point_indices())
83         {
84             for (const unsigned int i : fe_values.dof_indices())
85                 for (const unsigned int j : fe_values.dof_indices())
86                 {
87                     cell_matrix(i, j) =
88                         (fe_values.shape_grad(i, q_index) * // grad phi_i(x, q)
89                          fe_values.shape_grad(j, q_index)) // grad phi_j(x, q)
90                         * // grad phi_i(x, q)
91                         fe_values.JxW(q_index);
92                     cell_rhs(i) += (fe_values.dof_index(i) *
93                                     fe_values.shape_value(i, q_index)) * // phi_i(x, q)
94                                     fe_values.dof_index(j); // phi_j(x, q)
95                 }
96             cell_matrix(i, i) += fe_values.dof_index(i) *
97                               fe_values.dof_index(i); // I(x, q)
98         }
99         for (const unsigned int i : fe_values.dof_indices())
100             for (const unsigned int j : fe_values.dof_indices())
101                 system_matrix.add(local_dof_indices[i],
102                                   local_dof_indices[j]);
103         for (const unsigned int i : fe_values.dof_indices())
104             system_matrix.local_dof_indices[i] = cell_matrix(i, i);
105         for (const unsigned int i : fe_values.dof_indices())
106             system_rhs.local_dof_indices[i] = cell_rhs(i);
107     }
108     for (const unsigned int i : fe_values.dof_indices())
109         system_rhs.local_dof_indices[i] += cell_rhs(i);
110 }
111 std::map<types::global_dof_index, double> boundary_values;
112 VectorTools::interpolate_boundary_values(dof_handler,
113                                         boundary_values,
114                                         Function::ZeroFunction());
115 MatrixTools::apply_boundary_values(boundary_values,
116                                     system_matrix,
117                                     solution,
118                                     system_rhs);
119 }
120 void Step3::solve()
121 {
122     DataOut<2> data_out;
123     data_out.set_name("step3");
124     data_out.add_data_vector(solution, "solution");
125     data_out.build_patches();
126     data_out.stream_output("solution.vtk");
127     data_out.write_vtk(patch);
128 }
129 void Step3::output_results() const
130 {
131     DataOut<2> data_out;
132     data_out.set_name("step3");
133     data_out.add_data_vector(solution, "solution");
134     data_out.build_patches();
135     data_out.stream_output("solution.vtk");
136     data_out.write_vtk(patch);
137 }
138 void Step3::run()
139 {
140     make_grid();
141     setup_system();
142     assemble_system();
143     solve();
144     output_results();
145 }
146 int main()
147 {
148     desktop_grid();
149     Step3 step3;
150     step3.run();
151     return 0;
152 }
```

Why a new FEM library?

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

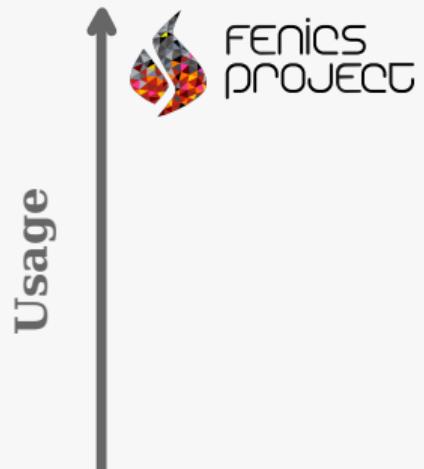


