

Couple scientific simulation codes with preCICE

A journey towards sustainable research software

Gerasimos Chourdakis

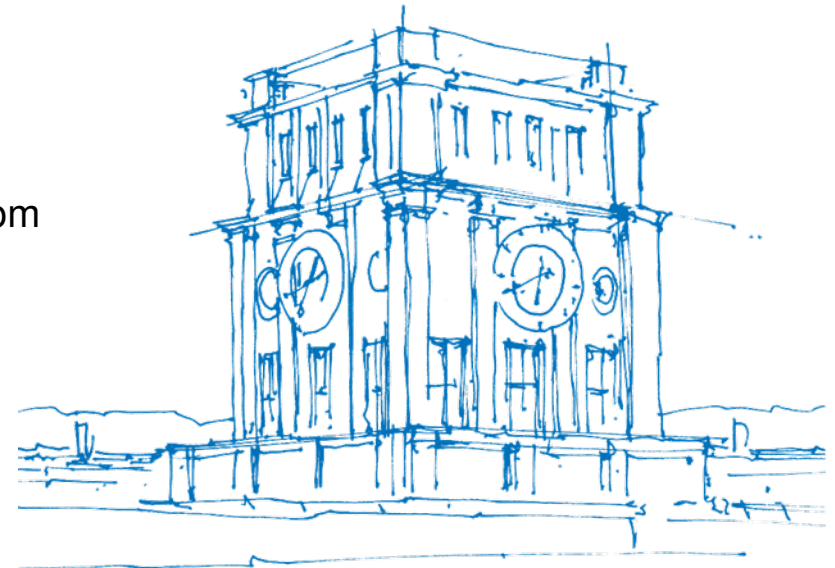
Technical University of Munich

Department of Informatics

Chair of Scientific Computing in Computer Science

February 3, 2019

FOSDEM'19: HPC, Big Data and Data Science devroom



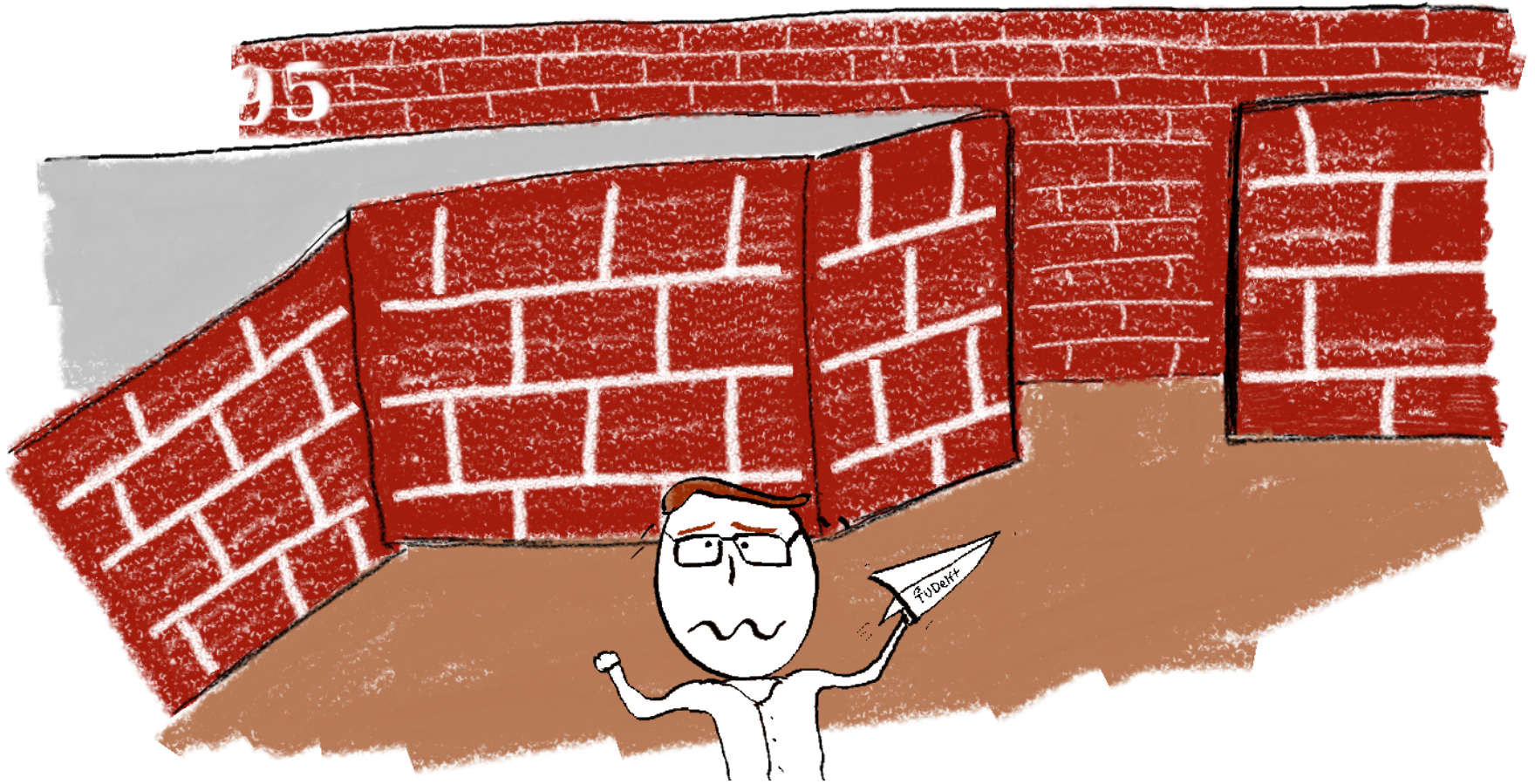
TUM Uhrenturm

This is Derek, an aerospace engineer



Totally accurate depiction of @derekriseeuw.

He is trapped in a lab(yrinth)...



... and he wants to escape by flying.

