preCICE

# A Comprehensive Coupling Library for Large-scale Surface-coupled Multi-physics Problems

Miriam Mehl*, Benjamin Uekermann†, *Alexander Jaust**

alexander.jaust@ipvs.uni-stuttgart.de

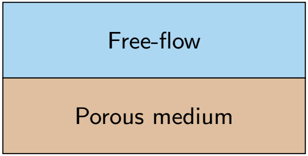
*Institute for Parallel and Distributed Systems, University of Stuttgart
†Department of Mechanical Engineering, Eindhoven University of Technology

University of Stuttgart
Germany

March 11, 2019

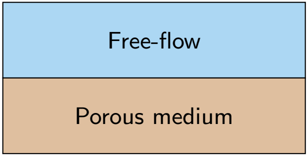
# Interface-coupled problems

- Porous media and free flow

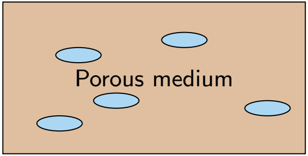


# Interface-coupled problems

- Porous media and free flow

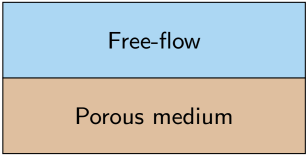


- Fractured porous media

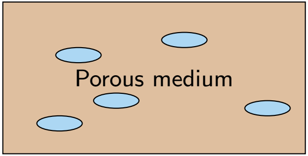


# Interface-coupled problems

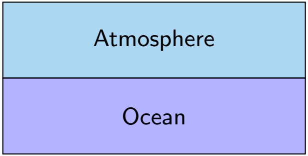
- Porous media and free flow



- Fractured porous media

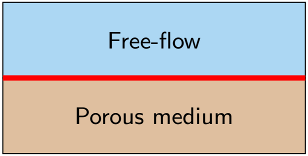


- Atmosphere and ocean

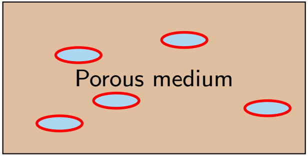


# Interface-coupled problems

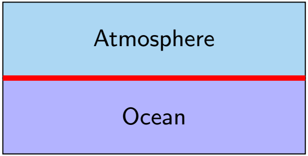
- Porous media and free flow



- Fractured porous media



- Atmosphere and ocean



# Table of contents

Introduction to preCICE

- Background information
- Coupling approach
- Features
- Coupling schemes

Getting started

- Get preCICE
- Tutorials
- Couple own code

Application to fractured porous media

Summary and outlook

# Table of contents

Introduction to preCICE

- Background information
- Coupling approach
- Features
- Coupling schemes

Getting started

- Get preCICE
- Tutorials
- Couple own code

Application to fractured porous media

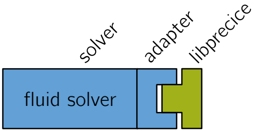
Summary and outlook

# Origin

## Fluid-structure interaction

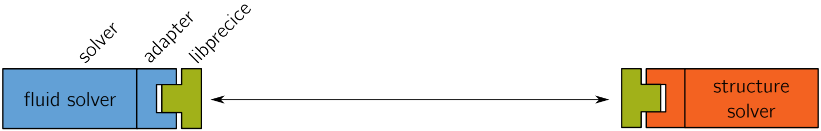
https://vimeo.com/user79566151

# preCICE – A Plug-and-Play Coupling Library



OpenFOAM
SU2
foam-extend

# preCICE – A Plug-and-Play Coupling Library



solver

adapter

libprecice

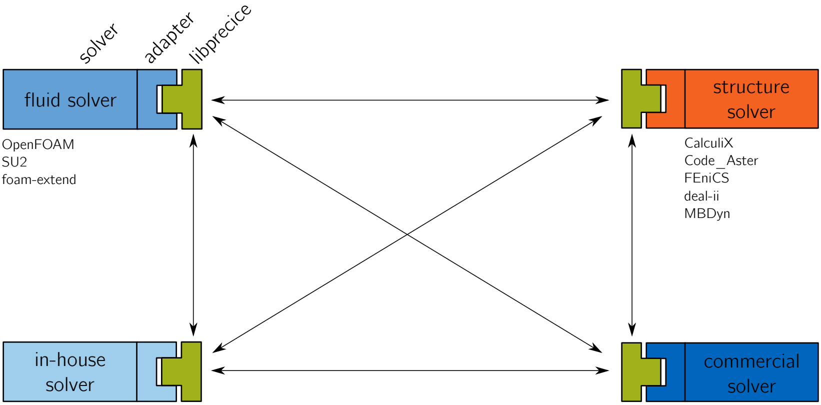
fluid solver

structure solver

OpenFOAM
SU2
foam-extend

CalculiX
Code_Aster
FEniCS
deal-ii
MBDyn

# preCICE – A Plug-and-Play Coupling Library



solver

adapter

libprecice

fluid solver

OpenFOAM
SU2
foam-extend

structure solver

CalculiX
Code_Aster
FEniCS
deal-ii
MBDyn

