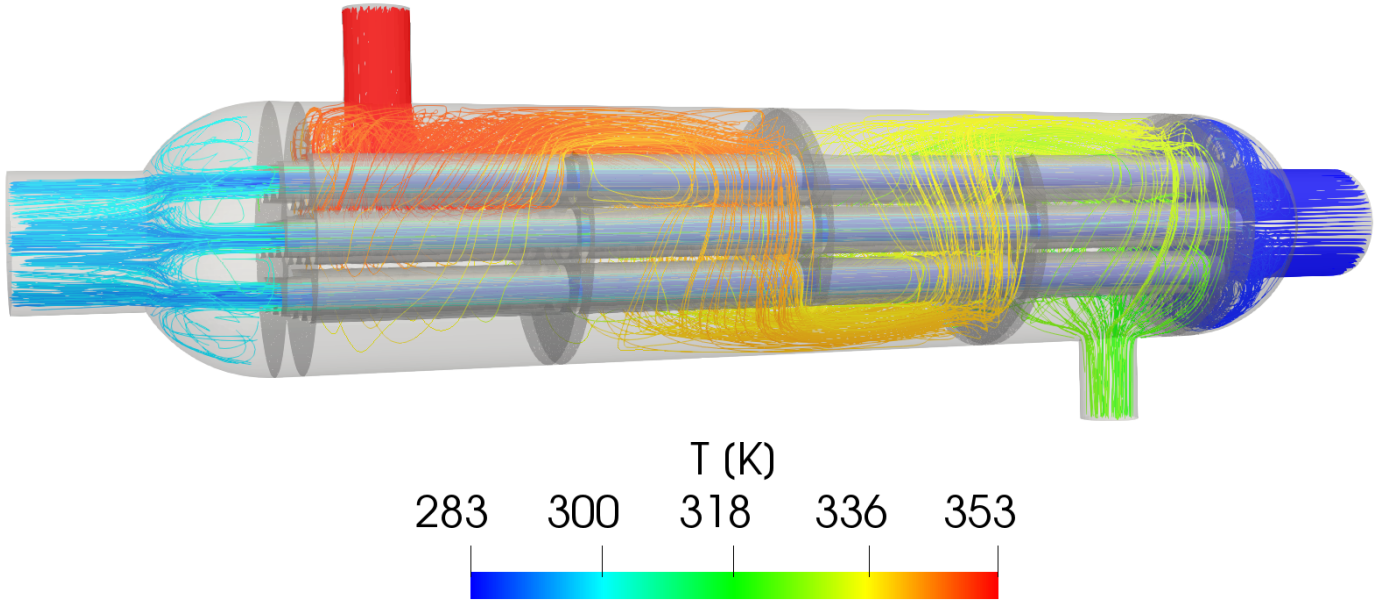
A Coupling Library for Partitioned Multi-Physics Simulations

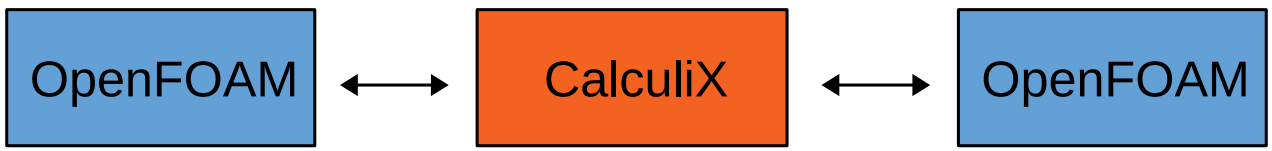
1. What is preCICE?
2. How to get started?
3. How can I couple my own code?