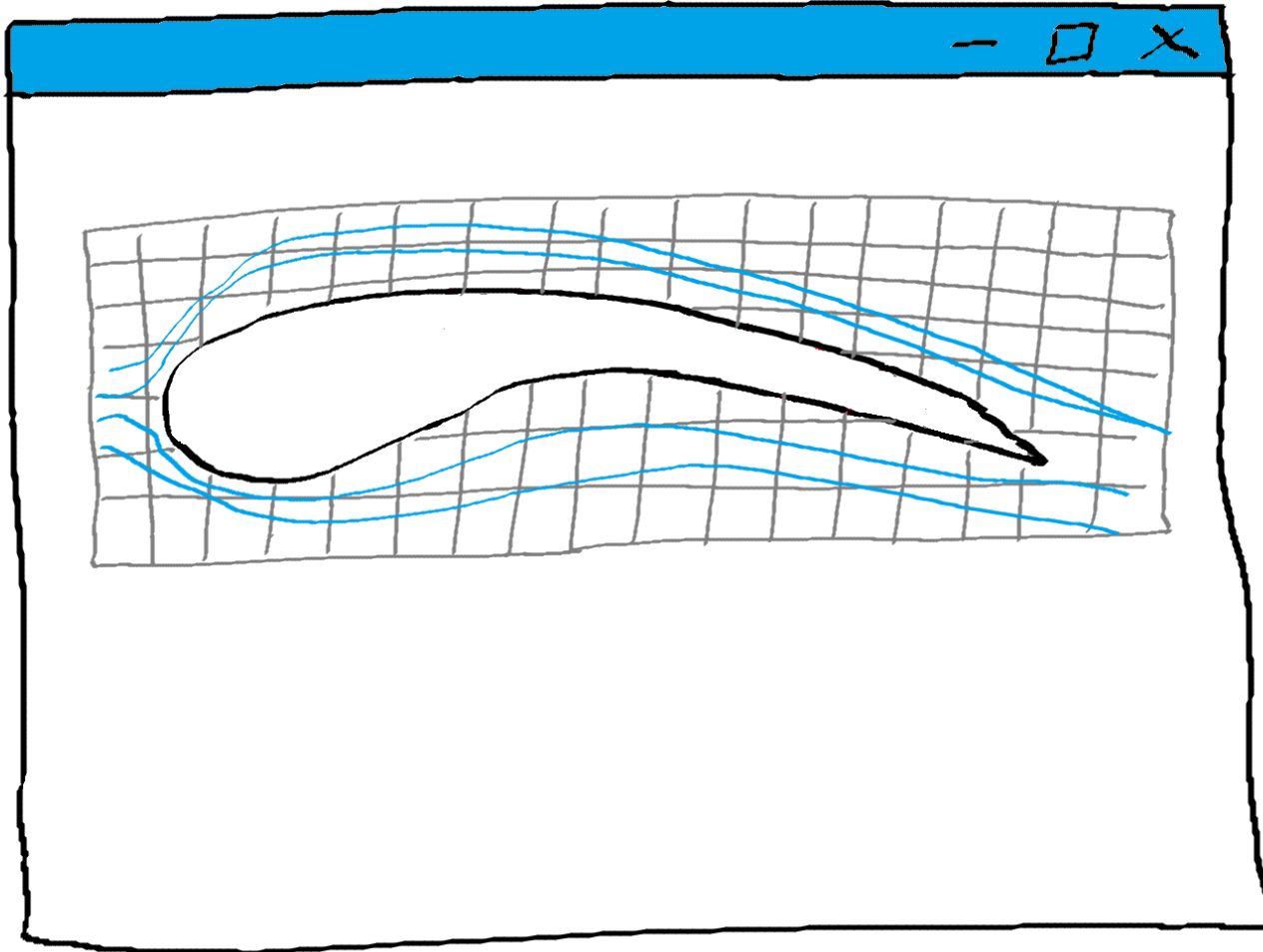


However, Derek lives in 2019:

He wants to simulate