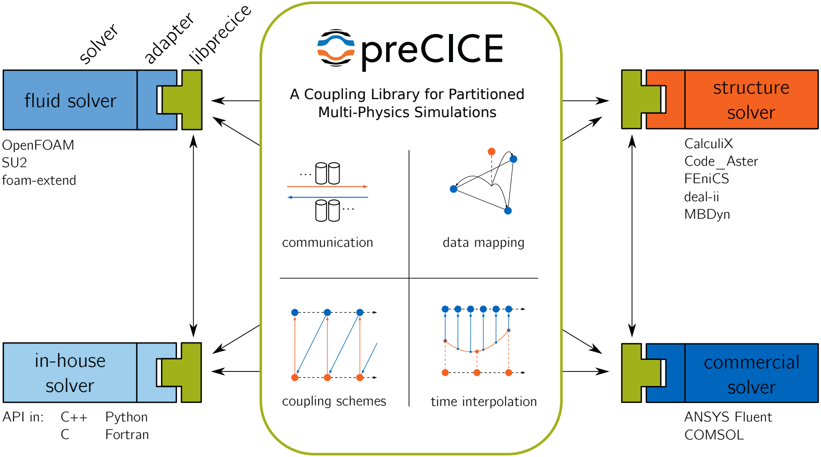
in-house solver

API in: C++ Python
C Fortran

commercial solver

ANSYS Fluent
COMSOL

# preCICE – A Plug-and-Play Coupling Library



solver

adapter

libprecice

fluid solver

OpenFOAM
SU2
foam-extend

in-house solver

API in: C++ Python
C Fortran

preCICE

A Coupling Library for Partitioned Multi-Physics Simulations

communication

data mapping

coupling schemes

time interpolation

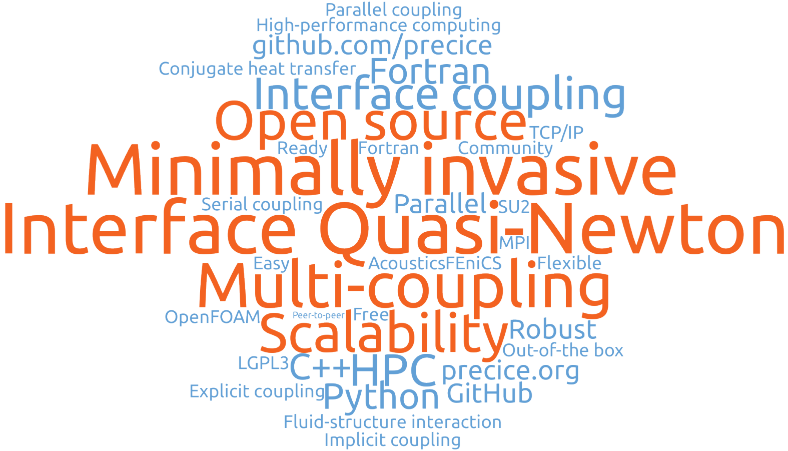
structure solver

CalculiX
Code_Aster
FEniCS
deal-ii
MBDyn

commercial solver

ANSYS Fluent
COMSOL

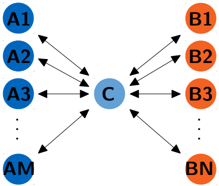
# Features



Parallel coupling
High-performance computing
github.com/precice
Conjugate heat transfer
Fortran
Interface coupling
Open source
TCP/IP
Ready
Fortran
Community
Minimally invasive
Serial coupling
Parallel
SU2
Interface Quasi-Newton
MPI
Easy
Acoustics
FEniCS
Flexible
Multi-coupling
OpenFOAM
Peer-to-peer
Free
Scalability
Robust
Out-of-the box
LGPL3
C++
HPC
precice.org
Explicit coupling
Python
GitHub
Fluid-structure interaction
Implicit coupling

# Scalability

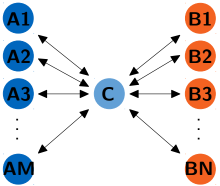
## Server-based concept



- All communication through central server
- Interface computations on server (in sequential)
- ⇒ Coupling becomes bottleneck for overall simulation already on moderate parallel systems

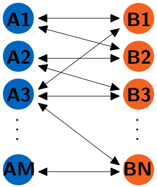
# Scalability

### Server-based concept



- All communication through central server
- Interface computations on server (in sequential)
- ⇒ Coupling becomes bottleneck for overall simulation already on moderate parallel systems

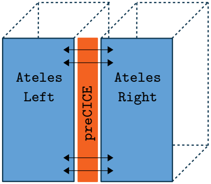
### Our peer-to-peer concept

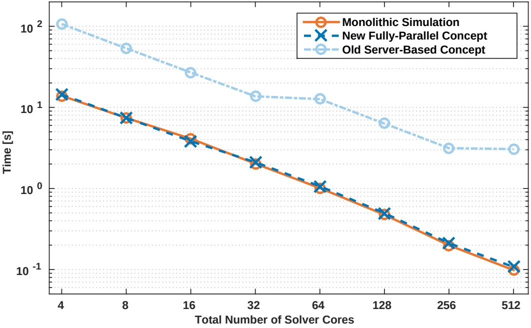


- No central entity
- ⇒ Easier to handle (user does not need to care about server)
- ⇒ No scaling issues

# Scalability

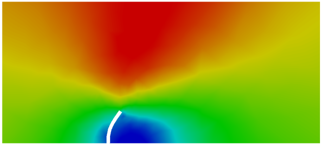
- Travelling density pulse (Euler equations) through artificial coupling interface
- DG solver Ateles (U Siegen), $7.1 \cdot 10^6$ dofs
- Nearest neighbor mapping and communication





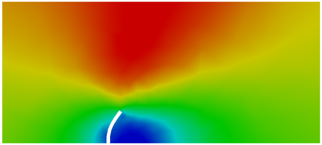
# Different couplings

- Fluid solver: $F : d \mapsto f$
- Structure solver: $S : f \mapsto d$
- Solve fixed-point problem: $(S \circ F)(d) \overset{!}{=} d$



