SeisSol Documentation

The SeisSol Team

May 02, 2024

INTRODUCTION

SeisSol is a software package for simulating wave propagation and dynamic rupture based on the arbitrary high-order accurate derivative discontinuous Galerkin method (ADER-DG).

Characteristics of the SeisSol simulation software are:

- use of arbitrarily high approximation order in time and space
- use of tetrahedral meshes to approximate complex 3D model geometries (faults & topography) and rapid model generation
- use of (an)isotropic elastic, viscoelastic and viscoplastic material to approximate realistic geological subsurface properties
- parallel geo-information input (ASAGI)
- to produce reliable and sufficiently accurate synthetic seismograms or other seismological data sets

---

We gratefully acknowledge the funding of the German Research Foundation (as part of project no. 391134334 - "Co-CoReCS"), which massively contributed to creating all documentation, tutorials, example workflows and reproducible setups published on this website.

CHAPTER

ONE

HISTORY

The software package SeisSol (http://www.seissol.org/) allows for realistic simulations of the three-dimensional seismic wavefield propagating in complex Earth structures generated by a finite dynamic earthquake source governed by a constitutive law that describes the relationship between fault stress and slip across a geometrically complex fault. SeisSol is a high-order accurate *Discontinuous Galerkin Finite Element* solver, based on the ADER-DG method presented in Kaser and Dumbser (2006), enabling precise modeling of on-fault frictional failure coupled to seismic waves traveling over large distances in terms of propagated wavelengths with minimal dispersion errors, whereas it is intrinsically dissipative and removes frequencies unresolved by the mesh without affecting longer and physically meaningful wavelengths. de la Puente et al. (2009) and Pelties et al. (2012) introduced Riemann solution to handle discontinuous fault slip conditions and achieve earthquake dynamics with seismic wave propagation.

The software has recently proven to be highly scalable on current and future HPC infrastructure. It reached multi-petaflop/s performance on some of the largest supercomputers worldwide in a pioneering simulation of the 1992 M7.2 Landers earthquake (Heinecke et al., 2014; Breuer et al., 2016). High detail rupture evolution and synthetic ground shaking in the engineering frequency band (0-10 Hz) were modeled on a non-planar earthquake fault structure. In early 2017, SeisSol performed the longest and largest dynamic rupture scenario to date, enabled by local time stepping (Uphoff et al.,2017), resolving the 2004 Sumatra-Andaman earthquake including complex splay fault geometries. The paper won the prestigious "Best Paper Award" of the International Supercomputing Conference (SC17). SeisSol results imply that acknowledging geometrical complexity, realistic fault properties, and velocity models affect not only earthquake source dynamics but the synthetic ground shaking crucially. The software package is available to the community as an open-source distribution (https://github.com/SeisSol/SeisSol).

CHAPTER TWO

INSTALLING DEPENDENCIES

In order to run SeisSol, you need to first install:

- Python (>= 3.5)
- Numpy (>= 1.12.0)
- hdf5 (>= 1.8, for instructions see below)
- netcdf (C-Release) (>= 4.4, for instructions see below)
- Intel compiler (>= 2021, icc, icpc, ifort) or GCC (>= 9.0, gcc, g++, gfortran)
- Some MPI implementation (e.g. OpenMPI)
- libxsmm (libxsmm_gemm_generator) for small matrix multiplications
- PSpaMM (pspamm.py) for small sparse matrix multiplications (required only on Knights Landing or Skylake)
- CMake (>= 3.20) for the compilation of SeisSol

For run-time partitioning you need to choose one of the following libraries:

- ParMETIS (with IDXTYPEWIDTH=64)
- SCOTCH
- ParHIP

The partitioning of SeisSol meshes with ParMETIS was tested in large simulations and is generally recommended for academic users. SCOTCH and ParHIP are free and open-source alternatives to ParMETIS and should be used by users from industry or for-profit institutions (cf. ParMETIS license). A study comparing partition quality for SeisSol meshes can be found here.

In addition, the following packages need to be installed for the GPU version of SeisSol:

- CUDA (>= 11.0) for Nvidia GPUs, or HIP (ROCm>= 5.2.0) for AMD GPUs
- SYCL: either hipSYCL >= 0.9.3 or DPC++
- gemmforge (>= 0.0.207)
- chainforge (>= 0.0.3, for Nvidia and AMD GPUs)

These dependencies can be installed automatically with spack or can be installed manually one by one.

2.1 Spack installation

Spack is a HPC software package manager. It automates the process of installing, upgrading, configuring, and removing computer programs. In particular, our spack package *seissol-env* allows automatically installing all dependencies of SeisSol (e.g. mpi, hdf5, netcdf, easi, asagi, etc). See https://github.com/SeisSol/seissol-spack-aid/tree/main/spack for details on the installation with spack. See also for reference our documentation on how to compile seissol-env on *SuperMUC-NG*, *Shaheen* (Cray system) and *Frontera*.

2.2 Manual installation

2.2.1 Initial Adjustments to .bashrc

Add the following lines to your .bashrc (vi ~/.bashrc).

```
# For intel compiler
# source /opt/intel/compiler/VERSION/bin/compilervars.sh intel64

export PATH=$HOME/bin:$PATH
export LIBRARY_PATH=$HOME/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=$HOME/lib:$LD_LIBRARY_PATH
export PKG_CONFIG_PATH=$HOME/lib/pkgconfig:$PKG_CONFIG_PATH
export CMAKE_PREFIX_PATH=$HOME
export EDITOR=vi
export CPATH=$HOME/include:$CPATH

# run "exec bash" or "source ~/.bashrc" to apply environment to the current shell
```

2.2.2 Installing CMake

```
# you will need at least version 3.20.0 for GNU Compiler Collection
(cd $(mktemp -d) && wget -qO- https://github.com/Kitware/CMake/releases/download/v3.20.0/
↪cmake-3.20.0-Linux-x86_64.tar.gz | tar -xvz -C "." && mv "./cmake-3.20.0-linux-x86_64"
↪"${HOME}/bin/cmake")

# use version 3.16.2 for Intel Compiler Collection
(cd $(mktemp -d) && wget -qO- https://github.com/Kitware/CMake/releases/download/v3.16.2/
↪cmake-3.16.2-Linux-x86_64.tar.gz | tar -xvz -C "." && mv "./cmake-3.16.2-Linux-x86_64"
↪"${HOME}/bin/cmake")

ln -s ${HOME}/bin/cmake/bin/cmake ${HOME}/bin
```

Note that this extracts CMake to the directory ${HOME}/bin/cmake, if you wish you can adjust that path.

2.2.3 Installing HDF5

```
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.8/src/hdf5-1.10.
↪8.tar.bz2
tar -xaf hdf5-1.10.8.tar.bz2
cd hdf5-1.10.8
CPPFLAGS="-fPIC ${CPPFLAGS}" CC=mpicc FC=mpif90 ./configure --enable-parallel --prefix=
↪$HOME --with-zlib --disable-shared --enable-fortran
make -j8
make install
cd ..
```

2.2.4 Installing netCDF

```
wget https://syncandshare.lrz.de/dl/fiJNAokgbe2vNU66Ru17DAjT/netcdf-4.6.1.tar.gz
tar -xaf netcdf-4.6.1.tar.gz
cd netcdf-4.6.1
CFLAGS="-fPIC ${CFLAGS}" CC=h5pcc ./configure --enable-shared=no --prefix=$HOME --
↪disable-dap
#NOTE: Check for this line to make sure netCDF is built with parallel I/O:
#"checking whether parallel I/O features are to be included... yes" This line comes at␣
↪the very end (last 50 lines of configure run)!
make -j8
make install
cd ..
```

2.2.5 Installing Eigen3

```
wget https://gitlab.com/libeigen/eigen/-/archive/3.4.0/eigen-3.4.0.tar.gz
tar -xf eigen-3.4.0.tar.gz
cd eigen-3.4.0
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=~
make install
cd ../..
```

2.2.6 Installing Libxsmm

```
git clone --branch 1.17 https://github.com/hfp/libxsmm
cd libxsmm
make generator
cp bin/libxsmm_gemm_generator $HOME/bin
cd ..
```

2.2.7 Installing PSpaMM

You may install PSpaMM as a Python package.

```
pip3 install --user git+https://github.com/SeisSol/PSpaMM.git
```

2.2.8 Installing ParMetis

```
wget https://ftp.mcs.anl.gov/pub/pdetools/spack-pkgs/parmetis-4.0.3.tar.gz
tar -xvf parmetis-4.0.3.tar.gz
cd parmetis-4.0.3
#edit ./metis/include/metis.h IDXTYPEWIDTH to be 64 (default is 32).
make config cc=mpicc cxx=mpiCC prefix=$HOME
make install
cp build/Linux-x86_64/libmetis/libmetis.a $HOME/lib
cp metis/include/metis.h $HOME/include
cd ..
```

(Make sure $HOME/include contains metis.h and $HOME/lib contains libmetis.a. Otherwise, compile error: cannot find parmetis.)

2.2.9 Installing ASAGI (Optional)

See section *Installing ASAGI*.

2.2.10 Installing easi

Follow the installation instructions.

2.2.11 Installing GemmForge, ChainForge (for GPUs)

```
pip3 install --user git+https://github.com/SeisSol/gemmforge.git
pip3 install --user git+https://github.com/SeisSol/chainforge.git
```

2.2.12 Installing SYCL (for GPUs)

See section *Installing SYCL*.

CHAPTER

THREE

COMPILING AND RUNNING SEISSOL

3.1 Compiling SeisSol

Get the latest version of SeisSol on git by cloning the whole repository including all submodules:

```
git clone --recursive https://github.com/SeisSol/SeisSol.git
```

If you have compiled *seissol-env with spack*, load the module and compile SeisSol with (e.g.):

```
mkdir build-release && cd build-release
CC=mpicc CXX=mpiCC FC=mpif90 cmake -DNUMA_AWARE_PINNING=ON -DASAGI=ON -DCMAKE_BUILD_
↪TYPE=Release -DHOST_ARCH=skx -DPRECISION=double -DORDER=4 -DGEMM_TOOLS_LIST=LIBXSMM,
↪PSpaMM ..
make -j 4
```

Please adapt `CC`, `CXX` and `FC` to the mpi compilers you used for compiling the dependencies. In case of a manual installation of dependencies, you may have to prepend `CMAKE_PREFIX_PATH` and `PKG_CONFIG_PATH` to the cmake command, e.g. for dependencies installed in `${HOME}`:

```
CMAKE_PREFIX_PATH=~:$CMAKE_PREFIX_PATH PKG_CONFIG_PATH=~/lib/pkgconfig/:$PKG_CONFIG_PATH␣
↪CC=...
```

It is also important that the executables of the matrix multiplication generators (Libxsmm, PSpaMM) have to be in `$PATH`.

You can also compile just the proxy by `make SeisSol-proxy` or only SeisSol with `make SeisSol-bin`

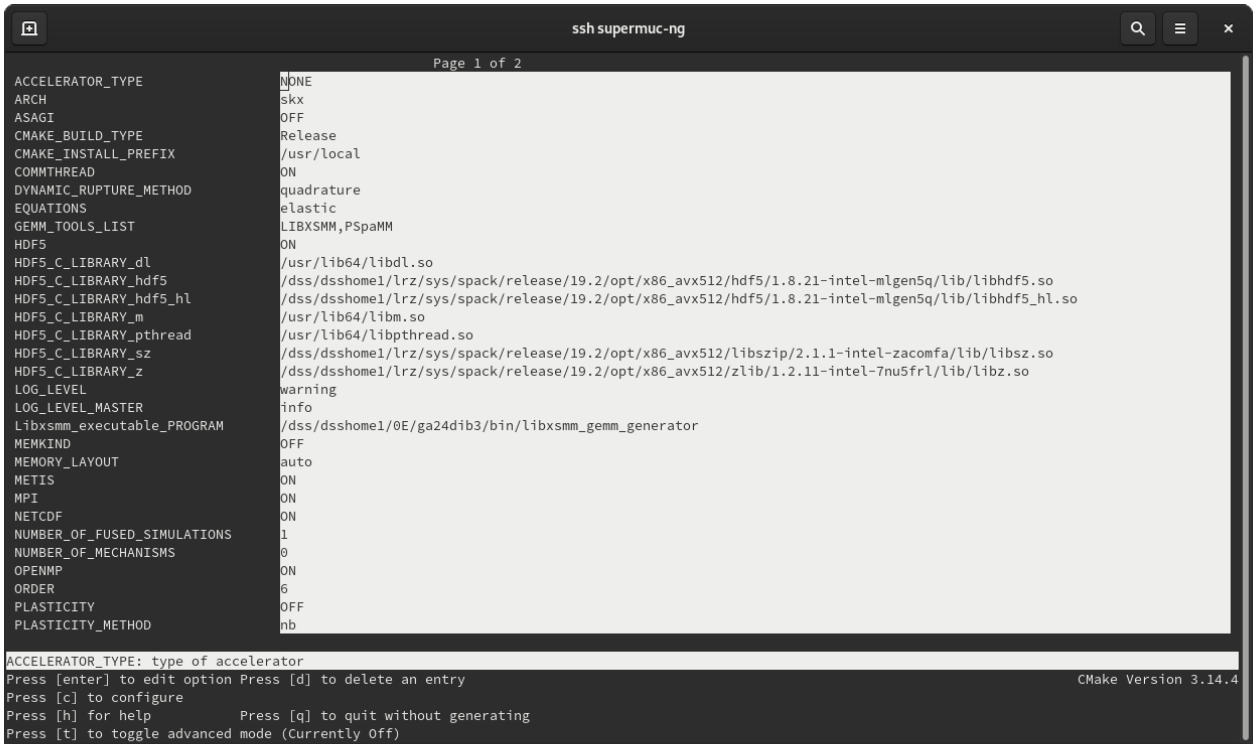
Note: CMake tries to detect the correct MPI wrappers.

You can also run `ccmake ..` to see all available options and toggle them.

3.1.1 Compile with Score-P

The Score-P measurement infrastructure is a highly scalable and easy-to-use tool suite for profiling and event tracing of HPC applications. To compile with Score-P, use:

```
SCOREP_WRAPPER=off CXX=scorep-mpic++ CC=scorep-mpicc FC=scorep-mpif90 cmake ..
SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--user --thread=omp --nomemory" make
```



```
                                   Page 1 of 2
ACCELERATOR_TYPE                 NONE
ARCH                             skx
ASAGI                            OFF
CMAKE_BUILD_TYPE                 Release
CMAKE_INSTALL_PREFIX             /usr/local
COMMTHREAD                       ON
DYNAMIC_RUPTURE_METHOD           quadrature
EQUATIONS                        elastic
GEMM_TOOLS_LIST                  LIBXSMM,PSpaMM
HDF5                             ON
HDF5_C_LIBRARY_dl                /usr/lib64/libdl.so
HDF5_C_LIBRARY_hdf5              /dss/dsshome1/lrz/sys/spack/release/19.2/opt/x86_avx512/hdf5/1.8.21-intel-mlgen5q/lib/libhdf5.so
HDF5_C_LIBRARY_hdf5_hl           /dss/dsshome1/lrz/sys/spack/release/19.2/opt/x86_avx512/hdf5/1.8.21-intel-mlgen5q/lib/libhdf5_hl.so
HDF5_C_LIBRARY_m                 /usr/lib64/libm.so
HDF5_C_LIBRARY_pthread           /usr/lib64/libpthread.so
HDF5_C_LIBRARY_sz                /dss/dsshome1/lrz/sys/spack/release/19.2/opt/x86_avx512/libszip/2.1.1-intel-zacomfa/lib/libsz.so
HDF5_C_LIBRARY_z                 /dss/dsshome1/lrz/sys/spack/release/19.2/opt/x86_avx512/zlib/1.2.11-intel-7nu5frl/lib/libz.so
LOG_LEVEL                        warning
LOG_LEVEL_MASTER                 info
Libxsmm_executable_PROGRAM       /dss/dsshome1/0E/ga24dib3/bin/libxsmm_gemm_generator
MEMKIND                          OFF
MEMORY_LAYOUT                    auto
METIS                            ON
MPI                              ON
NETCDF                           ON
NUMBER_OF_FUSED_SIMULATIONS      1
NUMBER_OF_MECHANISMS             0
OPENMP                           ON
ORDER                            6
PLASTICITY                       OFF
PLASTICITY_METHOD                nb

ACCELERATOR_TYPE: type of accelerator
Press [enter] to edit option Press [d] to delete an entry                                               CMake Version 3.14.4
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

3.2 Running SeisSol

1. Follow the instructions on *Configuration*.
2. Run SeisSol version of interest. To run the example: `./SeisSol_Release_.... parameter.par`

Further information regarding meshing and parameter files etc. can be found in the documentation folder. See also *A first example*.

CHAPTER

FOUR

A FIRST EXAMPLE

This tutorial will guide you through the steps of your first SeisSol simulation. We will use the SCEC TPV33 benchmark as an example in this tutorial. We assume that you have successfully *compiled SeisSol*.

This tutorial is targeted to people who have compiled SeisSol and want to test their installation for the first time. If you are completely new to SeisSol and want to explore its features, we recommend our training material, which bundles a pre-compiled version of SeisSol and some scenarios.

4.1 Setup

- Clone our examples repository: https://github.com/SeisSol/Examples/.
- Navigate to the folder `Examples/tpv33`. We will refer to this directory as *working directory* in the following.
- Download the mesh files from https://zenodo.org/record/8042664 and store them in the working directory.
- You can visualize the mesh file with paraview, using e.g. `paraview tpv33_half_sym.xdmf`. The mesh is described by two files: `tpv33_half_sym` and `tpv33_half_sym.xdmf`. The first one is a binary file, which contains all the data (e.g. coordinates, connectivity) and the `xdmf` file contains information on how to read that data for visualization software such as Paraview. You can read more about this mesh format here: *PUML Mesh format*.
- Create the output directory: `mkdir output`.
- **Optional** To create the mesh on your own, execute `./generating_the_mesh.sh`. To do so, you need to install gmsh, PUMGen and mirrorMesh.
- **Optional:** For performance reasons, we suggest that you store large files (mesh, output) in a scratch file system (if one is available at your cluster) and create symbolic links in your working directory:

  ```
  ln -s <path/to/tpv33_half_sym> tpv33_half_sym
  ln -s <path/to/output/directory> output
  ```

  You may not see a huge difference in this small test case but for larger meshes, this is the recommended strategy.

4.2 Execution

- Link the SeisSol binary to your working directory (`Examples/tpv33`).
- Now run: `export OMP_NUM_THREADS=<threads>`, where `<threads>` is the number of threads. If you are on a cluster or want to run with multiple MPI ranks, then you should set the number of OMP threads to the number of available threads minus 1 (the last thread is used for communication).
- Now run: `mpiexec -np <n> ./SeisSol_<configuration> parameters.par`, where:
  - `<n>` is the number of MPI ranks / the number of compute nodes used.
  - `<configuration>` depends on your compilation setting (e.g. SeisSol_Release_dhsw_4_elastic for a Haswell architecture and order 4 accuracy in space and time).
  - When running on your local desktop computer, you may also run with only one MPI rank, i.e. leave away the `mpiexec`, i.e. only type `./SeisSol_<configuration> parameters.par`. Then, you can use all available threads.

Hint: Depending on the system you are using, the MPI launcher might be different from `mpiexec` (e.g. `mpiexec.hydra`, `mpirun`, `srun`). For more infos about how to get optimal performance, have a look at the *Optimal environment variables on SuperMUC-NG*.

4.3 Result verification

SeisSol produces various output files:

- *3D wave field output* (`.xdmf`)
- *2D free surface output* (`-surface.xdmf`)
- *2D fault output* (`-fault.xdmf`)
- *Off fault receivers* (`-receiver-<id>.dat`)
- *Fault receivers* (`-faultreceiver-<id>.dat`)
- *Energy output* (`-energy.csv`)

The `xdmf` files can be visualized with Paraview. For the `dat` files, you can use viewrec.

The outputs of your simulation can be compared with our outputs (using SeisSol) and the outputs of other codes by checking out the uploaded files for this SCEC benchmark on the SCEC Code Verification Project website.

CHAPTER

FIVE

ACKNOWLEDGEMENTS

SeisSol is still under heavy development and comes without any guaranteed functionality. At the moment we can only provide very limited support for general users. Please contact Alice Gabriel (gabriel@geophysik.uni-muenchen.de) if you are interested in a close collaboration.

All copyrights belong to the seismology group @LMU and to the scientific computing group @TUM.

CHAPTER SIX

REPRODUCIBLE RESEARCH

On this page we list datasets containing reproducible simulation scenarios realised with SeisSol and elementary benchmark examples. In these data sets, all required input files, etc., are provided, to promote reproducible research and open science, and to inspire new simulation scenarios.

Please contact us via SeisSol Email list, if you're interested to provide a scenario. While many data sets are hosted on Zenodo, we do not require specific formats or repositories.

We provide datasets containing all input files of simulations in recent SeisSol publications as Zenodo repositories to promote reproducible research and open science. A non-exhaustive list is given below:

Description	Source	Faulting mechanism[Page 16, 1]	Friction law[2]	Number of faults	Research Item	Data
2004 Mw 9.2 Sumatra megathrust	dynamic	R	LSW	4	Uphoff et al. (2017)	zenodo.org/record/439946
Collective Knowledge (CK) workflow	n/a	n/a	n/a	n/a	Fursin et al. (2018)	zenodo.org/record/2422877
2017 Mw 5.5 Pohang induced earthquake	dynamic	O	fvw-RS	2	Palgunadi et al. (2020)	zenodo.org/record/3930819
2016 Mw 7.8 Kaikōura earthquake	dynamic	mixed SS & R	fvw-RS	10	Ulrich et al. (2019)	zenodo.org/record/2538024
2018 Mw 7.5 Palu earthquake	dynamic	mixed SS & N	fvw-RS	3	Ulrich et al. (2019b)	zenodo.org/record/3234664
2010 Mw 7.1 Darfield earthquake	dynamic	SS	LSW	6	Uphoff (2019)	zenodo.org/record/3565774
Acoustic-Elastic model of Palu earthquake	dynamic	mixed SS & N	fvw-RS	3	Krenz et al. (2021)	zenodo.org/record/5159333
Megathrusts initialized with a 2D STM model	dynamic	R	LSW	1	Wirp et al. (2021)	zenodo.org/record/4686551
Benchmarks for poroelasticity	point	n/a	n/a	n/a	Wolf et al. (2021)	zenodo.org/record/5236133
Low-angle normal fault scenarios	dynamic	N	fvw-RS	1	Biemiller et al. (2022)	zenodo.org/record/6094294
Megathrusts scenarios	dynamic	R	LSW	4	Madden et al. (2022)	zenodo.org/record/5914661
2016 Mw 6.2 Amatrice broadband model	dynamic	N	LSW	1	Taufiqurrahman et al. (2022)	zenodo.org/record/6386938
2016 Mw 6.5 Norcia earthquake	dynamic	N	LSW	1	Tinti et al. (2021)	github repository
2004 Mw 9.2 Sumatra megathrust	dynamic	R	LSW	4	Ulrich et al. (2022)	zenodo.org/record/5541271

[1] SS: strike-slip, N: normal, R: reverse, O: oblique
[2] LSW: linear slip-weakening friction, fvw-RS: fast-velocity weakening rate-and-state friction

SeisSol setups for community benchmark are described in the cookbook (see *cookbook overview*), and the input files are available at https://github.com/SeisSol/Examples.

Description	Source	Faulting mechanism[1]	Friction law[Page 16, 2]	Number of faults	Further details
TPV5	dynamic	SS	LSW	1	3 stress asperities, see *SCEC TPV5*
TPV6	dynamic	SS	LSW	1	Bi-material fault, heterogeneous initial stress, see *SCEC TPV6*
TPV12	dynamic	N	LSW	1	depth-dependent initial stress conditions, see *SCEC TPV12*
TPV13	dynamic	N	LSW	1	Same as TPV12 with non-associative Drucker-Prager plastic with yielding in shear, see *SCEC TPV13*
TPV16	dynamic	SS	LSW	1	Randomly-generated heterogeneous initial stress conditions, see *SCEC TPV16/17*
TPV24	dynamic	SS	LSW	2	Rightward branch forming a 30 degree angle, see *SCEC TPV24*
TPV29	dynamic	SS	LSW	1	Stochastic roughness, see *SCEC TPV29*
TPV34	dynamic	SS	LSW	1	Imperial Fault model with 3D velocity structure, see *SCEC TPV34*
TPV104	dynamic	SS	fvw-RS	1	see *SCEC TPV104*
LOH.1	point	n/a	n/a	n/a	point-source benchmark, see *SISMOWINE WP2_LOH1*
Northridge	kinematic	R	n/a	1	see *Kinematic source example - 1994 Northridge earthquake*

We provide the following small-scale examples, specifically designed for SeisSol training and tutorials, such as the CHEESE Advanced training on HPC for Computational Seismology and ICTP Advanced Workshop on Earthquake Fault Mechanics . These SeisSol training examples are part of the SeisSol Docker container which also includes related open-source tools (Gmsh and ParaView) and all required input files.

Description	Source	Faulting mechanism[Page 16, 1]	Friction law[Page 16, 2]	Number of faults	Data
TPV13	dynamic	N	LSW	1	https://github.com/SeisSol/Training/tree/main/tpv13
2018 Mw 7.5 Palu earthquake (reduced mesh-size)	dynamic	mixed SS & N	fvw-RS	3	https://github.com/SeisSol/Training/tree/main/sulawesi
Northridge	kinematic	R	n/a	1	https://github.com/SeisSol/Training/tree/main/northridge

CHAPTER SEVEN

RELATED PUBLICATIONS

Ulrich, T., A.-A. Gabriel,, J. P., Ampuero, & W. Xu, (2019). Dynamic viability of the 2016 Mw 7.8 Kaikōura earthquake cascade on weak crustal faults. Nature communications. doi: 10.1038/s41467-019-09125-w.

Wollherr, S., A.-A. Gabriel, and C. Uphoff (2018), Off-fault plasticity in three-dimensional dynamic rupture simulations using a modal Discontinuous Galerkin method on unstructured meshes: implementation, verification and application, Geophys. J. Int., 214(3), 1556-1584, doi: 10.1093/gji/ggy213.

Uphoff, C., S. Rettenberger, M. Bader, E. H. Madden, T. Ulrich, S. Wollherr, and A.-A. Gabriel (2017), Extreme scale multi-physics simulations of the tsunamigenic 2004 sumatra megathrust earthquake, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, edited, pp. 1-16, ACM, Denver, Colorado, doi: 10.1145/3126908.3126948.

Breuer, A., A. Heinecke, and M. Bader (2016), Petascale Local Time Stepping for the ADER-DG Finite Element Method, paper presented at 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 23-27 May 2016. doi: 10.1109/IPDPS.2016.109.

Pelties, C., A. A. Gabriel, and J. P. Ampuero (2014), Verification of an ADER-DG method for complex dynamic rupture problems, Geosci. Model Dev., 7(3), 847-866, doi: 10.5194/gmd-7-847-2014.

Breuer, A., A. Heinecke, S. Rettenberger, M. Bader, A.-A. Gabriel, and C. Pelties (2014), Sustained Petascale Performance of Seismic Simulations with SeisSol on SuperMUC, Springer International Publishing, Cham, doi: 10.1007/978-3-319-07518-1_1.

Heinecke, A., A. Breuer, S. Rettenberger, M. Bader, A. Gabriel, C. Pelties, A. Bode, W. Barth, X. Liao, K. Vaidyanathan, M. Smelyanskiy, and P. Dubey (2014), Petascale High Order Dynamic Rupture Earthquake Simulations on Heterogeneous Supercomputers, paper presented at SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 16-21 Nov. 2014, doi: 10.1109/SC.2014.6.

Pelties, C., J. de la Puente, J.-P. Ampuero, G. B. Brietzke, and M. Käser (2012), Three-dimensional dynamic rupture simulation with a high-order discontinuous Galerkin method on unstructured tetrahedral meshes, J. Geophys. Res., 117(B2), doi: 10.1029/2011JB008857.

de la Puente, J., J.-P. Ampuero, and M. Käser (2009), Dynamic rupture modeling on unstructured meshes using a discontinuous Galerkin method, J. Geophys. Res., 114(B10), doi: 10.1029/2008JB006271.

Käser, M., M. Dumbser, J. de la Puente, and H. Igel (2007), An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes - III. Viscoelastic attenuation, Geophys. J. Int., 168(1), 224-242, doi: 10.1111/j.1365-246X.2006.03193.x..

Käser, M., and M. Dumbser (2006), An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes - I. The two-dimensional isotropic case with external source terms, Geophys. J. Int., 166(2), 855-877, doi: 10.1111/j.1365-246X.2006.03051.x.

Dumbser, M. and Käser, M. (2006), An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes – II. The three-dimensional isotropic case. Geophysical J. Int., 167: 319-336. doi: 10.1111/j.1365-246X.2006.03120.x.

CHAPTER

EIGHT

CAD MODELS

The following help pages describe how to build a structural model with either Gocad or SimModeler.

8.1 SimModeler CAD workflow

Since September 2019, SimModeler features powerful tools for processing discrete data (geometry in the form of meshes), which allow building structural models without the need to rely on additional CAD software. We illustrate the SimModeler CAD workflow by building the structural model of the Palu earthquake dynamic rupture scenario (Ulrich et al., 2019). See *SimModeler CAD workflow*.

8.2 GOCAD CAD workflow

GOCAD is the tool we historically used to process complex geophysical data into structural models. Since then, SimModeler developed tools for processing discrete data, in particular, a discrete surface intersection algorithm, which is much faster and more reliable than the one from GoCAD. We, therefore, recommend the use of the SimModeler workflow. Because GoCAD may still be useful for fine processing of surface data (e.g. surface smoothing with constraints), we detail the full GoCAD workflow at: *Generating a CAD model using GOCAD: basic tutorial*.

8.3 Useful scripts

A collection of python scripts typically used to create the surfaces used in the CAD model is available here. They are documented (try -h option). The most important scripts are:

- `create_fault_from_trace.py` allows creating a meshed surface from a fault trace. The fault trace is resampled, smoothed, and extended using either a constant, a depth-varying, or an along-strike varying dip.
- `create_surface_from_rectilinear_grid.py` allows creating a meshed surface from a (possibly sparse, e.g. Slab2.0 dataset) structured dataset (e.g. netcdf).
- `create_surface_from_structured_grid.py` allows creating a meshed surface from structured grid of nodes. Vertices in a row or a column do not necessary share the same x and y values (non rectilinear grid).
- `surface_from_one_dim_structured_grid.py` allows creating a meshed surface from a partially structured grid of nodes. The point set should consist of serveral lines of nodes of same y (or x, depending on args.axis) coordinates. Contrary to `create_surface_from_structured_grid.py`, the number of nodes on a line (resp. on a column) is not necessarily constant. On the other hand, the lines (resp. the columns) of the point cloud should share constant ordinates (resp. abscissa).
- `convertTs.py` allows converting the geometric model from Gocad into another supported format (e.g. stl, bstl).

8.4 Processing high-resolution topographic data

High resolution topographic and bathymetric data are usually available. Generating geometric models including such large datasets can be challenging. In particular, intersecting such surfaces with other surfaces can be time-consuming and error-prone. Here we present various strategies and tools to overcome this challenge.

8.5 Using Gdal

Gdal is a powerful library to process gridded data. It allows, for instance, to easily resample or crop a dataset, and to convert files in handy file formats. Here is a commented example of our use of Gdal to create a ts surface from a high-resolution topography of Nepal (file data/merged_original.tif).

```
#resample data
gdalwarp -s_srs EPSG:4326 -r near -tr 0.0025 0.0025 data/merged_original.tif data/
↪file250b.tif
#crop data
gdalwarp -te 83.7 26. 88.1 29.4 data/file250b.tif data/file250.tif
#change format
gdal_translate -of netCDF -co "FOMRAT=NC4" data/file250.tif data/file250.nc
#python script from 'creating_geometric_model'
#The specified hole allows to use algorithm described in 'remeshing the topography'
python3 create_surface_from_rectilinear_grid.py data/file250.nc data/file250.stl --proj
↪"+init=EPSG:32645" --hole 84.8 86.5 27.1 28.3
```

8.6 Topographic data coarsening with SimModeler

To avoid dealing with too large files when building the CAD model, topography data can be coarsened where fine resolution is not necessary. For further details, see *Remeshing the topography*.

The same procedure can be also useful when the intersection between 2 surfaces fails in Gocad. In fact, creating a clean mesh of one of the surfaces can facilitate the intersection step in Gocad. In such a case, all surface already intersected with the surface that we want to mesh again have to be exported to SimModeler. The mesh attributes “Use Discrete Geometry Mesh” and “No mesh” have to be assigned to these surfaces. This will ensure that the border nodes of the new meshed surfaces keep unchanged.

8.7 Alternative using Gocad

It can occur that the procedure described in *Remeshing the topography* is not applicable. For example, if a first model with fine topography has been compiled, and we want to extend it without starting from scratch. In this case, an alternative procedure can be used: *Adapting the CAD model resolution using Gocad*.

8.8 Dealing with intersection artifacts

Manually fixing an intersection in Gocad

8.9 On the use of projections

Special care must be taken when projecting from WGS84 to a projected coordinate system (e.g. Mercator) as the coordinates of the projected model can then be centered on a point distant from (0,0), which can cause numerical precision issues when building the geometric model or when meshing. For instance, for the Kaikoura scenario, we used EPSG:3994, leading to a model centered on (6e6,-4e6) m for a model size of roughly 500 km. It can then be a good idea to manually center back the model on (0,0,0). This can usually be done by using the option +x_0=xxx and +y_0=yyy in the projection description.

CHAPTER

NINE

MESHING WITH SIMMODELER

The meshing workflow is presented through a simple example, by meshing the CAD model obtained from *Generating a CAD model using GOCAD: basic tutorial*. The created stl-file is imported via `File > Import Discrete Data`.

9.1 Prerequisite

The procedure to download SimModeler (GUI) and the SimModeler modeling suite (library) is detailed here. Note that to be able to properly define the boundary conditions and to be able to export the mesh in the proper format, SimModeler has to be SeisSol customized.

9.2 SimModeler version

We have used so far 3 main versions of SimModeler (3, 4 and 5). Sometimes, quality meshes can be obtained on older versions of SimModeler whereas the latest version of SimModeler is not able to get quality meshes (in that case the support of SimModeler is very reactive and helpful). It is then important to notice that smd file created in older versions of SimModeler can be read in all SimModeler versions. On the other hand, smd file from the latest simModeler releases are not backward compatible. Anyway, in most cases, we strongly recommend using the latest version of SimModeler.

9.3 Analysis tab

tab Analysis > Click twice on "New Case" on the Analysis Attributes panel. give a name. If your SimModeler is set for SeisSol, the solver seisSol should appear in the drop-down menu.

Select the top surface (several surfaces can be selected by holding Shift), click on the + sign > Boundary conditions > Free Surface. And then on Apply-close (no need to enter a name). Process similarly for the Absorbing and Dynamic rupture boundary conditions.

9.4 Meshing tab

The default Surface Meshing and Volume meshing attributes are initially set. Their default attributes can be changed by clicking on them in the mesh attribute tab. In particular, the Smoothing algorithm can be changed from Laplacian to gradient (quoting SimModeler manual, "This algorithm will generally produce better results, but at some performance cost"). The Smoothing level can also be changed (max 1 for Volume meshing and 4 for Surface meshing according to the manual). Finally, the Discrete Face Rotation Angle Limit is also a parameter to consider, for knowing to which extend the CAD model has to be matched.

+ > Mesh Size > Absolute > e.g. 5000 will define a maximum mesh size in the model.

+ > Gradation > Rate > e.g. 0.15 will define the coarsening rate within the mesh. The smaller the value, the slower the coarsening within the mesh.

Click on the fault then + > Mesh Size > Absolute > e.g. 250 to define the on-fault size.

Click on the fault then + > Mesh Size Propagation > propagation distance: e.g. 1000, scaling Factor e.g. 2. This allows the mesh to remain fine in a box bounding the fault. For example here, the mesh is coarsened away from the fault according to the gradation rate set, with a maximum value of 2*250m = 500m within this box.

+ > Surface Shape Metric > Aspect Ratio > e.g. 3 and

+ > Volume Shape Metric > Aspect Ratio > e.g. 6 will define quality levels that the mesher will try to enforce. The mesher will not necessarily create a mesh which passes all the Shape Metric set. From our experience, setting additional shape metrics does not help improving the mesh. An easy mesh can reach AR < 10. For more complex meshes, AR < 40 should be expected.

9.5 Generating the mesh

Meshing tab > Generate Mesh

9.6 Checking mesh quality

Display tab > Mesh Stats > check Aspect Ratio > look at extreme value and results spread.

Display tab > Region Select > change the Aspect Ratio range and visualize where are the badly shaped elements. If they are related to some geometric features of the CAD model (e.g. narrow layers, shallow dipping fault) then the CAD model should be modified to allow meshes of better quality.

9.7 Exporting the mesh

File > Export Analysis > e.g. test.neu

In case of a large mesh (several million cells), it is probably quicker to mesh using Pumgen. In this case, before meshing save the model file before meshing, and run the mesher using Pumgen.

CHAPTER

TEN

MESHING WITH PUMGEN

10.1 PUMGen

PUMGen (https://github.com/SeisSol/PUMGen/wiki) is a tool for creating and exporting meshes in an efficient format (see *PUML Mesh format*). We recommend using PUMGen, in particular when large meshes have to be generated. PUMGen uses SimModeler libraries for meshing. Here is a basic example of use:

```
pumgen -s simmodsuite -l SimModelerLib.lic --mesh "Mesh case 1" --analysis "Analysis␣
↪case 1" test.smd test
```

The Analysis and meshing attributes of the model file test.smd have been first defined using the GUI of SimModeler, as detailed in *Meshing with SimModeler*. In particular, the –mesh and –analysis attributes are set according to the mesh attributes and analysis attributes defined in the mesh and analysis tabs (SimModeler GUI). This script will generate 2 files: test and test.xdmf describing the mesh (see *PUML Mesh format*).

The "Minimum insphere found" output of PUMgen allows checking the mesh quality. The minimum insphere of a quality mesh is about 1/10 or more of the minimum mesh size specified. Smaller values might impact your time step, and then your simulation time, but not so much if using local time stepping. Also note the –analyseAR option of PUMgen, allowing to produce a histogram of the mesh aspect ratios, similarly to SimModeler GUI.

PUMGen can be used to convert an ASCII *.neu mesh file (for instance from the GUI of SimModeler) to the xmdf mesh format:

```
pumgen test.neu test
```

PUMGen can be also used to convert a netcdf mesh file (older SeisSol mesh file format) to the xmdf mesh format:

```
pumgen -s netcdf test.nc test
```

Or to convert a gmsh mesh file (msh2) to the xmdf mesh format:

```
pumgen -s msh2 test.msh test
```

Multiprocessing can be used to speed up the process:

```
mpirun -n 4 pumgen -s netcdf test.nc test
```

Note that version < 10 of Simulation Modeling Suite do not support parallel volume meshing.

10.2 Parametrizing PUMGen with an xml file

The parametrization of meshing and analysis attributes using the GUI of SimModeler can be tedious, particularly for heavy models (>Gb smd file, with finely sampled topography or fault surface) or when running parametric studies with different meshes or analysis attributes. The –xml option of PUMGen offers a way to tag boundary conditions surfaces and to set mesh attributes using an xml file. In addition, this allows keeping track of the meshing parameters in an xml file. A typical xml file can be found here.

A typical use of the parametrization through xml file could be:

```
pumgen -s simmodsuite -l SimModelerLib.lic --xml meshAttributes.xml test.smd test
```

The GUI of SimModeler can be used to find the correspondence between region/surface and id.

10.3 Velocity-aware meshing

PUMGen supports automatic mesh refinement depending on the velocity structure specified in an easi file. PUMGen generates a mesh with a local element size that satisfies the specified number of `elementsPerWaveLength` for the target `frequency` within the `VelocityRefinementCuboid`. As a rule of thumb, running SeisSol with `-DORDER=6` resolves the target frequency when using two elements per wavelength (for details see Käser et al., 2008).

Velocity-aware meshing is enabled within the xml file:

```
<VelocityAwareMeshing easiFile="material.yaml" elementsPerWaveLength="2">
  <VelocityRefinementCuboid frequency="2.0" centerX="0" centerY="0" centerZ="0"
                            halfSizeX="1e5" halfSizeY="1e5" halfSizeZ="1e5"
                            bypassFindRegionAndUseGroup="1"/>
</VelocityAwareMeshing>
```

CHAPTER

ELEVEN

GMSH

11.1 Introduction

Gmsh is an open-source mesher, able to generate quality meshes for not too complex models. Contrary to SimModeler, it can build geometric models from scratch. It is particularly useful for simple idealized models, e.g. planar fault, no topography. Two examples of model generation using Gmsh are provided at this link. The purpose of this tutorial is not to explain all functions of gmsh, but to describe the features useful for setting a model and getting a mesh for SeisSol.

11.2 Coarsening the mesh

The Attractor Field feature is used to coarsen the mesh away from the fault. As it requires a “Ruled Surface”, we define an artificial “Ruled Surface” used only for that feature. The rate of coarsening is defined empirically by combining a linear increase of the mesh size in the near field and a quadratic increase in the far size. Example:

```
Field[2].F = Sprintf("0.1*F1 +(F1/5.0e3)^2 + %g", lc_fault);
```

11.3 Boundary conditions

The free-surface (resp. dynamic rupture, resp. absorbing) boundary conditions are set using Physical Surfaces 101 (resp. 103, resp. 105). The volumes should also be put into Physical Volume to be exported into the mesh. Here is an example from tpv33:

```
Physical Surface(101) = {252, 258, 260, 262};
Physical Surface(105) = {242, 244, 246, 248, 250, 254, 256};
Physical Surface(103) = {2, 3, 4};
Physical Volume(2) = {276,278};
Physical Volume(3) = {274};
```

11.4 Generating the mesh

Once the geometry and boundary conditions are set, the mesh can be obtained using the following command:

```
gmsh test.geo -3 -optimize -format neu
```

Note that the '-format neu' is only possible since gmsh 4.0 For previous versions, we used gmsh2gambit on the msh mesh generated with:

```
gmsh test.geo -3 -optimize
```

The optimize option is very important. If not used, mesh of very poor quality may be obtained.

11.5 gmsh to SimModeler

It is possible to create the geometry with gmsh and then mesh it with SimModeler. A way of doing so is to put all surfaces of the model in a "physical surface", mesh them (-2) and output them to an stl file (e.g. -o test.stl). Then the stl file can be opened with SimModeler and the mesh can be generated.
If SimModeler merges some features of the geometry, it is then necessary to isolate the features in different stl files (i.e. running several times `gmsh ___.geo -2 -o ___.stl` with different surfaces listed in the physical surface listing). Then the solid name attribute of the stl files has to be modified. Finally, the stl files can be merged into a single stl file, to be opened in SimModeler.

11.6 mirroring a mesh

In order to get maximum accuracy, it is sometimes necessary (e.g. for benchmarks) to mirror a mesh. To get a mirrored mesh, a half mesh is first generated. The half mesh is then converted to PUML format using PUMGen (if not already in this format). Finally, this script allows creating the mirrored mesh

11.7 Add topography in GMSH

The roughed fault interface model is generated with Gmsh is complicated than planar faults in previous sections. There are 5 steps to generate the model.

1.Create topography data. The format is the following:

```
Line 1: num_x, num_y
Line 2 to nx: positions of nodes along the strike (in meters)
Line nx+3 to ny+nx+3: positions of nodes along the downdip (in meters)
Line to the end: the topography of each node (nx\*ny, in meters)
```

Save this file as *mytopo.dat*.

2.Make a model with a plane surface first (step1.geo).

```
cl = 1;

// This file builds a rectangular box domain region which is exactly the same as␣
↪topographic data.

level = 0.0; // horizontal elevation
region = 220; // range in meter
depth = 100;

Point(1) = { 0.5*region, 0.5*region, level, cl} ; //water level
Point(2) = { -0.5*region,0.5*region, level, cl} ;
Point(3) = { -0.5*region,-0.5*region, level, cl} ;
Point(4) = { 0.5*region, -0.5*region, level, cl} ;

Line(1) = {1,2}; Line(2) = {2,3}; Line(3) = {3,4}; Line(4) = {4,1};

Point(5) = { 0.5*region, 0.5*region,-depth, cl} ;
Point(6) = { -0.5*region,0.5*region,-depth, cl} ;
Point(7) = { -0.5*region,-0.5*region,-depth, cl} ;
Point(8) = { 0.5*region,-0.5*region, -depth, cl} ;

Line(5) = {5,6}; Line(6) = {6,7}; Line(7) = {7,8}; Line(8) = {8,5};

Line(9) = {1,5}; Line(10) = {2,6}; Line(11) = {3,7}; Line(12) = {4,8};

Line Loop(1) = {  1,  2,   3,  4} ; Plane Surface(1) = {1} ;// the free surface
Line Loop(2) = {  5,  6,   7,  8} ; Plane Surface(2) = {2} ;
Line Loop(3) = {  -4, 12,  8,  -9} ; Plane Surface(3) = {3} ; //
Line Loop(4) = {  9,  5, -10,  -1} ; Plane Surface(4) = {4} ;
Line Loop(5) = { 10,  6,  -11, -2} ; Plane Surface(5) = {5} ;
Line Loop(6) = { 11,  7,  -12, -3} ; Plane Surface(6) = {6} ;

Physical Surface(101) = {1};// free surface
Physical Surface(105) = {2,3,4,5,6};//absorb boundary

Mesh.MshFileVersion = 1.0;
```

then generate msh file by:

```
$ gmsh step1.geo -2 -o step1.msh
```

3.Use *gmsh_plane2topo.f90* and interpol_topo.in* to shift the planar surface according to positions given in *mytopo.dat*.

```
$ ./gmsh_plane2topo interpol_topo.in
```

gmsh_plane2topo.f90 can be found in https://github.com/daisy20170101/SeisSol_Cookbook/tree/master/tpv29

The format of interpol_topo.in is following:

```
&input ! this is the input file for "interpol_topo"

!
!- name of the topography file:
!
```

(continues on next page)

(continued from previous page)

```
   TopoFile = 'mytopo.dat'
!
!- name of the input and output mesh files:
!
   SkinMeshFileIn  = 'step1.msh'
   SkinMeshFileOut = 'step1_modified.msh'
!
!- face #s corresponding to the surface:
!
   SurfaceMeshFaces = 1  ! free-surface will be modified
!
!- optionals:
!
   MeshFacesToSmooth =  3, 4, 5,6  ! face #s

   IterMaxSmooth = 100 ! default=200
   TolerSmooth   = 0.01 ! default=0.01

/ ! end of data
```

This will generate a step1_modified.msh file containing topography. Load this in Gmsh to double-check.

4.Make a new step2.geo file that contains the topography and mesh follow the general GMSH process.

The format of step2.geo is following:

```
Merge "step1_modified.msh"; // merge modified msh

Surface Loop(1) = {1,2,3,4,5,6};
Volume(1)={1};
Physical Volume(1) = {1};

Mesh.MshFileVersion = 2.2;
```

The new geometry with topography:

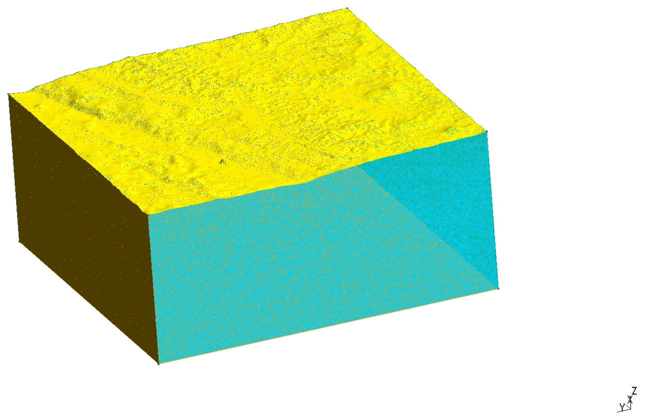


Fig. 1: Diagram showing the geometry with topography.

5. Generate MSH mesh with the command line:

```
& gmsh step2.geo -3 -optimize_netgen -o step2.msh
```

option optimize_netgen is necessary for optimizing meshing with good quality.

CHAPTER

TWELVE

CONFIGURATION

To set up a SeisSol run, you need to do the following steps (assuming you are in the root directory of the code repository):

1. Create the launch directory:

```
mkdir launch_SeisSol
```

2. Copy all executables to the launch directory:

```
cp build/SeisSol* launch_SeisSol/
```

3. Create your *Parameter File*
4. Copy any additional input files referenced in the parameter file (for example file with receiver coordinates) to your launch directory
5. (For large mesh only) Create symbolic links (ln -s) to the mesh file(s) in your launch directory
6. Make sure output and checkpoint directories exist
7. Optional: set *Environment Variables* for tuning

12.1 Checklist for required files

12.1.1 Necessary files

- SeisSol executable (compiled on the system where the job will run)
- Parameter file
- *.yaml files for setting model parameters

12.1.2 Optional files depending on settings in the parameter file

- receiver files in *.dat format
- fault receiver files in *.dat format (in the parameter file)

CHAPTER THIRTEEN

PARAMETER FILE

13.1 General Information

The parameter file in SeisSol is based on the Fortran NAMELIST format. The file is divided into different sections. Each section has a set of configuration parameters that influences the execution of SeisSol. Each configuration parameter can be one of the following:

- **Integer**
- **Float**
- **Array** Arrays can contain integers or floats and have a fixed size. The elements are separated by spaces (`1 2 3 4`)
- **String**
- **Path** A path references a file or a directory and is given as a string in the parameter file with additional restrictions. A path might be absolute or relative to the starting directory of the execution. If the path is used for an output file, the user has to make sure that the directory exists. (E.g. If the path is set to "output/wavefield", then the directory "output" must exist.)
- **Path prefix** Path prefixes are similar to paths. However, SeisSol will automatically append a filename extension or other suffixes (e.g. "-fault").

13.2 Commented parameter file

```
&equations
!yaml file defining spatial dependance of material properties
MaterialFileName = '33_layered_constant.yaml'
!1: Compute average materials for each cell, 0: sample material values at element␣
↪barycenters
UseCellHomogenizedMaterial = 1
!off-fault plasticity parameters (ignored if Plasticity=0)
Plasticity=0
Tv=0.05
!Attenuation parameters (ignored if not compiled with attenuation)
FreqCentral=0.5
FreqRatio=100
GravitationalAcceleration = 9.81 ! value of gravitational acceleration
!ITM Parameters
ITMEnable = 1 ! 1 is on, 0 is off
```

(continues on next page)

(continued from previous page)

```
ITMStartingtime = 2.0 ! Time when ITM is turned on
ITMTime = 0.01 !Time duration for which ITM is on
ITMVelocityscalingfactor = 2 ! Scaling factor
ITMReflectionType = 1 ! 1 reflects both waves, 2 reflects both waves with constant␣
↪velocity, 3 reflects only P-waves, 4 reflects only S-waves
/

&IniCondition
cICType = 'Zero'                  ! Initial Condition
!If not specified the default value is Zero
!Possible values are
!Zero        All zero - the standard case to work with point source or dynamic rupture
!Planarwave  A planar wave for convergence tests, needs periodic boundary conditions
!Travelling  A planar wave travelling wave (one period of a plane wave), needs more␣
↪parameters to be specified
!AcousticTravellingwithITM A travelling acoustic wave with ITM
!Scholte     A Scholte wave to test elastic-acoustic coupling
!Snell       Snells law to test elastic-acoustic coupling
!Ocean       An uncoupled ocean test case for acoustic equations

!The following parameters are only needed for the travelling wave IC:
origin = 0 0 0.5             ! Origin of the wave
kVec = 6.283 0 0             ! Gives direction of wave propagation, the wavelength of␣
↪the travelling wave is 2*pi / norm(kVec)
k = 6.283 ! Wave number to be used for the travelling acoustic wave with ITM test case.␣
↪Not to be used in other scenarios
ampField = 2 0 0 0 0 0 0 1 0 ! Amplification of the different wave modes
/

&DynamicRupture
FL = 16                      ! Friction law
!0: none, 16:LSW, 103: RS with strong velocity weakening
!yaml file defining spatial dependance of fault properties
ModelFileName = '33_fault0.yaml'

!reference vector for defining strike and dip direction
XRef = -0.1                  ! Reference point
YRef = 0.0
ZRef = -1.0
refPointMethod = 1

etahack = 1                 ! use any value smaller than one to mitigate quasi-divergent␣
↪solutions in friction laws

OutputPointType = 5         ! Type (0: no output, 3: ascii file, 4: paraview file, 5:␣
↪3+4)
SlipRateOutputType=0        ! 0: (smoother) slip rate output evaluated from the␣
↪difference between the velocity on both side of the fault
                            ! 1: slip rate output evaluated from the fault tractions and␣
↪the failure criterion (less smooth but usually more accurate where the rupture front␣
↪is well developped)
TpProxyExponent = 0.3333333333333333 ! exponent alpha in the TP proxy slip weakening law␣
```

(continues on next page)

(continued from previous page)

```
↪(default is 1/3)
/

!see: https://seissol.readthedocs.io/en/latest/fault-output.html
! parameterize paraview file output
&Elementwise
printtimeinterval_sec = 0.2      ! Time interval at which output will be written
OutputMask = 1 1 1 1 1 1 1 1 1 1 1 1   ! turn on and off fault outputs
refinement_strategy = 2
refinement = 1
/

! parameterize ascii fault file outputs
&Pickpoint
printtimeinterval = 1       ! Index of printed info at timesteps
OutputMask = 1 1 1 1 1 1 1 1 1 1 1 1   ! turn on and off fault outputs
PPFileName = 'tpv33_faultreceivers.dat'
/

&SourceType
!Type = 50   ! 50: point source described by an ASCII file
!Type = 42   ! 42: finite source in netcdf format
!FileName = 'source_norm.dat'
/

&MeshNml
MeshFile = 'tpv33_gmsh'         ! Name of mesh file
pumlboundaryformat = 'auto'      ! the boundary data type for PUML files
meshgenerator = 'PUML'          ! Name of meshgenerator (Netcdf or PUML)
PartitioningLib = 'Default' ! name of the partitioning library (see src/Geometry/
↪PartitioningLib.cpp for a list of possible options, you may need to enable additional␣
↪libraries during the build process)
/

&Discretization
CFL = 0.5                            ! CFL number (<=1.0)
FixTimeStep = 5                      ! Manually chosen maximum time step
ClusteredLTS = 2                     ! 1 for Global time stepping, 2,3,5,... Local time␣
↪stepping (advised value 2)
!ClusteredLTS defines the multi-rate for the time steps of the clusters 2 for Local time␣
↪stepping
LtsWeightTypeId = 1                  ! 0=exponential, 1=exponential-balanced, 2=encoded
vertexWeightElement = 100 ! Base vertex weight for each element used as input to ParMETIS
vertexWeightDynamicRupture = 200 ! Weight that's added for each DR face to element vertex␣
↪weight
vertexWeightFreeSurfaceWithGravity = 300 ! Weight that's added for each free surface with␣
↪gravity face to element vertex weight

! Wiggle factor settings:
! Wiggle factor adjusts time step size by a small factor. This can lead to a slightly␣
↪better clustering.
LtsWiggleFactorMin = 1.0 ! Minimal wiggle factor applied to time step size. Should be >␣
```

(continues on next page)

(continued from previous page)

```
↪1/rate
LtsWiggleFactorStepsize = 0.01 ! Stepsize for wiggle factor grid search
LtsWiggleFactorEnforceMaximumDifference = 1 ! 0 or 1: Enforces the maximum difference␣
↪between neighboring clusters during wiggle factor search
LtsMaxNumberOfClusters = 20 ! Enforces a maximal number of clusters
LtsAutoMergeClusters = 0 !  0 or 1: Activates auto merging of clusters
LtsAllowedRelativePerformanceLossAutoMerge = 0.1 ! Find minimal max number of clusters␣
↪such that new computational cost is at most increased by this factor
LtsAutoMergeCostBaseline = 'bestWiggleFactor' ! Baseline used for auto merging clusters.␣
↪Valid options: bestWiggleFactor / maxWiggleFactor


/

&Output
OutputFile = '../output_tpv33/tpv33'
WavefieldOutput = 1                  ! disable/enable wavefield output (right now,␣
↪format=6 needs to be set as well)
Format = 6                           ! Format (10= no output, 6=hdf5 output)
!             |stress     |vel
iOutputMask = 0 0 0 0 0 0 1 1 1
!                 |strain     |eta
iPlasticityMask = 0 0 0 0 0 0 1
TimeInterval = 2.                    ! Index of printed info at time
refinement = 1
OutputRegionBounds = -20e3 20e3 -10e3 10e3 -20e3 0e3 !(optional) array that describes␣
↪the region
! of the wave field that should be written. Specified as 'xmin xmax ymin ymax zmin zmax'

! off-fault ascii receivers
ReceiverOutput = 1                   ! Enable/disable off-fault ascii receiver output
RFileName = 'tpv33_receivers.dat'    ! Record Points in extra file
pickdt = 0.005                       ! Pickpoint Sampling
! (Optional) Synchronization point for receivers.
!            If omitted, receivers are written at the end of the simulation.
ReceiverOutputInterval = 10.0
ReceiverComputeRotation = 1          ! Compute Rotation of the velocity field at the␣
↪receivers

! Free surface output
SurfaceOutput = 1
SurfaceOutputRefinement = 1
SurfaceOutputInterval = 2.0

!Checkpointing
Checkpoint = 1                       ! enable/disable checkpointing
checkPointFile = 'checkpoint/checkpoint'
checkPointBackend = 'mpio'           ! Checkpoint backend
checkPointInterval = 6

xdmfWriterBackend = 'posix' ! (optional) The backend used in fault, wavefield,
! and free-surface output. The HDF5 backend is only supported when SeisSol is compiled␣
```

(continues on next page)

(continued from previous page)

```
↪with
! HDF5 support.

EnergyOutput = 1 ! Computation of energy, written in csv file
EnergyTerminalOutput = 1 ! Write energy to standard output
EnergyOutputInterval = 0.05
ComputeVolumeEnergiesEveryOutput = 4 ! Compute volume energies only once every␣
↪ComputeVolumeEnergiesEveryOutput * EnergyOutputInterval

LoopStatisticsNetcdfOutput = 0 ! Writes detailed loop statistics. Warning: Produces␣
↪terabytes of data!
/

&AbortCriteria
EndTime = 15.0
terminatorMaxTimePostRupture = 5.0   ! Stops SeisSol x sec after slip rate everywhere on␣
↪the fault or seismic moment rate is below
                                     ! the given threshold
terminatorSlipRateThreshold = 0.5    ! Slip rate threshold for the above criteria (the␣
↪proposed value should filter out locally high SR
                                     ! in "creeping" fronts (numerical artifacts when␣
↪rupture dies out).
terminatorMomentRateThreshold = 2e18 ! Seismic moment rate threshold for the above␣
↪criteria
/
```

13.3 Sections

Additional, more detailed information on several sections are listed here.

13.3.1 DynamicRupture

Reference point

The slip rate is defined as the velocity difference between the two sides of a fault, that is,

$\Delta v = v^+ - v^-$.

A practical issue is to define which side at an interface corresponds to "+" and which one to "-". The reference point defines which side is which and it is **crucial** to set it correctly.

The parameters `XRef, YRef, ZRef` define the coordinate vector of the reference point, which we denote with $\mathbf{r}$. Furthermore, the `refPointMethod` has to specified, whose effect is outlined in the following.

1. **refPointMethod=0**

   In order to decide if a side of a fault is "+" or "-" we compute the vectors $\mathbf{n}$, $\mathbf{x}$, and $\mathbf{y}$ for a face of a tetrahedron, whose boundary condition indicates it to be part of the fault. The vector $\mathbf{n}$ is the face's normal, which always points outward with respect to the tetrahedron. The vector $\mathbf{x}$ is an arbitrary vertex of the face and the vector $\mathbf{y}$ is face's the missing vertex, that is, the vertex which belongs to the tetrahedron but not to the face.

   We define

   $\text{isPlus} := \langle \mathbf{r} - \mathbf{x}, \mathbf{n} \rangle \cdot \langle \mathbf{y} - \mathbf{x}, \mathbf{n} \rangle > 0$

isPlus is only true whenever **r**-**x** and **y**-**x** point in the same direction (lie in the same half-space w.r.t. **n**).

This method works, as long as the sign of the first dot product is the same for all faces tagged as being part of the fault.

Example: One has a planar fault with normal **N** and an arbitrary point **z** on the plane. Then a good reference point would be **z**+**N**. In this case, **n**=**N** and

$\langle \mathbf{z} + \mathbf{N} - \mathbf{x}, \mathbf{n} \rangle = \langle \mathbf{z} - \mathbf{x}, \mathbf{N} \rangle + \langle \mathbf{N}, \mathbf{N} \rangle = \langle \mathbf{N}, \mathbf{N} \rangle$

that is, the first dot product becomes independent of the face.

2. **refPointMethod=1**

   Here, **r** should be rather be called reference direction instead of reference point.

   We define, with **n** again being a face's normal,

   isPlus $:= \langle \mathbf{r}, \mathbf{n} \rangle > 0$

   Example: One has a planar fault with normal **N**. Then a good reference direction would be **N**.

Application Example: Assume you have chosen a *enu* coordinate system (x=east, y=north, z=up). Your fault is in the x-z-plane with y=0 (strike-slip fault) and you set the reference point to (0,10000,0) with `refPointMethod=0`. Then, the faces with normal (0,+1,0) make up the "+"-side. In this case, all vertices of the "+"-tetrahedron lie in the half-space $y \geq 0$.

In the fault output, a strike and dip direction is defined (see `create_fault_rotationmatrix.f90`). For the normal (0,-1,0), one would obtain (-1,0,0) as strike direction (west). Recalling the definition of the slip rate, a positive slip rate indicates left-lateral motion.

Read the article Left-lateral, right-lateral, normal, reverse for more information.

CHAPTER

FOURTEEN

INITIAL CONDITIONS

Currently we provide the following initial conditions:

14.1 Zero

All quantities are set to zero. This is the standard case to work with point sources or dynamic rupture.

14.2 Planar wave

A planar wave for convergence tests. The inital values are computed such that a planar wave in a unit cube is imposed. For elastic, anisotropic and viscoelastic materials, we impose a P and an S wave travelling in opposite directions. For poroelastic materials, we impose a slow P and an S wave travelling in one direction and a fast P wave travelling in opposite direction. This scenario needs periodic boundary conditions to make sense. This is the only case where the old netcdf mesh format is prefered. After the simulation is finished the errors between the analytic solution and the numerical one are plotted in the L^1-, L^2- and L^∞-norm.

Use `cube_c` to generate the meshes for the convergence tests: https://github.com/SeisSol/SeisSol/tree/master/preprocessing/meshing/cube_c

14.3 Superimposed planar wave

Superimposed three planar waves travelling in different directions. This is especially interesting in the case of directional dependent properties such as for anisotropic materials.

14.4 Travelling wave

Impose one period of a sinusoidal planar wave as initial condition, see for example the video below. There, we impose a P wave travelling to the left. The wave speed at the top and bottom is $2m/s$ and $2.83m/s$ in the middle.

The Travelling wave can be configured in the parameter file:

- `origin = 0 0 0` describes a point on the (initially) planar wave.
- `kVec = 6.283 0 0` is the wave vector. The wavelength can be computed as $\lambda = 2\pi/\|k\|$. In this case it travels in the direction of the x-axis, the wave length is $1m$.

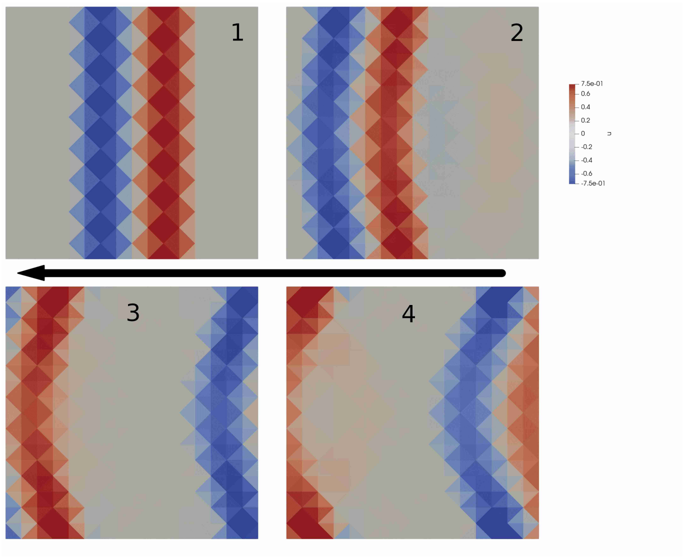


Fig. 1: Travelling wave example, wave speed is higher in the middle than at the top and bottom.

- `ampField = 2 0 0 0 0 0 0 1 0` describes the amplitudes of the different wave modes. We can impose a `P` wave and two `S` waves with different polarizations travelling either in the same direction (+) or in the opposite direction (-) of k. Note: there are three non propagating modes (`N`). The entries of k are `-P -S -S N N N +S +S +P`. In this example, we impose a P wave travelling in the opposite direction as k with relative amplitude 2 and an S wave travelling towards the same direction as k with relative amplitude 1.

14.5 Scholte

A Scholte wave to test elastic-acoustic coupling

14.6 Snell

Snells law to test elastic-acoustic coupling

14.7 Ocean

An uncoupled ocean test case for acoustic equations

14.8 How to implement a new initial condition?

New initial conditions can be easily implemented. Extend the class

```
seissol::physics::Initalfield
```

and implement the method

```
void evaluate(  double time,
                std::vector<std::array<double, 3>> const& points,
                const CellMaterialData& materialData,
                yateto::DenseTensorView<2,real,unsigned>& dofsQP ) const;
```

Here `dofsQP(i,j)` is the value of the j^{th} quantity at the `points[i]`.

CHAPTER
FIFTEEN

LOCAL TIME-STEPPING (LTS)

You can (and should!) enable local time-stepping for your simulations. Generally, this can lead to a better time to solution. The following settings are relevant:

```
&Discretization
...
ClusteredLTS = 2

LtsWiggleFactorMin = 0.51
LtsWiggleFactorStepsize = 0.01
LtsWiggleFactorEnforceMaximumDifference = 1

LtsMaxNumberOfClusters = 20
LtsAutoMergeClusters = 1
LtsAllowedRelativePerformanceLossAutoMerge = 0.01
LtsAutoMergeCostBaseline = 'bestWiggleFactor'
/
```

To enable LTS, use the setting `ClusteredLTS = 2`. This is called rate-2 LTS. Higher values are also supported, `ClusteredLTS = N` leads to rate-N LTS (where N is a positive integer). To disable LTS, use the value `ClusteredLTS = 1` For most simulations, we recommend to start with rate-2 LTS.

When using local time-stepping, SeisSol updates elements only as often as needed. To do this, SeisSol computes a time step size for each element independently. The elements are then grouped into so-called time-clusters, which are updated together. The time step size of a cluster is the minimum of the time step sizes of the elements in the cluster. Assuming rate-2 LTS (`ClusteredLTS = 2`), the first cluster contains elements of time step sizes in the interval

$$[\lambda(\Delta t)^{\text{min}}, 2\lambda(\Delta t)^{\text{min}}])$$

the second cluster elements of size

$$[2\lambda(\Delta t)^{\text{min}}, 4\lambda(\Delta t)^{\text{min}}])$$

and so on. For LTS with higher rates, the base of the exponentiation changes. The wiggle factor λ is typically one.

15.1 Maximum Difference Property

SeisSol enforces some constraints on the clustering, for example, neighboring elements are always either in the same cluster, or in a cluster which has at most a time step size difference of 2. Elements that are connected by a dynamic rupture face have to be in the same time-cluster. This is called the maximum difference property.

15.2 Wiggle factor (experimental)

This feature is only supported for rate-2 LTS (`ClusteredLTS = 2`) at the moment. The *LtsWiggleFactorMin* parameter sets the minimum allowed value for the wiggle factor $0.5 < \lambda \leq 1$. This wiggle factor can be used to reduce the overall number of time-steps in some cases and hence reduce the cost of the simulation. Even though it seems counterproductive to reduce the global time step size, it can move the boundaries of the clusters such that elements move from a cluster with a smaller time step size to clusters with a larger time step size.

SeisSol tries to find the optimal wiggle factor automatically by performing a grid search in the interval

$$\text{LtsWiggleFactorMin} \leq \lambda \leq 1$$

The step size of the grid search is controlled by the parameter *LtsWiggleFactorStepsize*. The optimal wiggle factor is the one that minimizes the cost of updates per unit time. When the setting *LtsWiggleFactorEnforceMaximumDifference* is set to one, SeisSol enforces the *Maximum Difference Property*. during the grid search. This leads to a better cost estimate, but computing this cost estimate can be costly for large scale simulations with many MPI ranks. Hence, this is a trade-off between the time required to initialize the simulation and the time required to perform the actual simulation. Typically this setting should be activated and only be deactivated when the initialization time becomes a bottleneck.

The wiggle factor was inspired by the implementation in (Breuer, Heinecke, 2022)[1]

15.3 Enforcing maximum number of clusters (experimental)

You can set a maximum number of clusters by the parameter LtsMaxNumberOfClusters. This can lead to better performance in some cases, especially on GPUs. You can set a maximum number of clusters by setting `LtsMaxNumberOfClusters=20`. SeisSol also supports the automatic merging of clusters. For this, you need to set *LtsAutoMergeClusters* to one. The parameter *LtsAllowedRelativePerformanceLossAutoMerge* controls the allowed relative performance loss when merging clusters compared to the baseline cost. There are two different types of computing the baseline cost. `LtsAutoMergeCostBaseline = 'bestWiggleFactor'` allows merging clusters without losing the performance gained from the wiggle factor. It computes the optimal wiggle factor without merging and then computes the best wiggle factor with auto-merging using the previous result as baseline. This requires two iterations of finding the best wiggle factor but because the results of the most expensive operations are cached in the implementation, it is not much slower than `LtsAutoMergeCostBaseline = 'maxWiggleFactor'`. Alternatively, you can use `LtsAutoMergeCostBaseline = 'maxWiggleFactor'`, which computes the cost without merging and wiggle factor and uses this as baseline cost. The default and recommended choice is `LtsAutoMergeCostBaseline = 'bestWiggleFactor'`.

These features should be considered experimental at this point.

[1] Breuer, A., & Heinecke, A. (2022). Next-Generation Local Time Stepping for the ADER-DG Finite Element Method. In 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 402-413). IEEE.

CHAPTER

SIXTEEN

LEFT LATERAL, RIGHT LATERAL, NORMAL, REVERSE

Assume we are given a reference point **r** and a normal **n** pointing from the “+”-side to the “-“-side of a fault segment. How does one describe the motion of the fault?

16.1 Left-lateral and right-lateral

“In a left-lateral (right-lateral) fault, an observer on one of the walls will see the other wall moving to the left (right).” [J. Pujol, Elastic Wave Propagation and Generation in Seismology]

Assume we stand on the “-“-side and look towards the “+”-side, then if the strike vector **s** points to left, we have a left-lateral motion (for a positive slip-rate). We formalize “points to the left” with:

$l := u \times (-n)$

where **u** is the unit vector which points up (e.g. (0,0,1) for *enu* or (0,0,-1) for *ned*).

In SeisSol, the strike vector is (not normalized)

$s := (-e_3) \times n$

So, e.g., for *enu* we always have a left-lateral motion, as **s** and **l** are parallel, and for *ned* we always have a right-lateral motion as **s** and **l** are anti-parallel.

16.2 Normal and reverse

“The foot wall (hanging wall) is defined as the block below (above) the fault plane. (. . .) the hanging wall moves up with respect to the foot wall and the fault is known as *reverse*. (. . .) the opposite happens and the fault is said to be *normal*.” [J. Pujol, Elastic Wave Propagation and Generation in Seismology]

In SeisSol, the dip vector is (not normalized)

$d := n \times s = n \times (-e_3 \times n) = -e_3 + n_z n$

We used Grassmann’s Identity for the last step. In particular, we observe that the dip vector **d** is independent of the reference point, as we obtain the same vector for -**n** and, as **n** is normalized and

$d_z := -e_3 + n_z^2,$

the dip vector always points in -z direction. That is, the “+”-side moves down for *enu* and the “+”-side moves up for *ned* (assuming positive slip rate).

Normal or reverse depends also on the reference point. If the reference point is inside the hanging wall (foot wall) the “+”-side corresponds to the hanging wall (foot wall).

In summary, we obtain the following table

	foot wall= +	hanging wall = +
z = up	reverse	normal
z = down	normal	reverse

or logically

$\text{isNormal} := (+ = \text{hanging wall}) \leftrightarrow (z = \text{up})$

16.3 Example

We have a 60° dipping normal fault with 90° strike (West-East direction, pointing towards the East with *enu* convention) and 0° rake. The normal of the fault plane, which points from the foot wall to the hanging wall, is given by

$n := \frac{1}{2}\begin{pmatrix}0 & -\sqrt{3} & 1\end{pmatrix}$

Such normal vector is obtained as the result of the cross product between the dip vector

$d := \frac{1}{2}\begin{pmatrix}0 & -1 & -\sqrt{3}\end{pmatrix}$

and the respective strike vector

$s := \begin{pmatrix}1 & 0 & 0\end{pmatrix}$

Hence, we set the reference point to **x** + a **N**, where a > 0 and **x** is an arbitrary point on the fault. In this case, the reference point is inside the hanging wall and we obtain a normal fault.

16.4 Warning

The rake angle describes the direction of slip. In SeisSol, the convention for the rake angle is to assume a positive rake angle refers to right-lateral strike slip. !Warning! This is opposite to the common convention in seismology which assumes positive rake angle implying left-lateral strike slip faulting. Thus, e.g., the case of Ts0>0 and Td0=0 means initial shear stress loading in rake 180° direction.

CHAPTER

SEVENTEEN

EASI

easi is a library for the Easy Initialization of model parameters in three (or less) dimensional domains. easi offers the possibility to parameterize the simulation without having to recompile SeisSol. Thanks to easi, all user can run their simulations with the same executable (no more hardcoded fortran initialization routines).

17.1 Writing easi models

easi uses configuration files written in Yet Another Markup Language (YAML). For example, a simple model with constant material parameters could be described in the following way:

```
!ConstantMap
map:
  lambda: 3.2044e+010
  mu:     3.2038e+010
  rho:    2670.
  Qp:     69.3
  Qs:     155.9
```

Complex models may be easily built, too. For example, built-in with easi come linear or polynomial models (e.g. to build depth-dependent models). Furthermore, one may sample parameters from large two- or three-dimensional uniform grids. Lastly, code may be supplied written in a C-like language, which is compiled at run-time.

Please refer to easi's documentation and to easi's examples for further details.

17.2 Invoking SeisSol

It is recommended to place easi models in the folder that also contains the parameter file. Within the parameter-file, add the parameter `MaterialFileName` to the equations block, e.g.