his wings before trying!

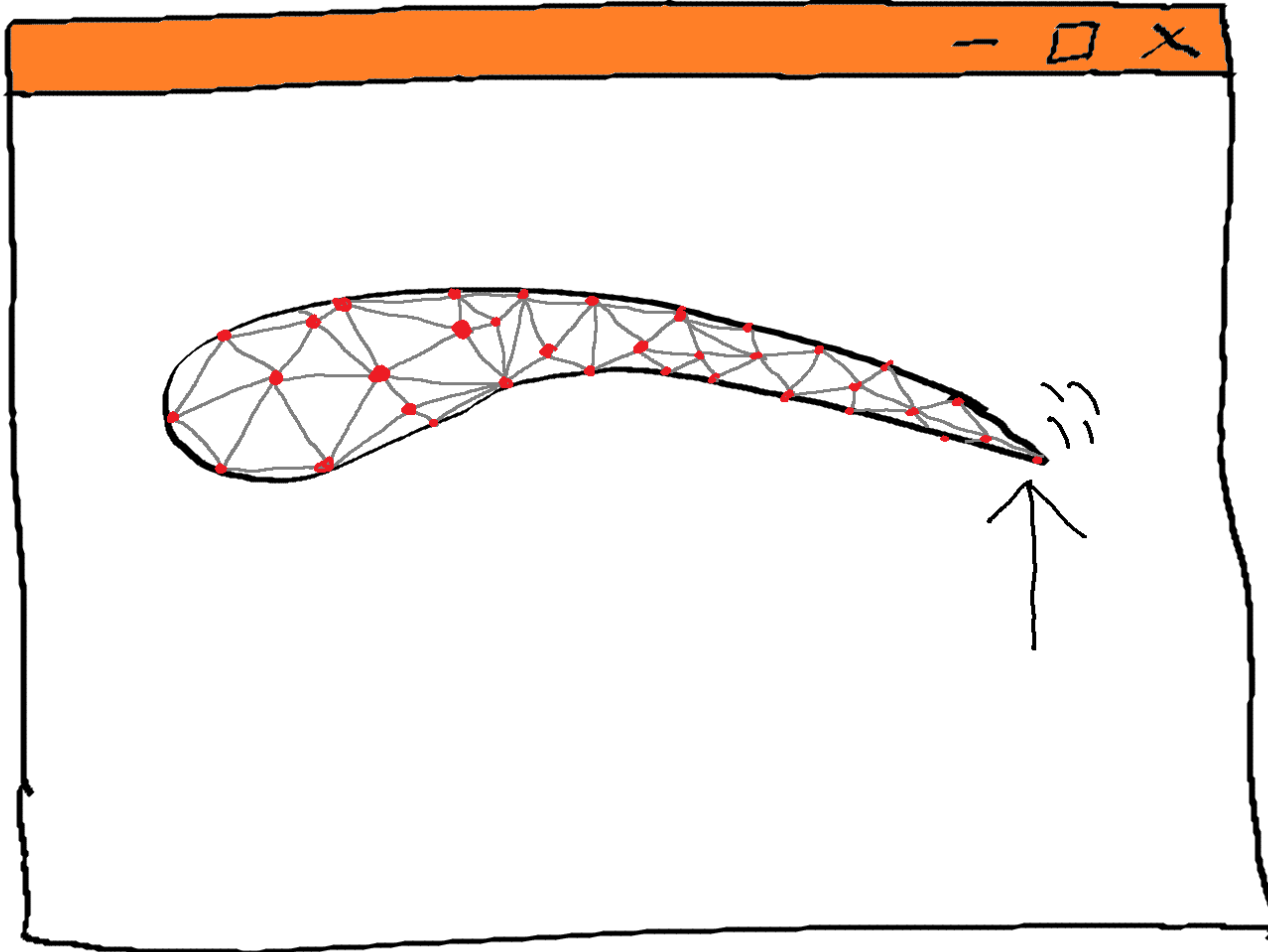
So he simulates the flow...