```
1 from fenics import *
2 mesh = UnitSquareMesh(8, 8)
3 V = FunctionSpace(mesh, 'P', 1)
4 u_D = Expression('1 + x[0]*x[0] + 2*x[1]*x[1]', degree=2)
5 def boundary(x, on_boundary):
6     return on_boundary
7 bc = DirichletBC(V, u_D, boundary)
8 u = TrialFunction(V)
9 v = TestFunction(V)
10 f = Constant(-6.0)
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
```

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

Why a new FEM library?

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



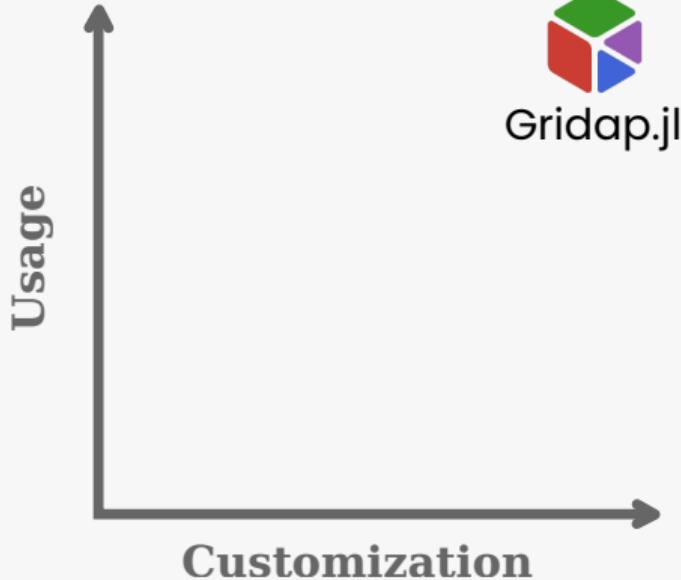
```
1 from fenics import *
2 mesh = UnitSquareMesh(8, 8)
3 V = FunctionSpace(mesh, 'P', 1)
4 u_D = Expression('1 + x[0]*x[0] + 2*x[1]*x[1]', degree=2)
5 def boundary(x, on_boundary):
6     return on_boundary
7 bc = DirichletBC(V, u_D, boundary)
8 u = TrialFunction(V)
9 v = TestFunction(V)
10 f = Constant(-6.0)
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
```

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

```
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

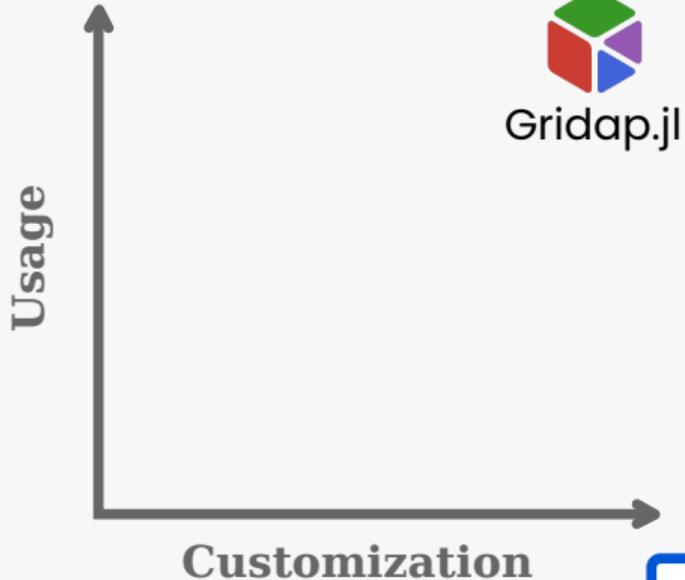


```
1 using Gridap
2 u(x) = x[1] + x[2]; f(x) = 0
3 domain = (0,1,0,1); cells = (4,4); order = 1
4 model = CartesianDiscreteModel(domain,cells)
5 Ω = Interior(model); dΩ = Measure(Ω,2*order)
6 reffe = ReferenceFE(lagrangian,Float64,order)
7 V = TestFESpace(Ω,reffe)
8 U = TrialFESpace(V,u)
9 a(u,v) = ∫( ∇(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])
```

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

Why a new FEM library?