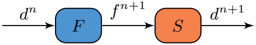
# Different couplings

- Fluid solver: $F : d \mapsto f$
- Structure solver: $S : f \mapsto d$
- Solve fixed-point problem: $(S \circ F)(d) \overset{!}{=} d$



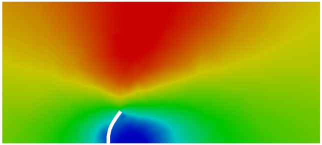
**explicit**

**serial**



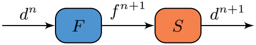
# Different couplings

- Fluid solver: $F : d \mapsto f$
- Structure solver: $S : f \mapsto d$
- Solve fixed-point problem: $(S \circ F)(d) \overset{!}{=} d$

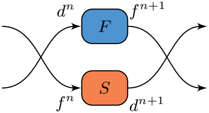


**explicit**

**serial**

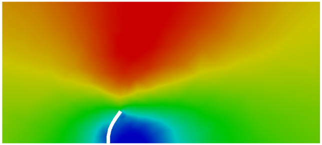


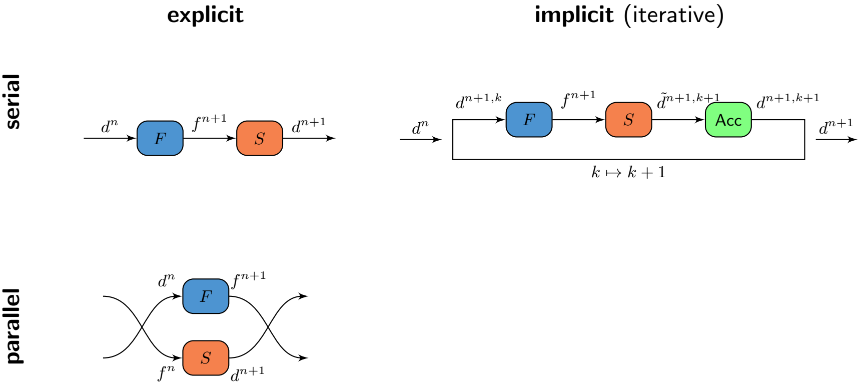
**parallel**



# Different couplings

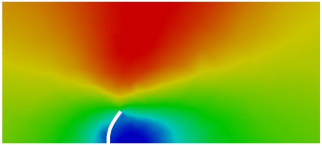
- Fluid solver: $F : d \mapsto f$
- Structure solver: $S : f \mapsto d$
- Solve fixed-point problem: $(S \circ F)(d) \overset{!}{=} d$





# Different couplings

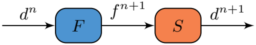
- Fluid solver: $F : d \mapsto f$
- Structure solver: $S : f \mapsto d$
- Solve fixed-point problem: $(S \circ F)(d) \overset{!}{=} d$

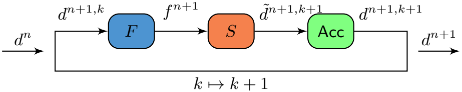


**explicit**

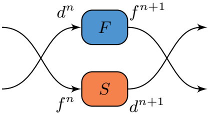
**implicit** (iterative)

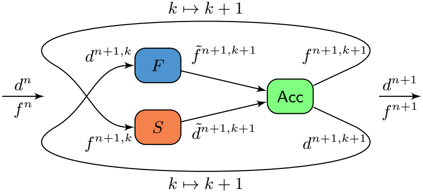
**serial**





**parallel**





# Why do I need implicit coupling?

## Fluid-structure interaction

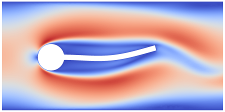
**Explicit** serial coupling

`https://vimeo.com/user79566151`

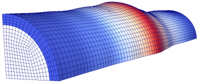
# Quasi-Newton Coupling (Accelerator/Postprocessing)

Coupled problem: $F : d \mapsto f,\ S : f \mapsto d \quad \rightsquigarrow \quad (S \circ F)(d) \overset{!}{=} d$

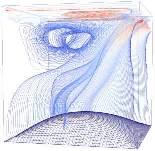
FSI3



3D-tube



Driven cavity



| Mean Iterations | Aitken | Quasi-Newton |
|---|---:|---:|
| FSI3 | 17.0 | 3.3 |
| 3D-tube | Div. | 7.5 |
| Driven cavity | 7.4 | 2.0 |

# Table of contents

Introduction to preCICE

- Background information
- Coupling approach
- Features
- Coupling schemes

Getting started

- Get preCICE
- Tutorials
- Couple own code

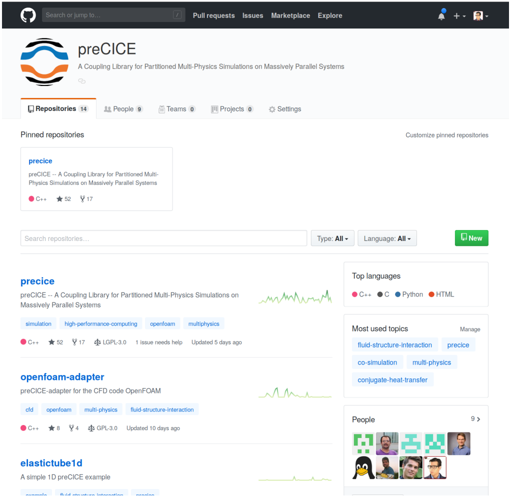
Application to fractured porous media

Summary and outlook

# Infrastructure

We are on GitHub: `https://github.com/precice`

- Main developers: University of Stuttgart and TU Munich
- LGPL3 license
- User documentation in the wiki