1. What is preCICE?

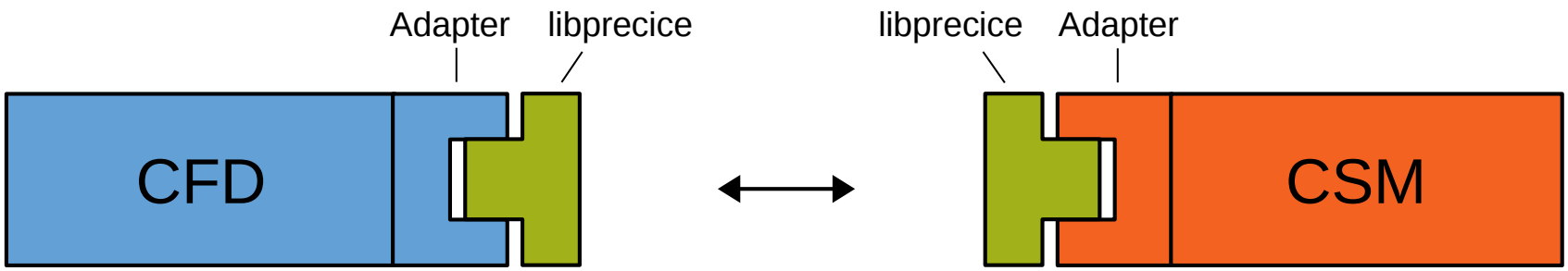
Example: Shell-And-Tube Heat Exchanger



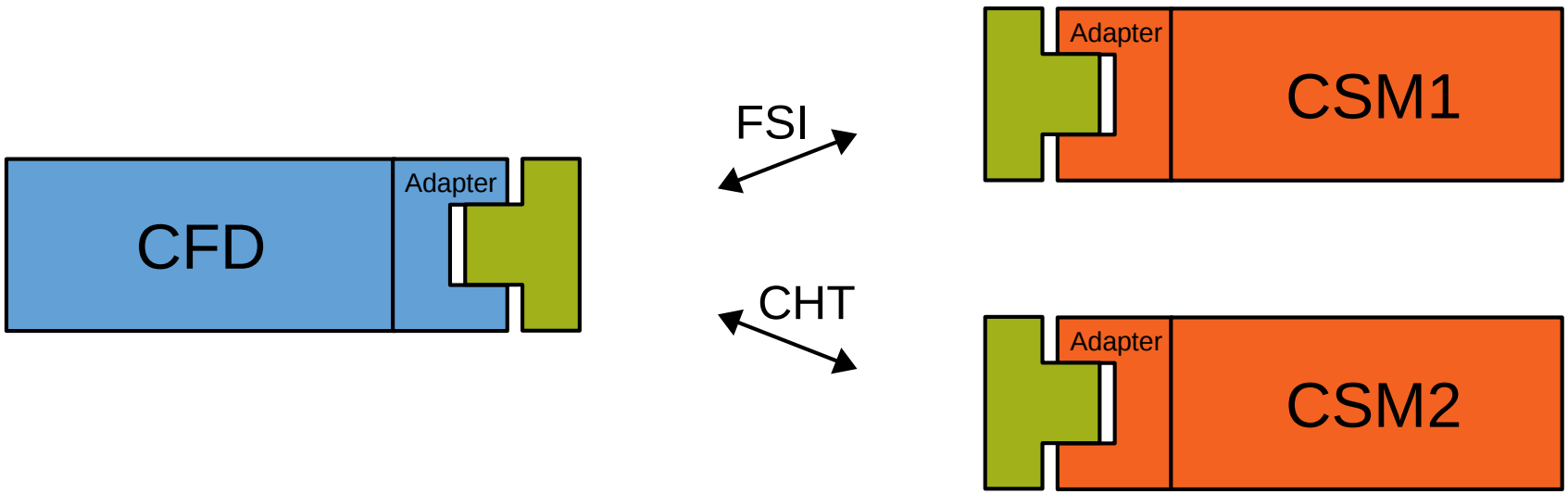
- Partitioned coupling: Usage of three independent solvers
- Reuse of existing solvers