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



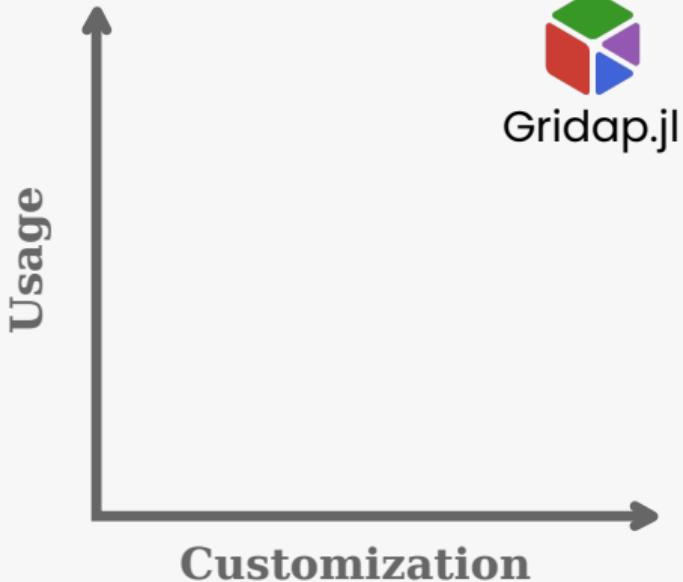
```
1 using Gridap
2 u(x) = x[1] + x[2]; f(x) = 0
3 domain = (0,1,0,1); cells = (4,4); order = 1
4 model = CartesianDiscreteModel(domain,cells)
5 Ω = Interior(model); dΩ = Measure(Ω,2*order)
6 reffe = ReferenceFE(lagrangian,Float64,order)
7 V = TestFESpace(Ω,reffe)
8 U = TrialFESpace(V,u)
9 a(u,v) = ∫( ∇(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])
```

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

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

Why a new FEM library?