Search or jump to... / Pull requests Issues Marketplace Explore

preCICE

A Coupling Library for Partitioned Multi-Physics Simulations on Massively Parallel Systems

Repositories 14 People 9 Teams 0 Projects 0 Settings

Pinned repositories — Customize pinned repositories

**precice**
preCICE -- A Coupling Library for Partitioned Multi-Physics Simulations on Massively Parallel Systems
● C++ ★ 52 17

Search repositories... Type: All Language: All New

**precice**
preCICE -- A Coupling Library for Partitioned Multi-Physics Simulations on Massively Parallel Systems
simulation high-performance-computing openfoam multiphysics
● C++ ★ 52 17 LGPL-3.0 1 issue needs help Updated 5 days ago

**openfoam-adapter**
preCICE-adapter for the CFD code OpenFOAM
cfd openfoam multi-physics fluid-structure-interaction
● C++ ★ 8 4 GPL-3.0 Updated 10 days ago

**elastictube1d**
A simple 1D preCICE example

Top languages
● C++ ● C ● Python ● HTML

Most used topics — Manage
fluid-structure-interaction precice co-simulation multi-physics conjugate-heat-transfer

People 9

# Building

## Dependencies

- Eigen, Boost (version $\geq 1.60$), libxml2
- Optional: PETSc, Python (incl. Numpy), MPI

## The easy way

- **NEW in 1.4**: Debian package available
- Ubuntu 18.04: All dependencies available through distribution
- Ubuntu 16.04: All dependencies available except Boost

## Still doable

- macOS
- Other Linux distributions

## Experimental

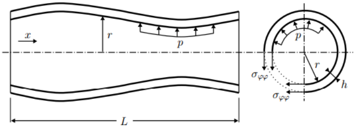
- Conda, Docker, Debian package, Spack
- Windows

# Tutorials

`https://github.com/precice/precice/wiki`

## 1D Elastic Tube

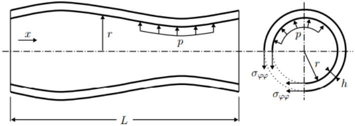
- Simple provided solvers
- Learn about API and configuration



# Tutorials

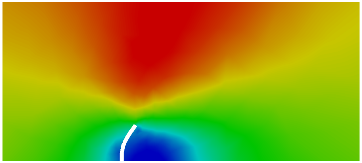
`https://github.com/precice/precice/wiki`

## 1D Elastic Tube



- Simple provided solvers
- Learn about API and configuration

## Flexible beam



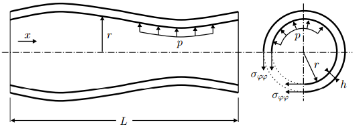
- Fluid-structure interaction
- Couple SU2 to CalculiX
- Learn about coupling schemes
- Also interactive version available in browser `http://run.coplon.de/`

# Tutorials

`https://github.com/precice/precice/wiki`

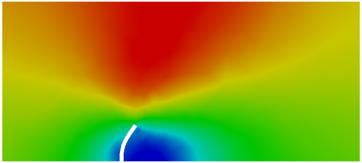
## 1D Elastic Tube

- Simple provided solvers
- Learn about API and configuration



## Flexible beam

- Fluid-structure interaction
- Couple SU2 to CalculiX
- Learn about coupling schemes
- Also interactive version available in browser `http://run.coplon.de/`



## More tutorials in the wiki!

- Conjugate heat transfer
- Structure-structure interaction
- . . .

# How to couple my own code?

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");
precice.setMeshVertices();
precice.initialize();

while (precice.isCouplingOngoing()) { // main time loop
solve();

precice.writeBlockVectorData();
precice.advance();
precice.readBlockVectorData();