(he already uses a tool to do this, e.g. OpenFOAM)



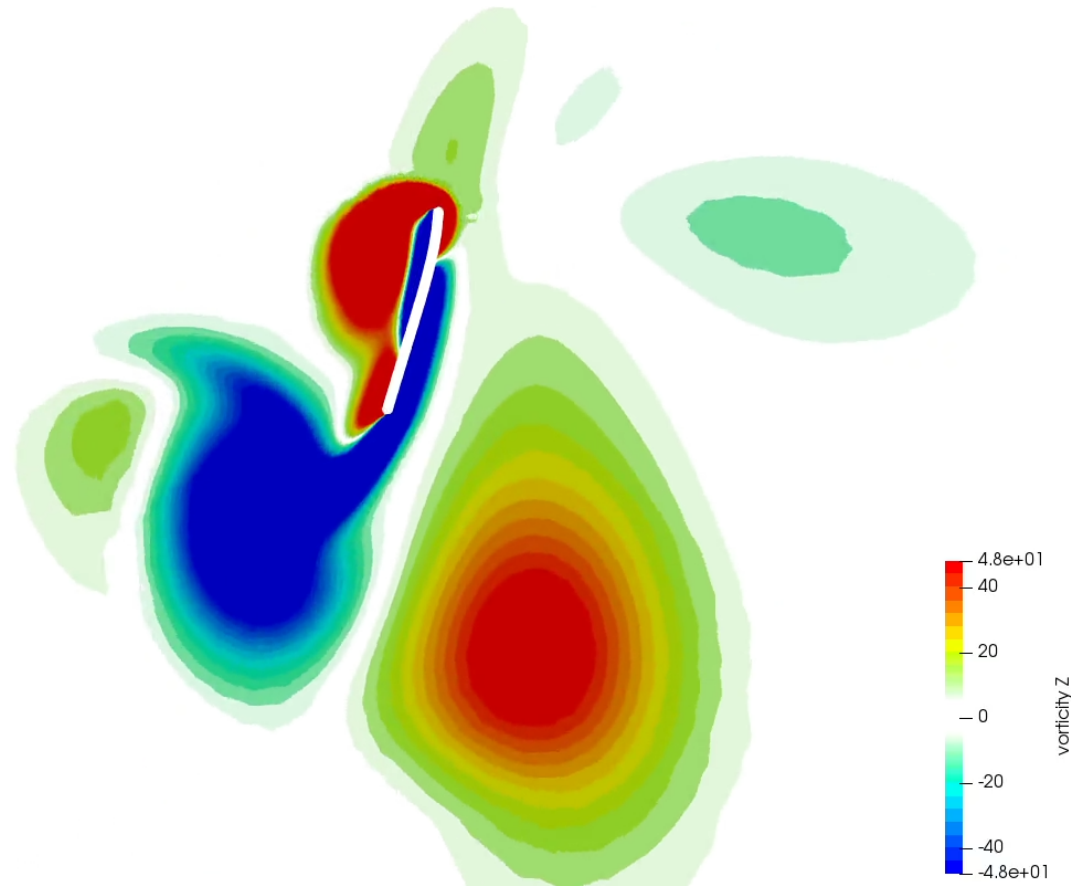
...and he simulates the structure

(he already uses a tool to do this, e.g. CalculiX)



But this is a coupled problem!

Aim (video) [1]:

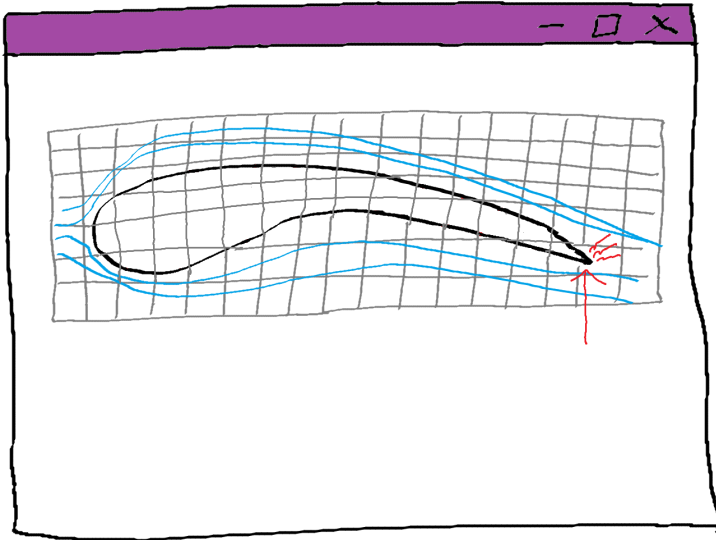


[1] Derek Risseuw. Fluid Structure Interaction Modelling of Flapping Wings. Master's thesis, Faculty of Aerospace Engineering, Delft University of Technology, 2019.