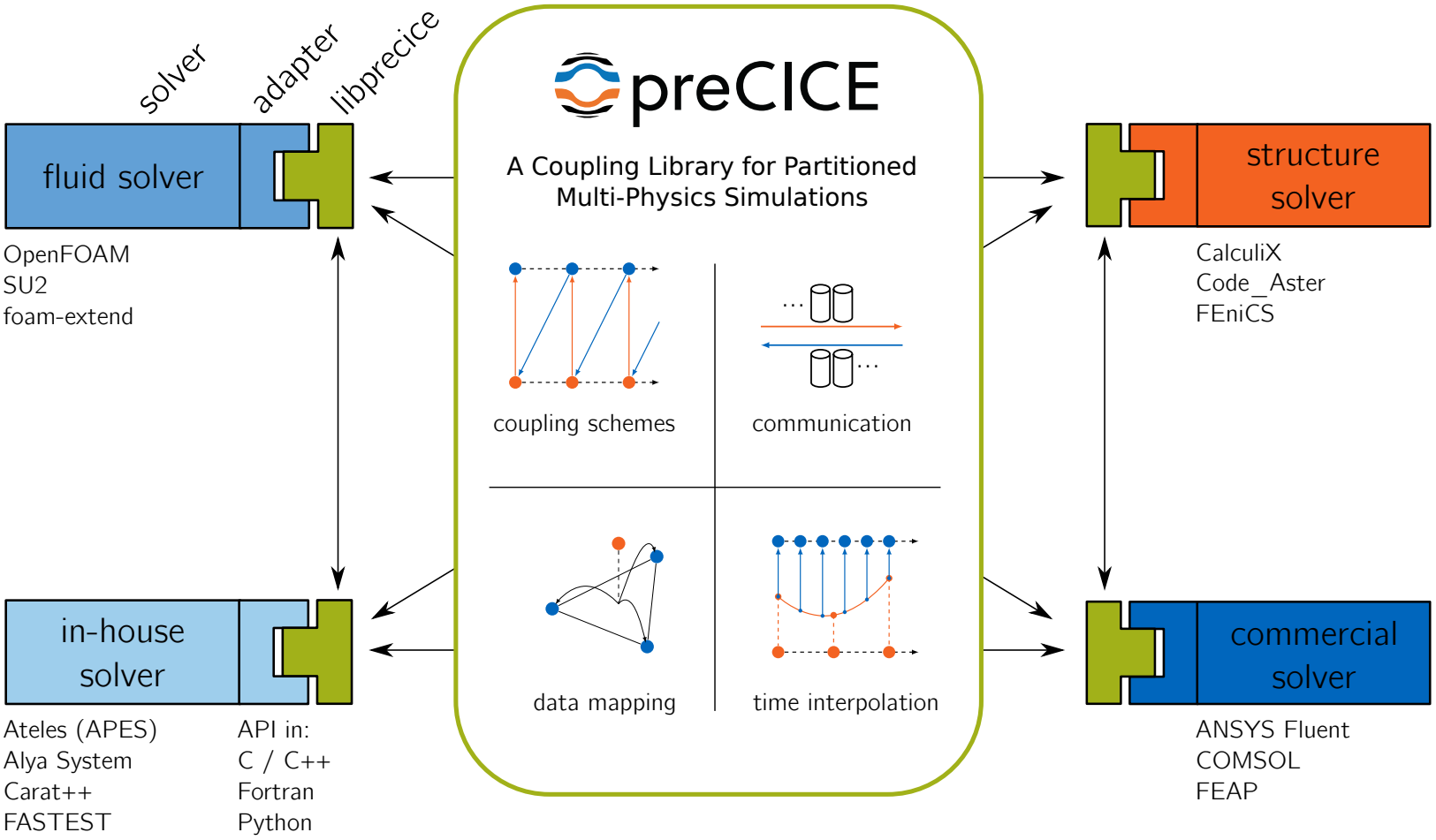
preCICE – A Plug-and-Play Coupling Library



preCICE – A Plug-and-Play Coupling Library



preCICE – A Plug-and-Play Coupling Library

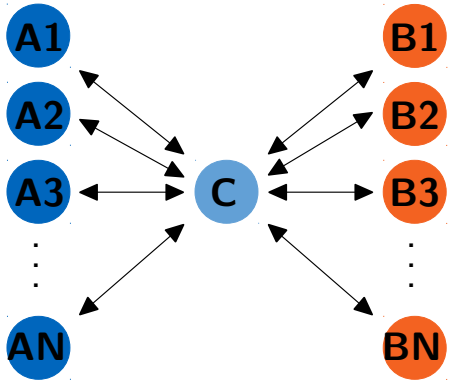


USPs

1. Scalability
2. Robust quasi-Newton coupling
3. Coupling of arbitrary many components
   (*arbitrary many = more than two*)
4. Minimally-invasive coupling
5. Open-source, community

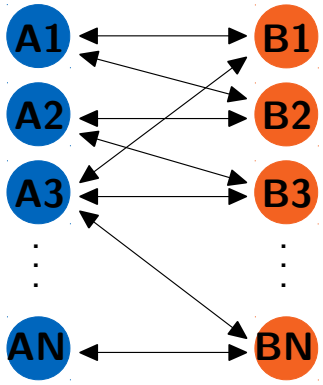
USP 1: Scalability

Server-Based Concept



- Complete communication through central server process
- Interface computations on server (in sequential)
- $\Rightarrow$ Coupling becomes bottleneck for overall simulation already on moderate parallel systems