endTimeStep(); // e.g. write results, increase time
}

precice.finalize();
```

- Timesteps, most arguments, and less important methods omitted.
- Full example in the wiki.
- API in C++, C, Fortran, and Python

# Table of contents

Introduction to preCICE

- Background information
- Coupling approach
- Features
- Coupling schemes

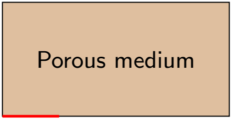
Getting started

- Get preCICE
- Tutorials
- Couple own code

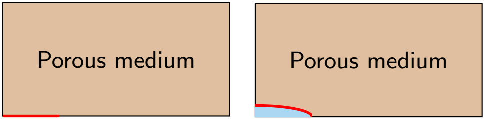
Application to fractured porous media

Summary and outlook

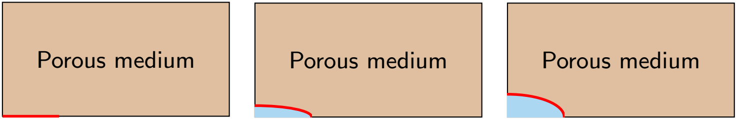
# Coupled flow and poromechanics



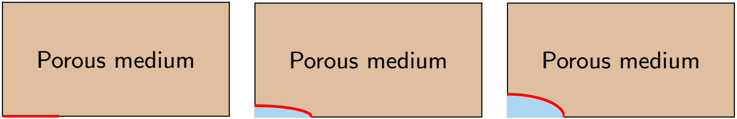
# Coupled flow and poromechanics



# Coupled flow and poromechanics

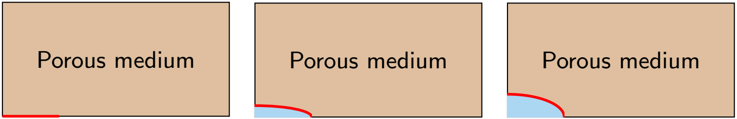


# Coupled flow and poromechanics



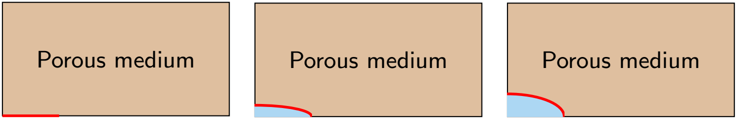
1. Poromechanics solver

# Coupled flow and poromechanics

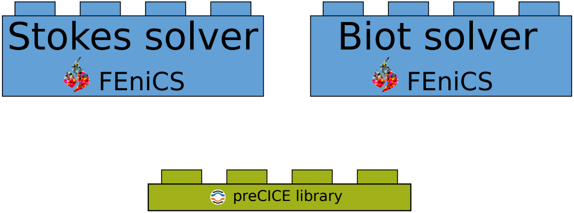


1. Poromechanics solver
2. Flow solver

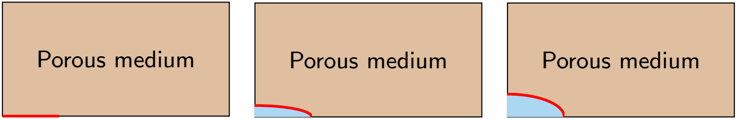
# Coupled flow and poromechanics



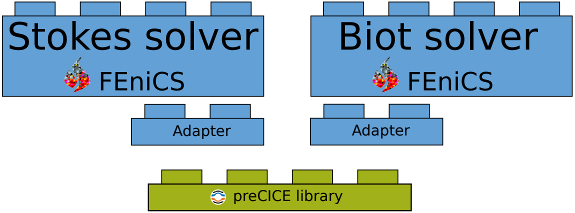
1. Poromechanics solver
2. Flow solver



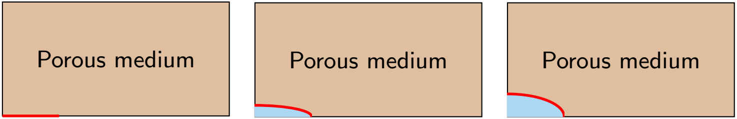
# Coupled flow and poromechanics



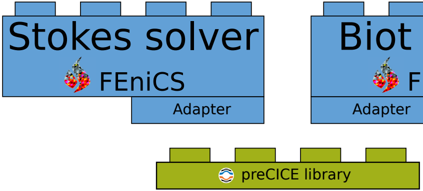
1. Poromechanics solver
2. Flow solver



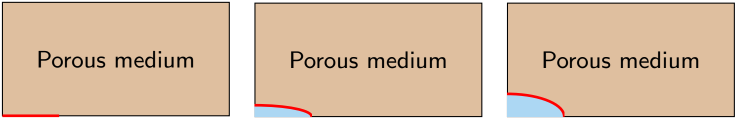
# Coupled flow and poromechanics



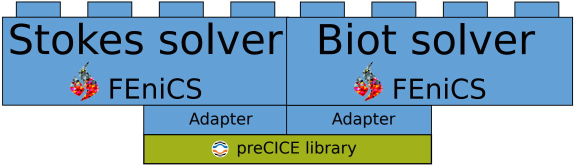
1. Poromechanics solver
2. Flow solver



# Coupled flow and poromechanics



1. Poromechanics solver
2. Flow solver



# Pressure over the fracture (Biot)

# Table of contents

Introduction to preCICE

Background information

Coupling approach

Features

Coupling schemes

Getting started

Get preCICE

Tutorials

Couple own code

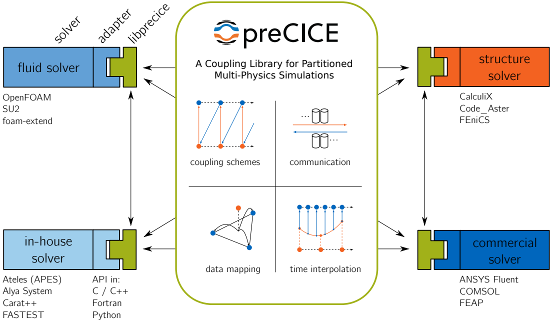
Application to fractured porous media

Summary and outlook

# preCICE

## Summary

- Black box coupling tool
- Coupling arbitrary number of solvers
- Suitable for strongly coupled problems
- Good scalability
- Minimally invasive



# Roadmap

## Current Developments v1.4 (just released)

- Debian package
- Building with cmake

## Current Developments Adapters

- Fluid-fluid module for the OpenFOAM adapter
- Update of Fluent adapter
- Official adapters for dealii, FEniCS, and Nutils

## Long-term Goals v2.0

- 3D-1D and 3D-2D data mapping
- Parallel initialization → support of very large cases
- Support of re-meshing and dynamically changing coupling interfaces
- Consistent time interpolation

alexander.jaust@ipvs.uni-stuttgart.de

- 🌐 www.precice.org
- github.com/precice
- 🐦 @preCICE_org
- 💬 Mailing-list, Gitter
- 🎓 Literature Guide on wiki

SPPEXA  DFG  EuroTech Universities  SFB 1313 Interface-Driven Multi-Field Processes in Porous Media

TUM  TU/e  Universität Stuttgart

# Thank you for your attention!

alexander.jaust@ipvs.uni-stuttgart.de

- www.precice.org
- github.com/precice
- @preCICE_org
- Mailing-list, Gitter
- Literature Guide on wiki

SPPEXA
DFG
EuroTech Universities
SFB 1313 Interface-Driven Multi-Field Processes in Porous Media
TUM
TU/e
Universität Stuttgart

# preCICE

preCICE = **Pre**cise **C**ode **I**nteraction **C**oupling **E**nvironment

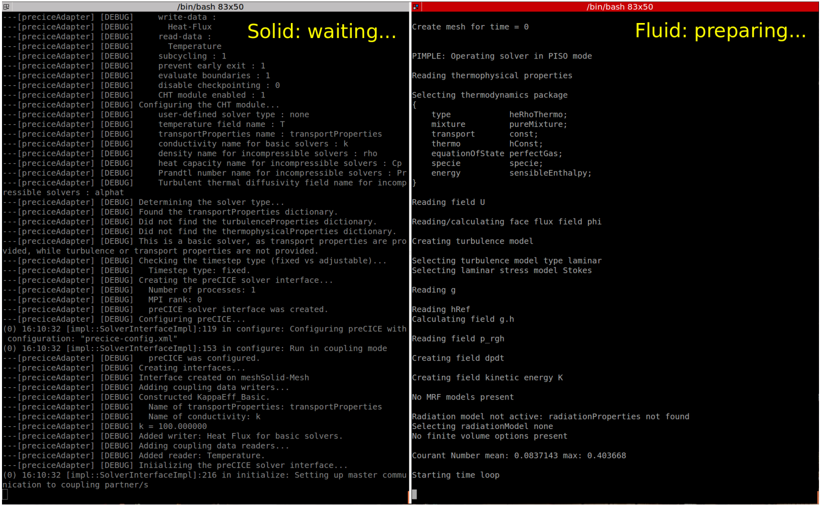
# Table of contents

Getting started

OpenFOAM adapter

Coupling own codes

# Flow over a heated plate



/bin/bash 83x50

Solid: waiting...