What are his options?

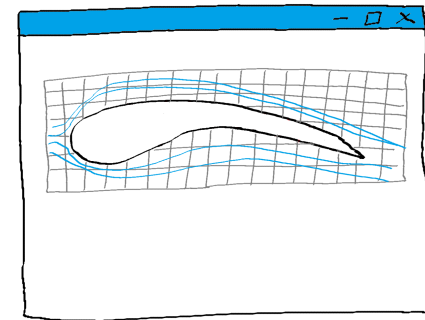
Monolithic approach

one software package for everything
(Emacs, is that you?)



Partitioned approach

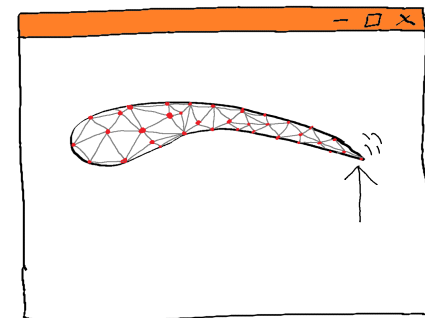
one specialist for each problem + coupling tool



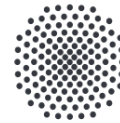
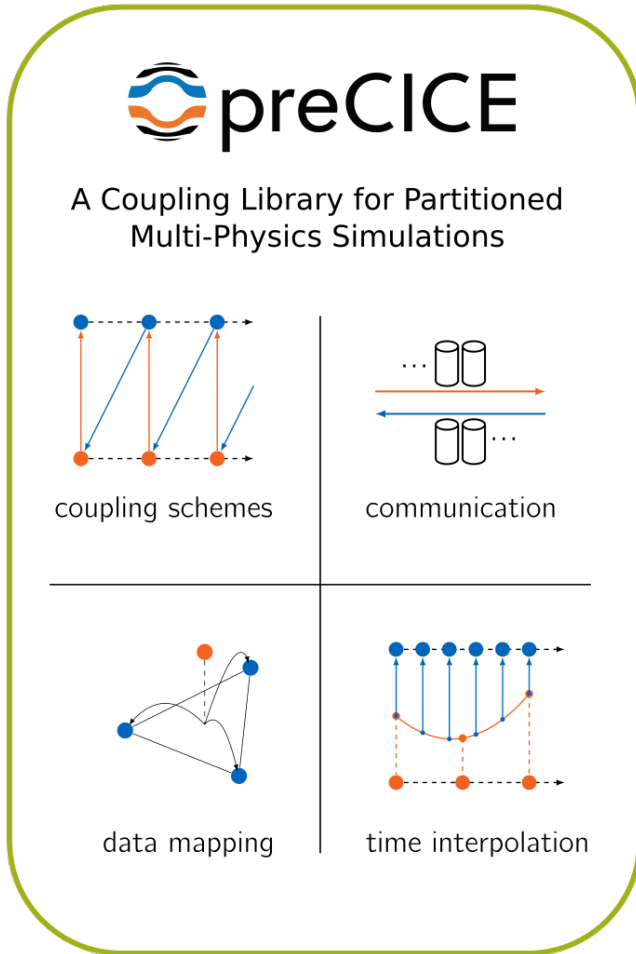
+