```
&equations
MaterialFileName = 'material.yaml'
/
```

When using *Dynamic rupture*, add the parameter `ModelFileName` to the DynamicRupture block, e.g.

```
&DynamicRupture
ModelFileName = 'fault.yaml'
/
```

17.2.1 Rheological model parameters

The following parameters need to be set by easi. The columms E, A, V, and P denote if the respective parameter is required when using an (isotropic) elastic, anisotropic (elastic), viscoelastic, and viscoplastic rheological model.

Parameter	Unit	E	A	V	P	Description
rho	$\frac{kg}{m^3}$	✓	✓	✓	✓	Density.
mu, lambda	Pa	✓		✓	✓	Lamé parameters.
c11, …, c66[1]	Pa		✓			stiffness tensor.
Qp, Qs				✓		P-wave and S-wave quality factors.
bulkFriction					✓	Bulk friction coefficient.
plastCo	Pa				✓	Plastic cohesion.
s_xx, s_yy, s_zz, s_xy, s_yz, s_xz	Pa				✓	Initial stress tensor.

17.2.2 Fault parameters (dynamic rupture)

The following parameters need to be set by easi. The column FL denotes for which friction law the respective parameter is required. Please note that there are two ways to specify the initial stress on the fault: You may either specify a stress tensor (s_xx, s_yy, s_zz, s_xy, s_yz, s_xz), which has to be given for the same cartesian coordinate system as the mesh, or you may specify a traction vector (T_n, T_s, T_d), which has to be given in a fault local coordinate system. You must not specify both.

Parameter	Unit	FL	Description
s_xx, s_yy, s_zz, s_xy, s_yz, s_xz	Pa	all (excludes initial traction)	Initial stress tensor.
T_n, T_s, T_d	Pa	all (excludes initial stress)	Initial traction in n=normal, s=strike, d=dip direction.
cohesion	Pa	2, 6	Magnitude of cohesive force.
mu_s, mu_d		2, 6	Linear slip weakening: Static and dynamic friction coefficient.
d_c	m	2, 6	Linear slip weakening: Critical distance.
forced_rupture_time	s	16	Time of forced rupture.
rs_a, rs_srW, RS_sl0		101, 103	Rate-and-state friction parameter.
nuc_{xx, yy, zz, xy, yz, xz} or Tnuc_{n, s, d}	Pa	2, 3, 4, 103	Nucleation stress or tractions.

[1] See *Anisotropic* for more details.

17.3 Debugging easi script

Most easi components return easy to track error, for example
`test.yaml: yaml-cpp: error at line 6, column 9: illegal map value`
Yet impljit function maps are more complex to debug. The following example:
`27.1: syntax error, unexpected '}', expecting ;`
indicates that an error occurred in the 27th line of the function, but does not indicate which file and which function. Hopefully this will be improved in the future.

17.3.1 An example illustrating some subtleties of easi error logs

Let suppose that we try to retrieve s_zz located at (x,y,z)=(0,0,0) in group 1 from the following easi file:

```
[s_zz,s_yy,s_yz,s_xx,s_xz,s_xy,d_c,mu_s]: !AffineMap
  matrix:
    xf: [0.4054811 , -0.91410343,  0.    ]
    yf: [-0.62424723, -0.2769057 ,  0.73050574]
    zf: [-0.6677578 , -0.29620627, -0.68290656]
  translation:
    xf: 348441.377459
    yf: 4760209.93637
    zf: 0.0
  components: !ASAGI
          file: norciax_210fault_nncia.nc
          parameters: [s_zz,s_yy,s_yz,s_xx,s_xz,s_xy,d_c,mu_s]
          var: data
          interpolation: nearest
```

and get the following error log:

```
terminate called after throwing an instance of 'std::runtime_error'
  what():  fault2.yaml@2: Could not find model for point [ 348441 4.76021e+06 0 ] in␣
↪group 1.
```

How to interpret this error log? The component at Line 2 is throwing the error (the AffineMap). The AffineMap component is complaining that its output point is not accepted by any of its child components. In this case, the point is outside the bounds of the ASAGI file.

Note that in the slightly different example below, without the AffineMap, easi will not verify that the point is outside the bounds of ASAGI file:

```
[s_zz,s_yy,s_yz,s_xx,s_xz,s_xy,d_c,mu_s]: !ASAGI
          file: norciax_210fault_nncia.nc
          parameters: [s_zz,s_yy,s_yz,s_xx,s_xz,s_xy,d_c,mu_s]
          var: data
          interpolation: nearest
```

In fact, in this case, ASAGI is directly queried and easi, therefore, does no verify that the point queried in inside the bounds of the ASAGI file. If the point is out of bounds, ASAGI will pick the value of the nearest grid point and issue a warning:

```
Thu Jan 09 14:32:22, Warn:  ASAGI: Coordinate in dimension 2  is out of range. Fixing.
```

CHAPTER EIGHTEEN

FAULT TAGGING

In SeisSol, boundary conditions are tagged as:
0: regular
1: free surface
2: free surface + gravity (water surface)
3 or n>64: dynamic rupture
5: absorbing
6: periodic

Dynamic rupture can therefore be tagged using a range of possible tags. This allows initializing fault parameters segment-wise easily. For example, if we have 2 segments, and we want them to have different dynamic friction, we can tag them with 3 and 65 and then use:

```
[mu_d]: !Any
  components:
    - !GroupFilter
      groups: 3
      components: !ConstantMap
        map:
          mu_d:    0.3
    - !GroupFilter
      groups: 65
      components: !ConstantMap
        map:
          mu_d:    0.4
```

Currently, the only way to tag fault faces other tags than 3 with SimModeler is to use the *–xml* option of pumgen. For example, to tag face 2 as 3 and face 8 and 9 as 65, we would use:

```
<boundaryCondition tag="3">2</boundaryCondition>
<boundaryCondition tag="65">8,9</boundaryCondition>
```

Then pumgen is run using the xml option:

```
pumgen -s simmodsuite -l SimModelerLib.lic --xml MeshandAnalysisAttributes.xml prefix.
↪smd output_prefix
```

Note that `<boundaryCondition tag="3">` is equivalent to `<dynamicRupture>`. Therefore, if you want to tag face 2 as 3, you can use either:

```
<boundaryCondition tag="3">2</boundaryCondition>
```

or

```
<dynamicRupture>2</dynamicRupture>
```

Note also that if a face is tagged twice, only the first tag will be considered.

18.1 Using more than 189 dynamic rupture tags

To handle more than 189 dynamic rupture tags (i.e. more than 255 boundary condition types), you will need to adjust the boundary format when building your mesh in PUMgen.

That is, add in PUMgen the option `--boundarytype=int64` when building your mesh. No modification in SeisSol is needed, as it tries to infer the boundary format from the shape of the boundary array automatically. However, to prevent mistakes with reading the format, we nevertheless recommend specifying the boundary format explicitly. (it *is* possible to confuse the boundary format, but only in some esoteric edge cases)

To do that, it suffices to specify `pumlboundaryformat = $option` in the `&meshnml` section of your SeisSol parameter file, where `$option` is one of the following:

- `'auto'`: SeisSol will try to infer the boundary format automatically. This is the default option. That is, if an attribute `boundary-format` is present, its value will be used (`i32x4 == 0`, `i32 == 1`, `i64 == 2`). Otherwise, the type will be inferred from the rank of the boundary storage array: if it is two-dimensional, then we choose `i32x4`; if it is rank 1, then we choose `i32` (note that `i64` will not be chosen automatically in this case).
- `'i32'`: 8 bits per boundary face. That is, 189 dynamic rupture tags are possible (255 boundary condition types). It is (usually) stored as a one-dimensional 32-bit integer array (one entry per cell) in the Hdf5/binary file.
- `'i64'`: 16 bits per boundary face. That is, 65469 dynamic rupture tags (65535 boundary condition types). It is stored as a one-dimensional 64-bit integer array (one entry per cell) in the Hdf5/binary file.
- `'i32x4'`: 32 bits per boundary face. In short, you will have $2^{32} - 65$ different dynamic rupture tags available. The data is stored as a two-dimensional array (four entries per cell, one for each face) of 32-bit integers.

To try inferring which boundary format you have built your mesh for (in case it is not indicated by an attribute), you can use `h5dump -H <yourmeshfile>.puml.h5`, and look at the datatype and the shape of the `boundary` dataset. Note however, that some libraries may store `i32` boundaries in a 64-bit integer.

CHAPTER

NINETEEN

ENVIRONMENT VARIABLES

SeisSol can be tuned with several environment variables.

19.1 Communication Thread

By default, any SeisSol run with more than one MPI rank will use a communication thread to advance the MPI progress engine. For that, you will need to leave at least one thread vacant in your OpenMP thread placing map, cf. the SuperMUC-NG example below.

If you do not want to use a communication thread, you may set *SEISSOL_COMMTHREAD=0*; then SeisSol polls on the progress from time to time.

19.2 Load Balancing

When running with multiple ranks, SeisSol will estimate the performance of a node, to enable better load balancing for it. For that, it runs the so-called "Mini SeisSol" benchmark. As its name already hints at, it simulates a small test workload on each node; thus estimating the performance of all nodes relative to each other. The number of elements per node assigned during the partitioning will be resized according to these values.

As a result, the partitioning of runs may become non-deterministic, and the initialization procedure may take a little longer; especially when running only on a single node with multiple ranks. To disable it, set *SEISSOL_MINISEISSOL=0*.

19.3 Persistent MPI Operations

Since SeisSol has a static communication pattern (in the sense of: per iteration, we issue the same MPI transfer requests), we may use persistent MPI communication—it may reduce the communication latency.

You may enable persistent communication by setting *SEISSOL_MPI_PERSISTENT=1*, and explicitly disable it with *SEISSOL_MPI_PERSISTENT=0*. Right now, it is disabled by default.

19.4 Output

The wave field and fault output use the XdmfWriter. Tuning variables for the XdmfWriter are listed in the corresponding wiki.

19.4.1 Asynchronous Output

In addition to the variables in SeisSol, the ASYNC library provides some tuning variables listed in the wiki.

19.4.2 Checkpointing

Some environment variables related to checkpointing are described in the *Checkpointing section*.

19.5 Optimal environment variables on SuperMUC-NG

On SuperMUC-NG, we recommend using SeisSol with async output in thread mode. Also, we recommend using hyperthreading capabilities (that is using 96 CPUs instead of 48. 2 threads out of 96 are used as communication threads). Here are some proposed environment variables, to be added prior to invoking SeisSol in your batch file:

```
export MP_SINGLE_THREAD=no
unset KMP_AFFINITY
export OMP_NUM_THREADS=94
export OMP_PLACES="cores(47)"

export XDMFWRITER_ALIGNMENT=8388608
export XDMFWRITER_BLOCK_SIZE=8388608
export SC_CHECKPOINT_ALIGNMENT=8388608
export SEISSOL_CHECKPOINT_ALIGNMENT=8388608
export SEISSOL_CHECKPOINT_DIRECT=1

export ASYNC_MODE=THREAD
export ASYNC_BUFFER_ALIGNMENT=8388608
```

A complete batch script for SuperMUC-NG can be found in the chapter about *SuperMUC-NG*.

In previous versions of SeisSol, you had to explicitly compile the software with *-DCOMMTHREAD=ON*. That is not necessary anymore, as any configuration with more than one MPI rank uses the communication thread by default.

SEISSOL WITH GPUS

20.1 General

The current GPU version of SeisSol targets the latest NVidia Graphics cards. Therefore, you need to have at least **CUDA 10** installed in your environment. Moreover, make sure that your installed MPI implementation is **CUDA-Aware**. Here is a list of known CUDA-Aware MPI vendors:

- OpenMPI
- MVAPICH2
- CRAY MPI
- IBM Platform MPI
- SGI MPI

Please, referer to the corresponding documentation if you need to install CUDA-Aware MPI manually. We also encourage you to bind your MPI with UCX communication layer if you need to manually configure CUDA-Aware MPI for a cluster or your local server with an open-source MPI implementation e.g., OpenMPI.

GPU version of SeisSol follows *single rank/single GPU* strategy. Therefore, if you want to run SeisSol on **M** nodes where each node is equipped with **N** GPUs then make sure that you launch SeisSol with **M x N** MPI processes.

To achieve the most efficient CPU-to-GPU communication and vice versa you have to pin your MPI processes to CPU cores which are the closest to the target GPUs. This problem is also known as GPU affinity. Latest versions of workload managers (e.g., SLURM) are aware of this problem and try to provide an automatic, GPU-aware process pinning. Consider the following SLURM options:

- *–ntasks-per-gpu*
- *–gpu-bind*

You can also enforce good GPU affinity with rankfiles if your GPU cluster or local server does not use a workload manager but is equipped with multiple GPUs per node.

Some systems have complex numbering of processing units and/or NUMA domains. Sometime it is very difficult to achieve desirable pinning of the communication and/or output-writing threads with HPC resource managers like SLURM. Therefore, SeisSol provides *SEISSOL_FREE_CPUS_MASK* environment variable which helps to describe locations of the auxiliary threads per local MPI process. The variable accepts a comma separated list of elements where an element can be either 1) an integer, or 2) a range of integers defined as *[start, end]* or 3) a comma separated list of integers surrounded by the curly brackets. The *i*-th list element describes the free cpus locations for the *i*-th local MPI process.

```
# SEISSOL_FREE_CPUS_MASK="(int | range: <int-int> | list: {int,+})+"
# Examples,
```

(continues on next page)

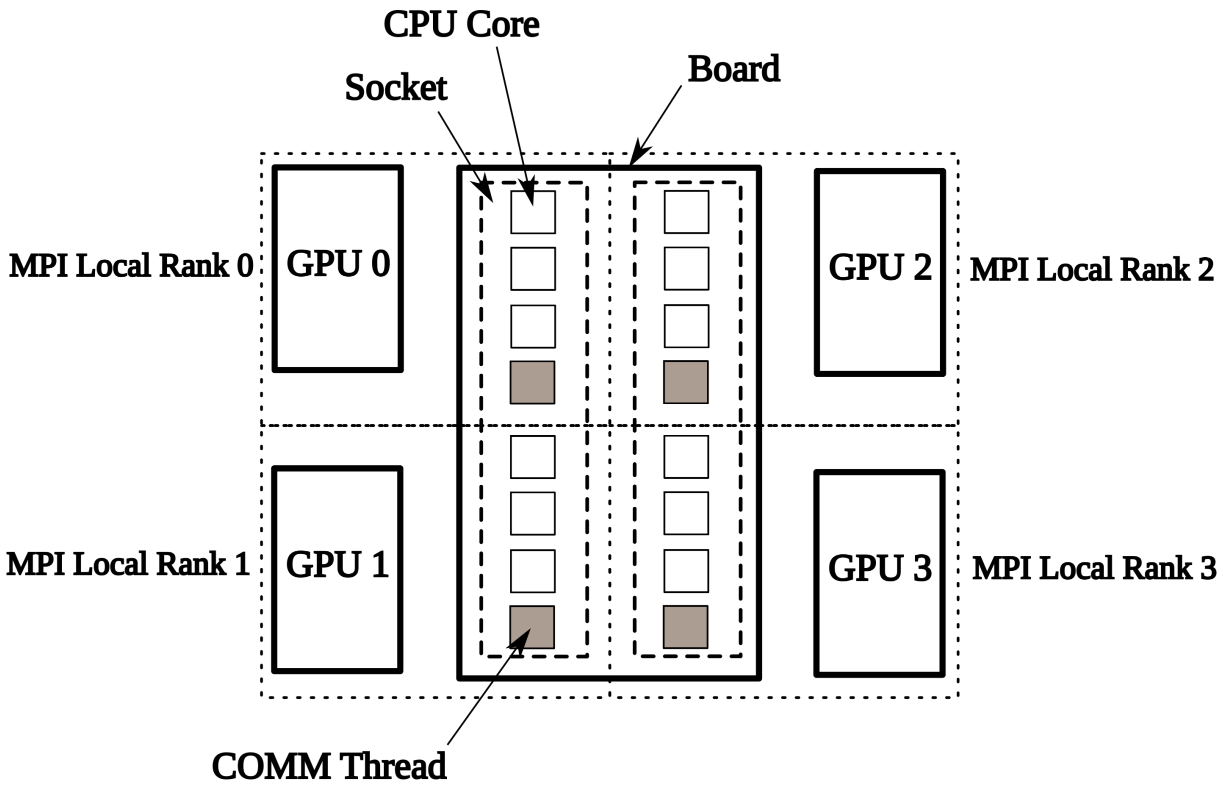


Fig. 1: Correct process pinning of 4 MPI processes where each process controls 3 OpenMP threads and one communication thread.

(continued from previous page)

```
export SEISSOL_FREE_CPUS_MASK="24,28,32,36"
export SEISSOL_FREE_CPUS_MASK="24-27,28-31,32-35,36-39"
export SEISSOL_FREE_CPUS_MASK="{24,25},{28,29},{32,33},{36,37}"

# Note, it is allowed to mix the formats of list elements. For example,
export SEISSOL_FREE_CPUS_MASK="24,28-31,{28,29},36"
```

20.2 Supported SeisSol Features

- elastic wave propagation model
- kinematic point sources
- dynamic rupture: linear slip weakening, slow and fast velocity weakening friction laws
- off-fault plasticity model

20.3 Compilation

To start off, make sure that you already have **GemmForge** installed on your system. If you don't have then follow this *link*.

After that, get the latest version of SeisSol

```
git clone --recurse-submodules https://github.com/SeisSol/SeisSol.git seissol-gpu
```

Compile SeisSol with (e.g.)

```
mkdir -p seissol-gpu/build && cd seissol-gpu/build
cmake -DDEVICE_BACKEND=cuda -DDEVICE_ARCH=sm_70 -DHOST_ARCH=skx \
-DCMAKE_BUILD_TYPE=Release -DPRECISION=double ..
make -j
```

The following two CMake options can be useful to improve performance:

- *USE_GRAPH_CAPTURING*: enables CUDA/HIP graphs. These are used to speed up the kernel execution for elastic or anisotropic equations.
- *PREMULTIPLY_FLUX*: enables the pre-multiplying of flux matrices (it was disabled for CPUs to free up cache space). This usually results in a speedup for AMD and Nvidia GPUs. By default, it is switched on when compiling for an AMD or Nvidia GPU and switched off in all other cases.

20.4 Execution

The launching process of the GPU version of SeisSol is similar as the one of the CPU version.

```
mpirun -n <M x N> ./SeisSol_dsm70_cuda_* ./parameters.par
```

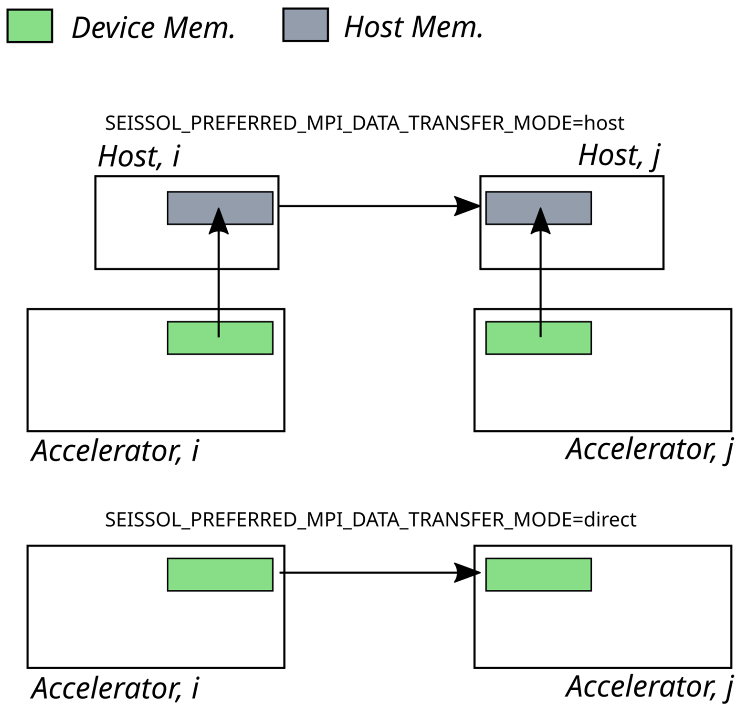
It is important to know that the GPU version of SeisSol by default allocates 1GB of GPU memory at the beginning of SeisSol execution. It is necessary for fast allocation/deallocation of GPU memory needed for holding temporary data. The default value can be changed by setting a necessary one to **DEVICE_STACK_MEM_SIZE** environment variable. For example, the following will force SeisSol to allocate 1.5GB of stack GPU memory for temporary data:

```
export DEVICE_STACK_MEM_SIZE=1.5
mpirun -n <M x N> ./SeisSol_dsm70_cuda_* ./parameters.par
```

The following device-specific environment variables are supported right now:

- SEISSOL_PREFERRED_MPI_DATA_TRANSFER_MODE
- SEISSOL_SERIAL_NODE_DEVICE_INIT

Currently, SeisSol allocates MPI buffers using the global memory type. Some MPI implementations are not GPU-aware and do not support direct point-to-point communication on device buffers. SeisSol provides the *SEISSOL_PREFERRED_MPI_DATA_TRANSFER_MODE* environment variable that can be used to select the memory type for the buffers. The *host* value means that the data will be copied to/from the host memory before/after each *MPI_Isend* / *MPI_Irecv*. The default value is *direct* which means that the communication goes over the global memory and thus does not involve explicit data copies.


Device Mem.
Host Mem.
SEISSOL_PREFERRED_MPI_DATA_TRANSFER_MODE=host
Host, i
Host, j
Accelerator, i
Accelerator, j
SEISSOL_PREFERRED_MPI_DATA_TRANSFER_MODE=direct
Accelerator, i
Accelerator, j

The variable "SEISSOL_SERIAL_NODE_DEVICE_INIT" exists to mitigate some possible execution bugs with regard to AMD GPU drivers. It is disabled by default and scheduled for removal long-term. To enable it, set "SEISSOL_SERIAL_NODE_DEVICE_INIT=1". To explicitly disable it, write "SEISSOL_SERIAL_NODE_DEVICE_INIT=0".

CHAPTER

TWENTYONE

MEMORY REQUIREMENTS

21.1 General

Memory requirements per node are difficult to predict because the elements are not distributed equally when Local Time-Stepping (LTS) is enabled. However, an adequate rule of thumb to estimate memory requirements is to multiply the number of elements with the number of degrees of freedom per element and then multiply this number with a factor of 10. Therefore, a run using the viscoelastic wave equation with 100 million elements of order 5 requires about 1.4 terabytes of memory.

21.2 LTS weight balancing strategies

By default, SeisSol uses a single-constraint node-weight mesh partitioning when LTS is enabled. The node-weights are evaluated as follows:

$$w_k = c_k R^{L-l_k}$$

where w_k - node-weight of element k; c_k - cost of element k, which depends whether 1) a cell is regular, 2) has n dynamic rupture faces and 3) has m free surface with gravity faces; $n, m \in [0, 4]$; R - LTS cluster update ratio; L - total number of LTS clusters; $l_k \in [0, L)$ - time cluster number, which element k belongs to.

Because of the form of the node-weight function, we call this weight balancing strategy as *exponential*. The strategy reflects a computational intensity of each element, taking element-wise update frequencies into account, and thus it aims to balance computations between MPI ranks. However, the strategy may lead to memory imbalances, which can be a problem for systems with a limited amount of memory, e.g. GPUs.

To address this issue, two other multi-constraint mesh partitioning strategies are available, namely: 1) *exponential-balanced* and 2) *encoded*.

exponential-balanced strategy can be described as follows:

$$w_k \in \mathbb{R}^2 \mid w_k = \begin{bmatrix} c_k R^{L-l_k} \\ 1 \end{bmatrix}$$

It features an additional constrain (the second component of w_k) that aims at equalizing the number of elements in each rank.

The *encoded* one:

$$w_k \in \mathbb{R}^L \mid w_k^i = \begin{cases} c_k, & \text{if } i = l_k \\ 0, & \text{otherwise} \end{cases} \text{and } i \in [0, L)$$

This strategy aims at distributing time clusters equally between all ranks without taking into account element-wise update frequencies. The strategy may be beneficial while working with LTS ratio 3 or 4.

A user can specify a particular partitioning strategy in *parameters.par* file:

```
&Discretization
...
ClusteredLTS = 2
LtsWeightTypeId = 1  ! 0=exponential, 1=exponential-balanced, 2=encoded
/
```

Note, the default (*exponential*) strategy is going to be used if *ClusteredLTS* is ≥ 2 and *LtsWeightTypeId* is not specified.

CHAPTER

TWENTYTWO

ACCESSING GITHUB BEHIND A FIREWALL

Some HPC clusters (e.g. SuperMUC-NG) restricts access to outside sources and thus does not allow connections to https servers. Nevertheless, GitHub can be used if remote port forwarding is correctly set. Here, we described the procedure to set up such port forwarding.

1. On your local machine, add to your ~/.ssh/config the following lines:

```
Host supermucNG
   Hostname skx.supermuc.lrz.de
   User <Your Login>
   RemoteForward ddddd github.com:22
```

where ddddd is an arbitrary 5-digital port number, smaller than 65535.

2. Use the following command to login onto the HPC cluster:

```
ssh supermucNG
```

Add the following lines to your ~/.ssh/config (on the HPC cluster):

```
Host github.com
   HostName localhost
   User git
   Port ddddd
```

With ddddd the same port number as before.

3. Create SSH key by typing (use a non-empty passphrase, not too long as you will need to type it often)

```
ssh-keygen -t rsa
```

4. Go to https://github.com/settings/ssh, add a new SSH key, and paste the public SSH key you just created (the content of ~/.ssh/id_rsa.pub on the HPC cluster).

5. To allow cloning using ssh on SuperMUC-NG with the https address of git repository, add to `~/.gitconfig`:

```
[url "git@github.com:"]
    insteadOf = https://github.com/
```

6. (Optionnally) To avoid having to Enter your passphrase many times during the session, you can execute on the HPC cluster:

```
eval `ssh-agent -s`
ssh-add ~/.ssh/id_rsa
```

You should now be able to clone any GitHub repository, e.g. SeisSol including the submodules using:

```
git clone --recursive https://github.com/SeisSol/SeisSol.git
```

If it works, you will see several lines, for example:

```
Cloning into 'SeisSol'...
remote: Enumerating objects: 25806, done.
remote: Counting objects: 100% (4435/4435), done.
remote: Compressing objects: 100% (1820/1820), done.
remote: Total 25806 (delta 2972), reused 3710 (delta 2551), pack-reused 21371
Receiving objects: 100% (25806/25806), 110.50 MiB | 9.79 MiB/s, done.
Resolving deltas: 100% (19382/19382), done.
```

CHAPTER

TWENTYTHREE

ACCESSING PYPI BEHIND A FIREWALL

Many post-processing scripts of SeisSol require Python dependencies. We describe how to use pip on a HPC cluster with restricts access to outside sources in the following.

1. On your local machine in ~/.ssh/config add the following *RemoteForward* line:

```
Host supermucNG
    ...
    RemoteForward ddddd localhost:8899
```

where ddddd is an arbitrary port number with 5 digits. (This number should be different from port number used in other RemoteForward entries.)

2. Install proxy.py on your local machine.

```
pip install --upgrade --user proxy.py
```

3. Start proxy.py on your local machine. (And keep it running.)

```
~/.local/bin/proxy --port 8899
```

4. Login to the HPC cluster (e.g. *ssh supermucNG*). Check that you do not get: *Warning: remote port forwarding failed for listen port ddddd*. In this case you would need to change ddddd to a different port. Note that the problem might also be you have already an opened connection to the HPC cluster.

Once connected to the HPC cluster, pip can be used with

```
pip install <package name> --user --proxy http://localhost:ddddd/
```

where ddddd is your arbitrary port number.

CHAPTER

TWENTYFOUR

SUPERMUC-NG

24.1 Setting up GitHub on SuperMuc-NG

see *Accessing github behind a firewall*.

24.2 Building SeisSol

Load modules (including seissol-env). Add these lines to .bashrc:

```
##### module load for SeisSol
module load gcc
module load cmake/3.21.4
module load python/3.8.11-extended
module load numactl/2.0.14-intel21
#To use dependencies preinstalled with spack
module use /hppfs/work/pn49ha/ru76tuj2/modules/linux-sles15-skylake_avx512/
# you need to have access to project pn49ha
module load seissol-env/develop-intel21-impi-x2b
export CC=mpicc
export CXX=mpiCC
```

Install pspamm (see *Accessing PyPI behind a firewall* for the proxy):

```
pip3 install git+https://github.com/SeisSol/PSpaMM.git --no-build-isolation --user --
↪proxy http://localhost:DDDDD
```

(`--no-build-isolation` is used to circumvent the problem described in https://github.com/SeisSol/PSpaMM/issues/13)

Clone SeisSol including the submodules using

```
git clone --recursive https://github.com/SeisSol/SeisSol.git
```

Install SeisSol with cmake, e.g. with (more options with ccmake)

```
cd SeisSol
mkdir build-release && cd build-release
cmake -DNUMA_AWARE_PINNING=ON -DASAGI=ON -DCMAKE_BUILD_TYPE=Release -DHOST_ARCH=skx -
↪DPRECISION=double -DORDER=4 ..
make -j 48
```

24.3 Running SeisSol

This is an example job submission script for SeisSol on SuperMUC-NG. For your applications, change `#SBATCH --nodes=` to the number of nodes you want to run on. A rule of thumb for optimal performance is to distribute your jobs to 1 node per 100k elements. This rule of thumb does not account for potentially shorter queue times, for example when using the test queue or when asking for a large amount of nodes.

```
#!/bin/bash
# Job Name and Files (also --job-name)
#SBATCH -J <job name>

#Output and error (also --output, --error):
#SBATCH -o ./%j.%x.out
#SBATCH -e ./%j.%x.err

#Initial working directory:
#SBATCH --chdir=<work directory>

#Notification and type
#SBATCH --mail-type=END
#SBATCH --mail-user=<your email address>

#Setup of execution environment
#SBATCH --export=ALL
#SBATCH --account=<project id>
#SBATCH --no-requeue

#Number of nodes and MPI tasks per node:
#SBATCH --partition=general
#SBATCH --nodes=40
#SBATCH --time=03:00:00

#SBATCH --ntasks-per-node=1
#EAR may impact code performance
#SBATCH --ear=off

module load slurm_setup

#Run the program:
export MP_SINGLE_THREAD=no
unset KMP_AFFINITY
export OMP_NUM_THREADS=94
export OMP_PLACES="cores(47)"
#Prevents errors such as experience in Issue #691
export I_MPI_SHM_HEAP_VSIZE=32768

export XDMFWRITER_ALIGNMENT=8388608
export XDMFWRITER_BLOCK_SIZE=8388608
export SC_CHECKPOINT_ALIGNMENT=8388608

export SEISSOL_CHECKPOINT_ALIGNMENT=8388608
export SEISSOL_CHECKPOINT_DIRECT=1
export ASYNC_MODE=THREAD
```

(continues on next page)

(continued from previous page)

```
export ASYNC_BUFFER_ALIGNMENT=8388608
source /etc/profile.d/modules.sh

echo 'num_nodes:' $SLURM_JOB_NUM_NODES 'ntasks:' $SLURM_NTASKS
ulimit -Ss 2097152
srun SeisSol_Release_sskx_4_elastic parameters.par
```

24.4 Accessing PyPI

Many post-processing scripts of SeisSol require Python dependencies. We describe how to use pip on SuperMUC at see *Accessing PyPI behind a firewall*.

24.5 Using the Sanitizer

Note that to use the Sanitizer (https://en.wikipedia.org/wiki/AddressSanitizer), SeisSol needs to be compiled with gcc (or clang but the "static-libasan" argument does not work then). For that modules and compiler need to be switched:

```
module switch seissol-env seissol-env/develop-gcc11

export CC=gcc
export CXX=gxx
```

Then cmake on a new build folder. To enable sanitizer, add `-DADDRESS_SANITIZER_DEBUG=ON` to the argument list of cmake, and change the `CMAKE_BUILD_TYPE` to `RelWithDebInfo` or `Debug`.

24.6 Compiling seissol-env spack package

For reference, to compile seissol-env on SuperMUC-NG, follow the procedure below:

```
# load spack
module load user_spack
# clone seissol-spack-aid and add the repository
# we use a supermuc specific branch as supermuc spack is too old (0.17.1) for the main␣
↪branch
git clone --branch supermuc_NG https://github.com/SeisSol/seissol-spack-aid.git
cd seissol-spack-aid
spack repo add ./spack
# locate externally build pkg-config
spack external find pkg-config
# install all dependencies of seissol.
# We specify the intel and intel-mpi version matching preinstalled version on supermuc-ng
# These can be found with:
# >spack find intel-mpi
# >spack compiler list
spack install seissol-env %intel ^intel-mpi
# or
spack install seissol-env %gcc ^intel-mpi
```

(continues on next page)

(continued from previous page)

```
# now create a module:
spack module tcl refresh seissol-env@develop%intel
#to access the module at start up, add to your ~/.bashrc
module use $HOME/spack/modules/x86_avx512/linux-sles15-skylake_avx512/
# change this path to your_custom_path_2_modules if you update ~/.spack/modules.yaml
```

Custom install directory for packages and modules can be set by changing `~/.spack/config.yaml`

```
config:
  install_tree: path_2_packages
```

and `~/.spack/modules.yaml`:

```
modules:
  default:
    roots:
     tcl: your_custom_path_2_modules
```

This can be useful to share packages with other users of a SuperMUC project.

The seissol-env compilation can also be reduced by adding the python module to `~/.spack/packages.yaml`:

```
packages:
  python:
    externals:
    - spec: python@3.8.11
      buildable: False
      modules:
       - python/3.8.11-extended
```

24.7 Compiling the seissol spack package

The seissol package is similar to the seissol-env package (it gathers all dependencies of seissol), but also compiles a specific version of seissol itself. To compile the seissol spack package on SuperMUC-NG, follow the procedure below.