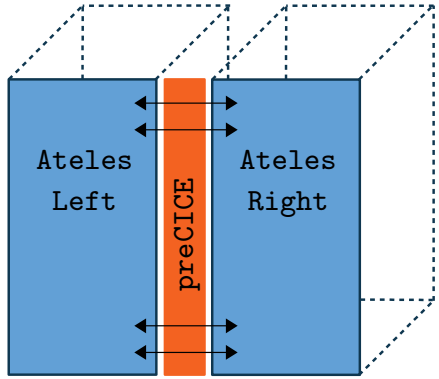
Our Peer-To-Peer Concept

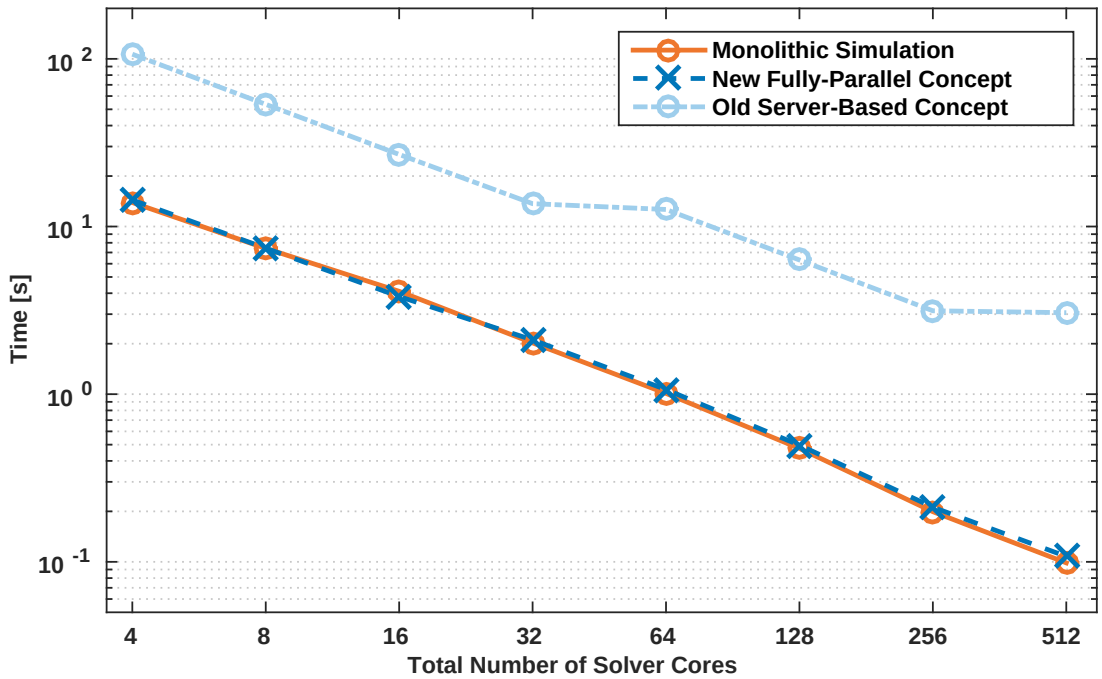


- No central entity
- $\Rightarrow$ Easier to handle (user does not need to care about server)
- $\Rightarrow$ No scaling issues