+



The preCICE coupling library



Universität Stuttgart

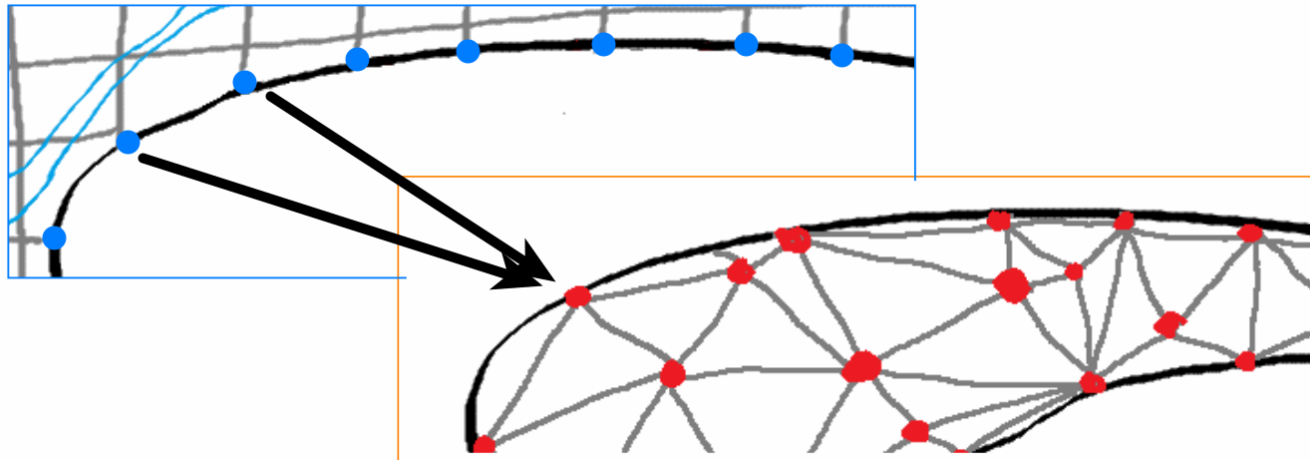


Deutsche
Forschungsgemeinschaft



Website: precice.org - **Source:** github.com/precice

Feature example: Data mapping



Available methods:

- Nearest-Neighbor
- Nearest-Projection
- Radial-Basis Function interpolation

Coupling buzzwords

implicit / explicit coupling

Fluid-Structure-Acoustics Interaction
Conjugate Heat Transfer
Fluid-Fluid Coupling

Aitken under-relaxation

Interface Quasi-Newton
acceleration

serial & parallel coupling

asynchronous, P2P communication

high-level API
C++, C, Fortran, Python

MPI Ports & TCP/IP sockets

library (not framework)

Multiple participants

Nearest-Projection mapping

subcycling

Radial-Basis Function
mapping

time interpolation*

Example of an adapted solver code

```
1 precice::SolverInterface precice("FluidSolver",rank,size);
2 precice.configure("precice-config.xml");
3 precice.setMeshVertices();
4 precice.initialize();
5
6 while (precice.isCouplingOngoing()) { // main time loop
7     solve();
8
9     precice.writeBlockVectorData();
10    precice.advance();
11    precice.readBlockVectorData();
12
13    endTimeStep(); // e.g. write results, increase time
14 }
15
16 precice.finalize();
```

Timesteps, most arguments and less important methods omitted.
Full example in the preCICE wiki.

easily enable your code to be coupled with:

*OpenFOAM, CalculiX, SU2, FEniCS, deal-ii, Code_Aster,
foam-extend, Ateles, FASTEST, FEAP, MBDyn,
ANSYS Fluent, COMSOL,*

...

(also for HPC!)

Part II

A journey towards sustainable research software

We are not Computer Scientists. What are we?

Mechanical engineers, mathematicians (, ...) who develop software for research
(and want to make it good and usable by others)



Research Software Engineers International

Research Software Engineers

This is the website of the international research software engineering community.

Research Software Engineers are people who combine professional software expertise with an understanding of research. They go by various job titles but the term Research Software Engineer (RSE) is fast gaining international recognition.

National RSE Associations

UK RSE - UK Research Software Engineers Association 

DE-RSE - Society for Research Software in Germany 

NL-RSE - The community of Research Software Engineers in the Netherlands 

NORDIC-RSE - Nordic Research Software Engineers Community 

researchsoftware.org

preDOM: Domesticating preCICE



dfg.de

1. Infrastructure & interoperability

- Building & Packaging
- Communication & collaboration
- ...

2. Sustainability

- Dependencies
- Testing
- Documentation
- ...

3. Usability

- Better error messages
- Live tracking of the simulation
- ...

4. Outreach

- Tutorials
- Website
- Conferences!?
- ...

preDOM: Domesticating preCICE



dfg.de

1. Infrastructure & interoperability

- **Building & Packaging**
- Communication & collaboration
- ...

2. Sustainability

- Dependencies
- **Testing**
- Documentation
- ...

3. Usability

- Better error messages
- Live tracking of the simulation
- ...

4. Outreach

- Tutorials
- Website
- Conferences!?
- ...

Building: From SCons to CMake

Currently (build and test):

```
1 $ scons -j 4 compiler=mpicxx mpi=yes petsc=yes build=Debug
2 $ export PRECICE_ROOT="/path/to/precice"
3 $ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PRECICE_ROOT/build/last/
4 $ ./tools/compileAndTest.py -t
```

Why are we using SCons?