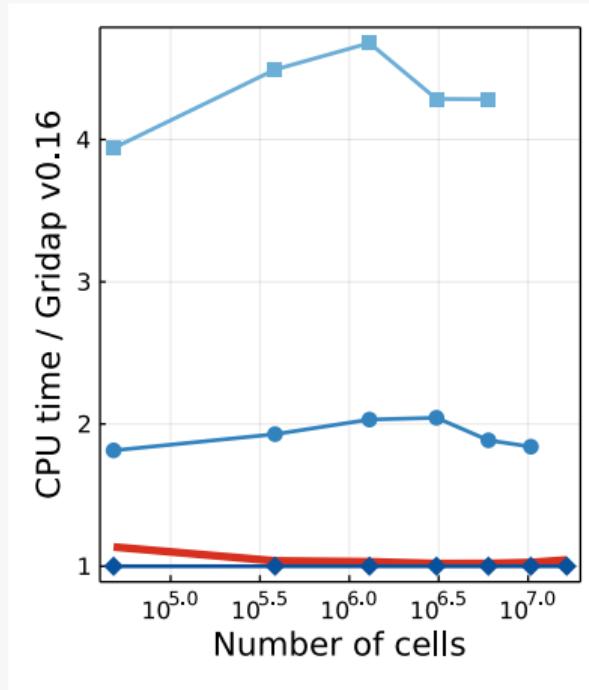
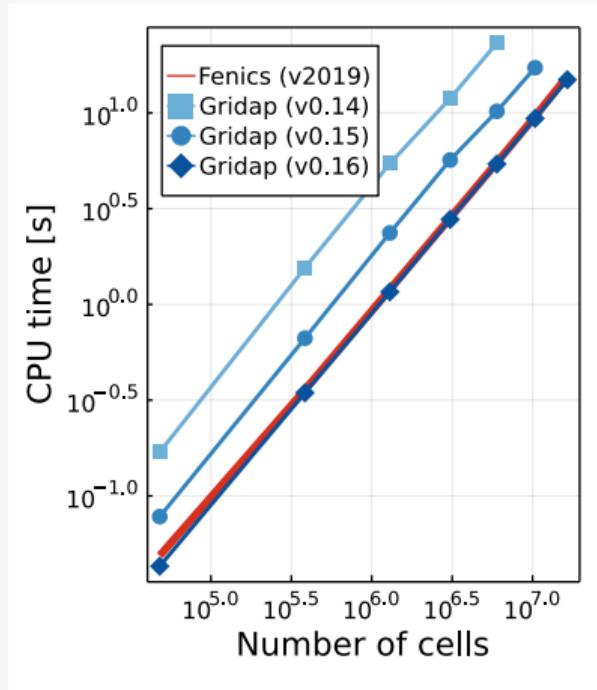
- We needed a library both easy to use and easy to customize



```
1 using Gridap
2 using Gridap.Fields
3 using Gridap.Polynomials
4 using Gridap.ReferenceFEs
5 using LinearAlgebra
6 using FillArrays
7
8 u(x) = x[1] + x[2]; f(x) = 0
9 domain = (0,1,0,1); cells = (4,4); order = 1
10 model = CartesianDiscreteModel(domain,cells)
11 reffe = ReferenceFE(QUAD,lagrangian,Float64,order)
12
13 # Reference quadrature rule
14 q = Point{2,Float64}[(.5,.5)]
15 w = [.1]
16
17 # Cell-wise shape functions
18 ncells = length(conn)
19 cell_s = Fill(s,ncells)
20
21 # Cell-wise quadrature
22 cell_q = Fill(q,ncells)
23 cell_w = Fill(w,ncells)
24
25 # Geometrical map and Jacobian
26 cell_nodes = lazy_map(
27    Broadcasting(Reindex(nodes)),conn)
28 cell_g = lazy_map(
29    linear_combination,cell_nodes,cell_s)
30 cell_Jt = lazy_map(V,cell_g)
31
32 # Cell-wise shape function derivatives
33 cell_Vref_s = lazy_map(Broadcasting(V),cell_s)
34 cell_invJt = lazy_map(Operation(inv),cell_Jt)
35 cell_Vs = lazy_map(
36    Broadcasting(Operation(-)),cell_invJt,cell_Vref_s)
37
38 # Cell-wise stiffness matrix
39 cell_Vst = lazy_map(transpose,cell_Vs)
40 cell_VsVst = lazy_map(
41    Broadcasting(Operation(-)),cell_Vs,cell_Vst)
42 cell_mat = lazy_map(
43    integrate,cell_VsVst,cell_q,cell_w,cell_Jt)
44
45 # Assembly
46 V = TestFESpace(model,reffe)
47 U = TrialFESpace(V,u)
48 zh = zero(U)
49 assem = SparseMatrixAssembler(U,V)
50 cell_dofs = get_cell_dof_ids(V)
51 cell_vals = get_cell_dof_values(zh)
52 cell_vec = attach_dirichlet(cell_mat,cell_vals)
53 metadata = ([cell_mat],[cell_dofs],[cell_mat])
54 vecdata = ([cell_vec],[cell_dofs])
55 A = assemble_matrix(assem,metadata)
56 b = assemble_vector(assem,vecdata)
57 x = A\b
58 uh = FEFFunction(U,x)
59 Q = Interior(model)
60 writevtk(Q,"results",cellfields=[uh=>uh])
```