USP 1: 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

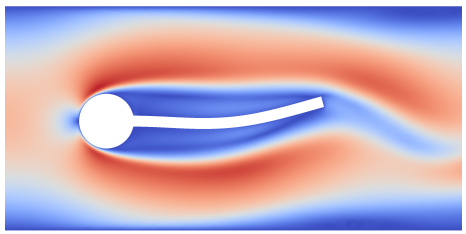




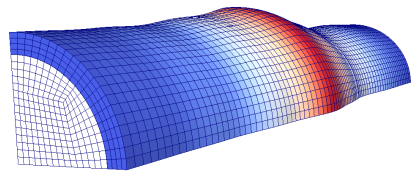
USP 2: Quasi-Newton Coupling

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

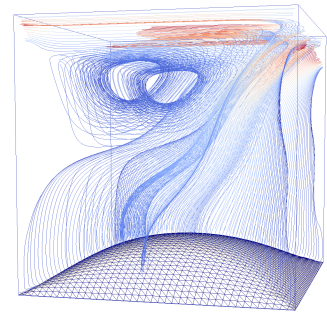
FSI3



3D-Tube



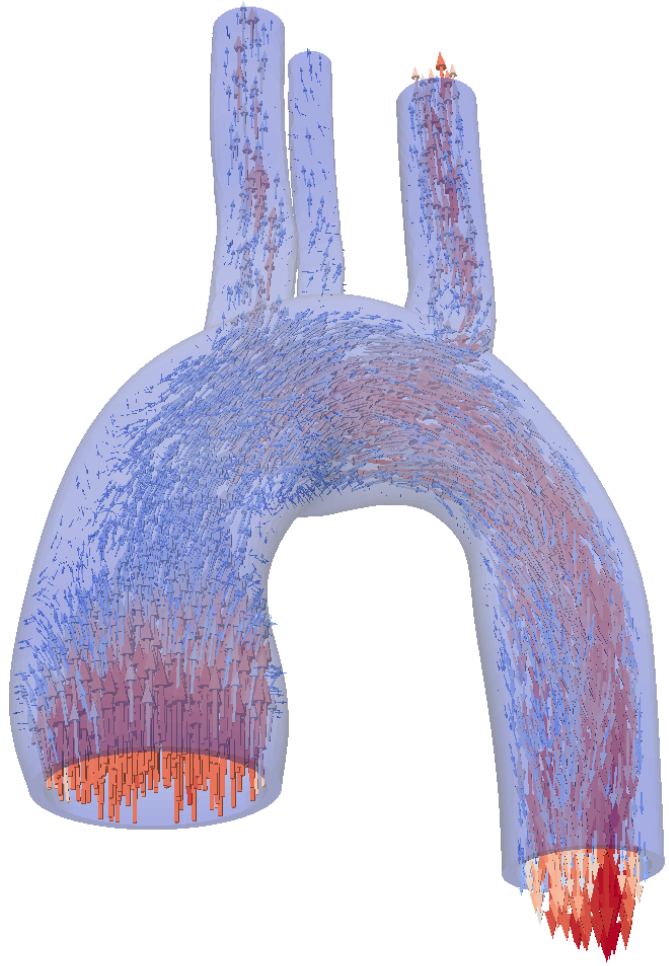
Driven Cavity



Mean Iterations	Aitken	Quasi-Newton
FSI3	17.0	3.3
3D-Tube	Div.	7.5
Driven Cavity	7.4	2.0

USP 2: Quasi-Newton Coupling

- Quasi-Newton can even handle biomedical applications, such as an Aortic bloodflow
- Stable coupling (no added-mass instabilities)
- Six times less iterations than Aitken

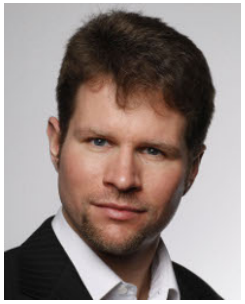


- Joint work with Juan-Carlos Cajas (Barcelona Supercomputing Center)
- Geometry by Jordi Martorell

Contributors

Miriam Mehl

U Stuttgart



Florian Lindner

U Stuttgart

Amin Totounferoush

U Stuttgart

Alexander Rusch

ETH Zürich

Hans Bungartz

TUM

Benjamin Rüth

TUM

Gerasimos Chourdakis

TUM

Frédéric Simonis

TUM

Benjamin Uekermann

TUM

Previous and minor contributors:

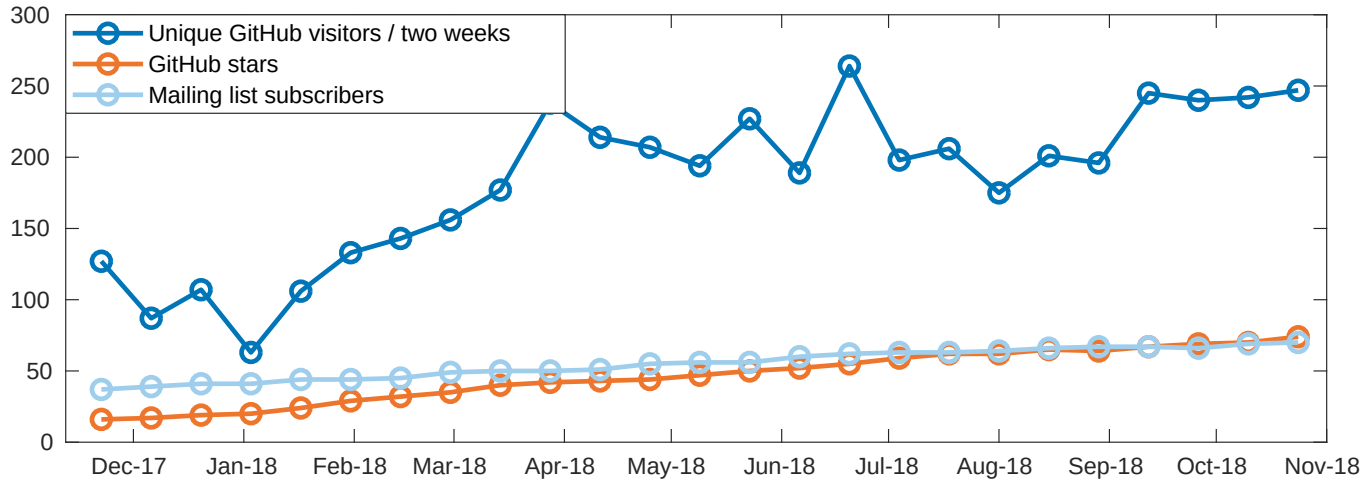
- Bernhard Gatzhammer, Klaudius Scheufele, Lucia Cheung, Alexander Shukaev, Peter Vollmer, Georg Abrams, Alex Trujillo, Dmytro Sashko, David Sommer, David Schneider, Richard Hertrich, Saumitra Joshi, Peter Meisrimel, Derek Risseeuw, Jiho Yang, ...

Users

- LSM & STS, U Siegen, Germany
- SC & FNB, TU Darmstadt, Germany
- SCpA, CIRA, Italy
- Cardiothoracic Surgery, University of the Free State, South Africa
- A*STAR, Singapore
- CASA, TU Eindhoven, The Netherlands
- Nuclear Research and Consultancy Group, Petten, The Netherlands
- Aerospace Engineering, TU Delft
- Mechanical and Aeronautical Eng., University of Manchester, UK
- University of Strathclyde, Glasgow, UK

Upcoming:

- Global Research for Safety (GRS), Garching, Germany
- MTU Aero Engines, Munich, Germany
- TU Braunschweig, Institute of Aircraft Design and Lightweight Structures, Germany
- FAST, Karlsruhe Institute of Technology, Germany
- Numerical Analysis, Lund, Sweden
- Austrian Institute of Technology, Vienna, Austria
- IAG, University of Stuttgart, Germany
- Polytechnique Montreal, Canada



2. How to get started?

Infrastructure

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

- LGPL3 license
- User documentation in the wiki

Building

Dependencies

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

The easy way

- Ubuntu 18.04: All dependencies available through distribution
- Ubuntu 16.04: All dependencies available except Boost