- Easy and flexible! (Python)
- preCICE happened to start with SCons, 10+ years ago

Note: Building in a user directory and using environment variables is (unfortunately) quite common.

Building: From SCons to CMake

From next release on (expected on February 15):

```
1 $ CXX=mpicxx cmake -DMPI=ON -DPETSC=ON -DCMAKE_BUILD_TYPE=Debug ..
2 $ make -j 4
3 $ make install
4 $ mpirun -np 4 ./testprecice
```

How? Learn CMake, create a (quite long) CMakeLists.txt, ...

See [precice/precice PR #240](#), contributed by Frédéric Simonis, TUM & Florian Lindner, Univ. of Stuttgart. First implementation already since longer, by Florian Lindner.

Building: From SCons to CMake

Advantages:

- Closer to the expected behavior (xSDK requirement)
- More configurable dependency discovery
- Many build systems and IDEs out-of-the-box.
- User-friendlier output in Make (progress bar!)
- Clear configuration - building - testing - installing steps
- Easy configuration with `ccmake` or CMake GUI
- Debugging-related logs (`CMakeCache.txt`, `CMakeOutput.log`, `CMakeError.log`)
- Easier to create packages

xSDK policies: xsdk.info/policies

preCICE is not yet xSDK-compatible, but we are working on it.



Packaging: Debian packages with CPack

Build it locally:

```
1 $ cmake ..
2 $ make
3 $ make package
```

And give it to the user!

```
1 $ sudo apt install ./libprecice1.4.0-Linux.deb
```

How? Learn CPack, create a (not so long) CPackConfig.cmake, ...

Don't forget to validate: `lintian libprecice1.4.0-Linux.deb`

See [precice/precice PR #240](#), contributed by Frédéric Simonis, TUM.

Packaging: preCICE with Spack

Get preCICE on supercomputers and select the dependencies:

```
1 $ spack install precice ^openmpi@1.10
2 $ spack load precice
```

How? Create a (quite short) `package.py` (and submit it to the Spack repository).

Like the concept of Spack? Check out also EasyBuild!