```
---[preciceAdapter] [DEBUG]     write-data :
---[preciceAdapter] [DEBUG]       Heat-Flux
---[preciceAdapter] [DEBUG]     read-data :
---[preciceAdapter] [DEBUG]       Temperature
---[preciceAdapter] [DEBUG]     subcycling : 1
---[preciceAdapter] [DEBUG]     prevent early exit : 1
---[preciceAdapter] [DEBUG]     evaluate boundaries : 1
---[preciceAdapter] [DEBUG]     disable checkpointing : 0
---[preciceAdapter] [DEBUG]     CHT module enabled : 1
---[preciceAdapter] [DEBUG] Configuring the CHT module...
---[preciceAdapter] [DEBUG]     user-defined solver type : none
---[preciceAdapter] [DEBUG]     temperature field name : T
---[preciceAdapter] [DEBUG]     transportProperties name : transportProperties
---[preciceAdapter] [DEBUG]     conductivity name for basic solvers : k
---[preciceAdapter] [DEBUG]     density name for incompressible solvers : rho
---[preciceAdapter] [DEBUG]     heat capacity name for incompressible solvers : Cp
---[preciceAdapter] [DEBUG]     Prandtl number name for incompressible solvers : Pr
---[preciceAdapter] [DEBUG]     Turbulent thermal diffusivity field name for incomp
ressible solvers : alphat
---[preciceAdapter] [DEBUG] Determining the solver type...
---[preciceAdapter] [DEBUG] Found the transportProperties dictionary.
---[preciceAdapter] [DEBUG] Did not find the turbulenceProperties dictionary.
---[preciceAdapter] [DEBUG] Did not find the thermophysicalProperties dictionary.
---[preciceAdapter] [DEBUG] This is a basic solver, as transport properties are pro
vided, while turbulence or transport properties are not provided.
---[preciceAdapter] [DEBUG] Checking the timestep type (fixed vs adjustable)...
---[preciceAdapter] [DEBUG]   Timestep type: fixed.
---[preciceAdapter] [DEBUG] Creating the preCICE solver interface...
---[preciceAdapter] [DEBUG]   Number of processes: 1
---[preciceAdapter] [DEBUG]   MPI rank: 0
---[preciceAdapter] [DEBUG]   preCICE solver interface was created.
---[preciceAdapter] [DEBUG] Configuring preCICE...
(0) 16:10:32 [impl::SolverInterfaceImpl]:119 in configure: Configuring preCICE with
 configuration: "precice-config.xml"
(0) 16:10:32 [impl::SolverInterfaceImpl]:153 in configure: Run in coupling mode
---[preciceAdapter] [DEBUG]   preCICE was configured.
---[preciceAdapter] [DEBUG] Creating interfaces...
---[preciceAdapter] [DEBUG] Interface created on meshSolid-Mesh
---[preciceAdapter] [DEBUG] Adding coupling data writers...
---[preciceAdapter] [DEBUG] Constructed KappaEff_Basic.
---[preciceAdapter] [DEBUG]   Name of transportProperties: transportProperties
---[preciceAdapter] [DEBUG]   Name of conductivity: k
---[preciceAdapter] [DEBUG] k = 100.000000
---[preciceAdapter] [DEBUG] Added writer: Heat Flux for basic solvers.
---[preciceAdapter] [DEBUG] Adding coupling data readers...
---[preciceAdapter] [DEBUG] Added reader: Temperature.
---[preciceAdapter] [DEBUG] Iniializing the preCICE solver interface...
(0) 16:10:32 [impl::SolverInterfaceImpl]:216 in initialize: Setting up master commu
nication to coupling partner/s
```

/bin/bash 83x50

Fluid: preparing...