```
# load spack
module purge
module load user_spack/23.1.0
module load intel intel-mkl intel-mpi python/3.10.10-extended

# install a few python modules (change DDDDD to the value used after RemoteForward in ~/.
↪ssh/config)
python3.10 -m pip install --upgrade pip --user --proxy http://localhost:DDDDD
pip3.10 install --upgrade setuptools numpy wheel packaging --user --proxy http://
↪localhost:DDDDD
pip3.10 install git+https://github.com/SeisSol/PSpaMM.git --no-build-isolation --user --
↪proxy http://localhost:DDDDD

# clone seissol-spack-aid and add the repository
# we use a supermuc specific branch as supermuc spack is not fully up to date
git clone --branch NG https://github.com/SeisSol/seissol-spack-aid.git
```

(continues on next page)

(continued from previous page)

```
cd seissol-spack-aid
spack repo add ./spack
# install a specific version of seissol, and enable python binding enabled for easi
spack install -j 40 seissol@master convergence_order=4 dr_quad_rule=dunavant␣
↪equations=elastic precision=single %intel  ^easi +python

# now create a module:
spack module tcl refresh $(spack find -d --format "{name}{/hash:5}" seissol)

#to access the module at start up, add to your ~/.bashrc
module use $HOME/spack/modules/x86_avx512/linux-sles15-skylake_avx512/
# change this path to your_custom_path_2_modules if you update ~/.spack/modules.yaml
```

Custom install directory for packages and modules can be set by changing `~/.spack/config.yaml`:

```
config:
  install_tree:
    root: path_2_packages
```

and `~/.spack/modules.yaml`:

```
modules:
  default:
    roots:
     tcl: your_custom_path_2_modules
    tcl:
      all:
        suffixes:
          domain_dimension=2: d2
          domain_dimension=3: d3
          polynomial_degree=1: p1
          polynomial_degree=2: p2
          polynomial_degree=3: p3
          polynomial_degree=4: p4
          polynomial_degree=5: p5
          polynomial_degree=6: p6
          convergence_order=3: o3
          convergence_order=4: o4
          convergence_order=5: o5
          convergence_order=6: o6
          equations=elastic: elas
          equations=viscoelastic2: visco
          dr_quad_rule=stroud: stroud
          dr_quad_rule=dunavant: dunav
          precision=single: single
          precision=double: double
          cuda: cuda
          debug: debug
    enable:
    - tcl
```

This can be useful to share packages with other users of a SuperMUC project.

The number of dependencies to be compiled can be reduced by adding the python module to `~/.spack/packages.`

yaml:

```
packages:
  python:
    externals:
    - spec: python@3.10.10
      buildable: False
      modules:
       - python/3.10.10-extended
```

CHAPTER TWENTYFIVE

SHAHEEN

Add these lines to ~/.bashrc:

```
##### module load for SeisSol
module unload PrgEnv-cray
module load PrgEnv-gnu
module load python
module load cmake
module unload intel
# to use preinstall seissol-env module
module use /project/k1587/ulrich/spack/share/spack/modules/cray-cnl7-ivybridge/
module load seissol-env-develop-gcc-11.2.0-rckyrcj
export CC=cc
export CXX=CC
export FC=ftn
```

This will load a preinstalled seissol-env module.

Alternatively (and for reference), to compile seissol-env on shaheen, follow the procedure below:

```
git clone --depth 1 --branch v0.18.1 https://github.com/spack/spack.git
cd spack
echo "export SPACK_ROOT=$PWD" >> $HOME/.bashrc
echo "export PATH=\$SPACK_ROOT/bin:\$PATH" >> $HOME/.bashrc
# clone seissol-spack-aid and add the repository
git clone --branch supermuc_NG https://github.com/SeisSol/seissol-spack-aid.git
cd seissol-spack-aid
spack repo add ./spack
spack compiler find
```

Then update ~/.spack/packages.yaml as follow:

```
packages:
  python:
    externals:
    - spec: python@3.10.1
      buildable: False
      modules:
       - python/3.10.1-cdl
  cmake:
    buildable: false
    externals:
```

(continues on next page)

(continued from previous page)

```
    - spec: cmake@3.22.1
      modules:
       - cmake/3.22.1
  mpich:
    buildable: false
    externals:
    - spec: mpich@7.7.18
      modules:
      - cray-mpich/7.7.18
    - spec: mpich@7.7.20
      modules:
      - cray-mpich/7.7.20
  all:
    providers:
      mpi: [mpich]
```

Finally, install seissol-env with

```
spack install -j 8 seissol-env %gcc@11.2.0 ^mpich
```

and create a module with:

```
spack module tcl refresh seissol-env@develop%%gcc@11.2.0
```

To access the module at start up, add to your `~/.bashrc`:

```
module use $SPACK_ROOT/share/spack/modules/cray-cnl7-ivybridge/
```

Finally, install SeisSol with cmake, as usual, with `-DHOST_ARCH=hsw`.

Here is an example job submission script for SeisSol on Shaheen (to be launched from the `/scratch/` folder):

```
#!/bin/bash
# Job Name and Files (also --job-name)

#SBATCH -J <job name>
#Output and error (also --output, --error):
#SBATCH -o ./%j.%x.out
#SBATCH -e ./%j.%x.err

#Initial working directory:
#SBATCH --chdir=<work directory>

#Notification and type
#SBATCH --mail-type=END
#SBATCH --mail-user=<your email address>

# Wall clock limit:
#SBATCH --time=00:30:00
#SBATCH --no-requeue

#Setup of execution environment
#SBATCH --export=ALL
```

(continues on next page)

(continued from previous page)

```
#SBATCH --account=<project id>
#SBATCH --partition=debug

#Number of nodes and MPI tasks per node:
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1

export MP_SINGLE_THREAD=no
unset KMP_AFFINITY
export OMP_NUM_THREADS=31
# you could also consider OMP_NUM_THREADS=62 for high order
export OMP_PLACES="cores(31)"
#Prevents errors such as experience in Issue #691
export I_MPI_SHM_HEAP_VSIZE=8192

export XDMFWRITER_ALIGNMENT=8388608
export XDMFWRITER_BLOCK_SIZE=8388608
export SC_CHECKPOINT_ALIGNMENT=8388608

export SEISSOL_CHECKPOINT_ALIGNMENT=8388608
export SEISSOL_CHECKPOINT_DIRECT=1
export ASYNC_MODE=THREAD
export ASYNC_BUFFER_ALIGNMENT=8388608

export MPICH_MAX_THREAD_SAFETY=multiple
# update to output folder
lfs setstripe -c 32 output
srun path_2_SeisSol_Release_dhsw_4_elastic parameters.par
```

CHAPTER

TWENTYSIX

FRONTERA

Add these lines to ~/.bashrc:

```
module switch python3 python3/3.9.2
module use /work2/09160/ulrich/frontera/spack/share/spack/modules/linux-centos7-
↪cascadelake
module load seissol-env
export CC=mpiicc
export CXX=mpiicpc
```

This will load a preinstalled seissol-env module.

Alternatively (and for reference), to compile seissol-env on Frontera, follow the procedure below:

```
git clone --depth 1 --branch v0.21.0 https://github.com/spack/spack.git
cd spack
echo "export SPACK_ROOT=$PWD" >> $HOME/.bashrc
echo "export PATH=\$SPACK_ROOT/bin:\$PATH" >> $HOME/.bashrc
# clone seissol-spack-aid and add the repository
git clone https://github.com/SeisSol/seissol-spack-aid.git
spack repo add seissol-spack-aid/spack
spack compiler find
```

Following the workaround proposed in https://github.com/spack/spack/issues/10308, precise the module of the intel compilers in ~/.spack/linux/compilers.yaml by changing `modules:  []` to `modules:  ['intel/19.1.1']`.

Then, update ~/.spack/packages.yaml as follow:

```
packages:
  python:
    externals:
    - spec: python@3.9.2
      buildable: false
      prefix: /opt/apps/intel19/python3/3.9.2
      modules:
      - python3/3.9.2
  intel-mpi:
    buildable: false
    externals:
    - spec: intel-mpi@2019.0.9
      modules:
      - impi/19.0.9
  all:
```

(continues on next page)

(continued from previous page)

```
    providers:
      mpi: [intel-mpi]
```

Additional already installed modules can be discovered and added with:

```
spack external find
```

Finally, install seissol-env with

```
spack install -j 16 seissol-env %intel ^intel-mpi
```

We then create the required modules with:

```
spack module tcl refresh $(spack find -d --format "{name}{/hash:5}" seissol-env)
```

To access the module at start up, add to your `~/.bashrc`:

```
module use $SPACK_ROOT/share/spack/modules/linux-centos7-cascadelake/
```

Finally, install SeisSol with cmake, as usual, with `-DHOST_ARCH=skx`.

For large runs, it is recommanded to have executable, dynamic libraries, setup and outputs on scratch. That is, $WORK is not built for large, intensive IOs, and loading the shared libraries from 8000+ nodes at once is quite intensive. This could potentially break the filesystem. The dynamic libraries can be copied to $SCRATCH with the following commands:

```
# replace by the path to your seissol executable
mkdir -p $SCRATCH/libdump  && ldd SeisSol_Release_dskx_6_elastic | grep -E "/work|/
↪scratch" | awk '{print $(NF-1)}' | xargs -I _ cp _ $SCRATCH/libdump
```

Then you can unload the seissol-env module and add the required dynamic libraries, e.g. with:

```
export LD_LIBRARY_PATH=$SCRATCH/libdump/:$LD_LIBRARY_PATH
module unload seissol-env
```

Finally, we provide an example of launch script used for running a full-machine frontera run. In particular, note how timeout and retry count are increased.

```
#!/bin/bash
#SBATCH --chdir=./
#SBATCH -o ./%j.out       # Name of stdout output file
#SBATCH -e ./%j.out       # Name of stderr error file
#SBATCH -p debug         # Queue (partition) name
#SBATCH --nodes=8192
#SBATCH --ntasks-per-node=2
#SBATCH -t 24:00:00        # Run time (hh:mm:ss)
#SBATCH -A EAR22007       # Project/Allocation name (req'd if you have more than 1)

# Any other commands must follow all #SBATCH directives...
module list
pwd
date

#Prevents errors such as experience in Issue #691
```

(continues on next page)

(continued from previous page)

```
export I_MPI_SHM_HEAP_VSIZE=32768

export OMP_NUM_THREADS=27
export OMP_PLACES="cores(27)"
export OMP_PROC_BIND="close"

export XDMFWRITER_ALIGNMENT=8388608
export XDMFWRITER_BLOCK_SIZE=8388608
export ASYNC_MODE=THREAD
export ASYNC_BUFFER_ALIGNMENT=8388608

echo 'num_nodes:' $SLURM_JOB_NUM_NODES 'ntasks:' $SLURM_NTASKS
ulimit -Ss 2097152

source ~cazes/texascale_settings.sh
export UCX_TLS=knem,dc
export UCX_DC_MLX5_TIMEOUT=35000000.00us
export UCX_DC_MLX5_RNR_TIMEOUT=35000000.00us
export UCX_DC_MLX5_RETRY_COUNT=180
export UCX_DC_MLX5_RNR_RETRY_COUNT=180
export UCX_RC_MLX5_TIMEOUT=35000000.00us
export UCX_RC_MLX5_RNR_TIMEOUT=35000000.00us
export UCX_RC_MLX5_RETRY_COUNT=180
export UCX_RC_MLX5_RNR_RETRY_COUNT=180
export UCX_UD_MLX5_TIMEOUT=35000000.00us
export UCX_UD_MLX5_RETRY_COUNT=180


# Launch MPI code...
seissol_exe=SeisSol_Release_dskx_6_viscoelastic2
echo $seissol_exe
time -p ibrun $seissol_exe parameters.par
```

CHAPTER

TWENTYSEVEN

MARCONI 100

Marconi 100 is a distributed multi-GPU HPC system equipped with 4 Nvidia V100 GPUs and 2 IBM Power9 CPUs per node. This architecture usually comes with a pre-installed CUDA-Aware Spectrum-MPI. However, SeisSol cannot operate with Spectrum-MPI because of GPU memory leaks caused by this MPI implementation. This part of documentation describes how to setup and configure OpenMPI 4.1.x together with UCX 1.10.x to operate on IBM Power and similar HPC systems.

27.1 Installing Main Libraries and Packages

1. Create the following launch file:

```
$ touch $HOME/launch.sh
$ cat $HOME/launch.sh
#!/bin/bash

VERSION="openmpi4-1-1_ucx1-10-1"
export SEISSOL_INSTALL=$HOME/usr/local/${VERSION}

export PATH=$SEISSOL_INSTALL/bin:$PATH
export LIBRARY_PATH=$SEISSOL_INSTALL/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=$SEISSOL_INSTALL/lib:$LD_LIBRARY_PATH
export PKG_CONFIG_PATH=$SEISSOL_INSTALL/lib/pkgconfig:$PKG_CONFIG_PATH
export CMAKE_PREFIX_PATH=$SEISSOL_INSTALL
export CPATH=$SEISSOL_INSTALL/include:$CPATH
export CPLUS_INCLUDE_PATH=$SEISSOL_INSTALL/include:$CPLUS_INCLUDE_PATH
export C_INCLUDE_PATH=$SEISSOL_INSTALL/include:$C_INCLUDE_PATH
```

2. Load basic modules, source launch file and create an install directory:

```
$ source ./launch.sh
$ mkdir -p $SEISSOL_INSTALL

$ module load python/3.8.2
$ module load profile/candidate
$ module load gnu/11.2.0
$ module load cmake/3.20.0
$ module load cuda/11.6
```

3. Create any directory where you are going to configure and build libraries and packages. For example,

```
mkdir -p $HOME/Download
cd $HOME/Download
```

4. Install SYCL:

See section *Installing SYCL*. Note, you will need to adjust install-prefixes for both *LLVM*, *Boost* and *hipSYCL* to point to the content of *SEISSOL_INSTALL* environment variable.

5. Install hwloc:

```
$ wget http://www.open-mpi.org/software/hwloc/v2.7/downloads/hwloc-2.7.1.tar.gz && \
$ tar -xvf ./hwloc-2.7.1.tar.gz
$ cd hwloc-2.7.1 && \
$ CC=$(which gcc) CXX=$(which g++) FC=$(which gfortran) ./configure --prefix=${SEISSOL_
↪INSTALL} \
  --disable-opencl --disable-cairo \
  --disable-nvml --disable-gl \
  --enable-cuda \
  --with-cuda=/cineca/prod/opt/compilers/cuda/11.6/none/bin/nvcc \
  --disable-libudev --enable-shared

$ make -j
$ make install
$ cd ..
```

6. Install UCX:

Note, CINECAs' system administrators manually added *CUDA_CFLAGS* env. variable to all *cuda* modules. This prevents compilation of many versions of UCX and OpenMPI. Please, disable it during SeisSol's installation.

```
$ wget https://github.com/openucx/ucx/releases/download/v1.10.0/ucx-1.10.0.tar.gz
$ tar -xvf ucx-1.10.0.tar.gz
$ unset CUDA_CFLAGS
$ mkdir -p ucx-1.10.0/build && cd ucx-1.10.0/build

$ CC=$(which gcc) CXX=$(which g++) FC=$(which gfortran) \
../configure \
--prefix=$SEISSOL_INSTALL \
--build=powerpc64le-redhat-linux-gnu \
--host=powerpc64le-redhat-linux-gnu \
--disable-logging --disable-debug \
--disable-assertions --disable-params-check \
--disable-params-check --enable-optimizations \
--disable-assertions --disable-logging --with-pic \
--without-java \
--enable-mt \
--with-cuda=/cineca/prod/opt/compilers/cuda/11.6/none/bin/nvcc \
--with-gdrcopy \
--with-knem=/opt/knem-1.1.3.90mlnx1 \
--without-xpmem

$ make -j
$ make install
$ cd ../..
```

7. Install OpenMPI:

```
$ wget https://github.com/open-mpi/ompi/archive/refs/tags/v4.1.4.tar.gz
$ tar -xvf ./v4.1.4.tar.gz
$ cd ./ompi-4.1.4
$ ./autogen.pl
$ mkdir -p ./build && cd ./build

$ CC=$(which gcc) CXX=$(which g++) FC=$(which gfortran) \
CFLAGS="-I/opt/pmix/3.1.5/include" CPPFLAGS="-I/opt/pmix/3.1.5/include" \
../configure \
--prefix=$SEISSOL_INSTALL \
--with-memory-manager=none \
--enable-static=yes \
--enable-shared \
--with-slurm \
--with-pmix=/opt/pmix/3.1.5 \
--with-ucx=$SEISSOL_INSTALL \
--with-libevent=/usr \
--with-hwloc=${SEISSOL_INSTALL} \
--with-verbs \
--enable-mpirun-prefix-by-default \
--with-platform=/cineca/prod/build/compilers/openmpi/4.0.3/gnu--8.4.0/BA_WORK/openmpi-4.
↪0.3/contrib/platform/mellanox/optimized \
--with-hcoll=/opt/mellanox/hcoll \
--with-cuda=/cineca/prod/opt/compilers/cuda/11.6/none/bin/nvcc \
--with-knem=/opt/knem-1.1.3.90mlnx1 \
--without-xpmem

$ make -j
$ make install
$ cd ../..
```

8. Install HDF5:

```
$ wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.5/src/hdf5-1.
↪10.5.tar.gz
$ tar -xvf ./hdf5-1.10.5.tar.gz
$ cd hdf5-1.10.5
$ ./autogen.sh
$ mkdir build && cd build

$ CFLAGS="-fPIC ${CFLAGS}" CC=mpicc CXX=mpicxx FC=mpif90 \
../configure \
--prefix=$SEISSOL_INSTALL \
--build=powerpc64le-redhat-linux-gnu \
--host=powerpc64le-redhat-linux-gnu \
--enable-parallel --with-zlib --disable-shared \
--enable-fortran

$ make -j
$ make install
$ cd ../..
```

9. Installing netCDF:

```
$ wget https://syncandshare.lrz.de/dl/fiJNAokgbe2vNU66Ru17DAjT/netcdf-4.6.1.tar.gz
$ tar -xvf ./netcdf-4.6.1.tar.gz
$ cd hdf5-1.10.5
$ ./autogen.sh

$ CFLAGS="-fPIC ${CFLAGS}" CC=h5pcc \
./configure \
--prefix=$SEISSOL_INSTALL \
--build=powerpc64le-redhat-linux-gnu \
--host=powerpc64le-redhat-linux-gnu \
--enable-shared=no \
--disable-dap

$ make -j
$ make install
$ cd ..
```

10. Installing ParMetis:

```
$ https://ftp.mcs.anl.gov/pub/pdetools/spack-pkgs/parmetis-4.0.3.tar.gz
$ tar -xvf ./parmetis-4.0.3.tar.gz
$ cd parmetis-4.0.3
#edit ./metis/include/metis.h IDXTYPEWIDTH to be 64 (default is 32).
$ make config cc=mpicc cxx=mpiCC prefix=$SEISSOL_INSTALL
$ make install
$ cp build/Linux-ppc64le/libmetis/libmetis.a $SEISSOL_INSTALL/lib
$ cp metis/include/metis.h $SEISSOL_INSTALL/include
$ cd ..
```

11. Install GemmForge and ChainForge. Please, follow steps described *here*.

12. Install easi with LUA backend:

```
# yaml-cpp
$ wget https://github.com/jbeder/yaml-cpp/archive/refs/tags/yaml-cpp-0.7.0.tar.gz
$ tar -xvf yaml-cpp-0.7.0.tar.gz
$ cd yaml-cpp-yaml-cpp-0.7.0
$ sed -i 's/$<${not-msvc}/#$<${not-msvc}/g' ./CMakeLists.txt
$ mkdir -p build && cd build
$ cmake .. -DCMAKE_INSTALL_PREFIX=$SEISSOL_INSTALL \
  -DYAML_BUILD_SHARED_LIBS=ON \
  -DBUILD_TESTING=OFF \
  -DBUILD_MOCK=OFF \
  -DBUILD_GMOCK=OFF
$ make -j4 && make install
$ cd ../..

# LUA
$ wget https://www.lua.org/ftp/lua-5.3.6.tar.gz
$ tar -xzvf lua-5.3.6.tar.gz
$ cd lua-5.3.6
$ make linux CC=mpicc
```

(continues on next page)

(continued from previous page)

```
$ make local
$ cp -r install/* $SEISSOL_INSTALL
$ cd ..

# easi
$ git clone https://github.com/SeisSol/easi.git
$ cd easi
$ mkdir -p build && cd build
$ CC=mpicc CXX=mpicxx FC=mpifort cmake .. \
  -DASAGI=OFF \
  -DLUA=ON \
  -DIMPALAJIT=OF \
  -DCMAKE_INSTALL_PREFIX=$SEISSOL_INSTALL
$ make -j4 && make install
$ cd ../..
```

13. Install Eigen3:

```
$ git clone https://gitlab.com/libeigen/eigen.git
$ mkdir -p eigen/build && cd eigen/build
$ CXX=g++ CC=gcc FC=gfortran cmake .. -DCMAKE_INSTALL_PREFIX=$SEISSOL_INSTALL
$ make -j
$ make install
$ cd ../..
```

14. Install SeisSol:

```
$ module load cuda/11.6
$ git clone --recurse-submodules https://github.com/SeisSol/SeisSol.git
$ cd SeisSol
$ mkdir build && cd build

$ CC=mpicc CXX=mpicxx FC=mpifort cmake .. \
-DCMAKE_BUILD_TYPE=Release \
-DDEVICE_BACKEND=cuda \
-DDEVICE_ARCH=sm_70 \
-DHOST_ARCH=power9 \
-DPRECISION=single

$ make -j
```

15. Run SeisSol-proxy as a sanity check:

```
./launch ./SeisSol_proxy_Release_ssm70_cuda_6_elastic 100000 100 all
```

27.2 Running SeisSol

As discussed *here*, process pinning is important for SeisSol GPU version. IBM Power9 is an example of RISC architecture designed with with 4-way hyperthreading and 8 cores per CPU. In total, each node of Marconi 100 can run 256 threads. By and large process pinning needs a special care on such architectures because some libraries have different meanings of cores and threads.

Below you can see an example of a *batch script* with parameters resulting in an optimal process pinning. Note that each node of Marconi 100 has 2 Mellanox network cards i.e., each per NUMA domain. In this example, we enforce UCX to utilize both. Moreover, we reserve one 1 core for each MPI process for SeisSol communication thread.

In this particular case it is not necessary to provide a number of processes after **mpirun** because OpenMPI was compiled with PMIX (see step 5).

Please, do not forget to launch SeisSol via *launch* bash script generated with CMake during SeisSol's configuration.

```
$ cat ./job.sh
#!/bin/bash
#SBATCH --account=<you account>
#SBATCH --partition=m100_usr_prod
#SBATCH --qos=m100_qos_dbg
#SBATCH --time <time>
#SBATCH --nodes <number of nodes>
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=32
#SBATCH --gres=gpu:4
#SBATCH --gpu-bind=closest
#SBATCH --mem=161070
#SBATCH --job-name=<your job name>
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<user_email>
#SBATCH --output=seissol.out
#SBATCH --error=seissol.err
#SBATCH --exclusive

NUM_CORES=$(expr $SLURM_CPUS_PER_TASK / 4)
NUM_COMPUTE_CORES=$(expr $NUM_CORES - 1)

export OMP_NUM_THREADS=$NUM_COMPUTE_CORES
export OMP_PLACES="cores($NUM_COMPUTE_CORES)"
export PROC_BIND=spread

export DEVICE_STACK_MEM_SIZE=1.5
export UCX_MEMTYPE_CACHE=n

mpirun --report-bindings --map-by ppr:$SLURM_NTASKS_PER_NODE:node:pe=$NUM_CORES \
-x UCX_MAX_EAGER_RAILS=2 -x UCX_MAX_RNDV_RAILS=2 -x UCX_NET_DEVICES=mlx5_0:1,mlx5_1:1 \
-x UCX_MEM_MMAP_HOOK_MODE=none \
./launch ./SeisSol_Release_ssm70_cuda_6_elastic ./parameters.par
```

27.3 Troubleshooting OpenMPI and UCX

1. OpenMPI and UCX provide users with utilities which show how these packages were configured and installed. It is **ompi_info** for former and **ucx_info -b** for latter.

2. It may be required to switch off UCX memory caching because it can lead to run-time failures of UCX. One can achieve this by setting the following environment variable:

```
$ export UCX_MEMTYPE_CACHE=n
```

3. One can enable a launch time information from OpenMPI and UCX by setting the following parameters after **mpirun**.

```
--mca pml_base_verbose 10 --mca mtl_base_verbose 10 -x OMPI_MCA_pml_ucx_verbose=10
```

4. We recommend to login into a compute node and execute **ucx_info -d** command if you need to get information about all available network devices. This will help you to retrieve exact names of network devices e.g., *mlx5_0:1, mlx5_1:1, etc*.

HEISENBUG

heisenbug is a computing cluster of the computational seismology group at LMU. It is an AMD EPYC based machine with 128 cores that can run 256 threads (near) simultaneously. It also has 2 GPGPUs (NVIDIA GeForce RTX 3090), that can be used to run the GPU version of SeisSol. The RTX 3090 belongs to a consumer kind of graphics cards and thus does not perform well with double precision. Therefore, it is preferable to compile SeisSol with single precision.

A module integrating all libraries relevant for compiling SeisSol with CUDA and SYCL is available on heisenbug. It can be discovered at startup after adding the following to `~/.bashrc`:

```
module use /import/exception-dump/ulrich/spack/modules/linux-debian11-zen2
```

It is then loaded with:

```
# load the (first in the list) seissol-env module compiled with cuda support
module load $(module avail seissol-env/*-cuda-* | awk '/seissol-env/ {print $1}')
```

This module has been compiled based on the main branch of https://github.com/SeisSol/seissol-spack-aid with the command:

```
spack install -j 40 --fresh seissol-env +cuda %gcc@10
spack module tcl refresh $(spack find -d --format "{name}{/hash:5}" seissol-env +cuda)
```

Install YATeTo GPU backends (i.e., GemmForge and ChainForge) as shown *here*.

Then clone SeisSol with:

```
git clone https://github.com/SeisSol/SeisSol.git
cd SeisSol
git submodule update --init --recursive
```

To compile the GPU version of SeisSol on heisenbug, use the following cmake options

```
-DDEVICE_ARCH=sm_86 -DHOST_ARCH=hsw -DDEVICE_BACKEND=cuda -DPRECISION=single -DHIPSYCL_
↪CUDA_PATH=$CUDA_HOME
```

As there is no queuing system on heisenbug, you need to make sure that nobody is running anything on the GPUs. You can check that by running `nvidia-smi` (it should return `No running processes found`).

To run on one GPU (here with order 4, elastic), use simply:

```
export OMP_NUM_THREADS=1
export OMP_PLACES="cores"
export OMP_PROC_BIND=spread
./launch ./SeisSol_RelWithDebInfo_ssm_86_cuda_4_elastic ./parameters.par
```

launch is a simple bash helper script. It is generated by CMake, in the build directory).

On 2 ranks, use:

```
# Note that it is possible to increase OMP_NUM_THREADS
# This will speed up (the rare) portions of the code running only CPUs, e.g. the wiggle␣
↪factor calculation
export OMP_NUM_THREADS=1
export OMP_PLACES="cores"
export OMP_PROC_BIND=spread
mpirun -n 2 --map-by ppr:1:numa:pe=2 --report-bindings ./launch ./SeisSol_RelWithDebInfo_
↪ssm_86_cuda_4_elastic ./parameters.par
```

CHAPTER

TWENTYNINE

DYNAMIC RUPTURE

SeisSol is verified for a wide range of dynamic rupture problems (Pelties et al. 2014): it is possible to use branched faults, dipping fault geometries and laboratory-derived constitutive laws such as the rate-and-state friction law.

29.1 Sign conventions

The following definitions regarding fault and stress orientation are conventions. The initial stresses need to be given in the global coordinate system. The Matlab script '/preprocessing/science/stressrotation3D' is available for correct rotation of initial stress values on arbitrary orientated faults.

29.1.1 Definitions

- Traction = (normal vector)* (stress tensor)
  - Shear traction in strike direction right (left) lateral: positive (negative)
  - Shear traction along dip (anti-dip) direction: positive (negative)
- Normal stress = negative in compression
- Cohesion = negative (acts on shear stress components, encapsulates the effect of pore pressurization)
- Reference point = defines dip direction, starting point of the normal vector n pointing towards the fault, at +y
- Note: SCEC benchmark descriptions define normal stress as positive in compression, opposite to SeisSol 3D convention.

29.1.2 Fault geometry

The right-handed coordinate system is required. Please, make sure to follow this convention during model/mesh generation!

- Arbitrary fault orientation possible, **except** a fault in the xy-plane, i.e. parallel to the free surface
- Fault Output is in fault coordinate system
  - n = (nx, ny, nz).T normal vector, from + y → - y
  - s = (ny, - nx, 0).T * sqrt (nx^2 + ny^2) along-strike vector
  - d = (-sy*nz, sx*nz, sy*nx-ny*sx).T / || d || along-dip vector, pointing in -z direction (downwards)

29.1.3 Example 1: Initial stresses for a 60° dipping fault

Following the SCEC benchmark description of TPV10 we need to transfer the following initial stresses given in fault coordinates in global coordinates for SeisSol:

INPUT (from tpv10)

```
normal stress = - 7378 Pa/m * (meter downdip-distance)
shear stress = 0.55 * (normal stress) = + 4057.9 Pa/m * (meter downdip-distance)
```

Assuming the fault plane in xz-plane, dipping 60° in -y-direction we can use the script '/preprocessing/science/stressrotation3D' to rotate the stresses 30° clockwise at the x-axis (considering we look towards +x) to get the full stress tensor in global coordinates:

OUTPUT (to be used in SeisSol's parameter file)

```
s_yy = -  2019.26 Pa/m * (meter downdip-distance)
s_zz = -  5358.74 Pa/m * (meter downdip-distance)
s_yz = + 5223.72 Pa/m * (meter downdip-distance)
s_xx = s_xy = s_xz = 0.0 Pa
```

29.1.4 Example 2: Initial stresses for a branched fault

Normal and shear stresses in fault coordinate system for strike-slip fault aligned with the x- or y-axis are already in the global coordinates. But if we have additional branches the initial stresses need to be rotated again. Following for example the SCEC benchmark description for TPV14, the stresses for the right-lateral strike-slip fault in xz-plane are:

```
s_yy = - 120 MPa
s_xy = + 70 MPa
s_xy (nucleation patch) = + 81.6 MPa
s_xx = s_zz = s_xz = s_yz = 0.0 Pa
```

For the 30° branch in -y direction we rotate the given stresses (in that case, they are the same as on the main fault) by 330° counterclockwise (or 30° clockwise) around the z-axis using the script '/preprocessing/science/stressrotation3D'.

OUTPUT (to be used in SeisSol's parameter file)

```
s_xx = + 30.62 MPa
s_yy = - 150.62 MPa
s_xy =  - 16.96 MPa
s_zz = s_xz = s_yz = 0.0 Pa
```

In case of a left-lateral strike slip fault (for example TPV15) the stresses on the main fault are:

```
s_yy = - 120 MPa
s_xy =  - 70 MPa
s_xy (nucleation patch) = - 81.6 MPa
s_xx = s_zz = s_xz = s_yz = 0.0 Pa
```

29.2 Projecting the state variable increment

A sudden decrease of slip rate to very small values, as for example occurring at a rupture front about to be arrested, may cause numerical issues and pollute the solution - in the worst case even leading to re-activation of rupture. Such numerical issues are easy to diagnose in the fault output visualization: a checkerboard pattern with unphysical values for the slip rate and slip in the affected region will be visible. This is a sign of mesh resolution (h-refinement) being locally not sufficient. SeisSol mitigates such artifacts by projecting the state variable (e.g. cumulative slip for linear slip weakening) increment on the 2D fault interface basis function after evaluation of the friction law. This implementation lowers the required h-refinement across the fault in comparison to earlier implementations (cf. Pelties et al. , JGR 2012; GMD 2014)

29.3 Visualisation: SlipRateOutputType (default =1)

By default, the fault output will be showing regions affected by numerical problems. However, the user may choose to smooth out such artifacts for visualization purposes. Switching `SlipRateOutputType` in the `DynamicRupture` namelist from the default value to 0, will evaluate the slip-rate from the difference between the velocity on both sides of the fault, rather than evaluating the slip-rate from the fault tractions and the failure criterion directly. Note that this fix only applies to the output, but does not suppress numerical problems in the simulation. Also note that that `SlipRateOutputType=0` is slightly less accurate than the default `SlipRateOutputType=1` without numerical problems.

29.4 Friction laws

29.4.1 Linear slip-weakening friction (FL=6, FL=16)

The linear slip-weakening friction is widely used for dynamic rupture simulations.

The fault strength is determined by

$$\tau = -C - \min\left(0, \sigma_n\right) \left(\mu_s - \frac{\mu_s - \mu_d}{d_c} \min\left(S, d_c\right) \right),$$

where $S(t) = \int_0^t |V(s)| ds$ is the accumulated fault slip, and the other variables are parameters of the friction, detailed below.

Friction parameters:

symbol	quantity	SeisSol name
$\mu_s(x)$	static friction coefficient	`mu_s`
$\mu_d(x)$	dynamic friction coefficient	`mu_d`
$d_c(x)$	slip-weakening critical distance	`d_c`
$C(x)$	cohesion	`cohesion`
$T(x)$	forced rupture time	`forced_rupture_time`
v_0	threshold velocity	`lsw_healingThreshold`

Friction law 16 implements linear slip-weakening with a forced rupture time. If you are only interested in linear slip weakening friction without forced rupture time, do not supply the parameter *forced_rupture_time* in the fault *yaml* file. Friction law 6 uses Prakash-Clifton regularization for bimaterial faults. For friction law 16, we resample the slip rate in every step to suppress spurious oscillations. In the case of Prakash-Clifton regularization, we do not resample the slip rate. If the slip rate V drops below the threshold velocity v_0, we reset the friction parameter $\mu = \mu_s$. Also, we reset the state variable $S = 0$. The threshold v_0 is set to -1.0 by default, such that healing is disabled.

Examples of input files for the friction laws `6` and `16` are availbable in the *cookbook*.

Linear slip weakening can be seen as a special case of rate-and-state friction with

$$\begin{aligned} f(V,\psi) &= C - \left(\mu_s - \frac{\mu_s - \mu_d}{d_c}\right) \min\left(\psi, d_c\right), \\ g(V,\psi) &= V. \end{aligned}$$

Now the state variable stores the accumulated slip.

29.4.2 TP proxy slip-weakening friction (`FL=1058`)

The TP proxy slip-weakening friction has been proposed by Herrera et al. (2024), GJI, to approximate thermal pressurization in a computationally efficient way. The fault strength is determined by

$$\tau = -C - \min\left(0, \sigma_n\right) \left(\mu_d + \frac{(\mu_s - \mu_d)}{\left(1 + \frac{S}{d_c}\right)^{\alpha}}\right),$$

All variables are the same as defined in previous section for `FL=16`. The friction law also supports forced rupture time. You can modify the default value 1/3 of α, by adjusting the `TpProxyExponent` parameter in the main parameter file (namelist: `DynamicRupture`).

29.4.3 Rate-and-state friction

Rate-and-state friction laws allow modeling the frictional shear strength variations as a function of slip rate and of the evolving properties of the contact population (Dieterich, 1979, 1981; Ruina, 1983). In SeisSol, we currently support 3 types of rate-and-state friction laws, which differ by the set of ordinary differential equations describing the evolution of the state variable. The type of rate-and-state friction law is set by the FL variable in the DynamicRupture namelist (parameters.par): Friction law `3` implements the ageing law, friction law `4` implements the slip law, and friction law `103` implements a slip law with strong rate-weakening. More details about these friction laws can be found in the SCEC benchmarks descriptions (TPV101 to 105) or in Pelties et al. (2013, GMD).

Some parameters are considered homogeneous across the fault and defined in the main parameter file. Others can spatially vary (`rs_a`, `RS_sl0` for FL=3,4 and 103 and `rs_srW` for FL=103) and are defined in the fault yaml file. Examples of input files for the ageing law and for the rate and state friction with strong velocity weakening are available at the given links.

All rate-and-state friction laws are described by the following system of differential algebraic equations, which depend on the state variable ψ and the slip velocity V.

$$\begin{aligned} \tau &= \sigma_n f(V,\psi) \\ \frac{\partial \psi}{\partial t} &= g(V,\psi) \end{aligned}$$

Ageing law (`FL=3`)

Reference benchmarks: TVP101 and TPV102

Friction parameters:

symbol	quantity	seisSol name
$a(x)$	frictional evolution coefficient	`rs_a`
b	frictional state coefficient	`rs_b`
$L(x)$	characteristic slip scale	`rs_sl0`
V_0	reference slip velocity	`rs_sr0`
f_0	reference friction coefficient	`rs_f0`

$$f(V,\psi) = a \sinh^{-1}\left[\frac{V}{2V_0}\exp\left(\frac{f_0 + b\ln(V_0\psi/L)}{a}\right)\right]$$
$$g(V,\psi) = 1 - \frac{V\psi}{L}$$

Slip law (`FL=4`)

The slip law has the same parameters as the Ageing Law.

$$f(V,\psi) = a \sinh^{-1}\left[\frac{V}{2V_0}\exp\left(\frac{f_0 + b\ln(V_0\psi/L)}{a}\right)\right]$$
$$g(V,\psi) = -V\frac{\psi}{L}\ln\left(V\frac{\psi}{L}\right)$$

Strong velocity weakening (`FL=103`)

Reference TPV103 and TPV104

In addition to the ageing and the slip Law, strong velocity weakening requires two more parameters:

symbol	quantity	seisSol name
$V_w(x)$	weakening slip velocity	`rs_srW`
μ_w	weakening friction coefficient	`rs_muW`

$$f(V,\psi) = a \sinh^{-1}\left[\frac{V}{2V_0}\exp\left(\frac{\psi}{a}\right)\right]$$
$$g(V,\psi) = -\frac{V}{L}\left(\psi - a\ln\left[\frac{2V_0}{V}\sinh\left(\frac{\mu_{ss}(V)}{a}\right)\right]\right)$$

with

$$\mu_{ss}(V) = \mu_w + \frac{f_0 - (b-a)\ln\left(\frac{V}{V_0}\right) - \mu_W}{\left(1 + \left[\frac{V}{V_W}\right]^8\right)^{1/8}}.$$

Note that from the merge of pull request #306 of March 17th, 2021 to the merge of pull request #752 of December 22nd, 2022, the state variable was enforced positive in this friction law. This enforcement aimed at avoiding the state variable getting negative because of Gibbs effects when projecting the state increment onto the modal basis functions (resampling matrix). Since then, we realized that the state variable can get negative due to other factors, and, therefore, reverted this change.

29.5 Thermal Pressurization

Seissol can account for thermal pressurization (TP) of pore fluids. As deformation occurs within the fault gauge, frictional heating increases the temperature of the rock matrix and pore fluids. The pore fluids then pressurize, which weakens the fault. The evolution of the pore fluid pressure and temperature is governed by the diffusion of heat and fluid. TP can be activated using `thermalPress` in the `DynamicRupture` namelist. TP can be enabled with all rate-and-state friction laws (FL=3,4 and 103). The TP parameters for which no spatial dependence has been implemented are defined directly in the `DynamicRupture` namelist:

```
&DynamicRupture
thermalPress = 1                     ! Thermal pressurization 0: inactive; 1: active
tp_iniTemp = 483.15                  ! Initial temperature [K]
tp_iniPressure = -80.0e6             ! Initial pore pressure; have to be added to normal␣
↪stress in your initial stress yaml file [Pa]
tp_thermalDiffusivity = 1.0e-6       ! Thermal diffusivity [m^2/s]
tp_heatCapacity = 2.7e6              ! Specific heat [Pa/K]
tp_undrainedTPResponse = 0.1e6       ! Pore pressure change per unit temperature [Pa/K]
```

Two additional thermal pressurization parameters are space-dependent and therefore have to be specified in the dynamic rupture yaml file:

```
!ConstantMap
map:
  tp_hydraulicDiffusivity: 1e-4   # Hydraulic diffusivity [m^2/s]
  tp_halfWidthShearZone: 0.01     # Half width of shearing zone [m]
```

TP generates 2 additional on-fault outputs: Pore pressure and temperature (see fault output).

CHAPTER

THIRTY

MULTIPLE POINT-SOURCES

30.1 Standard Rupture Format

SeisSol supports the Standard Rupture Format (SRF) for kinematic rupture models. Details about the file format can be found at the SCEC Wiki. With the help of the SRF format, one may specify several subfaults (point sources), where one may give individual source time function for each subfault. The location, however, is given in latitude, longitude, and depth such that it becomes necessary to convert those into the coordinate system used by the mesh, which we call the Mesh Coordinate System (MCS). In a software package, one usually has the option to directly convert coordinate systems during runtime or to use an intermediate format that does not require coordinate conversion. Here, we opted for the second approach, because

- we do not complicate the build and use of SeisSol and
- we can use a binary format which greatly reduces the loading time.

Hint: Use krfviewer to inspect Standard Rupture Format files and to test projections.

Note: There is a slight difference in SRF version 1.0 and 2.0.
Point line 1 in 1.0: longitude latitude depth strike dip area tinit dt.
Point line 1 in 2.0: longitude latitude depth strike dip area tinit dt vs den.

30.2 NetCDF Rupture Format (NRF)

The NRF is an intermediate format for describing kinematic rupture models. It is not meant to be used directly but it should be generated from an SRF file. To do so, you require the tool rconv. Note that some python scripts required for compiling rconv are given as a symbolic link in rconv, the link being a relative path. This means that **you need the whole SeisSol repository to compile it**.

30.2.1 Specifying the MCS

The main input parameter of rconv is the specification of the MCS. It is very important to specify it right, otherwise you will get wrong moment tensors and wrong subfault locations. The MCS can be specified by a string describing its projection in the same way as you would use the cartographic software proj.4. For example,