Also tried: Conda, Docker containers



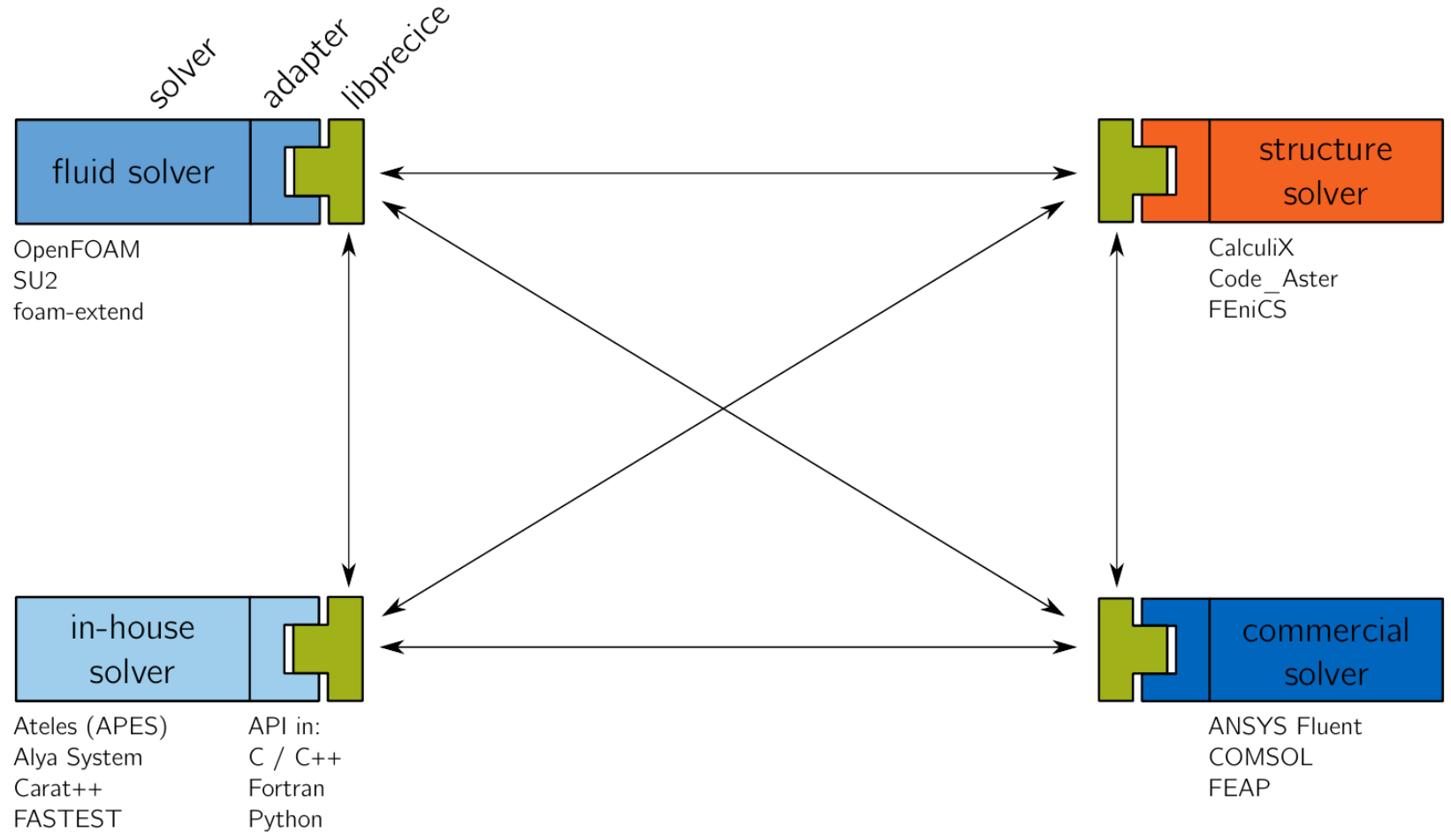
Spack

<https://spack.io>

Spack packages contributed by Mark Olesen, OpenCFD.

Testing: our situation

Unit tests only in preCICE itself, system tests with our tutorial simulations (nightly, Travis)



How to do performance tests? How to do unit tests inside the adapters?

Talking about tutorials...

☰

HIDE ^

In the xml file below, we have already changed the number of maximum timesteps

```
<max-timesteps value="50" />
```

in the coupling scheme section. Intuitively, this should allow us to see the flap in our model oscillate.

```

<participant name="Calculix">
  <use-mesh name="Calculix_Mesh" provide="yes"/>
  <write-data name="DisplacementDeltas0" mesh="Calculix_Mesh"/>
  <read-data name="Forces0" mesh="Calculix_Mesh"/>
  <watch-point
    mesh="Calculix_Mesh" name="point1" coordinate="-0.05;0.01;1">
  </participant>

<m2n:sockets
  from="SU2_CFD" to="Calculix"
  distribution-type="gather-scatter"/>

<coupling-scheme:serial-explicit>
  <participants first="SU2_CFD" second="Calculix"/>
  <max-timesteps value="50"/>
  <timestep-length value="1e-2" />
  <exchange
    data="Forces0" mesh="Calculix_Mesh"
    from="SU2_CFD" to="Calculix"/>
  <exchange
    data="DisplacementDeltas0" mesh="Calculix_Mesh"
    from="Calculix" to="SU2_CFD" />
</coupling-scheme:serial-explicit>

</solver-interface>

</precece-configuration>

```

BACK

NEXT

Coupling Scheme

Before we go ahead and run the simulation, we need to configure one last thing: the coupling scheme. As you can imagine, the coupling scheme can affect whether the simulation converges or not.

- **coupling-scheme:** Here we use the serial-explicit coupling scheme. Serial refers to the fact that the two solvers operate serially with respect to each other - one waits for the other to finish its timestep. Explicit means that we let every solver compute once and then move on to the next timestep. ?
- **participants:** This defines the coupled participants. The attribute "first" names the participant that leads the coupling.

Web-based preCICE tutorial, developed by Dmytro Sashko and other TUM BGCE students
run.precice.org

Workflow: Code quality checks with Travis



precice-bot commented 15 days ago

Thank you for your contribution.

Some suggestions for your pull request

- It seems, like you forgot to update `CHANGELOG.md`
- Your code formatting did not follow our clang-format style in following files:
 - `src/io/Constants.cpp`
 - `src/io/TXTWriter.cpp`

🎉 1






Bot implemented with Travis, contributed by Dmytro Sashko, TUM. Not merged yet.



What: a library that can surface-couple simulations

RSEs: adapt your code easily to couple it with any preCICE-enabled solver: *OpenFOAM, CalculiX, SU2, FEniCS, deal-ii, Code_Aster, Fluent, ...* (also for HPC!)

Software engineers: give us your feedback!

 www.precice.org
 github.com/precice
 @preCICE_org, @_makCh
 www5.in.tum.de
 chourdak@in.tum.de



Doughnuts contributed by Gerasimos Chourdakis, TUM.