Still doable

- macOS
- Other Linux distributions

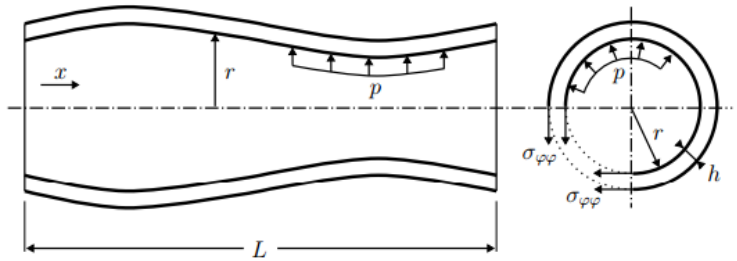
Experimental

- Conda, Docker, Debian package, Spack
- Windows

Tutorials

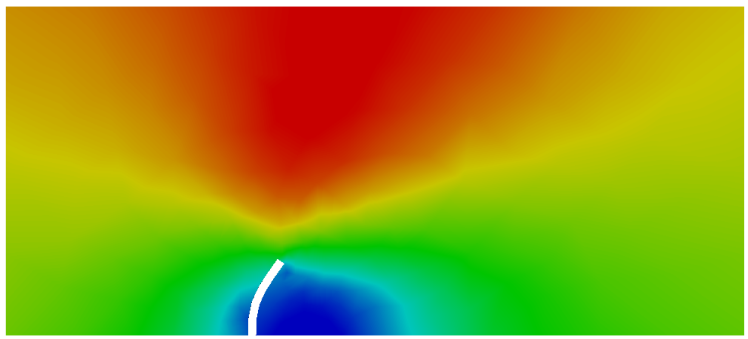
1D Elastic Tube

- Simple provided solvers
- Learn about API and configuration



Flexible beam

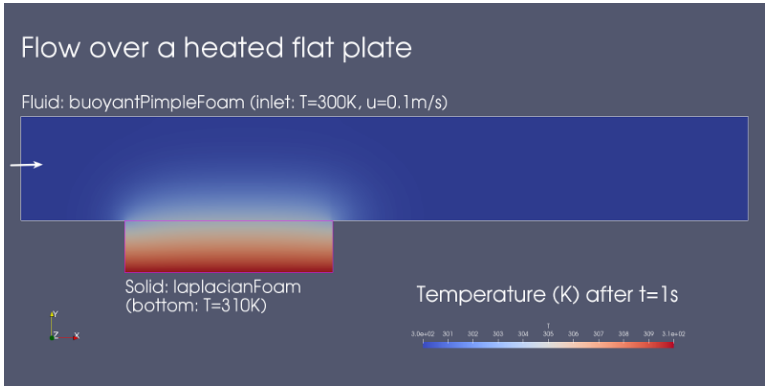
- Fluid-structure interaction
- Couple SU2 to CalculiX
- Learn about coupling schemes
- Also interactive version available in browser `http://run.coplon.de/`



Tutorials

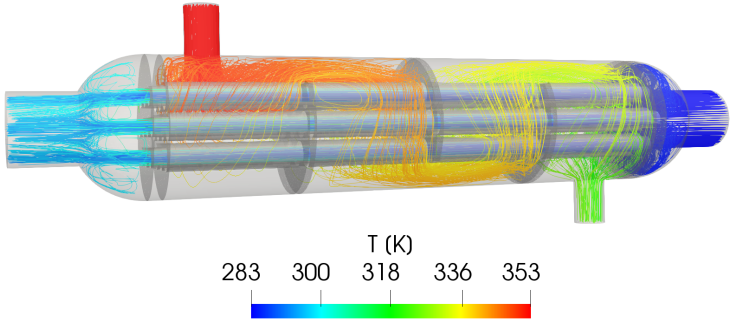
Flow over a Heated Plate

- Conjuagte-heat transfer
- Couple two OpenFOAM solvers
- Learn about OpenFOAM adapter



Heat exchanger

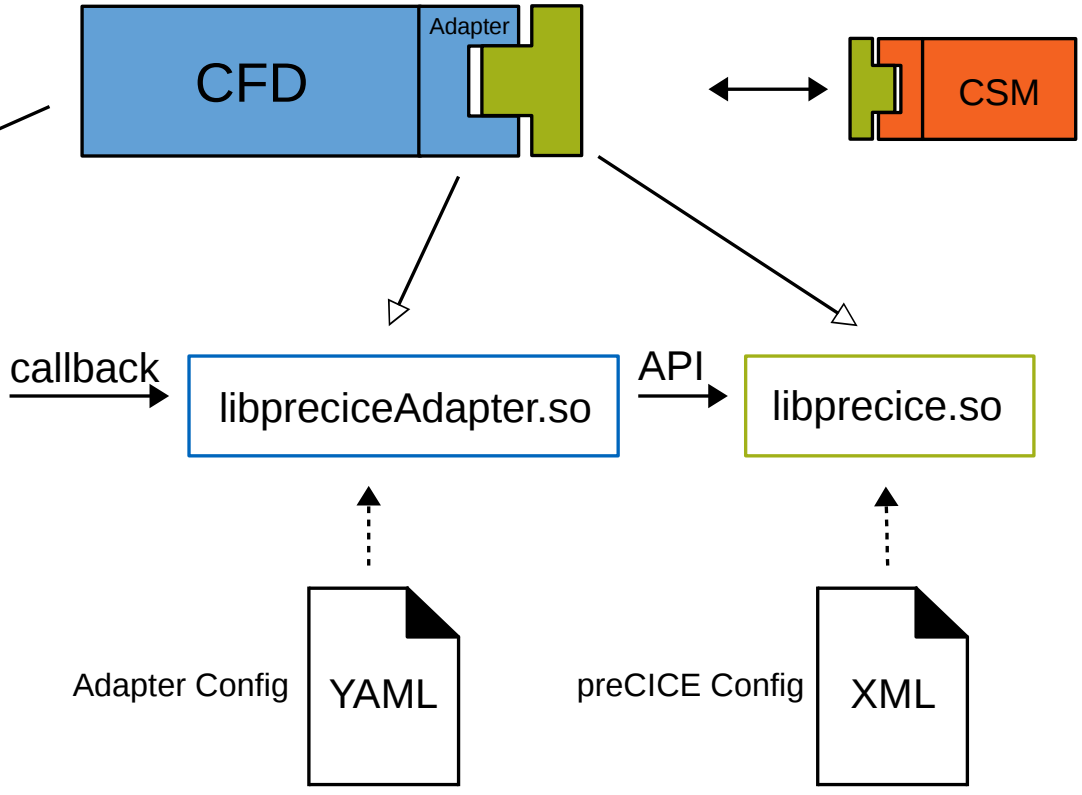
- Conjugate-heat transfer
- Couple two OpenFOAM instances with CalculiX
- Learn about multi coupling