```
Create mesh for time = 0


PIMPLE: Operating solver in PISO mode

Reading thermophysical properties

Selecting thermodynamics package
{
    type            heRhoThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleEnthalpy;
}

Reading field U

Reading/calculating face flux field phi

Creating turbulence model

Selecting turbulence model type laminar
Selecting laminar stress model Stokes

Reading g

Reading hRef
Calculating field g.h

Reading field p_rgh

Creating field dpdt

Creating field kinetic energy K

No MRF models present

Radiation model not active: radiationProperties not found
Selecting radiationModel none
No finite volume options present

Courant Number mean: 0.0837143 max: 0.403668

Starting time loop
```

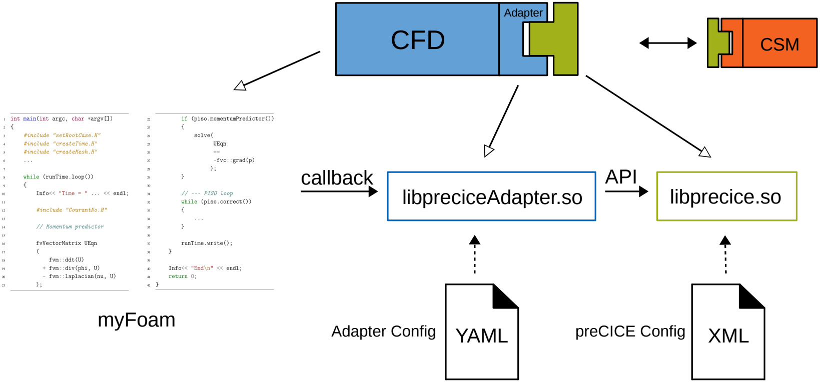
# Table of contents

Getting started

OpenFOAM adapter

Coupling own codes

# The (new) OpenFOAM adapter



```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    ...

    while (runTime.loop())
    {
        Info<< "Time = " ... << endl;

        #include "CourantNo.H"

        // Momentum predictor

        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );
        if (piso.momentumPredictor())
        {
            solve(
                  UEqn
                  ==
                  -fvc::grad(p)
                 );
        }

        // --- PISO loop
        while (piso.correct())
        {
            ...
        }

        runTime.write();
    }

    Info<< "End\n" << endl;
    return 0;
}
```

# Flow over a heated plate

Load adapter at runtime in `system/controlDict`:

```
functions
{
preCICE_Adapter
{
type preciceAdapterFunctionObject;
libs ("libpreciceAdapterFunctionObject.so");
}
}
```

Define coupling boundary in `system/blockMeshDict`:

```
interface
{
type wall;
faces
(
(4 0 1 5)
);
}
```

# Flow over a heated plate

Configure adapter in precice-adapter-config.yml:

```
participant: Fluid

precice-config-file: /path/to/precice-config.xml

interfaces:
- mesh: Fluid-Mesh
patches: [interface]
write-data: Temperature
read-data: Heat-Flux
```

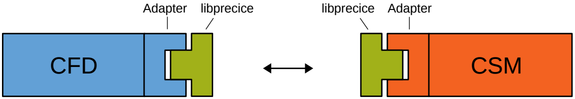
# Table of contents

Getting started

OpenFOAM adapter

Coupling own codes

# Coupling own code



1. Steering
2. Mesh and data access
3. Checkpointing (Implicit coupling, optional)

# 1. Steering

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh



double dt; // solver timestep size




while (not simulationDone()){
  dt = beginTimeStep(); // e.g. compute adaptive dt

  computeTimeStep(dt);

  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);


double dt; // solver timestep size




while (not simulationDone()){
  dt = beginTimeStep(); // e.g. compute adaptive dt

  computeTimeStep(dt);

  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size




while (not simulationDone()){
  dt = beginTimeStep(); // e.g. compute adaptive dt

  computeTimeStep(dt);

  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size
double maxDt; // maximum precice timestep size



while (not simulationDone()){
  dt = beginTimeStep(); // e.g. compute adaptive dt

  computeTimeStep(dt);

  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size
double maxDt; // maximum precice timestep size

maxDt = precice.initialize()

while (not simulationDone()){
  dt = beginTimeStep(); // e.g. compute adaptive dt

  computeTimeStep(dt);

  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size
double maxDt; // maximum precice timestep size

maxDt = precice.initialize()

while (not simulationDone() && precice.isCouplingOngoing()){
  dt = beginTimeStep(); // e.g. compute adaptive dt

  computeTimeStep(dt);

  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size
double maxDt; // maximum precice timestep size

maxDt = precice.initialize()

while (not simulationDone() && precice.isCouplingOngoing()){
  dt = beginTimeStep(); // e.g. compute adaptive dt
  dt = min(maxDt, dt);
  computeTimeStep(dt);

  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size
double maxDt; // maximum precice timestep size

maxDt = precice.initialize()

while (not simulationDone() && precice.isCouplingOngoing()){
  dt = beginTimeStep(); // e.g. compute adaptive dt
  dt = min(maxDt, dt);
  computeTimeStep(dt);
  maxDt = precice.advance(dt); // communication, data mapping, ...
  endTimeStep(); // e.g. update variables, increment time
}

turnOffSolver();
```