```
+proj=utm +zone=10 +datum=WGS84 +units=m +axis=ned +no_defs
```

would lead to UTM projection of latitude and longitude and

```
+proj=geocent +datum=WGS84 +units=m +axis=seu +no_defs
```

would lead to a geocentric coordinate system. You can test the correctness of your projection string by invoking `cs2cs` from the proj.4 application suite. For example,

```
echo 11.669 48.263 0.476 | cs2cs +proj=lonlat +datum=WGS84 +units=km +to +proj=utm␣
↪+zone=33 +datum=WGS84 +units=m +axis=ned
```

Besides correct projections, it is also important to specify the axis orientation with the +axis option. Examples:

- +axis=ned: x=north, y=east, z=down
- +axis=enu: x=east, y=north, z=up
- +axis=seu: x=south, y=east, z=up

If the axis description does not fit your mesh, your moment tensor will not be rotated correctly (according to strike, dip, and rake angles).

30.2.2 Named projections

Some named projections are not recognized by proj4 (for instance, most EPSG projections). A good ressource for transposing these named projections to generic projection strings that are understood by proj4 (and rconv) can be found at this page.

30.2.3 Dealing with projected data

If the SRF data are already projected, the projection within rconv can be by-passed using the following projection string: `+proj=lonlat +datum=WGS84 +units=m`.

30.2.4 How to use Rconv

To use e.g. Mercator projection, you can run

```
rconv -i input.srf -o output.nrf -m “+proj=merc +lon_0=central_longtitude +y_
↪0=translation_along_y +x_0=translation_along_x +units=m +axis=enu” -x visualization.
↪xdmf
```

More projection options can be found in proj.4 website.

30.2.5 Checking the NRF file

You may manually inspect an NRF file to verify its correctness. If you have NetCDF installed, you may enter

```
ncdump -v subfaults sources.nrf | less
```

and obtain something like

```
  compound Subfault {
    double tinit ;
    double timestep ;
    double mu ;
```

(continues on next page)

(continued from previous page)

```
    double area ;
    Vector3 tan1 ;
    Vector3 tan2 ;
    Vector3 normal ;
  }; // Subfault
...
    Subfault subfaults(source) ;
        Subfault_units subfaults:units = {"s", "s", "pascal", "m^2", "m", "m", "m"} ;
...
 centres = {23166.2135886125, -13374.980382819, 1250},
    {22733.1990108113, -13124.9814335668, 1250},
...
 subfaults =
    {10.617000000043, 0.001, 0, 250000, {-0.866025403784439, 0.5, -0}, {3.
↪06161699786838e-17, 5.30287619362453e-17, -1}, {-0.5, -0.866025403784439, -6.
↪12323399573677e-17}},
    {10.446600000043, 0.001, 0, 250000, {-0.866025403784439, 0.5, -0}, {3.
↪06161699786838e-17, 5.30287619362453e-17, -1}, {-0.5, -0.866025403784439, -6.
↪12323399573677e-17}},
```

which tells you the following:

- The first point source is located at x=23166.2135886125, y=-13374.980382819, z=1250.
- The STF of the source starts acting after 10.617000000043 seconds.
- The distance between samples in the STF is 0.001 seconds.
- The shear modulus is 0 Pa, which means that SeisSol will take the shear modulus from the element in which the point source resides.
- The subfault has an area of 250000 square meters. (Be careful, in the SRF you have to give it in square centimetres.)
- u_1 = {-0.866025403784439, 0.5, -0}, u_2 = {3.06161699786838e-17, 5.30287619362453e-17, -1}, u_3 = {-0.5, -0.866025403784439, -6.12323399573677e-17}, where u_1 is the strike direction, u_2 is orthogonal to the strike direction but lies in the fault plane, and u_3 is the normal direction.

30.2.6 Using an NRF file in SeisSol

Add the following section to your parameter file:

```
&SourceType
Type = 42
FileName = 'sources.nrf'
/
```

30.2.7 Pitfalls

Multi point-sources representation generate spurious waves at frequencies close to Vr/h with Vr the rutpure speed and h the spatial sampling of the Kinematic model. Also, the source time function are discretized by linear interpolation, and should be adequately sampled in time to avoid sharp kinks in the source time function, which can be the source of high frequency generation. Therefore, the kinematic model may need to be upsampled in space and/or in time, for example using this script: https://github.com/SeisSol/SeisSol/blob/master/preprocessing/science/kinematic_models/refine_srf.py A possible alternative is to impose the kinematic model on a dynamic rupture boundary, see *Slip-rate imposed on a DR boundary* for more details.

CHAPTER

THIRTYONE

SLIP-RATE IMPOSED ON A DR BOUNDARY

This “pseudo” friction law allows imposing slip-rate of a kinematic source model as a boundary condition on a fault plane. This implentation is equivalent to the approach used e.g. in Tinti et al. (2005, https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2005JB003644) and Causse et al. (2014, https://academic.oup.com/gji/article/196/3/1754/583512#9427920).

The advantage of this approach is that the displacement discontinuity can be accurately represented in SeisSol’s discontinuous finite element space. A multi point-source representation, in comparison, may give rise to spurious waves due the continuity of the basis functions within a finite element (smearing), and to the discrete spatial sampling between point sources (aliasing).

The current implementation allows either imposing kinematic models parameterized by regularized Yoffe functions (FL=33, see Tinti et al., 2005, https://doi.org/10.1785/0120040177) or by Gaussian source time functions (FL=34).

The Yoffe functions are parametrized by `rupture_onset`, `tau_S` and `tau_R`, where `rupture_onset` is the onset time of the rupture, `tau_S` is a parameter closely related (see hereafter) with `T_acc`, the duration of the positive slip acceleration (time to the peak slip-rate), and `tau_R` is a parameter that, in combination with `tau_S`, allow defining the effective duration of slip `tau^eff_R`.

For typical `tau_S/tau_R` ratio, we can assume `T_acc = 1.27 tau_S`. Yet, the factor can range from about 1.15 (for `tau_S/tau_R` close to 0) to about 1.4 (for `tau_S/tau_R` close to 0.4). In addition, we can typically assume `tau^eff_R = tau_R + 2 tau_S`. Note that in the code, we apply `tau_R = max(tau_R, tau_S)` to ensure that `tau_R` >= `tau_S` (the contrary could occur after interpolation from ASAGI).

The Gaussian source time functions are parametrized by `rupture_onset` and `rupture_rise_time`.

The slip distribution is defined by the `strike_slip` and `dip_slip` variables. All these parameters are read by easi from the dynamic rupture yaml file.

Warning:

- the direction of positive `strike_slip` and `dip_slip` is based on the convention of Seissol (e.g. positive strike_slip for right-lateral faulting).
- Select *SlipRateOutputType=0* in the parameter file to calculate slip-rate output properly.

A fully working example based on Northridge is given at https://github.com/SeisSol/Examples/tree/master/Northridge_FL33.

Note that the yaml and netcdf files describing the kinematic model parameters, and the ts file describing the fault geometry are automatically generated by this script. Misfits information printed by this script can help choosing the best threshold value (see option `--PSRthreshold`) and which of the Yoffe or Gaussian source time function to use.

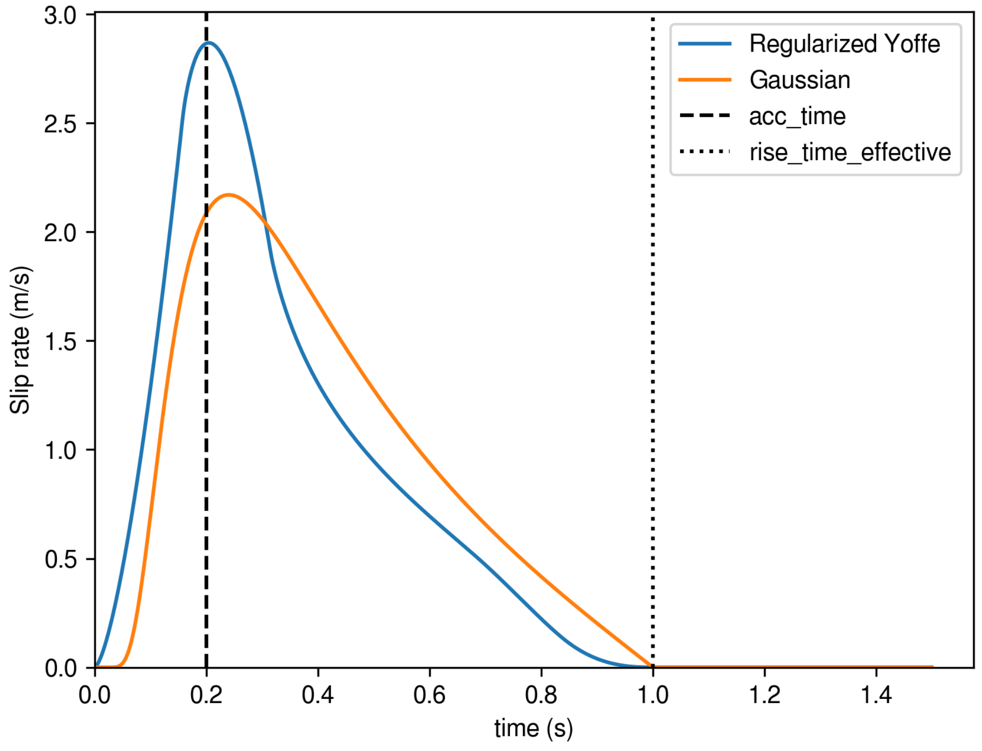


Fig. 1: Shape of a regularized Yoffe function (blue) and a Gaussian source time function (orange). We here use rupture_onset=0, acc_time=0.2 and effective_rise_time=1.0.

CHAPTER

THIRTYTWO

POINT SOURCE (OLDER IMPLEMENTATION)

Using `Type = 50` is the old and non-optimal way to include a point source in SeisSol. It might nevertheless still be useful for modeling a non double-couple point source (not currently possible with the nrf source description).

Add the following section to your parameter file:

```
&SourceType
Type = 50
FileName = 'source.dat'
/
```

Where source.dat has the following format

```
header line (e.g. 'Seismic Moment Tensor')
M11 M12 M13
M21 M22 M23
M31 M32 M33
header line (optional, but has to contain 'velocity' to be recognized)
d1 d2 d3    (optional)
header line (e.g. 'Number of subfaults')
nsubfaults
header line (e.g. 'x y z strike dip rake area Onset time')
x y z strike dip rake area Onset time(1)
x y z strike dip rake area Onset time(2)
....
x y z strike dip rake area Onset time(nsubfaults)
header line (e.g. 'source time function')
dt ndt
header line (e.g. 'samples')
STF(1,1)
STF(1,1)
...
STF(1,ndt)
STF(2,1)
...
STF(nsubfaults,1)
...
STF(nsubfault,ndt)
```

Using this implementation one can specify point sources in the following form:

$$\begin{aligned}
\frac{\partial}{\partial t}\sigma_{xx} - (\lambda+2\mu)\frac{\partial}{\partial x}u - \lambda\frac{\partial}{\partial y}v - \lambda\frac{\partial}{\partial z}w &= M_{xx}\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\frac{\partial}{\partial t}\sigma_{yy} - \lambda\frac{\partial}{\partial x}u - (\lambda+2\mu)\frac{\partial}{\partial y}v - \lambda\frac{\partial}{\partial z}w &= M_{yy}\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\frac{\partial}{\partial t}\sigma_{zz} - \lambda\frac{\partial}{\partial x}u - \lambda\frac{\partial}{\partial y}v - (\lambda+2\mu)\frac{\partial}{\partial z}w &= M_{zz}\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\frac{\partial}{\partial t}\sigma_{xy} - \mu\frac{\partial}{\partial x}v - \mu\frac{\partial}{\partial y}u &= M_{xy}\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\frac{\partial}{\partial t}\sigma_{yz} - \mu\frac{\partial}{\partial z}v - \mu\frac{\partial}{\partial y}w &= M_{yz}\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\frac{\partial}{\partial t}\sigma_{xz} - \mu\frac{\partial}{\partial z}u - \mu\frac{\partial}{\partial x}w &= M_{xz}\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\rho\frac{\partial}{\partial t}u - \frac{\partial}{\partial x}\sigma_{xx} - \frac{\partial}{\partial y}\sigma_{xy} - \frac{\partial}{\partial z}\sigma_{xz} &= d_x\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\rho\frac{\partial}{\partial t}v - \frac{\partial}{\partial x}\sigma_{xy} - \frac{\partial}{\partial y}\sigma_{yy} - \frac{\partial}{\partial z}\sigma_{yz} &= d_y\cdot S_k(t)\cdot\delta(x-\xi_k)\\
\rho\frac{\partial}{\partial t}w - \frac{\partial}{\partial x}\sigma_{xz} - \frac{\partial}{\partial y}\sigma_{yz} - \frac{\partial}{\partial z}\sigma_{zz} &= d_z\cdot S_k(t)\cdot\delta(x-\xi_k).
\end{aligned}$$

For details about the source term read section 3.3 of An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes – I. The two-dimensional isotropic case with external source terms

Mij and di are defined in a fault local coordinate system defined by strike, dip and rake, see for instance here: https://github.com/SeisSol/SeisSol/blob/master/src/SourceTerm/PointSource.cpp#L48

The above equations also hold for viscoelastic or anisotropic materials. In the viscoelastic case, the equations are extended by the memory variables. In the anisotropic case, λ and μ are replaced by the entries of the Hooke tensor c_{ij}.

For poroelastic materials, we add the possibility to consider forces in the fluid or pressure sources. To do so, add these two lines before *Number of subfaults*:

```
header line (optional, but has to contain 'pressure' to be recognized)
p
header line (optional, but has to contain 'fluid' to be recognized)
f1 f2 f3
```

In the case of a poroelastic material, the equations of motion differ from above equations. For the velocities in x direction, they read:

$$\begin{aligned}
\frac{\partial}{\partial t}u - \frac{1}{\rho^{(1)}}\frac{\partial}{\partial x}\sigma_{xx} - \frac{1}{\rho^{(1)}}\frac{\partial}{\partial y}\sigma_{xy} - \frac{1}{\rho^{(1)}}\frac{\partial}{\partial z}\sigma_{xz} - \frac{1}{\rho^{(1)}}\frac{\rho_f}{m}\frac{\nu}{\kappa}u_f &= d_x\cdot S_k(t)\delta(x-\xi_k)\\
\frac{\partial}{\partial t}u_f - \frac{1}{\rho^{(2)}}\frac{\partial}{\partial x}\sigma_{xx} - \frac{1}{\rho^{(2)}}\frac{\partial}{\partial y}\sigma_{xy} - \frac{1}{\rho^{(2)}}\frac{\partial}{\partial z}\sigma_{xz} - \frac{1}{\rho^{(2)}}\frac{\rho}{\rho_f}\frac{\nu}{\kappa}u_f &= f_x\cdot S_k(t)\delta(x-\xi_k)
\end{aligned}$$

with

$$\begin{aligned}
\rho^{(1)} = \left(\rho - \frac{\rho_f^2}{m}\right), \quad \rho^{(2)}\\
= \left(\rho_f - \frac{m\rho}{\rho_f}\right).
\end{aligned}$$

In particular, the terms d_x and f_x are not divided by $\rho^{(1)}$ or $\rho^{(2)}$, respectively.

CHAPTER

THIRTYTHREE

FAULT OUTPUT

33.1 Introduction

There are two primary methods for visualizing on-fault rupture dynamics:

1. Evaluation of rupture characteristics at specific on-fault locations via ASCII receiver files
2. Visualization across the entire fault surface with files that can be opened with ParaView

While threads or nodes can be allocated to write the ParaView output (see *Asynchronous Output*), it is typically not required.

33.2 DynamicRupture Namelist OutputPointType Configuration

The output type generated is determined by the **OutputPointType** variable in the DynamicRupture namelist. Here is an example of how to configure it:

```
&DynamicRupture
OutputPointType = 4
/
```

OutputPointType determines the type of output:

0 : No output
3 : ASCII fault receivers
4 : ParaView file
5 : Both ASCII fault receivers and ParaView file

33.3 Configuring ParaView output

You can adjust the ParaView output using the Elementwise namelist. Here's an example of how to do this:

```
&Elementwise
printtimeinterval_sec = 1.0
OutputMask = 1 1 1 1 1 1 1 1 1 1 1 1 !described herafter
refinement_strategy = 1 ! or 2
refinement = 1
/
```

33.3.1 printTimeInterval_Sec

- Output is generated every **printtimeinterval_sec**.

33.3.2 refinement

The **refinement** variable determines how the output mesh is created:

- refinement = 0 outputs a single triangle for each mesh cell. The unknowns are calculated at the center of each cell.
- refinement = 1 subdivides each triangle into 3 or 4 subtriangles, depending on the refinement_strategy. A higher refinement value will further subdivide each subtriangle.
- refinement_strategy = 1 divides each triangle into 3 triangles, all sharing the triangle barycenter as a node.
- refinement_strategy = 2 divides each triangle into 4 triangles.

33.3.3 OutputMask

The **OutputMask** variable allows you to select specific unknowns for visualization. You can toggle the output writing of each unknown by changing its corresponding bit in the OutputMask array.

Here's what each bit in the OutputMask array represents:

1. **SRs** and **SRd**: Slip rates in along-strike and along-dip directions
2. **T_s**, **T_d**: Shear stress in strike and dip directions, **P_n**: Normal stress
3. **u_n**: Fault normal velocity (Note: SeisSol does not allow for fault opening (mode I))
4. **Mud**: Current effective friction coefficient, **StV**: State variable for RS friction
5. **Ts0**,**Td0**,**Pn0**: Total shear and normal stresses, including initial stresses
6. **Sls** and **Sld**: Fault slip in along-strike and -dip directions
7. **Vr**: Rupture velocity, computed from the spatial derivatives of the rupture time
8. **ASl**: Accumulated slip
9. **PSR**: Peak slip rate
10. **RT**: Rupture time
11. **DS**: Dynamic stress time. With LSW, the time at which ASl>D_c. With RS, the time at which mu <= (f0 + mu_w). DS can be used to evaluate the process zone size.
12. **P_f** and **Tmp**: Only with thermal pressurization, pore pressure and temperature

33.4 Initial fault tractions

It is worth noticing that **Ts0**, **Td0**, and **Pn0** outputted at t=0s are the initial tractions after a first step through the friction routines. In particular, if the shear traction read in the input files locally exceeds fault strength, the outputted traction at t=0s is reduced compared with the one read in the input files. To visualize the initial shear tractions (or any other parameters, e.g. d_c) given in the easi file, the script read_ini_fault_parameter.py can be used. It depends on the easi python bindings.

```
./read_ini_fault_parameter.py output/data-fault.xdmf fault.yaml --ref_vector " -0.1,0,-1.
↪0"
```

33.5 seissolxdmf python module

You can read SeisSol ParaView files (XDMF/Hdf5 or XDMF/binary files, describing the fault outputs and the free-surface outputs and the volume wavefield outputs) using our Python module **seissolxdmf**. Find it on PyPi at: seissolxdmf.

33.6 Ascii fault receivers

To generate ASCII receiver files, configure the **Pickpoint** namelist as in this example:

```
&Pickpoint
printtimeinterval = 1
OutputMask = 1 1 1 1 1 1 1 1 1 1 1 1 !described herafter
nOutpoints = 24
PPFileName = 'fault_receivers.dat'
/
```

printtimeinterval determines how frequently the output is generated — every **printtimeinterval** (local) time step. Please note that using this output with local time-stepping may result in differently sampled receiver files.

33.6.1 OutputMask

This is the same as for the ParaView output.

CHAPTER

THIRTYFOUR

FREE SURFACE OUTPUT

34.1 Introduction

Velocities and ground deformations can be imaged as a surface representation, in a file that can be opened in ParaView. Threads or nodes can be dedicated to write this output (see *Asynchronous Output*), but it is usually not necessary. The output to enabled in the Output namelist:

```
&Output
SurfaceOutput = 1
SurfaceOutputRefinement = 1
SurfaceOutputInterval = 0.5
/
```

If `SurfaceOutputRefinement = 0`, one triangle is outputted for each mesh cell. The unknowns are evaluated at the center of each cell. `SurfaceOutputRefinement = 1` subdivides each triangle, into 4 subtriangles. Higher Surface-OutputRefinement would further subdivide each subtriangle.

34.2 variables

v1, **v2**, **v3**: ground velocities, x y and z components

u1, **u2**, **u3**: ground displacements, x y and z components

Additionally, the writer outputs a quantity called "locationFlag", which has the values 0 and 1 when at the elastic or acoustic side of an elastic-acoustic interface. In this way, we can distinguish between both sides of the interface even though they have the same coordinates. It has the value 2 for an ordinary free surface boundary condition and the value 3 for a free surface with gravity boundary condition. This value can be used to filter the output (which contains all these surfaces), for example using Paraview's Threshold filter.

CHAPTER

THIRTYFIVE

OFF FAULT RECEIVERS

35.1 Introduction

Ascii receivers are enabled using the namelist output. Here is a commented example:

```
&Output
pickdt = 0.01 ! Pickpoint Sampling
pickDtType = 1 ! Pickpoint Type
RFileName = 'receivers.dat' ! Record Points in extra file
/
```

If pickDtType = 2, the output is generated every N time steps, where N is set by pickdt. If pickDtType = 1, output is generated every pickdt second.

receivers.dat is an ASCII file describing the coordinates of the receivers in the form:

```
x1 y1 z1
x2 y2 z2
(...)
xn yn zn
```

The receivers files contain the time-histories of the stress tensor (6 variables) and the particle velocities (3). Currently, there is no way to write only a subset of these variables.

The variable `ReceiverOutputInterval` (in the section `Output` of the *Parameter File*) controls the frequency of flushing receiver time-histories. If not specified, they are written at the end of the simulation.

35.2 Rotational Output

You can additionally choose to write the rotation of the velocity field by setting `ReceiverComputeRotation=1` in the parameter file. The rotation of the vector field is defined as $\text{rot}v = \begin{pmatrix} \partial_2 v_3 - \partial_3 v_2 \\ \partial_3 v_1 - \partial_1 v_3 \\ \partial_1 v_2 - \partial_2 v_1 \end{pmatrix}$.

35.3 Placing free-surface receivers

Placing receivers on the free-surface requires special care when a realistic topography is used. The procedure to move receivers exactly to the surface is described here.

35.3.1 Compiling place_receivers on SuperMUC

Load the relevant *modules*.

```
git submodule update --init
mkdir build && cd build
cmake ..
make -j
```

CHAPTER

THIRTYSIX

POSTPROCESSING AND VISUALIZATION

To visualize the output of SeisSol, you can use ParaView with parallel rendering servers to distribute the workload. This chapter documents how to achieve this on the SuperMUC cluster. For other clusters, the approach might be similar.

36.1 Login to SuperMUC and start the pvservers

Connect to the SuperMUC via

```
ssh -Y hw.supermuc.lrz.de -l username
```

If you don't have access please see https://www.lrz.de/services/compute/supermuc/access_and_login/ for support.

To start the pvserver processes on the cluster use a jobscript to specify all the parameters. Here is an example of how this can look like (save as start_paraview.sh):

```
#!/bin/bash
##
###  optional: energy policy tags
##
##  DO NOT USE environment = COPY_ALL
#@ job_type = MPICH
#@ class = test
#@ node = 2
####  schedule the job to exactly 1 island
########## (more than one will fail)
#@ island_count=1
#@ tasks_per_node = 28
#@ wall_clock_limit = 0:30:00
#@ job_name = paraview
#@ network.MPI = sn_all,not_shared,us
#@ initialdir = $(home)/viz
#@ output = job.$(schedd_host).$(jobid).out
#@ error =  job.$(schedd_host).$(jobid).err
#@ queue
. /etc/profile
. /etc/profile.d/modules.sh
## setup of environment
module unload mpi.ibm
module load mpi.intel paraview/5.2.0_mesa
mpiexec -n 56 pvserver --use-offscreen-rendering
```

It is important to use a ParaView version that has been built with MPI and MESA support because usually the compute nodes in the cluster don't have graphics hardware and we want to use it in parallel. See https://www.paraview.org/Wiki/Setting_up_a_ParaView_Server for more information.

The script can be submitted via

```
llsubmit start_paraview.sh
```

You then have to wait a moment and check for the pvservers to be ready to connect.

```
tail -f job.srv23ib.831086.out
```

Please replace the job name by the one you got when you submitted the job.

If you can see something like

```
Waiting for client...
Connection URL: cs://i20r01c02s09:11111
Accepting connection(s): i20r01c02s09:11111
```

your pvserver is up and running and ready to connect. Check your running job with `llu` to get the complete ID of the leading compute node your job is running on, e.g. `i20r01c02s09ib`.

36.2 Setup remote visualization

If you have already used remote visualization of LRZ you can find a file `.vis_job.ll` in your home directory. Open it and modify it to use Intel MPI, i.e. set `#@ job_type = MPICH` instead of `#@ job_type = parallel`. It should look like this:

```
#!/bin/bash
#####  job_type = parallel
#@ job_type = MPICH
#@ class = vis
#@ node = 1
#@ tasks_per_node=28
#@ wall_clock_limit = 2:00:00
#@ network.MPI = sn_all,not_shared,us
#@ notification=never
#@ node_usage = not_shared
#@ island_count=1,1
#@ node_topology=island
#@ initialdir = .
#@ output = vncjob.$(schedd_host).$(jobid).out
#@ error = vncjob.$(schedd_host).$(jobid).err
#@ energy_policy_tag = testtag
#@ minimize_time_to_solution = yes
#@ notification=never
#@ node_resources = ConsumableMemory(16GB)
#@ queue
. /etc/profile . /etc/profile.d/modules.sh
hostname
/opt/TurboVNC/bin/vncserver -geometry 1280x900
sleep 48h
```

Submit the job with `llsubmit .vis_job.ll`.

Use `cat vncjob.srv23ib.831117.err` and look for something like this:

```
Desktop 'TurboVNC: vis01:2 (username)' started on display vis01:2
```

This tells you which node and which display you have to use for connecting with your VNC viewer. Start the VNC viewer with

```
vncviewer -via username@hw.supermuc.lrz.de vis01:2
```

Now you have a nice GUI on the visualization node. Open a Terminal and load the right modules:

```
module rm poe mpi.ibm
module load mpi.intel paraview/5.2.0
unset I_MPI_DEVICE
```

Start the ParaView client with `vglrun paraview`. Klick on `connect` and enter a new server. The host must be the leading compute node from above, in this example, it is `i20r01c02s09ib`. The port is `11111`. When you hit the connect button in the menu, you should have access to all the resources you asked for in your job script and are ready to open your data.

36.3 keyboard issue using vncviewer

A common problem is that the keyboard mapping gets all mixed-up after vncviewer windows is deselected. To avoid this problem, add in ~/.vnc/xstart before running vncviewer:

```
export XKL_XMODMAP_DISABLE=1
```

CHAPTER THIRTYSEVEN

WAVE FIELD OUTPUT

37.1 Introduction

The wavefield can be written in an hdf5 file, in order to visualize it in ParaView. To speed up the process, it is recommended to dedicate a few nodes to the writing tasks (see *Asynchronous Output*).

37.2 Refinement

0 (default): Refinement is disabled, i.e. only one cell is outputted for each element.
1: Refinement strategy is Face Extraction: 4 subcells per cell
2: Refinement strategy is Equal Face Area: 8 subcells per cell
3: Refinement strategy is Equal Face Area and Face Extraction: 32 subcells per cell
The unknowns are always evaluated at the center of the subcell.

37.3 iOutputMask

iOutputMask allows switching on and off the writing of SeisSol unknowns. The 6 first digits controls the components of the stress tensor (sigma_xx, sigma_yy, sigma_zz, sigma_xy, sigma_yz, and sigma_xz), and the 3 last digits the velocity components (u, v, w). When using poroelasticity, 4 more flags are added, for pore pressure (p) and fluid velocities (u_f, v_f, w_f).

37.4 iPlasticityMask

iPlasticityMask allows switching on and off the writing of plasticity variables. The 6 first digits controls the components of the off-fault plastic strain tensor (ep_xx, ep_yy, ep_zz, ep_xy, ep_yz, and ep_xz), and the last one the accumulated plastic strain (eta).

37.5 OutputRegionBounds

Using the OutputRegionBounds parameter, under the &Output heading, in the parameter.par file, the user can define the region for which the output is to be written. This region is provided in the following format:

```
OutputRegionBounds = xMin xMax yMin yMax zMin zMax
```

37.6 OutputGroups

Similar to the previous parameter, OutputGroups can be used to whitelist a set of mesh groups (as specified in the xdmf mesh file) that are included in the wavefield output. Cells whose group is not mentioned are not included in the output. This feature works with OutputRegionBounds, only cells that satisfy both criteria are included. It looks like this:

```
OutputGroups = 1 2 ! only include groups 1 and 2
```

37.7 Example

Here is an example of wavefield output parametrization:

```
&Output
OutputFile = '/output/prefix'
iOutputMask     = 0 0 0 0 0 0 1 1 1
iPlasticityMask = 0 0 0 0 0 0 1
OutputRegionBounds = -5e3 5e3 -10e3 10e3 -8e3 0e0
Format = 6                          ! Format (6=hdf5, 10= no output)
TimeInterval = 5.0                  ! Index of printed info at time
printIntervalCriterion = 2          ! Criterion for index of printed info: 1=timesteps,
↪2=time,3=timesteps+time
refinement = 1
/
```

CHAPTER

THIRTYEIGHT

CHECKPOINTING

Checkpointing consists of saving the simulation state at a given time, to allow restarting from that point in case of failure. It is parametrized within 'output' namelist of the parameter file by setting up the following variables:

```
checkPointFile = '../output/check'
checkPointBackend = 'mpio'
checkPointInterval = 0.4
```

checkPointFile defines the path and prefix to the chechpointfile.

checkPointBackend defines the implementation used ('posix', 'hdf5', 'mpio', 'mpio_async', 'sionlib', 'none'). If 'none' is specified, checkpoints are disabled. To use the HDF5, MPI-IO or SIONlib back-ends you need to compile SeisSol with HDF5, MPI or SIONlib respectively.

checkPointInterval defines the (simulated) time interval at which checkpointing is done. 0 (default value) disables checkpointing. When using an asynchronous back-end (mpio_async), you might lose 2 * checkPointInterval of your computation.

If the active checkpoint back-end finds a valid checkpoint during the initialization, it will load it automatically. (You cannot explicitly specify to load a checkpoint)

Hint: Currently only the output of the wavefield is designed to work with checkpoints. Other outputs such as receivers and fault output might require additional post-processing when SeisSol is restarted from a checkpoint.

38.1 Checkpointing Environment variables

The parallel checkpoint back-ends (HDF5, MPI-IO, SIONlib) support several tuning environment variables:

- **SEISSOL_CHECKPOINT_BLOCK_SIZE** Optimize the checkpoints for a specific file system block size. Set to 1 to disable the optimization. Set to -1 for auto-detection with the SIONlib back-end. (default: 1 (MPI-IO, HDF5) or -1 (SIONlib), MPI-IO, HDF5, SIONlib back-end only)
- **SEISSOL_CHECKPOINT_ROMIO_CB_READ** If set, the `romio_cb_read` in the MPI info object when opening the file. (default: no value, MPI-IO and HDF5 backend only)
- **SEISSOL_CHECKPOINT_ROMIO_CB_WRITE** See *SEISSOL_CHECKPOINT_ROMIO_CB_READ*
- **SEISSOL_CHECKPOINT_CB_NODES** See *SEISSOL_CHECKPOINT_ROMIO_CB_READ*
- **SEISSOL_CHECKPOINT_ROMIO_DS_READ** See *SEISSOL_CHECKPOINT_ROMIO_CB_READ*
- **SEISSOL_CHECKPOINT_ROMIO_DS_WRITE** See *SEISSOL_CHECKPOINT_ROMIO_CB_READ*

- **SEISSOL_CHECKPOINT_SION_BACKEND** The SIONlib back-end that should be used. Should be either *ansi* or *posix*. (default: 'ansi', SIONlib back-end only)
- **SEISSOL_CHECKPOINT_SION_NUM_FILES** Number of SIONlib files. (default: 1, SIONlib back-end only)
- **SEISSOL_CHECKPOINT_SION_COLL_SIZE** The collective size in SIONlib. Set to 0 for disabling collective operations. See SIONlib documentation for more details. (default: 0, SIONlib back-end only)
- **SEISSOL_CHECKPOINT_SION_COLL_MODE** The collective mode for SIONlib. Should be either *merge* or *normal*. See SIONlib documentation for more details. (default: 'merge', SIONlib back-end only)

CHAPTER

THIRTYNINE

ENERGY OUTPUT

39.1 Introduction

The energy output computes the energy of the simulation. It is divided into multiple parts:

- **Energy in the water layer (= acoustic medium, Ω_a)**
  - **Gravitational energy**
    $\int_{\Omega_a} \frac{1}{2}\rho g \eta^2 \, \mathbf{dx}$, with ρ the density, g the gravitational acceleration and η the sea-surface elevation.
  - **Acoustic energy**
    $\int_{\Omega_a} \frac{1}{2K} p^2 \, \mathbf{dx}$, with p the acoustic pressure and K the compressibility.
  - **Acoustic kinetic energy**
    $\int_{\Omega_a} \frac{1}{2}\rho v^2 \, \mathbf{dx}$, with ρ the density and v the velocity.
- **Energy in Earth Ω_e**
  - **Elastic kinetic energy**
    $\int_{\Omega_e} \frac{1}{2}\rho v^2 \, \mathbf{dx}$, with ρ the density and v the velocity.
  - **Elastic energy**
    $\int_{\Omega_e} \frac{1}{2}\epsilon_{ij}\sigma_{kl} \, \mathbf{dx}$, with ϵ_{ij} the strain tensor and σ_{ij} the stress tensor. It reduces for isotropic materials to $\int_{\Omega_e} \frac{1}{2\mu}(\sigma_{ij}\sigma_{ij} - \frac{\lambda}{3\lambda+2\mu}\sigma_{kk}^2) \, \mathbf{dx}$, with λ and μ the Lame coefficients.
- **Earthquake source energy**
  - **Total frictional work done by the stress change**
    $W_{\text{total}} = -\int_0^{t_f} \int_{\Sigma} \Delta\sigma(t) \cdot \Delta\dot{\mathbf{u}}(t) \, \mathbf{dx} dt$, with Σ the fault surface, $\Delta\sigma(t) = \sigma(t) - \sigma(0)$ the shear traction change, and $\Delta\dot{\mathbf{u}}(t)$ the fault slip rate, and t_f the end time of the simulation (see eq. 3 in Ma and Archuleta, 2006).
  - **Static frictional work done by the stress change**
    $W_{\text{static}} = -\int_{\Sigma} \frac{1}{2}\boldsymbol{\Delta}\sigma(t_f) \cdot \boldsymbol{\Delta}\mathbf{u}(t_f) \, \mathbf{dx}$ (see eq. 4 in Ma and Archuleta, 2006).
  - **Radiated energy can then be computed with:**
    $E_{\text{r}} = W_{\text{total}} - W_{\text{static}}$ (see eq. 5 in Ma and Archuleta, 2006).
- **Potency**
  $\int_{\Sigma} \Delta u_{\text{acc}}(t_f) \, \mathbf{dx}$, with Δu_{acc} the accumulated fault slip (scalar).
- **Seismic moment**
  $\int_{\Sigma} \mu \Delta u_{\text{acc}}(t_f) \, \mathbf{dx}$, with μ the second Lame coefficient.

- **Plastic moment**
  $\int_{\Omega_e} \mu\eta \, \mathbf{dx}$, with μ the second Lame coefficient and eta a scalar quantity measuring the accumulated material damage.

Currently, the output is only supported for the elastic wave equation.

39.2 Configuration

```
&Output
OutputFile = 'output/conv'
EnergyOutput = 1
EnergyTerminalOutput = 1
EnergyOutputInterval = 0.05
ComputeVolumeEnergiesEveryOutput = 4 ! Compute volume energies only once every␣
↪ComputeVolumeEnergiesEveryOutput * EnergyOutputInterval
/
```

39.2.1 Energy output

0 : no output
1 : csv output

For the example configuration, the output is written in the file "output/conv-energy.csv".

39.2.2 Terminal output

0 : no output
1 : additional output to stdout

Additionally, the energy can be written to stdout. This can be useful for debugging.

39.2.3 Output interval

The output interval is controlled by EnergyOutputInterval. If the output interval is not specified, the energy will be computed at the start of the simulation and at the end of the simulation.

39.2.4 Postprocessing and plotting

The code below suggests a way to process and plot variables of the energy output file:

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt

df = pd.read_csv("prefix-energy.csv")
df = df.pivot_table(index="time", columns="variable", values="measurement")
```

(continues on next page)

(continued from previous page)

```
df["seismic_moment_rate"] = np.gradient(df["seismic_moment"], df.index[1])
df.plot(y="seismic_moment_rate", use_index=True)

# if ComputeVolumeEnergiesEveryOutput > 1
volume_output = df.dropna()
volume_output.plot(y="elastic_energy", use_index=True)

plt.show()
```

CHAPTER

FORTY

PUML MESH FORMAT

SeisSol mesh format is xdmf + h5 file format. The h5 file contains the data (arrays) in binary format and the xdmf describes it. The content of an hdf5 file can be view using h5dump. The mesh file can be visualized using ParaView. The boundary conditions can be extracted from a PUML mesh and visualised in ParaView using this script.

here is an example of xdmf file (describing the hdf5 content):

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf Version="2.0">
 <Domain>
  <Grid Name="puml mesh" GridType="Uniform">
   <Topology TopologyType="Tetrahedron" NumberOfElements="901818">
    <DataItem NumberType="Int" Precision="8" Format="HDF" Dimensions="901818 4">
↪Sulawesi:/connect</DataItem>
   </Topology>
   <Geometry name="geo" GeometryType="XYZ" NumberOfElements="159618">
    <DataItem NumberType="Float" Precision="8" Format="HDF" Dimensions="159618 3">
↪Sulawesi:/geometry</DataItem>
   </Geometry>
   <Attribute Name="group" Center="Cell">
    <DataItem  NumberType="Int" Precision="4" Format="HDF" Dimensions="901818">Sulawesi:/
↪group</DataItem>
   </Attribute>
   <Attribute Name="boundary" Center="Cell">
    <DataItem NumberType="Int" Precision="4" Format="HDF" Dimensions="901818">Sulawesi:/
↪boundary</DataItem>
   </Attribute>
  </Grid>
 </Domain>
</Xdmf>
```

It shows that the hdf5 file consists of 4 arrays: geometry, connect, group and boundary.

- geometry contains the coordinates of the nodes. Dimension: (nNodes, 3)
- connect contains the volume cell connectivity. Dimension: (nCells, 4)
- group contains the region id of each volume cell (different properties can then be affected on different volumes). Dimension: nCells
- boundary contains the boundary conditions of each face of each volume cell. Dimension: nCells.

Each tetrahedron has 4 faces. In our format, the 4 boundary condition ids (4 bits each) are store within a single integer (16 bits) variable. The values can be unpacked, for example in python using:

```
for faceId in range(0,4):
   boundaryFace[faceId] = (boundary >> (faceId*8)) & 0xFF;
```

The possibles values of the boundary condition ids range from 0 to 255.
In SeisSol, boundary conditions are historically tagged as:
0: regular
1: free surface
3: dynamic rupture
5: absorbing
6: periodic
n>6: dynamic rupture (See *Fault tagging*)

Here is the convention defining the face id in a tetrahedron:

```
s_vert[0,:] = [0,2,1];   s_vert[1,:] = [0,1,3];    s_vert[2,:] = [1,2,3]; s_vert[3,:] =␣
↪[0,3,2];
```

CHAPTER

FORTYONE

ASAGI

The software package ASAGI can be used to map gridded simulation properties of the domain to the mesh used for a SeisSol simulation. ASAGI reads NetCDF files, which follow the COARDS Convention for netCDF files. This convention in particular states that:

> *"1-dimensional netCDF variables whose dimension names are identical to their variable names are regarded as "coordinate variables"."*

Here is an example of the structure of an ASAGI file (generated by *ncdump -h test_ASAGI.nc*), describing two 2D arrays (strike_slip and dip_slip) indexed by variables u and v.