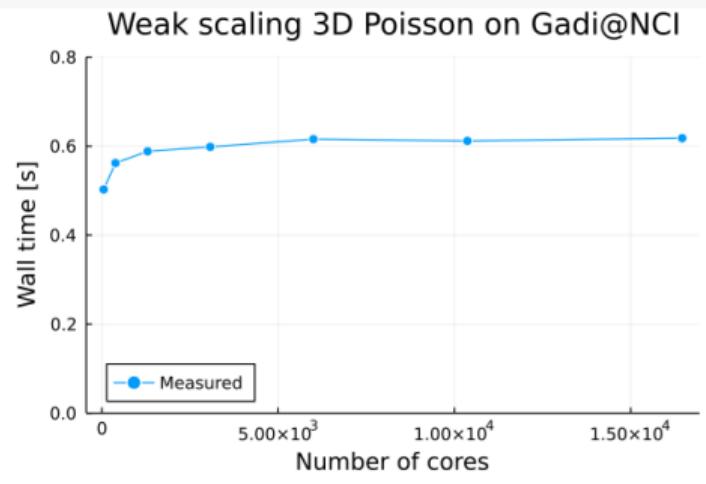
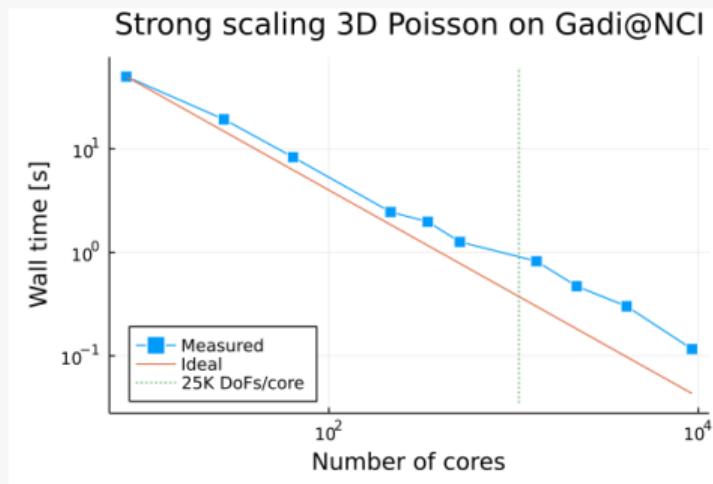
Performance

Wall time in FE assembly loop (Poisson eq.)



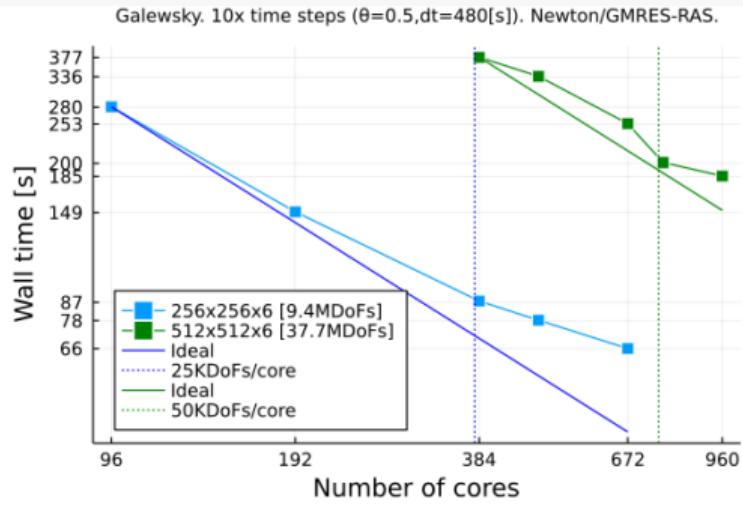
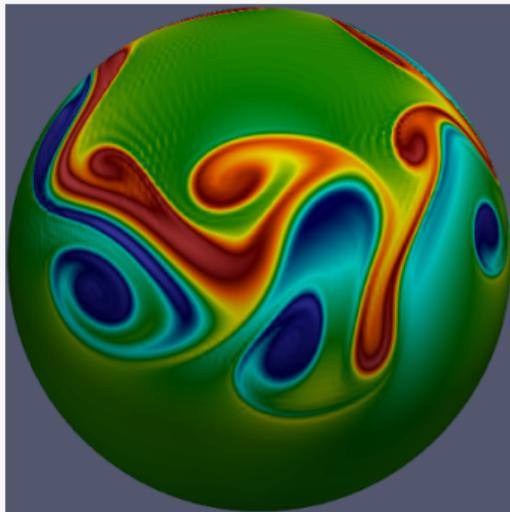
Parallel performance

- ▶ 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



Gridap

Software ecosystem to solve PDEs in Julia

Follow

[Overview](#) [Repositories 30](#) [Projects](#) [Packages](#) [Teams](#) [People 7](#) [Settings](#)

Pinned

Customize pins

[View as: Public](#)

You are viewing this page as a public user.

You can create a [README file](#) visible to anyone.