# API 1: Steering

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size
double maxDt; // maximum precice timestep size

maxDt = precice.initialize()

while (not simulationDone() && precice.isCouplingOngoing()){
  dt = beginTimeStep(); // e.g. compute adaptive dt
  dt = min(maxDt, dt);
  computeTimeStep(dt);
  maxDt = precice.advance(dt); // communication, data mapping, ...
  endTimeStep(); // e.g. update variables, increment time
}
precice.finalize();
turnOffSolver();
```

# 2. Mesh and data access

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");









precice.initialize()





[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");







precice.initialize()





[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface






precice.initialize()





[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize





precice.initialize()





[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)




precice.initialize()





[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)
// determine coordinates



precice.initialize()





[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)
// determine coordinates
int* vertexIDs = new int[vertexSize];


precice.initialize()





[...]
```

# API 2: Mesh and data access

```cpp
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)
// determine coordinates
int* vertexIDs = new int[vertexSize];
precice.setMeshVertices(meshID, vertexSize, coords, vertexIDs);

precice.initialize()






[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)
// determine coordinates
int* vertexIDs = new int[vertexSize];
precice.setMeshVertices(meshID, vertexSize, coords, vertexIDs);

precice.initialize()

[...]

int forceID = precice.getDataID("Forces", meshID);

[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)
// determine coordinates
int* vertexIDs = new int[vertexSize];
precice.setMeshVertices(meshID, vertexSize, coords, vertexIDs);

precice.initialize()

[...]

int forceID = precice.getDataID("Forces", meshID);
double* forces = new double[vertexSize*dim];
[...]
```

# API 2: Mesh and data access

```
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at coupling interface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)
// determine coordinates
int* vertexIDs = new int[vertexSize];
precice.setMeshVertices(meshID, vertexSize, coords, vertexIDs);

precice.initialize()

[...]

int forceID = precice.getDataID("Forces", meshID);
double* forces = new double[vertexSize*dim];
precice.writeBlockVectorData(forceID, vertexSize, vertexIDs, forces);
```

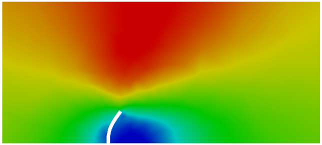
# preCICE configuration

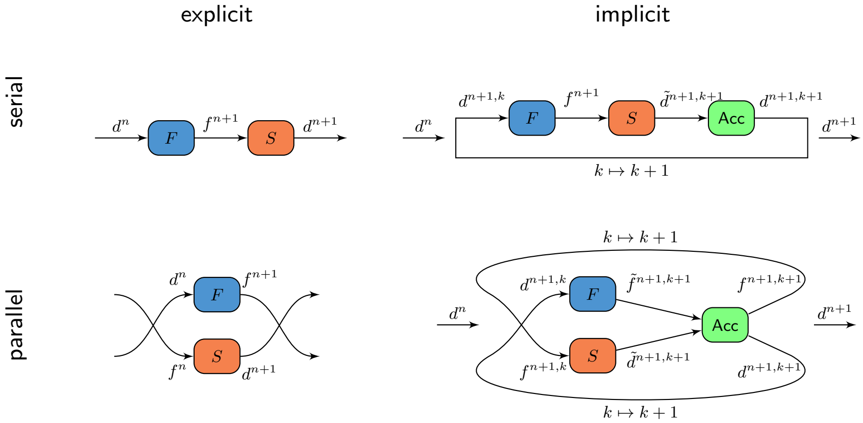
```xml
<precice-configuration>

  <data:vector name="Forces" />
  <data:vector name="Displacements" />

  <mesh name="FluidMesh">
    <use-data name="Forces" />
    <use-data name="Displacements" />
  </mesh>

  <participant name="FluidSolver">
    <use-mesh name="FluidMesh" provide="yes" />
    <write-data name="Forces" mesh="FluidMesh" />

    [...]

  </participant>

  [...]
```

# 3. Checkpointing (for implicit coupling)

# Different couplings

- Fluid solver: $F : d \mapsto f$
- Structure solver: $S : f \mapsto d$
- Solve fixed-point problem: $(S \circ F)(d) \overset{!}{=} d$





# API 3: Implicit coupling

```
while (not simulationDone() && precice.isCouplingOngoing()){





  dt = beginTimeStep();
  computeTimeStep(dt);
  precice.advance(dt); // communication, data mapping, ...






  endTimeStep(); // e.g. update variables, increment time

}
```

# API 3: Implicit coupling

```
while (not simulationDone() && precice.isCouplingOngoing()){
  if(precice.isActionRequired("WriteIterationCheckpoint")){
    saveCheckpoint(); // save internal state of solver
    precice.fulfilledAction("WriteIterationCheckpoint");
  }

  dt = beginTimeStep();
  computeTimeStep(dt);
  precice.advance(dt); // communication, data mapping, ...






  endTimeStep(); // e.g. update variables, increment time

}
```

# API 3: Implicit coupling

```
while (not simulationDone() && precice.isCouplingOngoing()){
  if(precice.isActionRequired("WriteIterationCheckpoint")){
    saveCheckpoint(); // save internal state of solver
    precice.fulfilledAction("WriteIterationCheckpoint");
  }

  dt = beginTimeStep();
  computeTimeStep(dt);
  precice.advance(dt); // communication, data mapping, ...

  if(precice.isActionRequired("ReadIterationCheckpoint")){
    reloadCheckpoint(); // set variables back to checkpoint
    precice.fulfilledAction("ReadIterationCheckpoint");
  }
  else{ // timestep converged
    endTimeStep(); // e.g. update variables, increment time
  }
}
```