```
netcdf test_ASAGI {
types:
  compound material {
    float strike_slip ;
    float dip_slip ;
  }; // material
dimensions:
        u = 40 ;
        v = 20 ;
variables:
        float u(u) ;
        float v(v) ;
        material data(v, u) ;
}
```

Typically, 1D, 2D, and 3D ASAGI files are used in SeisSol setups.

41.1 Installing ASAGI

Be careful that the python and gcc package is the same as for the compilation of SeisSol in a later step! First clone ASAGI with:

```
git clone git@github.com:TUM-I5/ASAGI
# git clone https://github.com/TUM-I5/ASAGI.git
cd ASAGI
git submodule update --init
```

Set compiler options, e.g. for intel compilers on SuperMUC:

```
export FC=mpif90
export CXX=mpiCC
export CC=mpicc
```

Run cmake, and compile with:

```
mkdir build && cd build
CMAKE_PREFIX_PATH=$NETCDF_BASE
cmake .. -DSHARED_LIB=no -DSTATIC_LIB=yes -DCMAKE_INSTALL_PREFIX=$HOME
make -j 48
make install
(Know errors: 1.Numa could not found - turn off Numa by adding -DNONUMA=on . )
```

41.2 building SeisSol with ASAGI support

Simply turn on the option `ASAGI=ON` in the using ccmake.

41.3 generating the NetCDF input file

41.3.1 using python

The most straightforward way to generate ASAGI file is to use the netCDF4 module of python. A typical example which generates a 2D ASAGI file can be found here.

41.3.2 using asagiconv

Asagiconv (Located here) allows querying data, vizualising and exporting to NetCDF data from the 3D Velocity Model for Southern California. For more detail, see ASAGI docu.

41.3.3 velocity models given as structured grids

Asagi expects a 1D, 2D, 3D (or higher dimensions) structured grid NetCDF files. Such files can be generated from an ASCII file using the command: `ncgen -b asagi_example.txt`
Here is a typical example for the ASCII file:

```
netcdf asagi_example {
types:
  compound material {
    float rho ;
    float mu ;
    float lambda ;
  }; // material
dimensions:
    x = 3 ; // Number of points in x-direction
    y = 2 ; // Number of points in y-direction
```

(continues on next page)

(continued from previous page)

```
    z = 1 ; // Number of points in z-direction
variables:
    float x(x) ;
    float y(y) ;
    float z(z);
    material data(z, y, x) ;
data:
  x = 2, 2.5, 3 ; // Grid points in x-direction (must have the same spacing)
  y = -1, 0 ; // Grid points in y-direction (must have the same spacing)
  z = 0 ; // Grid points in z-direction (must have the same spacing)

  data =
  {1, -1, 10}, // rho,mu,lambda for x0, y0, z0
  {2, -2, 11}, // rho,mu,lambda for x1, y0, z0
  {3, -3, 12}, // rho,mu,lambda for x2, y0, z0
  {4, -4, 13}, // rho,mu,lambda for x0, y1, z0
  {5, -5, 14}, // rho,mu,lambda for x1, y1, z0
  {6, -6, 15} ; // rho,mu,lambda for x2, y1, z0
}
```

41.4 SeisSol parameter file

A simple example file setting the elastic properties using EASI can be found here.

Such a file would be called adding in the namelist equation:

```
MaterialFileName = 101_asagi.yaml
```

In this example, the ASAGI file describes 2D arrays. The AffineMap is therefore needed to define the unit vectors used for indexing the 2D arrays. Note that the variables in the affine map can have different names than x, y or z (actually it should be preferred to avoid confusion). An AffineMap may also be used for 3D arrays, in case the coordinates variables are not aligned with the Cartesian coordinate system.

41.5 Further information

For further information, the use of asagiconv and asagi and its compilation, please see: ASAGI docu.

41.6 Known issues

There is a bug when using ASAGI with MPI. A workaround is described in https://github.com/SeisSol/SeisSol/issues/46.

CHAPTER

FORTYTWO

SYCL

SeisSol utilizes both native GPU and SYCL programming models. The former one is mainly used in performance critical parts of the application which typically contain some generated code. The latter one is used in places where we need to achieve GPU source code portability (e.g., Dynamic Rupture).

SYCL is a standard for cross-platform programming of heterogeneous processors. Below are the instructions for installing the *hipSYCL* implementation of the SYCL standard. If you want to use *DPC++* then refer to the followig instructions. You can find the latest successfully tested *DPC++* version here. Additionally to *DPC++*, you will need to compile *OpenMP* from the same repository for the host-side of the application. You can find the corresponding information there.

42.1 Installing LLVM

First download and prepare LLVM for building. Please, refer to the hipSYCL documentation regarding a specific version of LLVM. Here, we provide an example how to install hipSYCL using LLVM14. Note that unpacking the tar archive takes a long time.

```
wget https://github.com/llvm/llvm-project/archive/refs/tags/llvmorg-14.0.6.tar.gz
tar -xvf llvmorg-14.0.6.tar.gz
mkdir -p llvm-project-llvmorg-14.0.6/build && cd llvm-project-llvmorg-14.0.6/build
```

LLVM comes with many backends. The most common CPU ones are X86, ARM, PowerPC. Below are the instructions for installing LLVM with X86 and NVPTX backends for CPUs and GPUs, respectively.

```
export GCC_ROOT=$(dirname $(dirname $(which gcc)))
export CUDA_ROOT=${CUDA_PATH} # or CUDA_ROOT=$(dirname $(dirname $(which nvcc)))
cmake ../llvm -DCMAKE_BUILD_TYPE=Release \
-DLLVM_ENABLE_PROJECTS="clang;clang-tools-extra;compiler-rt;openmp;libunwind;polly" \
-DGCC_INSTALL_PREFIX="${GCC_ROOT}" \
-DCUDA_TOOLKIT_ROOT_DIR="${CUDA_ROOT}" \
-DCMAKE_INSTALL_PREFIX="${HOME}" \
-DLLVM_TARGETS_TO_BUILD="X86;NVPTX"
make -j $(nproc)
make install
cd ../..
```

Follow the instructions listed below if you need to configure LLVM with AMD GPU backend.

```
export GCC_ROOT=$(dirname $(dirname $(which gcc)))
export CUDA_ROOT=${CUDA_PATH} # or CUDA_ROOT=$(dirname $(dirname $(which nvcc)))
cmake ../llvm -DCMAKE_BUILD_TYPE=Release \
```

(continues on next page)

(continued from previous page)

```
-DLLVM_ENABLE_PROJECTS="clang;clang-tools-extra;compiler-rt;openmp;polly" \
-DGCC_INSTALL_PREFIX="${GCC_ROOT}" \
-DCMAKE_INSTALL_PREFIX="${HOME}" \
-DLLVM_TARGETS_TO_BUILD="X86;AMDGPU"

make -j $(nproc)
make install

cd ../..
```

42.2 Installing Boost

hipSYCL depends on 1.69.0 version of the Boost library.

```
wget https://boostorg.jfrog.io/artifactory/main/release/1.69.0/source/boost_1_69_0.tar.gz
tar -xvf ./boost_1_69_0.tar.gz
cd boost_1_69_0

GCC_EXEC=$(which g++)
./bootstrap.sh --prefix="${HOME}" \
--with-toolset=gcc \
--with-libraries=serialization,wave,date_time,iostreams,locale,math,random,context,regex,
↪program_options,atomic,timer,log,fiber,chrono,thread,exception,system,test,graph,
↪filesystem

echo "using gcc : : ${GCC_EXEC} ;" > user-config.jam

./b2 install threading=multi variant=release toolset=gcc link=shared \
cxxflags="-std=c++17" visibility=hidden -j $(nproc) --user-config="user-config.jam"

cd ..
```

42.3 Installing hipSYCL

SeisSol requires 0.9.3 version of hipSYCL for a correct multi GPU-setup.

```
git clone --depth 1 --branch v0.9.3 https://github.com/illuhad/hipSYCL.git
cd hipSYCL
mkdir build && cd build
```

Perform the following steps to configure and install hipSYCL for Nvidia GPUs. Make sure that the clang from the correct LLVM installation is used and check the paths carefully.

```
export CUDA_PATH=$CUDA_HOME
export CLANG_DIR=$(dirname $(dirname $(which clang)))
export CLANG_EXE=$(which clang++)

CC=gcc CXX=g++ cmake .. \
```

(continues on next page)

(continued from previous page)

```
-DCMAKE_BUILD_TYPE:STRING=Release \
-DCMAKE_INSTALL_PREFIX="${HOME}" \
-DWITH_CPU_BACKEND:Bool=TRUE \
-DWITH_ROCM_BACKEND:Bool=FALSE \
-DWITH_CUDA_BACKEND:Bool=TRUE \
-DWITH_ACCELERATED_CPU=OFF \
-DLLVM_DIR:String="${CLANG_DIR}/lib/cmake/llvm" \
-DCLANG_INCLUDE_PATH:String="${CLANG_DIR}/lib/clang/12.0.0/include" \
-DCLANG_EXECUTABLE_PATH:String="${CLANG_EXE}" \
-DCUDA_TOOLKIT_ROOT_DIR:String="${CUDA_PATH}" \
-DBoost_NO_BOOST_CMAKE=TRUE \
-DBoost_NO_SYSTEM_PATHS=TRUE \
-DBOOST_ROOT:PATHNAME="${HOME}" \
-DBoost_LIBRARY_DIRS:FILEPATH="${HOME}/lib"

make -j $(nproc)
make install

cd ../..
```

The following steps describe the steps to configure and install hipSYCL for AMD GPUs. Note *ROCM_PATH* is typically set by system administrators. Please, makes sure that this environment variable is not empty.

```
export CLANG_DIR=$(dirname $(dirname $(which clang)))
export CLANG_EXEC=$(which clang++)

cmake .. -DCMAKE_BUILD_TYPE:STRING=Release \
-DCMAKE_INSTALL_PREFIX="${HOME}" \
-DWITH_CPU_BACKEND:Bool=TRUE \
-DWITH_ROCM_BACKEND:Bool=TRUE \
-DWITH_CUDA_BACKEND:Bool=FALSE \
-DWITH_ACCELERATED_CPU=OFF \
-DLLVM_DIR:String="${CLANG_DIR}/lib/cmake/llvm" \
-DCLANG_INCLUDE_PATH:String="${CLANG_DIR}/lib/clang/12.0.0/include" \
-DCLANG_EXECUTABLE_PATH:String="${CLANG_EXEC}" \
-DROCM_PATH:String="${ROCM_PATH}" \
-DBoost_NO_BOOST_CMAKE=TRUE \
-DBoost_NO_SYSTEM_PATHS=TRUE \
-DBOOST_ROOT:PATHNAME="${HOME}" \
-DBoost_LIBRARY_DIRS:FILEPATH="${HOME}"/lib

make -j $(nproc)
make install

cd ../..
```

Add the following during the CMake configuration step if you want to enable the OpenMP backend of SYCL device kernels: *-DWITH_ACCELERATED_CPU=ON*

CHAPTER

FORTYTHREE

COMPUTING TIME VS ORDER OF ACCURACY

Using a higher order implies using smaller time steps and using more basis functions. The expected increase in computed time can then be estimated as follow:

Decrease of the time steps width by a factor: $(2N+1)/(2n+1)$
Increase of the number of basis functions by a factor: $(N+1)(N+2)(N+3)/((n+1)(n+2)(n+3))$
with:
n: initial order of accuracy -1
N: new order of accuracy -1

Here is an example of the theoretical increase of the compute time relative to order 3

	3	4	5	6	7	8	9	10
Increase due to time steps	1	1.4	1.8	2.2	2.6	3	3.4	3.8
Increase due to number of basis functions	1	2	3.5	5.6	8.4	12	16.5	22
theoretical increase relative to order 3	1	2.8	6.3	12.32	21.84	36	56.1	83.6
observed increase (on SM2)	1	2.0	4.6	11.5				

The last line shows the observed time increase on SuperMUC Phase 2 on a small run with dynamic rupture and LTS-DR. Low order calculation (up to order 5 included) are memory bounds, and are then less efficient. As a consequence, the higher-order simulations cost less than expected in comparison with order 3.

CHAPTER

FORTYFOUR

PERFORMANCE MEASUREMENT

44.1 Simulation time

SeisSol currently prints the total simulation time twice via stdout (i.e. to the terminal log), with mostly similar values. The first value is in the line starting with “Elapsed time (via clock_gettime)”, the second value is printed in the line starting with “Time spent in simulation:”, just one line before we see “Simulation done.”

The “total time spent in compute kernels” denotes the time in which all CPUs execute some function to advance the computation of the solution. In particular, it excludes the time spent with MPI communication.

44.2 FLOP/s counter

SeisSol outputs two FLOP and FLOP/s numbers:

- Non-zero FLOP (NZ-FLOP, and NZ-FLOP/s) denote the number of FLOP done towards computing the solution. That is, the number of additions and multiplications while considering the à priori-known sparsity patterns of all involved matrices (thus, e.g. multiplying by a diagonal matrix will be counted as much as a vector product). Note that the degrees of freedom are always assumed to be fully dense.
- Hardware FLOP (HW-FLOP, and HW-FLOP/s) denote the amount of FLOP actually calculated by the CPU. That is, every time we issue an instruction to add or multiply two values, we add a FLOP to the HW-FLOP counter.

Thus, the NZ-FLOP bounded from above by the HW-FLOP. In fact, we could write the HW-FLOP as sum of the NZ-FLOP and the zero-FLOP, i.e. operations where one of the operands is known to be 0 before running the simulation. This is mostly due to two reasons:

- We do not use the sparsity pattern of a matrix, even though we know it in advance. That can be the case if approximating the sparse matrix by a dense matrix can be implemented more efficiently in hardware by a code generator. Where that may be necessary if usually determined by running an auto tuner (i.e. SeisSol Proxy). By the approximation as a dense matrix, we include some operations where we know one of the operands is to be zero; thus we know the result in advance—therefore these operations are included in the HW-FLOP, but not the NZ-FLOP.
- Padding a matrix, e.g. to match the size of the underlying SIMD registers. The additional values are known to be zero; thus they are only included into the HW-FLOP, not the NZ-FLOP.

From the latter point, it follows that only the HW-FLOP depend on the underlying hardware. To note, both the HW-FLOP and the NZ-FLOP depend on the scenario and the equation system to be solved.

During the simulation (at synchronization points), we only print the HW-FLOP/s. After the simulation has finished, we print both the HW-GFLOP and the NZ-GFLOP, as well as HW-GLOP/s and NZ-GFLOP/s.

Note that the Dynamic Rupture computation or the Point Sources both are _not_ counted into the HW-/NZ-FLOP numbers at the moment; only the matrix operations do (as used, e.g., during the ADER computation).

44.3 Performance

One important value which we usually publish in our papers is the performance in "GFLOP/s per node". It comes in the flavor of our two metrics introduced in the previous section:

$$\text{HW-GFLOP/s per node} = \frac{\text{HW-GFLOP}}{\text{nodes} \cdot \text{elapsed-time}}$$

$$\text{NZ-GFLOP/s per node} = \frac{\text{NZ-GFLOP}}{\text{nodes} \cdot \text{elapsed-time}}$$

You can compare these values with the publications in order to see if your performance is ok.

Note that, empirically, the "HW-GFLOP/s per node" performance metric is used more often.

CHAPTER

FORTYFIVE

ATTENUATION

45.1 Introduction

- Q is measuring the decay of wave amplitude during its propagation in the medium and is physically defined as the ratio of wave energy to the energy dissipated per cycle of oscillation.
- The actual dissipation mechanisms, as scattering, heat generation, frictional losses in the vibrating crystal lattice, etc., "causing Q" can be manifold and do not have to follow similar physics.
- Q is not very frequency-dependent over a large range of (lower) frequencies but observations are ambiguous at higher frequencies.
- Common choices are $Q_s \sim 0.5Q_p$ and $Q_s \sim 40 - 50V_s$ (in km/s) (e.g. Olsen et al., 2009). $Q_s = 100 - 200$ for shallow sediments.
- In damaged fault zones Q_s would be expected to be as low as on the order of 10 and potentially smear out slip rate on the fault without affecting rupture speeds.

45.2 Implementation

- Implementing attenuation in SeisSol follows the ideas of a general Maxwell or Zener body.
- Each damping mechanism can be parametrized by its own relaxation frequency.
- It aims at resolving a frequency-independent Q with an adequate number of anelastic functions and relaxation frequencies to cover the frequency range under interest. Usually, 3 Maxwell bodies are enough for 5% error.

45.3 Stability with Local time stepping

The maximum timestep may need to be decreased in SeisSol to avoid stability issues when using attenuation and local time stepping. Practically, we found that if the maximum timestep is below $0.25T_3$ with $T_3 = 1/f_3 = 1/(\text{FreqCentral}\sqrt{\text{FreqRatio}})$, stability should be ensured. This is done by default by SeisSol when attenuation is turned on, and the parameter `FixTimeStep` of the the `&Discretization` namelist (main parameter file) is not set. If SeisSol is yet unstable, further decrease of the maximum timestep can be tried by manually setting the value of `FixTimeStep`.

45.4 Compiling

In ccmake, use:

```
EQUATIONS                         viscoelastic2
NUMBER_OF_MECHANISMS              3
```

Note that the equations='viscoelastic' is operational but deprecated.

45.5 Dispersion

The attenuation implementation implies dispersion in P and S wave velocities: that why we need to define a central frequency at which V_p/V_s are exact. In addition, the effective Q values are not exactly equal to the desired Q, but are oscillating around those values. The variation of V_p, V_s, Q_p and Q_s with frequency can be visualized using ViscoelasticModComp.m.

45.6 Parametrisation

Add Inside the parameter file of SeisSol, in the '&equations' section (frequencies values to be adapted to the source frequency content):

```
FreqCentral=0.5
FreqRatio=100
```

The spatial variation of Q_s and Q_p are defined with easi in the MaterialFileName. Here is an example of easi file, in which Q_s and Q_p are directly related to the shear wave speed V_s:

```
!ASAGI
file: ../material/vmodel_500.nc
parameters: [rho, mu, lambda]
var: data
components: !FunctionMap
  map:
    rho:    return rho;
    mu:     return mu;
    lambda: return lambda;
    Qs:     return 0.05 * sqrt(mu/rho);
    Qp:     return 0.1 * sqrt(mu/rho);
```

45.7 FreqCentral and FreqRatio

The relaxation frequencies are logarithmically equispaced, i.e.

$$\log(f_{i+1}) - \log(f_i) = \text{constant}.$$

In the parameter file, one has to give a frequency ratio of maximum to minimum frequency and a central frequency. For example, in the case of 3 mechanisms the following relations define the relaxation frequencies:

$$f_2 = \text{FreqCentral}$$

$$\log(f_3) - \log(f_2) = \log(f_2) - \log(f_1)$$

$$f_3 / f_1 = \text{FreqRatio}$$

This leads to $f_1 = \text{FreqCentral}/\sqrt{\text{FreqRatio}}$ and $f_3 = \text{FreqCentral}\sqrt{\text{FreqRatio}}$.

Outside of the frequency band $f_1 - f_3$, Q goes to infinity, yielding elastic behavior.

CHAPTER

FORTYSIX

PHYSICAL MODELS

46.1 Overview

SeisSol includes various physical models to simulate realistic earthquake scenarios. SeisSol assumes constant material values per element. By default, the elastic and viscoelastic materials are sampled by averaging over the whole element (c.f. https://mediatum.ub.tum.de/node?id=1664043). You can turn this feature off, by setting `UseCellHomogenizedMaterial = 0` in the *Parameter File* to fall back to sampling at the element barycenter. Anisotropic and poroelastic materials are never averaged and always sampled at the element barycenter.

46.1.1 Elastic

This is the standard model in SeisSol and it implements isotropic elastic materials. The constitutive behaviour is $\sigma_{ij} = \lambda\delta_{ij}\epsilon_{kk} + 2\mu\epsilon_{ij}$ with stress σ and strain ϵ. Elastic materials can be extended to elastoplastic materials (see *SCEC TPV13*).

46.1.2 Anisotropic

This is an extension of the elastic material, where direction-dependent effects also play a role. The stress strain relation is given by the $\sigma_{ij} = c_{ijkl}\epsilon_{kl}$. Whereas isotropic materials are described by two material parameters, the tensor c has 81 entries. Due to symmetry considerations there are only 21 independent parameters, which have to be specified in the material file: `c11`, `c12`, `c13`, `c14`, `c15`, `c16`, `c22`, `c23`, `c24`, `c25`, `c26`, `c33`, `c34`, `c35`, `c36`, `c44`, `c45`, `c46`, `c55`, `c56`, `c66`. For more details about the anisotropic stiffness tensor, see: https://en.wikipedia.org/wiki/Hooke%27s_law#Anisotropic_materials. All parameters have to be set, even if they are zero. If only the two Lamé parameters are provided, SeisSol assumes isotropic behaviour.

Example of an material file for a homogeneous anisotropic fullspace. The material describes the tilted transversally isotropic medium from chapter 5.4 in https://link.springer.com/chapter/10.1007/978-3-030-50420-5_3.

```
!ConstantMap
  map:
    rho:  2590
    c11:  6.66000000000000e10
    c12:  2.46250000000000e10
    c13:  3.44750000000000e10
    c14: -8.53035022727672e9
    c15:  0.0
    c16:  0.0
    c22:  6.29062500000000e10
    c23:  3.64187500000000e10
```

(continues on next page)

(continued from previous page)

```
    c24:  4.05949408023955e9
    c25:  0.0
    c26:  0.0
    c33:  4.95562500000000e10
    c34:  7.50194506028270e9
    c35:  0.0
    c36:  0.0
    c44:  7.91875000000000e9
    c45:  0.0
    c46:  0.0
    c55:  1.40375000000000e10
    c56:  5.43430940874735e9
    c66:  2.03125000000000e10
```

Anisotropy together with plasticity and dynamic rupture is not tested yet. You can define a dynamic rupture fault embedded in an isotropic material and have anisotropic regions elsewhere in the domain.

46.1.3 Poroelastic

In poroelastic materials a fluid and a solid phase interact with each other. The material model introduces the pressure p and the relative fluid velocities u_f, v_f, w_f to the model, so that we observe 13 quantities in total. A poroelastic material is characterised by the following material parameters:

Parameter	SeisSol name	Abbreviation	Unit
Solid Bulk modulus	`bulk_solid`	K_S	Pa
Solid density	`rho`	ρ_S	$kg \cdot m^{-3}$
Matrix 1^{st} Lamé parameter	`lambda`	λ_M	Pa
Matrix 2^{nd} Lamé parameter	`mu`	μ_M	Pa
Matrix permeability	`permeability`	κ	m^2
Matrix porosity	`porosity`	ϕ	
Matrix tortuosity	`tortuosity`	T	
Fluid bulk modulus	`bulk_fluid`	K_F	Pa
Fluid density	`rho_fluid`	ρ_F	$kg \cdot m^{-3}$
Fluid viscosity	`viscosity`	ν	$Pa \cdot s$

The implementation of poroelasticity is tested for point sources, material interfaces and free-surfaces. Plasticity and dynamic rupture together with poroelasticity are not tested.

46.1.4 Viscoelastic

Viscoelasticity is used to model the dissipation of wave energy over time. A full documentation can be found in *Attenuation*.

CHAPTER

FORTYSEVEN

SCALING

When working with SI units in earthquake scenarios numbers might get very large. Rescaling the equations might be advantageous, e.g. when working with single precision arithmetic. In this section, we show how to properly scale the elastic wave equation.

The elastic wave equation in velocity-stress form with source terms s_i and f_i is given by

$$\begin{aligned}\frac{\partial \sigma_{ij}}{\partial t} - \lambda\delta_{ij}\frac{\partial u_k}{\partial x_k} - \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) &= s_i \\ \rho\frac{\partial u_i}{\partial t} - \frac{\partial \sigma_{ij}}{\partial x_j} &= f_i\end{aligned}$$

We define scaled quantities

$$\bar{x}_i = \frac{x_i}{L}, \quad \bar{u}_i = \frac{u_i}{u_c}, \quad \bar{\sigma}_{ij} = \frac{\sigma_{ij}}{\sigma_c},$$

where L, u_c, σ_c are *scaling constants*. Inserting these into the elastic wave equation gives

$$\begin{aligned}\sigma_c\frac{\partial \bar{\sigma}_{ij}}{\partial t} - u_cL^{-1}\lambda\delta_{ij}\frac{\partial \bar{u}_k}{\partial \bar{x}_k} - u_cL^{-1}\mu\left(\frac{\partial \bar{u}_i}{\partial \bar{x}_j} + \frac{\partial \bar{u}_j}{\partial \bar{x}_i}\right) &= s_i \\ \rho u_c\frac{\partial \bar{u}_i}{\partial t} - L^{-1}\sigma_c\frac{\partial \bar{\sigma}_{ij}}{\partial \bar{x}_j} &= f_i\end{aligned}$$

Multiplying the first equation with σ_c^{-1}, multiplying the second equation with $L\sigma_c^{-1}$, and defining

$$\begin{aligned}\bar{\rho} = L\sigma_c^{-1}u_c\rho, \quad \bar{\lambda} = \sigma_c^{-1}u_cL^{-1}\lambda, \quad \bar{\mu} &= \sigma_c^{-1}u_cL^{-1}\mu, \\ \bar{f}_i = L\sigma_c^{-1}f_i, \quad \bar{s}_i &= \sigma_c^{-1}s_i\end{aligned}$$

leads to

$$\begin{aligned}\frac{\partial \bar{\sigma}_{ij}}{\partial t} - \bar{\lambda}\delta_{ij}\frac{\partial \bar{u}_k}{\partial \bar{x}_k} - \bar{\mu}\left(\frac{\partial \bar{u}_i}{\partial \bar{x}_j} + \frac{\partial \bar{u}_j}{\partial \bar{x}_i}\right) &= \bar{s}_i \\ \bar{\rho}\frac{\partial \bar{u}_i}{\partial t} - \frac{\partial \bar{\sigma}_{ij}}{\partial \bar{x}_j} &= \bar{f}_i\end{aligned}$$

47.1 Example

We change units with the scaling constants

$$L = 10^3, \quad u_c = 1, \quad \sigma_c = 10^6$$

In the rescaled equations, the spatial dimension of the mesh is [km], velocities are in [m/s], and stresses are in [MPa]. Parameters and source terms are scaled with

$$\bar{\rho} = 10^{-3}\rho, \quad \bar{\lambda} = 10^{-9}\lambda, \quad \bar{\mu} = 10^{-9}\mu, \quad \bar{f}_i = 10^{-3} f_i, \quad \bar{s}_i = 10^{-6} s_i$$

The wave speeds are given in [km/s]:

$$\bar{c}_s = \sqrt{\frac{\bar{\mu}}{\bar{\rho}}} = 10^{-3} c_s, \quad \bar{c}_p = \sqrt{\frac{\bar{\lambda} + 2\bar{\mu}}{\bar{\rho}}} = 10^{-3} c_p,$$

47.2 Acoustic-elastic with water layer

The acoustic wave equation is a special case of the elastic wave equation, therefore the scaling of parameters is identical. However, when using a water layer one needs to adjust gravitational acceleration in the *parameters file*:

```
GravitationalAcceleration = 0.00981
```

CHAPTER

FORTYEIGHT

BASIC CODE STRUCTURE

48.1 src/

Folder	Description
Checkpoint	Code related to checkpointing implementation, which allows to restart a simulation after failure.
Equations	Model-specific code.
Geometry	Everything related to reading tetrahedral meshes and setting up geometry information.
Initializer	Code that is called during initialization, e.g. allocating memory, setting up matrices, parsing material information.
Kernels	Common kernel code.
Model	Common model code.
Modules	Modules system implementation which allows adding code at pre-defined hooks.
Monitoring	Contains code for HPC statistics collected during a run.
Numerical_aux	Helper code for numerics, e.g. quadrature rules.
Parallel	MPI communicator related code.
Physics	Contains friction laws.
Reader	Code for reading parameter files.
ResultWriter	Fault, element, and surface output.
Solver	Time-stepping and code executed during a simulation.
SourceTerm	Everything related to kinematic rupture models.
tests	Unit tests.

48.2 preprocessing/

Folder	Description
meshing	Cube generator; gmsh converter; various scripts.
partitioning	*deprecated*
science	ASAGI converter; standard rupture format converter; various scripts.
workflow	*deprecated*

48.3 postprocessing/

Folder	Description
performance	*deprecated*
science	Various scripts processing simulation output.
validation	Cube mesh validation.
visualisation	Receiver viewer; scripts.

CHAPTER

FORTYNINE

KNOWN ISSUES

49.1 Download

- **The submodule directories are empty.** To clone the code including all submodules, use

  ```
  git clone --recursive https://github.com/SeisSol/SeisSol.git
  ```

  If you already cloned the repository without `--recursive`, you can use the following command to get the submodules.

  ```
  git submodule update --init --recursive
  ```

49.2 Installation

The build process fails because of missing files. SeisSol requires some external libraries which are integrated via git submodules. For some of the libraries it is important to have the correct version (e.g. the XMDF Writer) because of changes in the interface of the library. If you clone the submodules with git, you should always get the correct version. However, downloading the SeisSol and its submodules directly from the Github homepage might result in an incorrect combination of versions.

49.3 Asynchronous output

Some MPI versions have issues with threads and I/O. The MPI-IO implementation is either not threadsafe or too strict when it comes to locking. This can lead to crashes or deadlocks when using the asynchronous output. A simple workaround is to use the POSIX back-end of the XDMF writer, this is compiling SeisSol without HDF5. For check-pointing, one should also switch to POSIX or SIONlib.

49.4 easi and Intel/16.0

Using Intel/16.0 in a local cluster, compile error occurs in PUMLreader.cpp Using Intel/17.0 will solve this.

Using Intel/16.0 on SuperMUC, we observed some bugs in the fault stress initialization (you can see the imprint of some partitions in the initial stress). The bugs are not showing off with latest Intel/17.0 module.

49.5 “Holes” in the fault output

“Holes” in the fault output may appear if the reference point or reference vector, defined by (Xref, Yref, and Zref) is wrongly located (point exactly on the fault, or null vector).

CHAPTER

FIFTY

BREAKING CHANGES IN BACKWARD COMPATIBILITY

To keep up-to-date with changes in compute-centers and geoscientists' needs, breaking changes sometimes needed to be introduced in SeisSol. They are listed here.

50.1 Energy Output

Since we merged GitHub pull request #531 (April 2022), the seismic moment time history output, from which the moment rate can be post-processed, is integrated into the energy output (see *Energy output*). Therefore the parameters *magnitude_output_on*, *energy_rate_output_on* and *energy_rate_printtimeinterval* have been removed from the *DynamicRupture* namelist in the main parameter file.

50.2 C++ dynamic rupture implementation (dr/cpp)

While porting dynamic rupture to C++, we changed a few parameter names to make things more consistent. The new dynamic rupture implementation has been merged in September 2022 (GitHub pull request #625). The linear slip weakening friction laws FL=2 (nucleation by stress increase) and FL=16 (forced time rupture nucleation) have been merged (the new friction law is FL=16). Because of this change, FL=16 now requires nucleation stress or tractions to be specified in the fault-specific yaml file.

Parameter file (*parameters.par*):

old	new
0d0	*0.0*
v_star	*pc_vStar*
L	*pc_prakashLength*
mu_w	*rs_muW*
alpha_th	*tp_thermalDiffusivity*
rho_c	*tp_heatCapacity*
tp_lambda	*tp_undrainedTPResponse*
initemp	*tp_iniTemp*
inipressure	*tp_iniPressure*

Fault-specific yaml file (*fault.yaml*):

old	new
RS_sl0	*rs_sl0*
alpha_hy	*tp_hydraulicDiffusivity*
TP_half_width_shear_zone	*tp_halfWidthShearZone*
Ts0	*T_s*
Td0	*T_d*
Pn0	*T_n*

CHAPTER

FIFTYONE

SIMMODELER CAD WORKFLOW

In this section, we illustrate the SimModeler CAD workflow by building the structural model of the Palu earthquake dynamic rupture scenario (Ulrich et al., 2019). We strongly rely on the discrete toolbox of SimModeler, available since September 2019, and on additional python scripts. We generate a 3d model, incorporating topography and faults geometry.

51.1 prerequisite

See *Prerequisite*.

We use scripts available here to generate surface meshes of the faults. The dataset for generating the Palu structural model is available here.

51.2 Creating the topographic layer

We create the topography from a netcdf file downloaded from https://www.gebco.net/. The domain range from longitude 118.9 to 121.7 and from latitude -2.4 to 1.0. We then project the data (see *On the use of projections* for the choice of a projection), triangulate it, and export it as stl (list of triangles) using:

```
python3 SeisSol/Meshing/creating_geometric_models/create_surface_from_rectilinear_grid.
↪py --proj '+init=EPSG:23839' data/GEBCO_2014_2D_118.1904_-2.4353_121.6855_1.0113.nc␣
↪bathy.stl
```

We then load the stl file into SimModeler

```
File>import discrete data>bathy.stl
```

The option `Find Edges by Face Normal` allows isolating groups of triangles using the relative angle between their face normals. If the angle value is small, the imported surface will appear as divided into many faces. These faces must then be explicitly accounted for by the mesh. If some faces are tiny, this will mechanically lead to small cells in the final mesh. We, therefore, recommand to use a large enough value.

51.3 Creating the domain box

We generate a surface mesh of a simple box using gmsh:

```
gmsh -2 create_box.geo -format stl
```

The box dimensions are such as the topography is slightly wider than the box. The mesh size is chosen small enough to facilitate intersection with topography and large enough to limit the number of elements.