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

People



[Invite someone](#)

Top languages

Julia TeX

Most used topics

Manage

Julia finite-elements numerical-methods
partial-differential-equations pdes

Repositories

Find a repository... Type Language Sort New

GridapP4est.jl [Public](#)

Julia ⭐ 5 MIT 📂 0 ⏺ 1 📈 0 Updated 6 hours ago

8 of 12

Gridap tutorials



Gridap tutorials

Search docs

Introduction

- [Contents](#)
- [How to start](#)
- [How to run the notebooks locally](#)
- [How to pull the latest version of the tutorials](#)

1 Poisson equation

2 Code validation

3 Linear elasticity

4 p-Laplacian

5 Hyper-elasticity

6 Poisson equation (with DG)

7 Darcy equation (with RT)

8 Incompressible Navier-Stokes

9 Stokes equation

10 Isotropic damage model

11 Fluid-Structure Interaction

12 Electromagnetic scattering in 2D

Contents

- [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 distribution](#)
- [Tutorial 17: Transient Poisson equation](#)
- [Tutorial 18: Topology optimization](#)

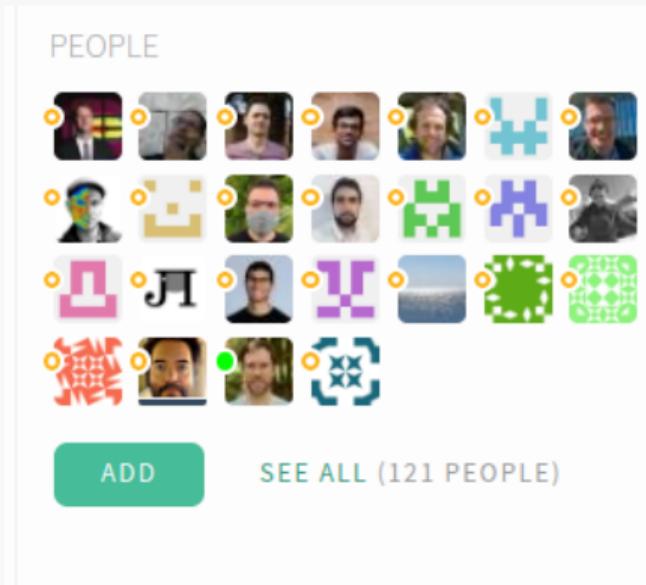
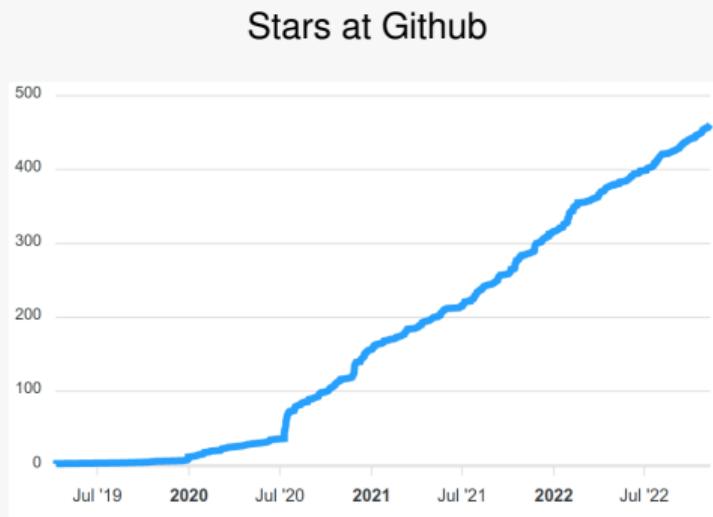
How to start

There are different ways to use the tutorials:

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

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=txcb3R0QBS4>
- ▶ Video at JuliaCon 2021 <https://www.youtube.com/watch?v=hsQiFP4S5RY>
- ▶ Video at JuliaCon 2022 <https://www.youtube.com/watch?v=heeiSoKn1Uk>
- ▶ 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