preCICE

User's Guide

August 2, 2012

Contents

1 Introduction **1**

2 Preparations **2**

2.1 Build Prerequesits under Linux 2

2.2 Build Prerequesits under Windows 2

2.3 Building preCICE 3

3 Running a Simulation with preCICE **3**

1 Introduction

preCICE stands for **pre**cise **c**ode **i**nteraction **c**oupling **e**nvironment. It has been written to facilitate partitioned coupled simulations, where partitioned is understood as a subdivisioning into separate systems of equations, surface coupled spatial discretization grids, and simulation softwares according to physical subproblems. preCICE has been written to deal with surface coupled problems, however, it is possible to perform volume coupling, too.

Another main task of preCICE is to support simulation programs using hexahedral grids. Surface representations of geometries read into preCICE can be queried to obtain positional (inside, outside, on geometry) and related information for points and hexahedrons (or rectangles in 2D). This functionality can be combined with the coupling functionality such that the surface representation is identical to the coupling surface.

This user's guide describes how to use preCICE to perform partitioned simulations, or as geometry interface.

Back to Contents.

2 Preparations

In order to use preCICE, first, a library has to be created from its source code. The solvers need to use the application programming interface (API) of preCICE in order to access coupling functionality, and the solver executables (or their interfaces for user programming) need to be linked to the preCICE library. This section describes how to accomplish this.

Back to Contents.

2.1 Build Prerequesits under Linux

- Obtain a current version of preCICE (http://www5.in.tum.de/wiki/index.php/PreCICE_Download).
- The Boost C++ libraries (http://www.boost.org/) need to be available (no build or installation is necessary). To choose a suitable version of Boost, please check out the preCICE release notes. To make the location of Boost available to preCICE, set the environment variable `BOOST_ROOT` to name the path to the Boost libraries root directory.
- In order to build preCICE, Python (version smaller than 3.0, http://www.python.org/) and SCons (http://www.scons.org/) need to be installed. If the Python extensions are intended to be used, NumPy (http://numpy.scipy.org/) needs to be installed in addition.
- If communication via MPI is required, a suitable implementation of the MPI2.0 standard is required to be installed (http://www.mcs.anl.gov/research/projects/mpich2/, e.g.).
- The GNU g++/gcc compiler (http://www.mingw.org/) needs to be available.

Back to Contents.

2.2 Build Prerequesits under Windows

- Obtain a current version of preCICE (http://www5.in.tum.de/wiki/index.php/PreCICE_Download).
- The Boost C++ libraries (http://www.boost.org/) need to be available, and the system and thread libraries need to be built when using socket communication. To choose a suitable version of Boost, please check out the preCICE release notes. To make the location of Boost available to preCICE, set the environment variable `BOOST_ROOT` to name the path to the Boost libraries root directory.
- In order to build preCICE, Python (version smaller than 3.0, http://www.python.org/) and SCons (http://www.scons.org/) need to be installed. If the Python extensions are intended to be used, NumPy (http://numpy.scipy.org/) needs to be installed in addition.
- If communication via MPI is required, a suitable implementation of the MPI2.0 standard is required to be installed (http://www.mcs.anl.gov/research/projects/mpich2/, e.g.).

- The MinGW g++/gcc compiler (http://www.mingw.org/) needs to be available.

Back to Contents.

2.3 Building preCICE

Switch to the root directory of preCICE, containing the `SConstruct` files. On a Linux system type

```
scons
```

On a Windows system type

```
scons -f SConstruct-windows
```

This will start building a debug version of preCICE, and may take several minutes. To speed up the build, add the option `-jn`, where `n` is the number of cores involved in the build and has to be larger than 1 in order to see speedup.

The following options can be enabled additionally, the first value is selected as default, when the option is not mentioned.

- `build=debug/release` For a production scenario, release mode should be used.
- `mpi=on/off` MPI is used as communication between solvers.
- `python=on/off` Python is used to enable configurable Python actions scripts.
- `sockets=on/off` Sockets are used as communication between solvers and recommended between solver and server.
- `spirit2=on/off` Boost.Spirit is used for writing checkpoints and reading geometries from VRML 1.0 files.

Back to Contents.

3 Running a Simulation with preCICE

When running a solver in parallel, currently a preCICE server executable has to be run in addition. The server has all coupling data and communicates with the coupled solver (or its server, when also running in parallel). The parallel solver processes have to communicate with the server in order to read/write data from/to the coupling interface. To configure this use-case, the XML-tag `<server/>` has to be used in the configuration of the parallel participant (see XML reference). There, a communication method between solver processes and server has to be specified. The server executable is created in the build directory of preCICE (also containing the library), and is named `binprecice`. The possible console options for `binprecice` are obtained by running it without parameters. To run it as server type

```
./binprecice server SolverName ConfigName
```

The `SolverName` is the name of the configured participant, which is used to identify the solver in preCICE. The `ConfigName` is the preCICE XML configuration file used for the simulation. Server and solver processes can be started in any order. The server executable can currently only be run by one process.