```
mesh_size = 10e3;
Xmax = -160e3;
Xmin = 215e3;
Ymin = 1235e3;
Ymax = 1605e3;
Zmin=-200e3;
Zmax=5e3;
```

51.4 Creating faults

We use this script to generate surface meshes of the faults, using inferred fault traces and dip description. The scripts first resample the 2D fault trace and then can smooth them. Finally, the smoothed and resampled traces are swept towards negative and positive (if the topography has positive elevation) z.

create_fault_from_trace.py takes 3 main arguments: `filename`, `dipType` and `dipDesc`.

- `filename` is the name of the ASCII file describing the trace.
- `dipType` allow switching between a constant (0), an along-depth dependant (1) or an along-strike dependent (2).
- `dipDesc` gives either the dip angle value (dipType=0) or the 1D variation of the dip angle (dipType=1 or 2).

In the Palu case example, the Southern segment dips 90, the Northern segment dips 65, and the middle segment has a varying dip along strike (shallower dip in the southern bend). We therefore generate the faults using:

```
dx=0.5e3
python3 SeisSol/Meshing/creating_geometric_models/create_fault_from_trace.py SeisSol/
↪Meshing/creating_geometric_models/ExampleFiles/SimModeler_workflow/segmentSouth_d90_
↪long.dat 0 90 --dd $dx --maxdepth 16e3 --extend 4e3
python3 SeisSol/Meshing/creating_geometric_models/create_fault_from_trace.py SeisSol/
↪Meshing/creating_geometric_models/ExampleFiles/SimModeler_workflow/smootherNorthBend.
↪dat 0 65 --dd $dx --maxdepth 16e3 --extend 4e3
python3 SeisSol/Meshing/creating_geometric_models/create_fault_from_trace.py SeisSol/
↪Meshing/creating_geometric_models/ExampleFiles/SimModeler_workflow/
↪segmentBayAndConnectingFault.dat 2 SeisSol/Meshing/creating_geometric_models/
↪ExampleFiles/SimModeler_workflow/segmentBayAndConnectingFaultDip.dat --dd $dx --
↪maxdepth 16e3 --extend 4e3
```

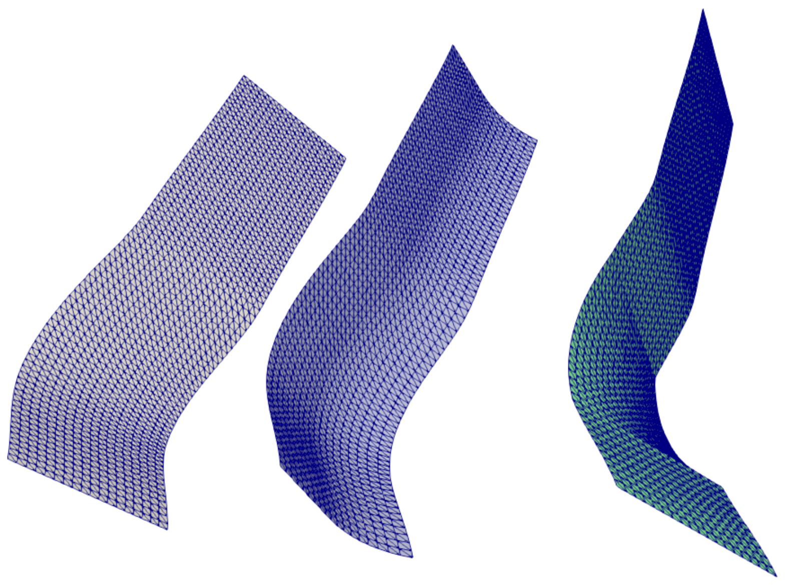


Fig. 1: Fig. 1: example of surface mesh generated using create_fault_from_trace.py and dipType=1 (constant dip, left), 2 (along-depth dependant dip, center), and 3 (along-strike dependent dip, right).

51.5 Mutual surface intersection

SimModeler requires a surface mesh representation of the structural model in which intersection between surfaces (e.g. faults, geologic layers, topography) are explicitly meshed for generating a 3D mesh. Historically, we used the 'Mutual surface intersection' feature of Gocad to intersect the surfaces. This workflow had several drawbacks. First, Gocad is an expensive software. Also, the 'Mutual surface intersection' algorithm is not highly fast and reliable. Sometimes, problems occur such as holes in the surfaces or small features in the generated surfaces, yielding tiny elements in the mesh (and small timesteps, see e.g. *Manually fixing an intersection in Gocad*). Here we use the tools for processing discrete data recently incorporated to SimModeler, which has proven to be superior to GoCAD in intersecting large datasets without artifacts. We first load all structural surfaces (*.stl and *.ts) using:

```
File>import discrete data> filenames
```

When then intersect these datasets using the Discrete tab of SimModeler:

```
Discrete>Union parts Add selected (green +), set tolerance 0.1 (e.g.), apply
```

Finally, we remove the part of the surface we do not want in the model using:

```
Discrete>Delete. Apply to delete the surface parts that are not needed.
```

This yields a self-intersecting surface representation of the structural model that can be easily meshed.

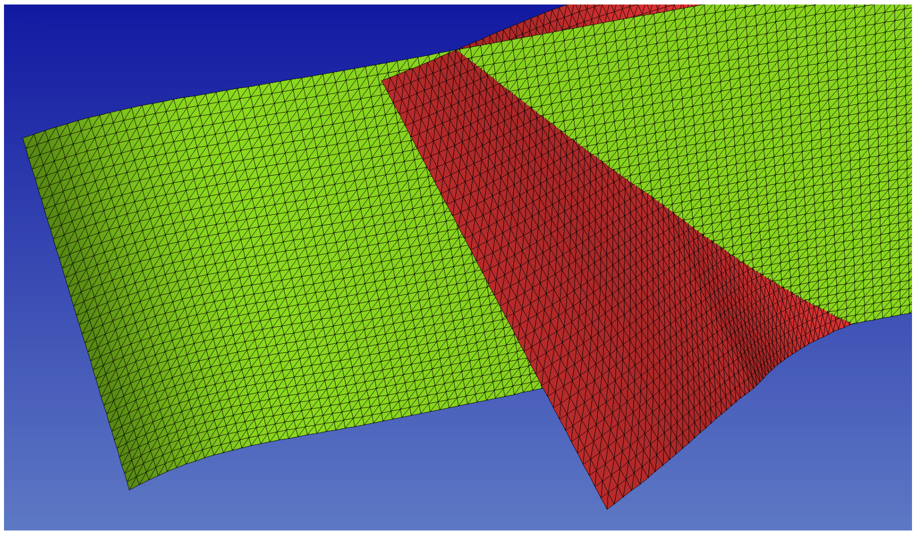


Fig. 2: Fig. 2: Imported mesh representation of faults without explicitly meshed intersection.

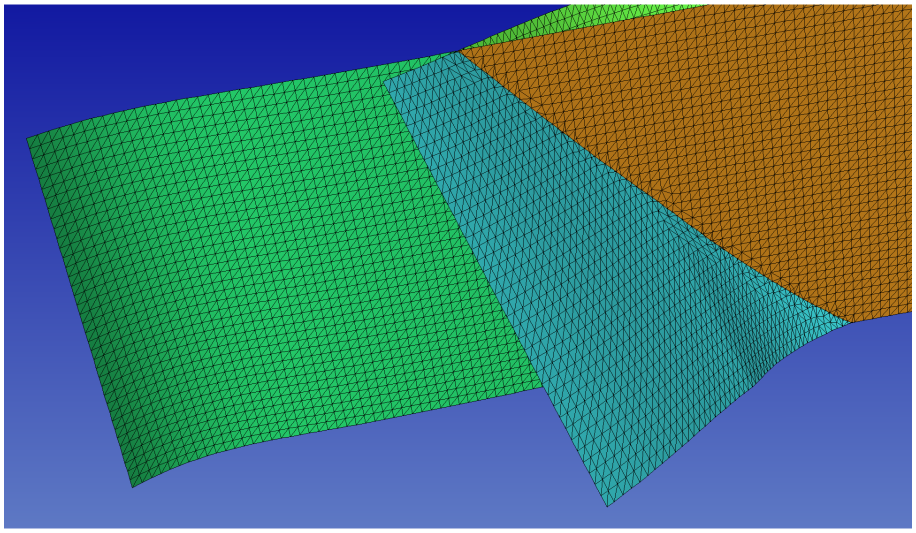


Fig. 3: Fig. 3: Intersected faults, after applying *union parts*

CHAPTER

FIFTYTWO

GENERATING A CAD MODEL USING GOCAD: BASIC TUTORIAL

52.1 Input files

The python scripts used in this tutorial are here and the input files are here.

Two *.dat files contain the structured point clouds representing the topographic layer and the fault. The *.pl file (Gocad ASCII File format) describes 2 manually set curves used for defining the model extend.

52.2 Triangulating structured point clouds

First, we construct triangulated surfaces from the structured point clouds using create_surface_from_structured_grid.py

```
python3 create_surface_from_structured_grid.py --NX 161 topography.dat topography.ts
```

If the points of the cloud share the same coordinates along a row or a column, then the -NX option can be omitted

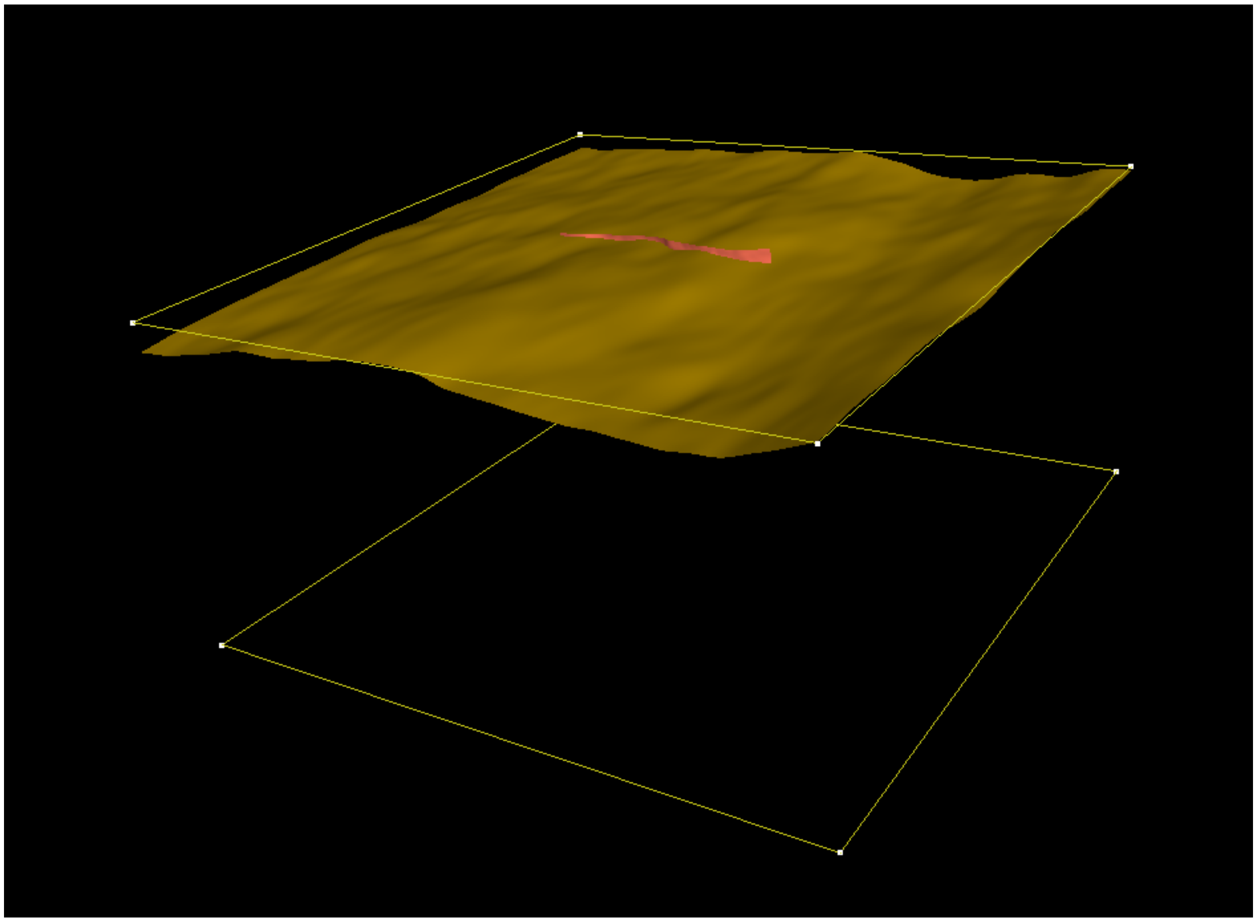
```
python3 create_surface_from_structured_grid.py topography.dat topography.ts
```

52.3 CAD model generation in GOCAD

Now let's combine all these data into a CAD model using GOCAD.

Launch Gocad and create a new project. Use the default modules. Change Depth to meters, and Depth axis to Upwards. Don't specify the coordinates system. Projecting is done using the python scripts. When creating the surfaces, many scripts have a "–proj" option. In addition, the script projTs.py also allows projecting gocad ts files. Using projected data in Gocad (as opposed to lat/lon/depth data) seems to help the software generating high-quality intersections.

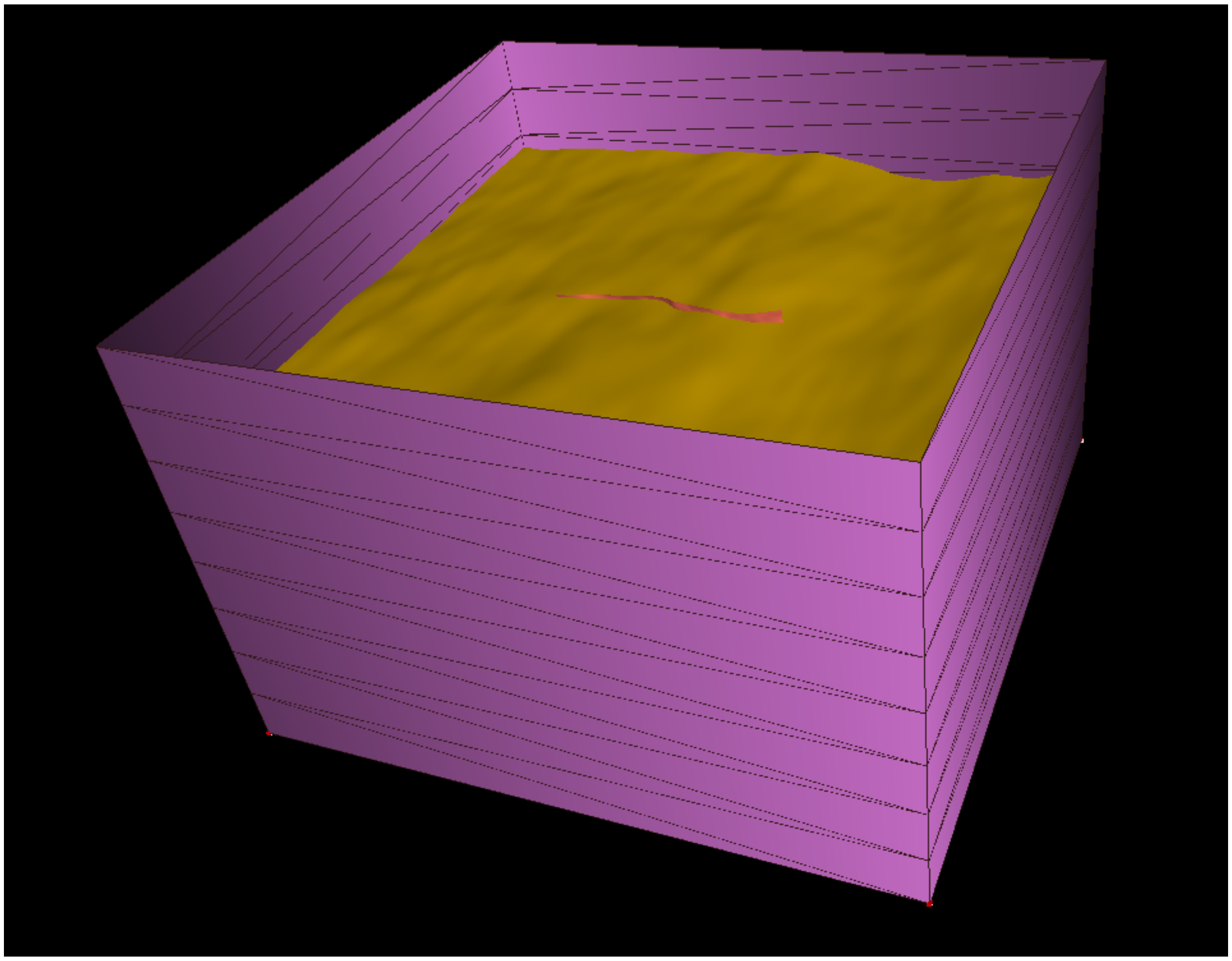
File > Import > Gocad Objects and select the two surfaces (fault.ts and topography.ts) and the *.pl file.



52.3.1 Creating the bottom and sides surfaces

Surface > New > Closed Curved, choose a name, uncheck 'Dissociate vertices' in advance (always do that), click on apply and click on the curve 'bottom'.

Surface > New > Builds in Forms/Tube, choose an name, curve 'bottom', dir expansion along z 80e3, number of level 8(for example), uncheck 'Dissociate vertices'.



52.3.2 Mutual intersections

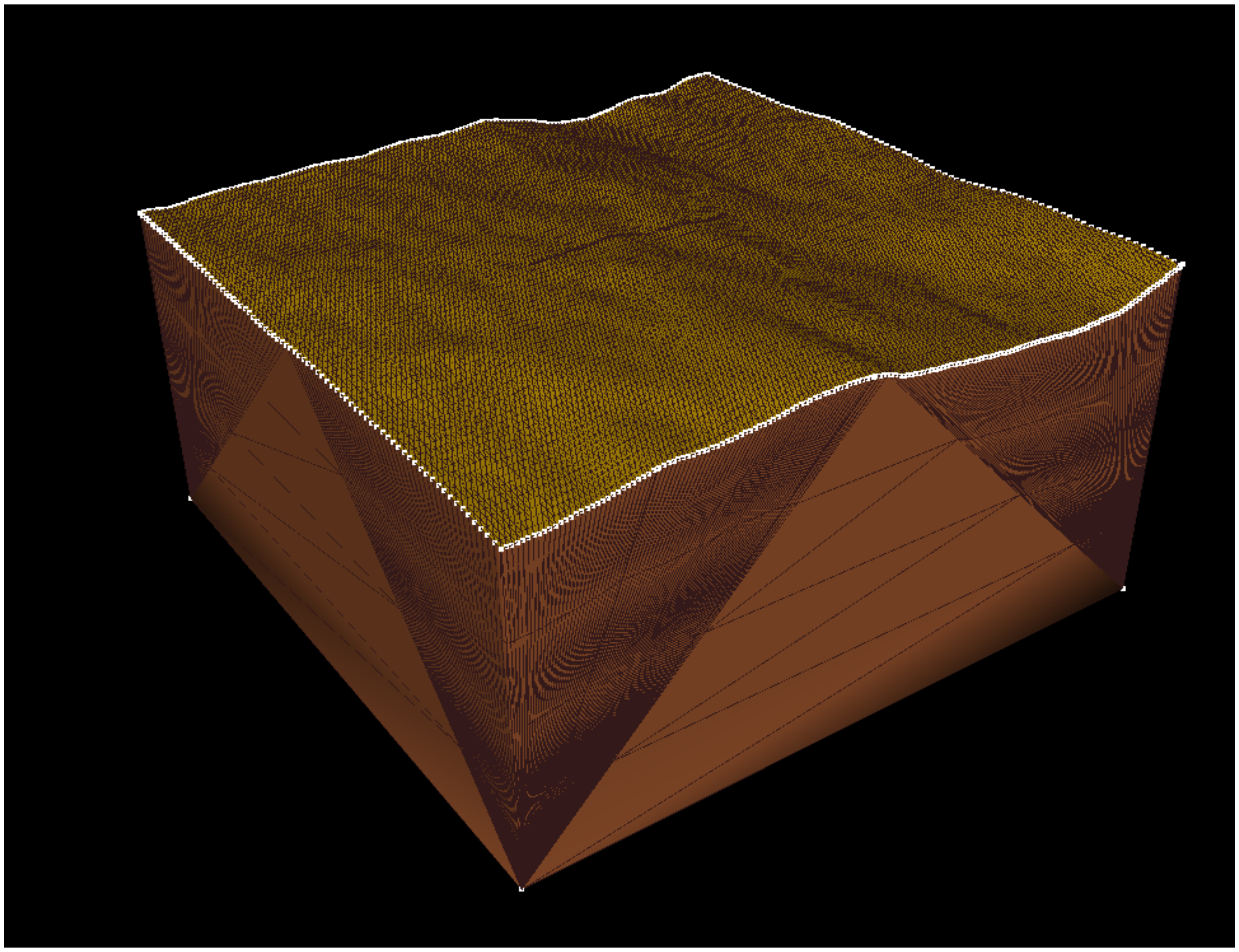
Tools > cut > mutual cut among surfaces > select side surface, fault and topography. Tools > Part > delete selection: click on the portion of the fault surface higher than the topography. Then on the part of the side surface higher than the topography.

52.3.3 Alternative way of Creating the side surface

Sometimes the mutual intersection of Gocad does not work properly (in particular when the intersection curve passes close to triangle edges on both surfaces). Then the surfaces to be intersected have to be manually adjusted, for example by switching triangles (Tools > Triangle > Switch 2 triangles), to allow the intersection to work properly. Alternatively, a different way of generating the surface could be considered. For instance, the side surfaces can be generated from the 2 curves bordering the topography and the bottom surface:

Curve > New > Borders > all, use surface topography to create curve ctopo. Surface > New > 2 curves parts > select a name, choose a number of level (e.g. 4) and click on both curves.

Here is the resulting model:



52.3.4 A third option for creating the side surface

If a projection has been already done on the topography surface, then the borders of the surface are probably not any more straights, and the previous method may not be applicable anymore. In such a situation, we can still create a tube from the border of the topography, and cut it at the required depth. Curve > New > Borders > all, use surface topography to create curve ctopo.

Surface > New > Builds in Forms/Tube, choose an name, curve 'ctopo', dir expansion along z -80e3, number of level 8(for example), uncheck 'Dissociate vertices'.

To trim the tube at the requested depth, we export the curve ctopo in a pl file (File > Export > Gocad ASCII >ctopo.pl) and we use the script change_depth_pl_curve.py:

```
python3 change_depth_pl_curve.py ctopo.pl ctopo_new.pl --depth 60e3
```

Then we import ctopo_new.pl into the project, create the bottom surface of the box from it (using Surface > New > Closed Curved). We can then intersected this new surface with the tube, and remove the part of the tube below the bottom box surface.

52.4 Exporting and converting to stl

Now the triangulation of all surfaces of the CAD model is conforming, and we can export the CAD surface in a ts format.

File > Export > Gocad ASCII > choose a filename (example test.ts).

Now let's convert the ts file to stl using convertTs.py:

`python3 convertTs.py test.ts`

The –merged option merges all surfaces in a single stl 'solid'. If not set, each Gocad surface will be isolated into a different stl 'solid'. Each surface will then be viewed as a different entity by SimModeler.

The option –proj allows projecting into a new coordinate system (e.g. –proj EPSG:32646).

52.5 Sanity check

We can now quickly check that the CAD model is correct, by loading it into SimModeler.

File > Import Discrete Data > select file and click on OK.

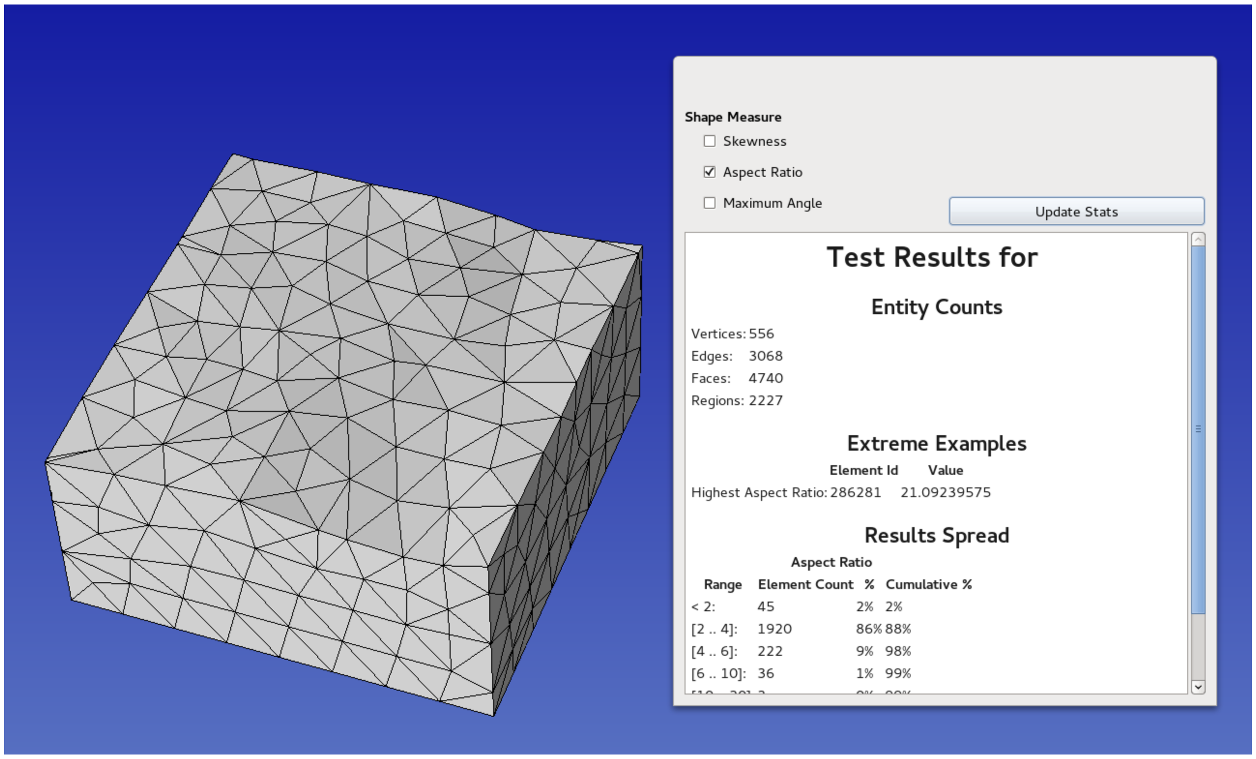
In Model list, we have a single region and no singled-out surfaces.

Model Tab > Remove Small Features: Shortest edge in the model should be big in comparison to your smallest mesh size. If not, you can specify a tolerance and have a look at the location of the small features. If they are located in the vicinity of some surface mutual intersection made with Gocad, this stage failed and you have to go back working on your CAD model, for example following this guide: [[Manually fixing an intersection in Gocad]].

If everything was fine in the previous checks proposed, we can make a last verification by meshing coarsely the model using

Meshing > Generate mesh. We can finally inspect the mesh:

Display tab > check Mesh Stats > then check Aspect Ratio, and see if the maximum aspect ratio has a reasonable value (~20).



Shape Measure

- ☐ Skewness
- ☑ Aspect Ratio
- ☐ Maximum Angle

Test Results for

Entity Counts

Vertices:	556
Edges:	3068
Faces:	4740
Regions:	2227

Extreme Examples

	Element Id	Value
Highest Aspect Ratio:	286281	21.09239575

Results Spread

Aspect Ratio

Range	Element Count	%	Cumulative %
< 2:	45	2%	2%
[2 .. 4]:	1920	86%	88%
[4 .. 6]:	222	9%	98%
[6 .. 10]:	36	1%	99%

CHAPTER

FIFTYTHREE

PROPOSED WORKFLOW FOR GENERATING A CAD MODEL OF A MEGATHRUST EARTHQUAKE

Here is present a workflow for generating a CAD model of the Japan subduction, including subduction interface, to be used in dynamic rupture models or for imposing a kinematic model on the subduction interface. We use scripts from https://github.com/SeisSol/Meshing. To best follow this tutorial, we suggest adding the geometry script folder to the path:

```
export PATH=$PATH:~/SeisSol/Meshing/creating_geometric_models
```

53.1 topography and slab interface

First, we download topography and bathymetry data from GEBCO (http://www.gebco.net/), and we triangulate it into a GoCad ts file. Note that we downsample the topography data by a factor 5 for dealing with a reasonable size dataset in this tutorial. Note also that we use a custom transverse Mercator roughly centered at the domain center.

```
#!/bin/sh
myproj='+proj=tmerc +datum=WGS84 +k=0.9996 +lon_0=143 +lat_0=39'
topofile='data/gebco_2021_n42.61596679687501_s34.16381791234017_w137.10571333765984_e146.
↪759033203125.nc'
create_surface_from_rectilinear_grid.py $topofile tmp/topo.ts --proj "$myproj" --sub 5
```

Next, we generate a mesh of the subduction interface. We use the Kamchatka-Kuril Islands-Japan Region model of Slab2.0 available here https://www.sciencebase.gov/catalog/item/5aa4060de4b0b1c392eaaee2. We download kur_slab2_dep_02.24.18.grd and rename it as kur_slab2_dep_02.24.18.nc. Finally, we crop and triangulate it using the following command:

```
create_surface_from_rectilinear_grid.py data/kur_slab2_dep_02.24.18.nc tmp/ur_slab2_dep_
↪02.24.18.ts --crop 140 145 35.5 41 --proj "$myproj"
```

The Slab2.0 data are sampled every 5 km, but a dynamic rupture simulation may require one order of magnitude smaller resolution, or even more. If we mesh at 500m a geometry sampled at 5 km, the obtained mesh won't be smooth but will have edges every 5 km. Therefore, we smooth and refine the mesh with:

```
refine_and_smooth_mesh.py --N 0 --P 1 tmp/ur_slab2_dep_02.24.18.ts
```

The help (-h option) offers further details about the script options. Here we use P=1 to avoid dealing with a too large dataset in the tutorial.

53.2 Terminal splay

The shallowest part of the subduction interface is 5-10km below the sea surface. We might want to acknowledge the possibility of surface rupture by extendiing the interface to the surface at steeper angle. We therefore build a terminal splay extending from the trace of the know trench location. For that, we first preprocess a USGS trench data file into a fault trace. Then we extend the trace along a constant dip angle.

```
generate_trace_terminal_splay.py --shift -0.05 0 --filter_trench 137.2 146.7 35.2 41.2␣
↪jap --usgs_trench_file data/trenches_usgs_2017_depths.csv --bathymetry $topofile --
↪plot tmp/extractedtrenchTohoku.dat
create_fault_from_trace.py tmp/extractedtrenchTohoku.dat 0 30 --proj "$myproj" --
↪maxdepth  20e3 --extend 8000 --extrudeDir data/strike90.dat --dd 5000 --
↪smoothingParameter 1e7
refine_and_smooth_mesh.py --N 0 --P 1 extractedtrenchTohoku0.ts
```

data/strike90.dat contains 2 lines, indicating that the extrude direction is towards West.

```
0 -90
1 -90
```

53.3 Merging terminal splay and subduction interface

First, we generate 2 planes that we will use to trim slab interface and extension to the same dimension.

```
python generate_cutting_plane.py  --center -65000 -367000 -11000  --normal 0. 1. 0.  --
↪dims 1000e3 300e3 --mesh 10e3 tmp/cut1.stl
python generate_cutting_plane.py  --center 125000 210000 -11000  --normal 0. 1. 0.  --
↪dims 1000e3 300e3 --mesh 10e3 tmp/cut2.stl
```

Then we load both processed subduction interface and terminal splay into SimModeler. We intersect them, and trim what can be trimmed. Then we import the cutting plane, and intersect all parts. We finally remove all parts that are not faults. At this point we have 2 connected surfaces, subduction interface and terminal splay, and a sharp angle between them. This sharp angle can be smoothed by extracting the discrete mesh as inp file (see *Remeshing the topography*), converting to ts and applying the refine_and_smooth_mesh.py script.

53.4 Final steps

Finally, we create a box mesh box domain with pygmsh as follow:

```
generate_box.py --proj "$myproj" --rangeFromTopo $topofile tmp/box.stl --zdim " -500e3"␣
↪5e3 --shrink 0.9
```

The final step consists in intersecting all objects (topography, faults and domain box) in the GUI of SimModeler, as presented in *SimModeler CAD workflow*.

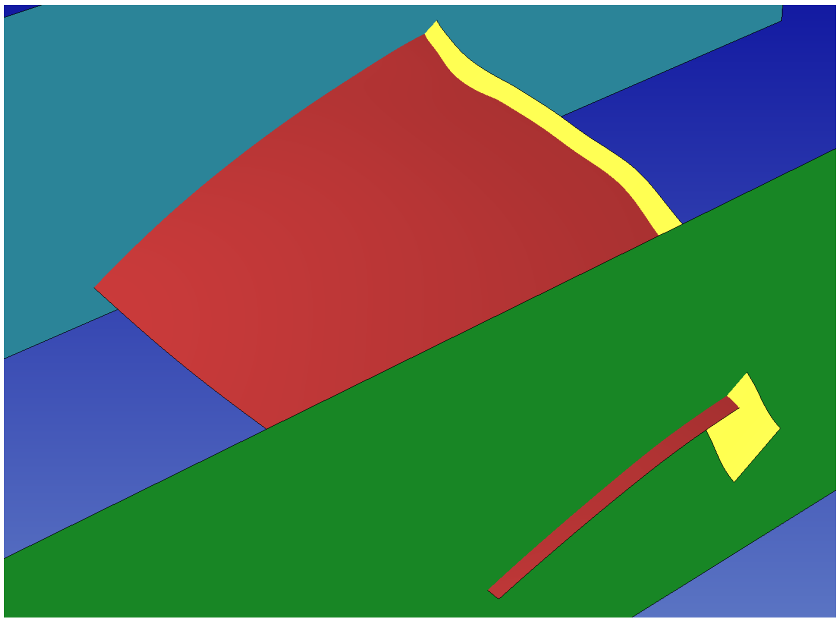


Fig. 1: Discrete geometry model of the slab interface (red), a potential terminal splay fault (yellow), and the cutting planes (blue and green) used to trim laterally these surfaces, before interesection.

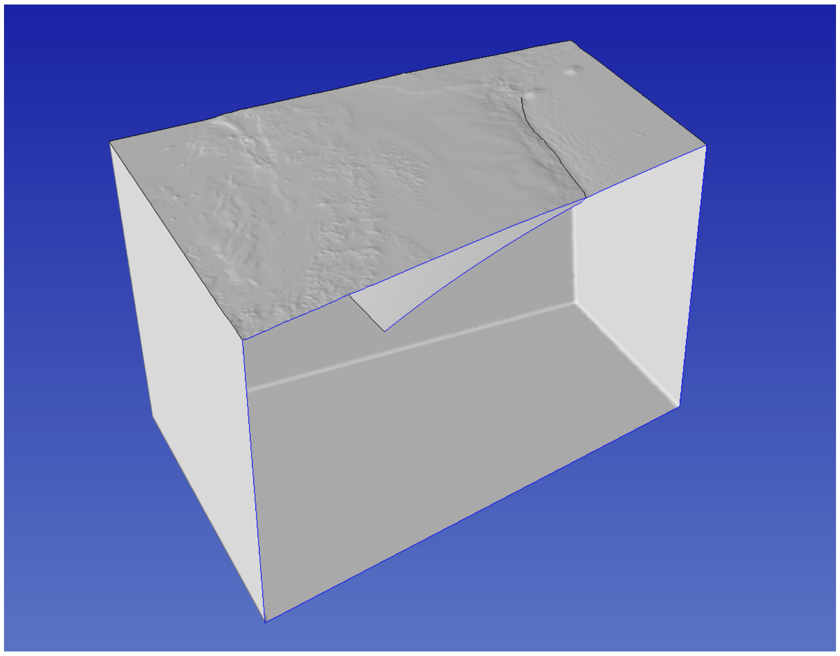


Fig. 2: Cut view of the final Tohoku's model

CHAPTER

FIFTYFOUR

GENERATING A CAD MODEL FOR A FULLY-COUPLED EARTHQUAKE-TSUNAMI SIMULATION

Here we present a workflow for generating a CAD model of the Eastern Aegean Sea, including ocean and topography to be used in fully-coupled earthquake-tsunami simulations of the 2020 Aegean Sea earthquake. We use scripts from https://github.com/SeisSol/Meshing. To best follow this tutorial, we suggest adding the geometry script folder to the path:

```
export PATH=$PATH:~/SeisSol/Meshing/creating_geometric_models
```

Please consider running *git pull* in SeisSol/Meshing to pull the latest version of the scripts, if the repository is not newly cloned. You may also install required python modules with *pip3 install -r requirements.txt* (or if you use anaconda, *conda install –file requirements.txt*). This tutorial has been tested with SimModeler10.0-211122. Note that one specific step requires SimModeler11.0-220403-dev, but another step of the workflow fails with the dev version.

54.1 Creating topography and boxes

First, we download topography and bathymetry data from GEBCO (http://www.gebco.net/), and we triangulate it into a GoCad ts file.

- Note that we downsample the topography data by a factor 2 for dealing with a reasonable size dataset in this tutorial.
- Note also that we use a custom transverse Mercator roughly centered at the domain center.
- With the option `--change_zero_elevation 1.0`, we move the nodes with zero elevation to 1.0 m. This avoids having to intersect locally coplanar surfaces.
- With the option `--smooth 100`, we replace the elevation of topography nodes with z coordinates in the range ± 100 m by spatially smoothed (using a 2D Gaussian kernel) values.

This facilitates the intersection of the sea surface with the topography and allows a smoother coastline (which can else have a saw-tooth shape due to rounded elevation data, stored as integers in Gebco files).

```
myproj='+proj=tmerc +datum=WGS84 +k=0.9996 +lon_0=26.25 +lat_0=37.75'
topofile='gebco_2021_n39.5_s36.0_w23.5_e29.0.nc'
create_surface_from_rectilinear_grid.py $topofile topo.ts --proj "$myproj" --sub 2 --
↪smooth 100 --change_zero_elevation 1.0
```

Next, we generate a mesh of 2 boxes, one for defining the water layer and the other for defining the domain region.