The 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;
}
```

myFoam



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`:

```yaml
participant: Fluid

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

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

Flow over a Heated Plate

3. How can I couple my own code?

API 1: Steering

```
1  turnOnSolver(); //e.g. setup and partition mesh
2
3
4
5  double dt; // solver timestep size
6
7
8
9
10 while (not simulationDone()){
11   dt = beginTimeStep(); // e.g. compute adaptive dt
12
13   computeTimeStep(dt);
14
15   endTimeStep(); // e.g. update variables, increment time
16 }
17
18 turnOffSolver();
```

API 1: Steering

```
1  turnOnSolver(); //e.g. setup and partition mesh
2  precice::SolverInterface precice("FluidSolver",rank,size);
3
4
5  double dt; // solver timestep size
6
7
8
9
10 while (not simulationDone()){
11   dt = beginTimeStep(); // e.g. compute adaptive dt
12
13   computeTimeStep(dt);
14
15   endTimeStep(); // e.g. update variables, increment time
16 }
17
18 turnOffSolver();
```

API 1: Steering

```
1  turnOnSolver(); //e.g. setup and partition mesh
2  precice::SolverInterface precice("FluidSolver",rank,size);
3  precice.configure("precice-config.xml");
4
5  double dt; // solver timestep size
6
7
8
9
10 while (not simulationDone()){
11   dt = beginTimeStep(); // e.g. compute adaptive dt
12
13   computeTimeStep(dt);
14
15   endTimeStep(); // e.g. update variables, increment time
16 }
17
18 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();
```

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

```
1  precice::SolverInterface precice("FluidSolver",rank,size);
2  precice.configure("precice-config.xml");
3
4  int meshID = precice.getMeshID("FluidMesh");
5
6
7
8
9
10
11
12 precice.initialize()
13
14
15
16
17
18 [...]
```

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

```
1  precice::SolverInterface precice("FluidSolver",rank,size);
2  precice.configure("precice-config.xml");
3
4  int meshID = precice.getMeshID("FluidMesh");
5  int vertexSize; // number of vertices at coupling interface
6  // determine vertexSize
7
8
9
10
11
12 precice.initialize()
13
14
15
16
17
18 [...]
```

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

```
1  precice::SolverInterface precice("FluidSolver",rank,size);
2  precice.configure("precice-config.xml");
3
4  int meshID = precice.getMeshID("FluidMesh");
5  int vertexSize; // number of vertices at coupling interface
6  // determine vertexSize
7  double* coords = new double[vertexSize*dim]; // coupling mesh (nodes)
8  // determine coordinates
9
10
11
12 precice.initialize()
13
14
15
16
17
18 [...]
```

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

```
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

```
<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>

  [...]
```

API 3: Implicit Coupling

```
1  while (not simulationDone() && precice.isCouplingOngoing()){
2
3
4
5
6
7    dt = beginTimeStep();
8    computeTimeStep(dt);
9    precice.advance(dt); // communication, data mapping, ...
10
11
12
13
14
15
16   endTimeStep(); // e.g. update variables, increment time
17
18 }
```

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
  }
}
```

Roadmap

Current Developments

- Debian package
- FSI module for the OpenFOAM adapter
- Fluid-fluid module for the OpenFOAM adapter
- Structure-structure coupling with CalculiX

Long-term Goals

- 3D-1D and 3D-2D data mapping
- Parallel initialization and support of dynamically changing coupling interfaces
- Consistent time interpolation

Funding

More information

- Open-source project: `www.precice.org`
- Source code: `github.com/precice`

Contact us

- Mailing list
- Gitter chat room