```
generate_box.py box_water_layer.stl --hdim " -40e3" 93e3 " -10e3" 53.0e3 --zdim " -10e3"␣
↪0 --meshSize 600.
```

(continues on next page)

(continued from previous page)

```
generate_box.py --proj "$myproj" --rangeFromTopo $topofile box_domain.stl --zdim " -200e3
↪" 10e3 --shrink 0.9
```

54.2 Intersecting topography and sea-surface

Then we load *box_water_layer.stl* and *topo.ts* into SimModeler (`Import Discrete Data`, select both files (press Shift), and unclick all options except `Find edges by face normals`, with `Normal Angle` 80, a value high enough to detect the box, but not creating new faces in the topography surface).

We then do `Unions Parts` (Tolerance 0.1, smaller than the `change_zero_elevation` parameter) with Parts topo and box_water_layer. This leads to the creation of 3 regions and 1 face. Region 3 is the solid Earth below the sea floor, region 4 is the larger ocean region, and region 5 is the small dark green water region.

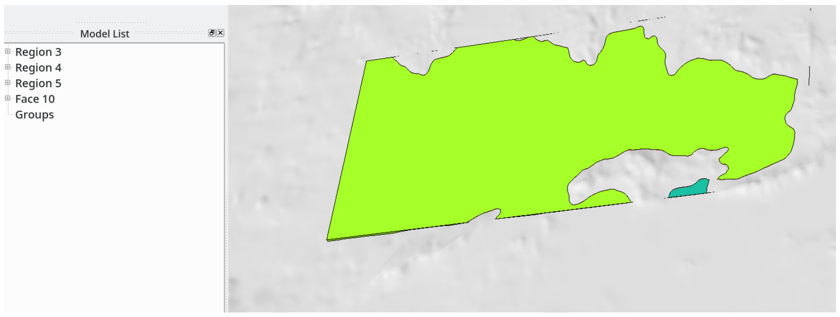


Fig. 1: Three-regions model obtained after intersecting bathymetry and water-layer box.

The next step is to delete the solid Earth regions 3 (note that region numbers may differ). Use `Discrete->Delete` add regions 3, and press `Ok`.

At this point, the model has 2 regions and 1 surface. We can double-check the quality of the intersections with `Prepare->Remove Small features`. Here the shortest edge in the model is 139 m, which is large enough. If this was not the case, small edges can be identified, and the associated node of the not yet intersected topography moved (with `Discrete->Deform Face`). Then the full workflow described above needs to be followed again with the updated topography (and hopefully the short edge has been suppressed).

The next step consists of loading *domain_box.stl* and intersecting it with the current model. The union is done with SimModeler11.0-220403-dev (as it fails with SimModeler10.0). We then clean the model of the upper part of domain_box, the part of the topography outside the domain box, and the smaller water layer region. The sea floor below the removed smaller water layer region can be merged with `Discrete->Combine Faces`). The obtained model has 2 regions.

54.3 Enforcing minimum depth on seafloor surface

Unfortunately, the water layer of the model described above is not meshable by SimModeler, because at some locations, the sea floor is too close to the sea surface. Because of that, the error `Cannot resolve intersecting mesh` is raised, even when using a small mesh size of 100 m. To deal with this problem, we extract the mesh of the sea floor and increase the sea floor depth where it is very close to the sea surface. This is done with:

- `Mesh->Miscellaneous->Use Discrete Geometry Mesh` on the sea-floor and
- `Mesh->Element Type->No Mesh->Entity` on all other surfaces.

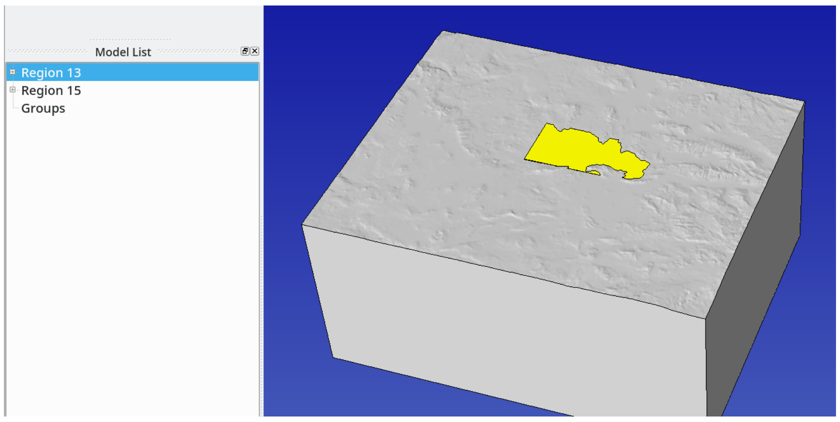


Fig. 2: Two-regions model of the Eastern Aegean Sea area, including a water layer.

- `Volume Meshing` should be removed.
- `Mesh->Generate Mesh`
- `Mesh->Export Mesh`: Filename seafloor.inp.

Then we enforce the minimum depth of the seafloor with:

```
convertInp.py seafloor.inp seafloor.stl --enforce_min_depth 40
```

Note that the minimum depth of 40 m applies only on nodes of the seafloor triangulation, that is the effective depth varies linearly between these nodes and the coast. A value of 40 m makes sense due to the coarse (horizontal) resolution of the topography here used (900 m resolution). For example, we see that with 25 m, SimModeler can successfully mesh the water layer only with a mesh size smaller than 200 m, while a value of 40 m allows at least 1 km. In case of a finer topography resolution, a smaller value should be used.

The next step is to generate an stl file (`other_surfaces.stl`) with all other surfaces from the model using the workflow presented above (without `--enforce_min_depth` option but with `--isolate` option).

```
convertInp.py other_surfaces.inp other_surfaces.stl --isolate
```

Both stl files can finally be combined into a stl file using cat:

```
cat seafloor.stl other_surfaces.stl > new_model.stl
```

Once loaded into SimModeler (untick all when importing), 2 regions are detected and the shallow water can be successfully meshed, even with large mesh size (e.g. 1 km in the water layer).

54.4 Dealing with union errors

Unfortunately, unions may fail. At best, a descriptive error is issued by SimModeler, e.g.: `Error: Code: 604 String: edge 72 has tangent faces at point (61781.436490285792, 3066.1427893521077, -2.2204460492502381e-16)` In this case, we can overcome the problem by manually moving a node of one of the surfaces intersected close to the location of the error with `Discrete->Deform Face`. The error `Error: Code: 60 String: General error` may also be raised, for which there is currently no obvious workaround, except trying to change the mesh size or the dimension of one of the intersected objects. Hopefully, these error messages will be improved in the future.

CHAPTER

FIFTYFIVE

REMESHING THE TOPOGRAPHY

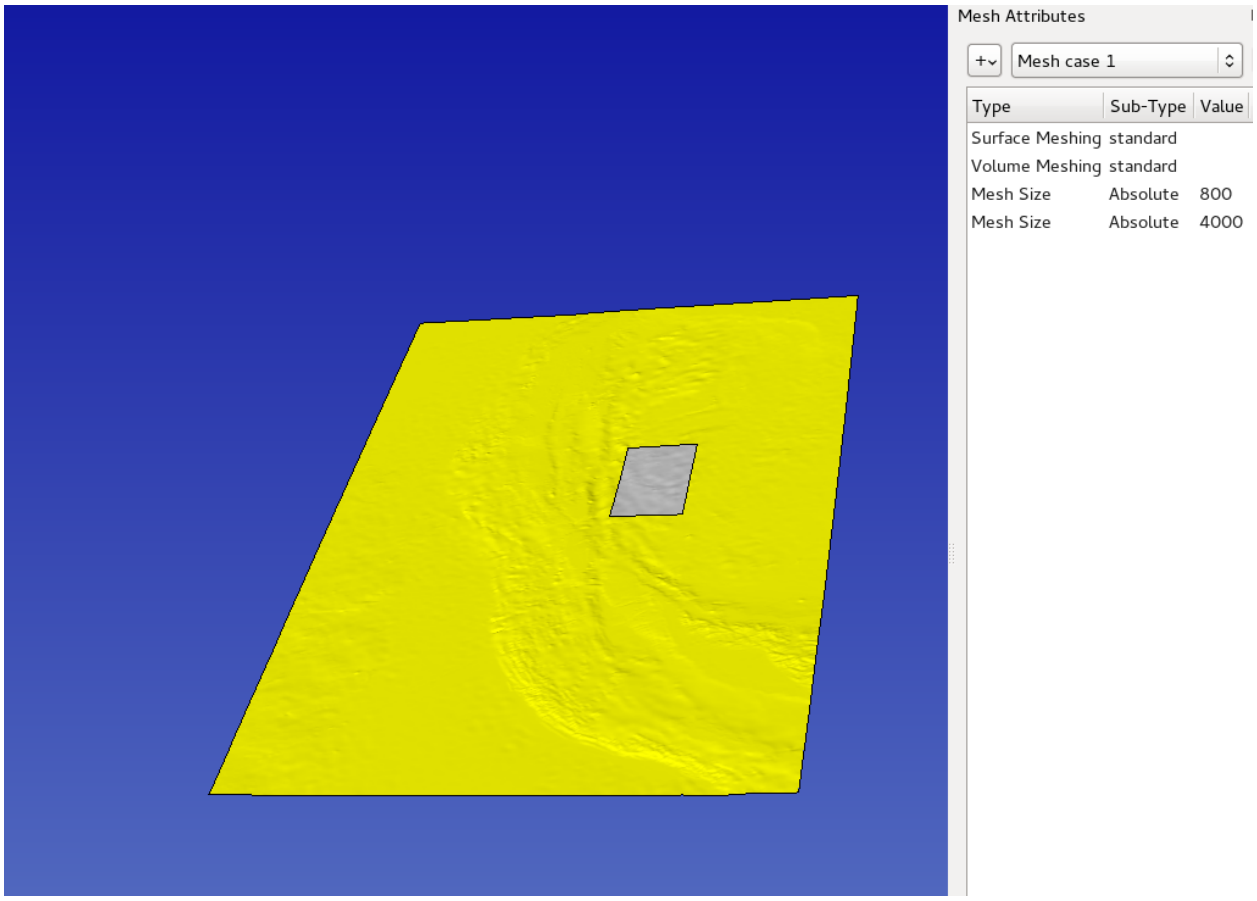
Nowadays, high resolution topographic and bathymetric data are most of the time available. Processing this large amount of data can be a challenge. For example in Gocad, intersecting such surfaces with other surfaces can be time-consuming and error-prone. To overcome these kinds of difficulties, an idea is to coarsen the meshed topography where a fine resolution is not necessary, before working on the surfaces in Gocad.

As an illustration, we will process a netcdf file from GEBCO (http://www.gebco.net/). It has been used in Sumatra related simulations. It features a 400m resolution regular grid. Using the script create_surface_from_rectilinear_grid.py, available here, we will downsample the data overall, project them and isolate a square region away from which a fine discretization is not necessary.

```
python3 create_surface_from_rectilinear_grid.py data/GEBCO_2014_2D_90.0_1.5_97.0_14.5.nc
trash.stl --subsample 2 --proj EPSG:32646 --hole 94 95 8 10
```

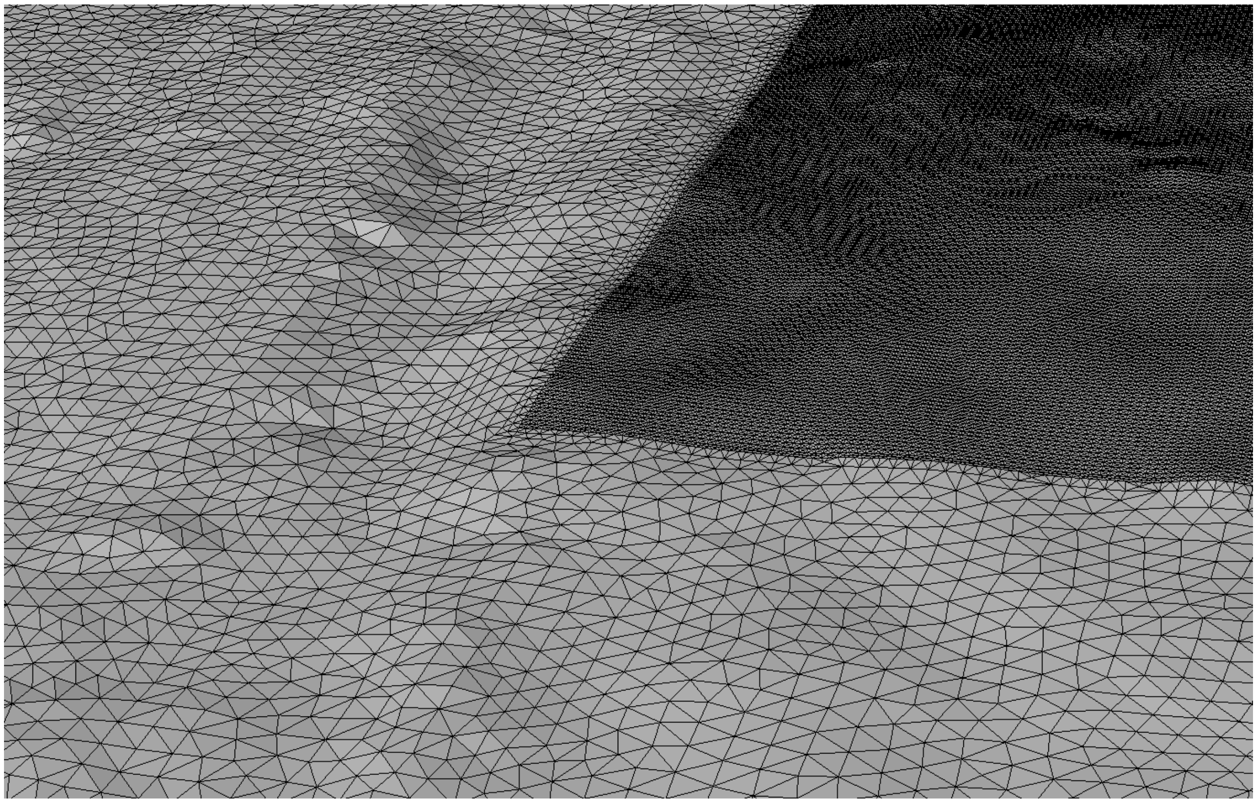
Now we can import the data in SimModeler5 (the version is important, as SimModeler4 does not have the ABAQUS 2D export):

File > Import discrete Data > uncheck all.



Type	Sub-Type	Value
Surface Meshing	standard	
Volume Meshing	standard	
Mesh Size	Absolute	800
Mesh Size	Absolute	4000

Now we can specify the size we want in the central and side areas. Here is the resulting mesh:



The mesh can then be exported:

File > Export mesh > ABAQUS 2D > test.inp (for example).

We finally convert the inp file to a ts file readable by gocad using:

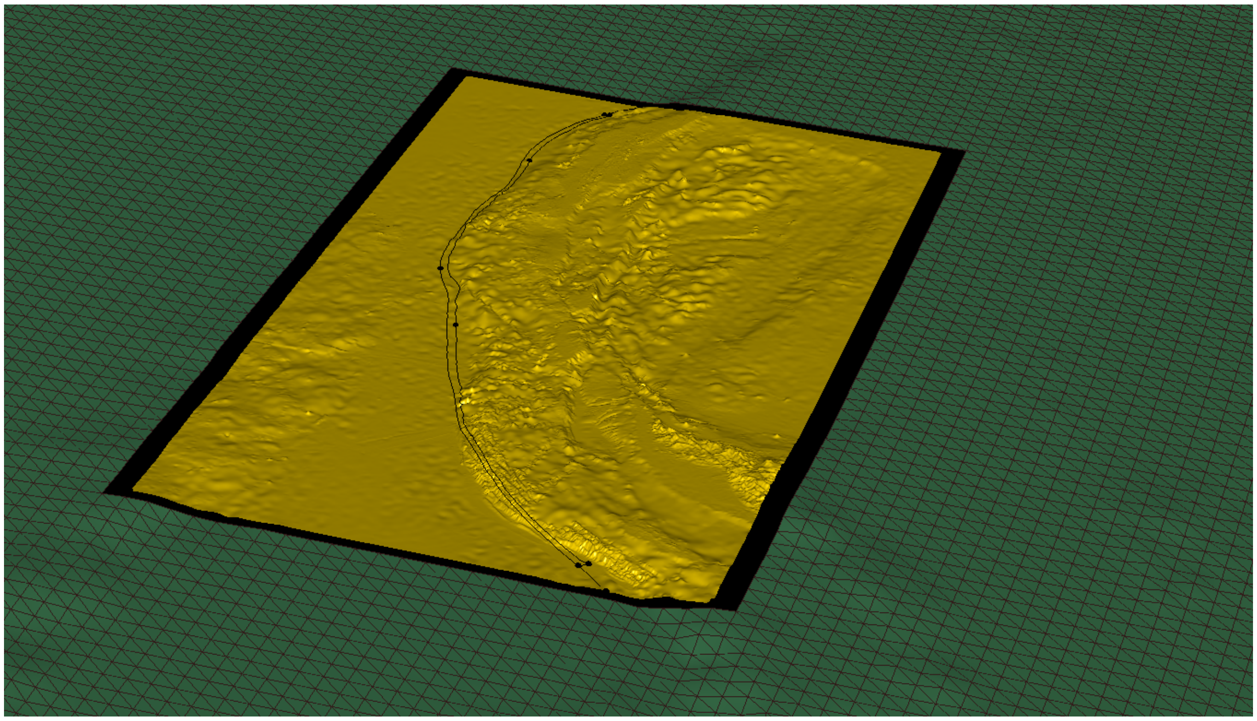
```
python3 convertInp.py test.inp --isolate
```

CHAPTER

FIFTYSIX

ADAPTING THE CAD MODEL RESOLUTION USING GOCAD

In this example, a finely sampled topography (yellow) has already undergone several operations in Gocad (the fault and another structural interface have been intersected with it). Now we want to extend the model with a coarsely sampled topography (green surface).

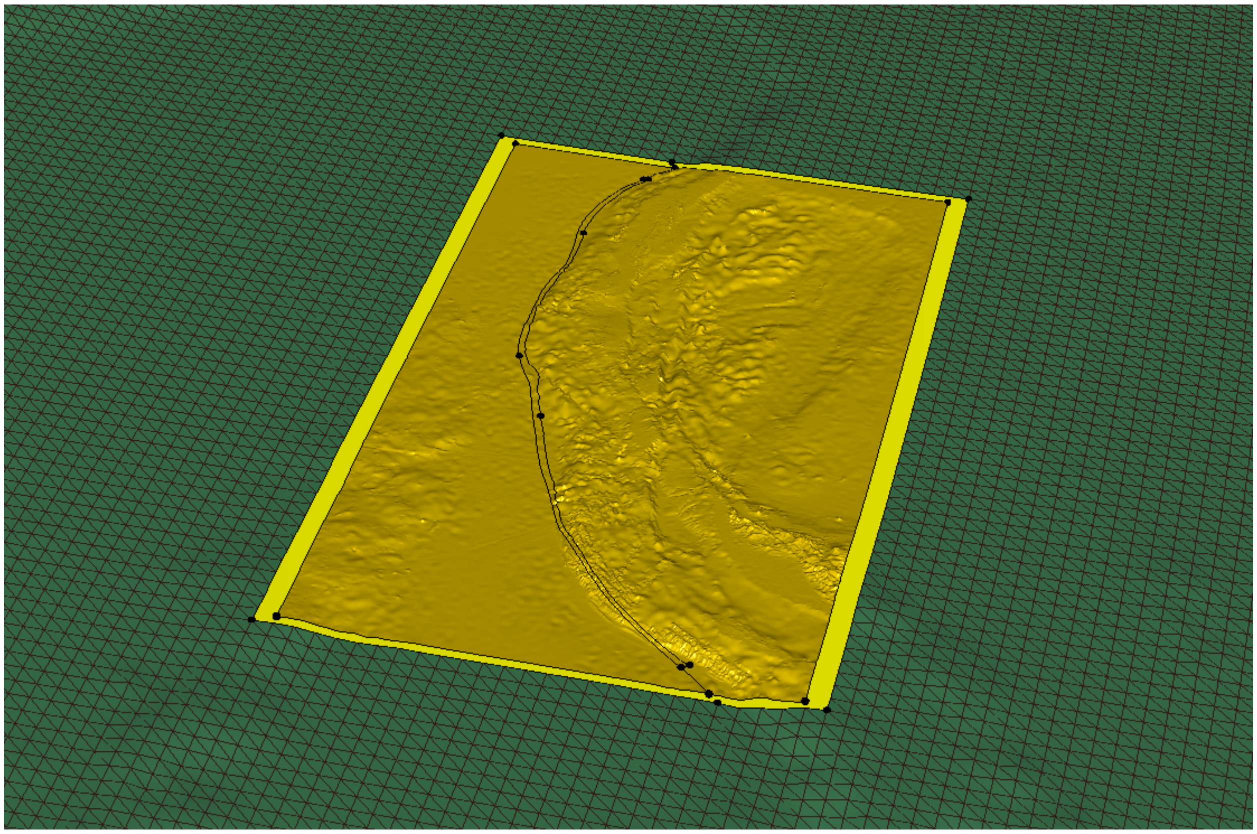


The coarse topography has been created using the "hole" and "subsample" options of the script create_surface_from_structured_grid.py, using the following command line:

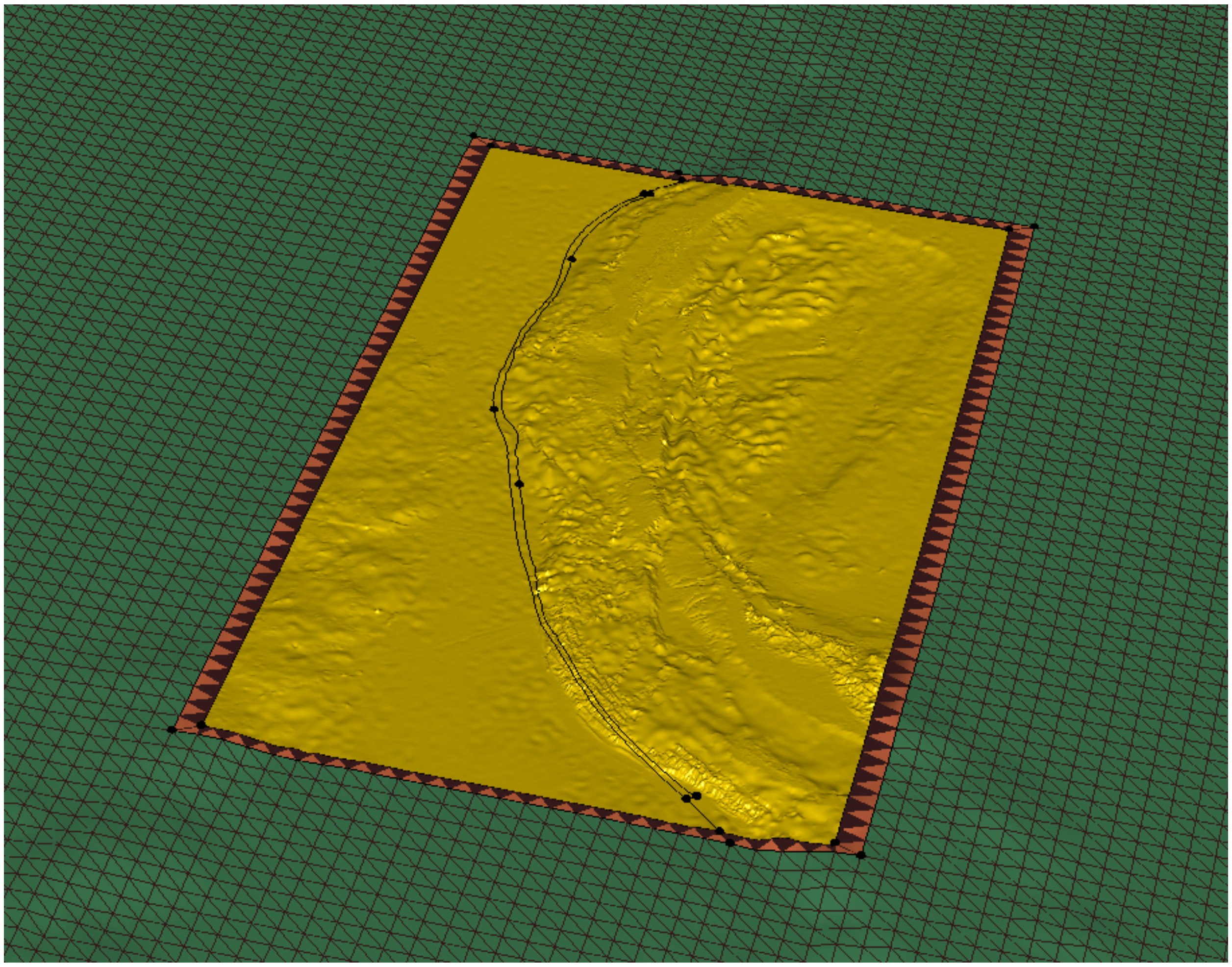
```
python3 create_surface_from_structured_grid.py --subsample 8 --objectname topoE --hole
89.5 97.0 1.5 14.5 structuredPointSet.xyz topoExt.ts --proj EPSG:32646
```

Now we want to seal the gap between both surfaces. To do that we will create surfaces "from 2 surface borders". To facilitate the work of Gocad, we first need to add extra boundary extremities to both surfaces, at the corners of their borders. We also need to add boundary extremities to the coarse topography to roughly match the segmentation of the fine topography border, resulting from the fault intersection. This is done by:

Right-clicking on the surface >Tools > Border Extremity >Add.



Then, the surface filling the gap between coarse and fine topography can be created using: Surface > New > From curves > 2 surface Borders > give a name, a click on each border. Repeat the operation for each section of the border. Here is the final result.



Finally, the new surface can possibly be merged into the coarse topography.

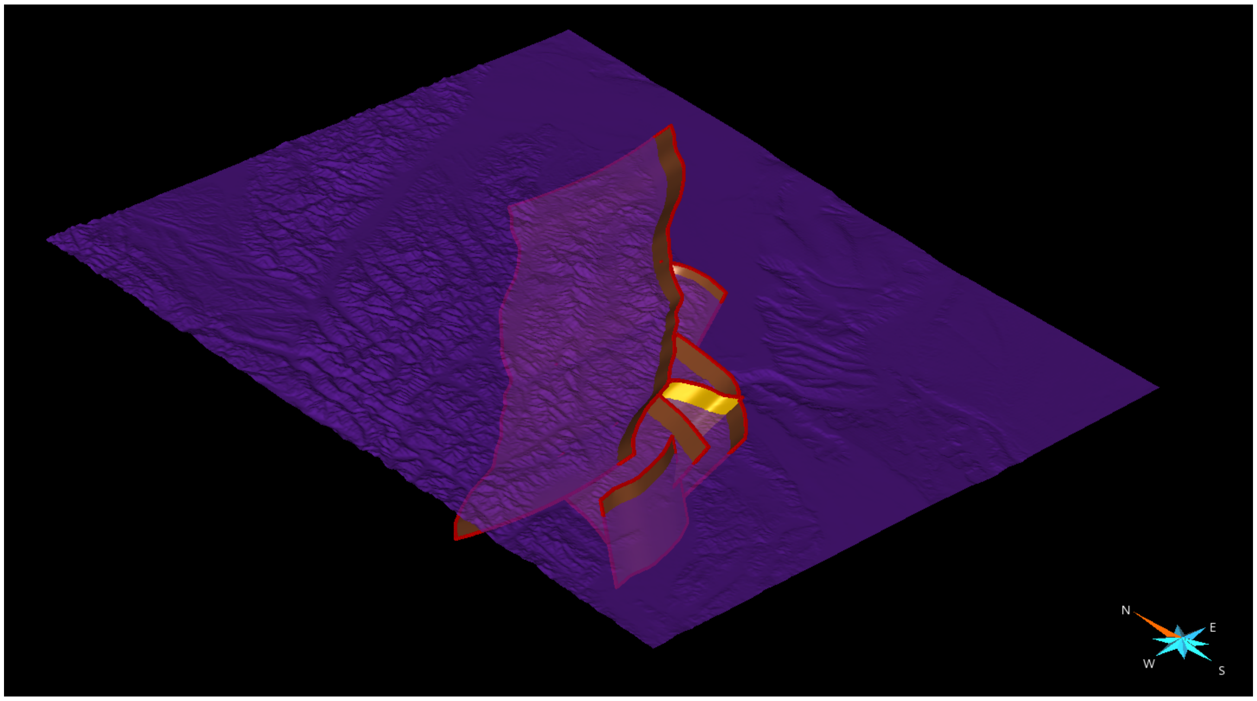
CHAPTER

FIFTYSEVEN

MANUALLY FIXING AN INTERSECTION IN GOCAD

The mutual intersection in Gocad is the keystone of our geometry building framework, but is also the bottleneck. In fact, it is a black box, which fails in given geometric configurations: Sometimes it generates tiny holes preventing building a volume out of the surfaces, or it creates tiny misoriented faces, that cannot be attached to an existing surface, and then lead to very small time steps, so small that they impact the simulation time even with local time stepping enabled. So far, we did not find any better solution than fixing manually the geometry.

Here we detail a systematic way of fixing the intersection, using a realistic example. We then try to intersect a complex fault system (yellow and orange surfaces, about 250k triangles) with a finely meshed topographic surface (blue surface, about 1M triangles).



The first step of the workflow is to carry out the intersection, to visualize the problematic area of the mesh to be fixed.

Surface>Tools>Cut>Mutual cut among surfaces>select the surface and Apply

The next step is to import the intersected surfaces in SimModeler. Then we use the standard procedure:

File > Export > Gocad ASCII > choose a filename (example test.ts).

We then convert the ts file to stl using convertTs.py:

```
python convertTs.py test.ts
```

Finally we import the stl file in SimModeler using:

File > Import Discrete Data > unclick all option, select file and click on OK.

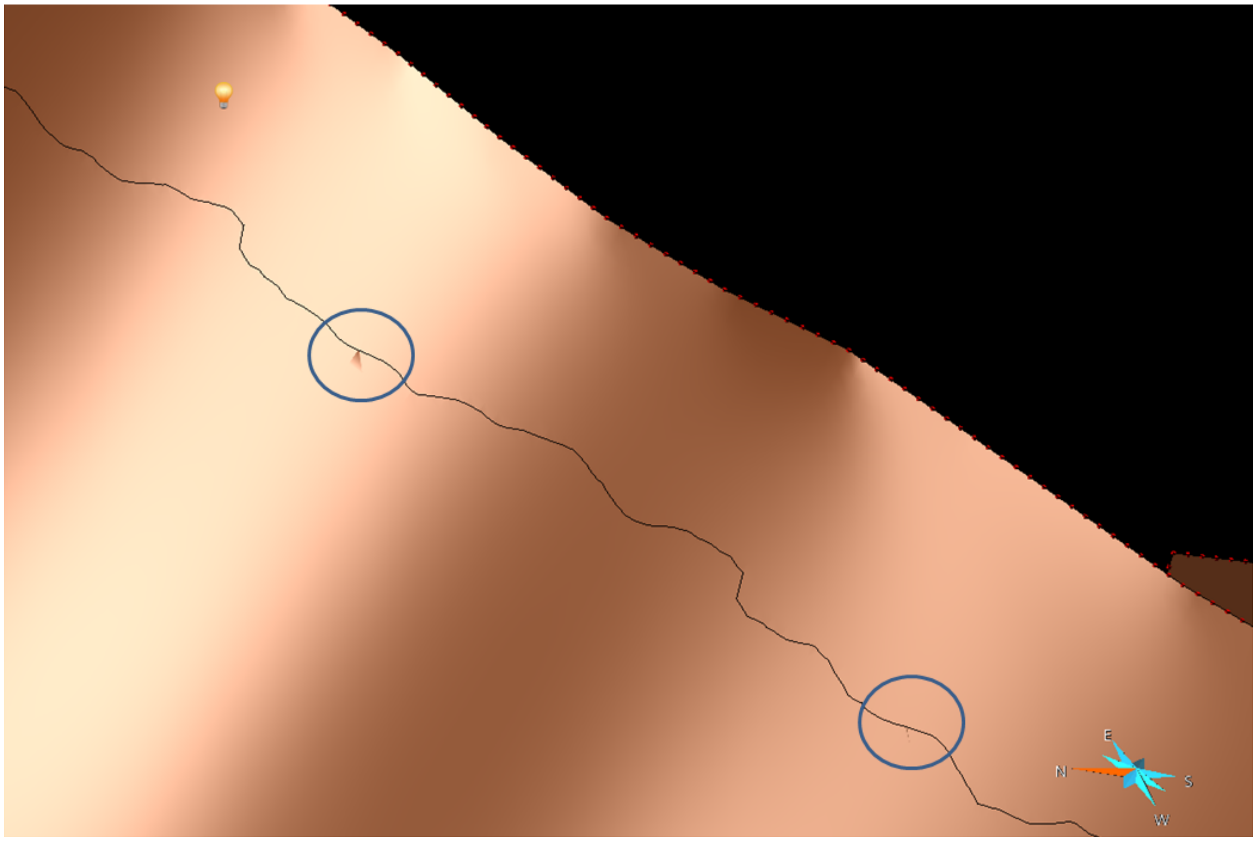
Now let's check for small features in the geometry:

Model Tab > Remove Small Features > Find (note that we do now remove the small features here, but we only localize them).

SimModeler will list faces and edges. Nevertheless, we only focus on the faces, the edges being usually the edges of the faces listed.

57.1 Graphic search for the small features

At this point, one possible option is to select one of the small feature, click on focus on selected, and then try to zoom out to see where the face is located. In practice, it might not be so easy. Once you have localized the face in SimModeler, you can try to find it in gocad by playing with the light. In fact, the small features have usually a different normal orientation that the surrounding faces, and their edges may be visible under certain light incidences. Here is an example of 2 features poping out:



57.2 meshing the small features to get their coordinates

A preferable option, because more systematic, for localizing the small features is to mesh them in SimModeler, export the mesh in an ascii file, and read the node's coordinates. Then mouse over the intersected surface, along the intersection line up to each of the coordinates. To mesh them, click on each small features faces, and choose 'Use discrete Geometry mesh'. Then click on all other faces, and choose 'No mesh/Entity'. Finally, remove the 'Volume meshing' attribute, and click on 'Generate Mesh'. The surface mesh can then be exported:

File>Export Mesh> Format ABAQUS 2D (for instance), enter a filename and save.

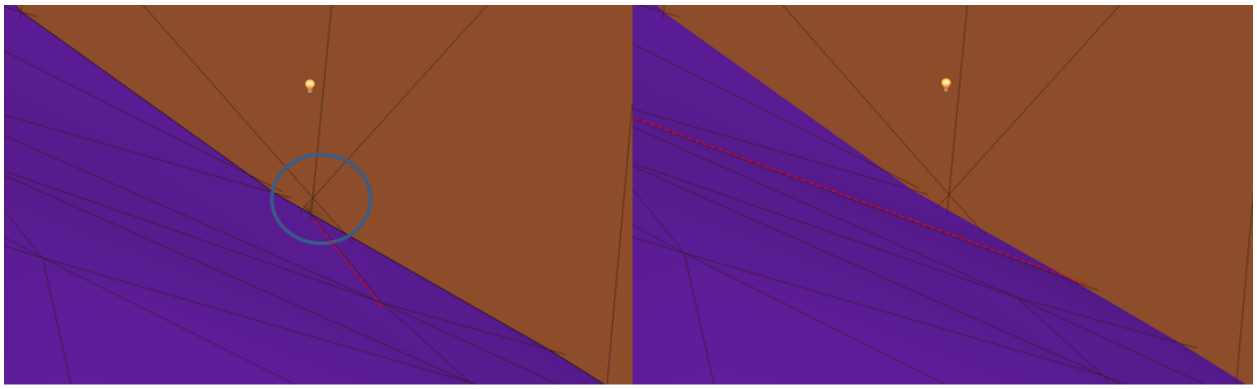
The ascii mesh look like that:

```
*Heading
*Node
1, 6186870.5469, -3926750.75, 134.60108948
2, 6190219.5312, -3912085.8125, 950.25006104
3, 6127079.0312, -3952731.3125, 411.05957031
(. . . )
236, 6101705.5938, -3992835.8125, 297.36010742
237, 6099955.5938, -3962585.8125, 595.60552979
*Element, Type=S3R, Elset=gface2
34279, 42, 59, 4
329262, 2, 59, 42
*Element, Type=S3R, Elset=gface3
550757, 1, 30, 31
1011559, 31, 30, 8
*Element, Type=S3R, Elset=gface4
909933, 5, 86, 6
```

It features 237 nodes, but actually only 5 faces. We then look for the coordinates of each face's node, here, for instance, the coordinates of nodes 42,59,4,2,42,etc.

57.3 Fixing the intersection

Now that the small features have been localized, we will try to fix them. To do that we first load the gocad model prior to the intersection, and we also load the intersected surfaces (previously save in a ts file), to keep track of the localization of the small features. Then for each small features, we identify why the intersection failure, and we amend the triangulation, to remove the cause of the failure. Usually, the failure is due to intersecting or almost intersecting edges. For instance here:



We see that the small feature seems related to the edge intersection circled. We then switch 2 triangles of the blue surface sharing the incriminated edge, which has the virtue of moving the edge. For that we use (for instance):

Right-click on the surface>Tools>Triangles>Switch Triangles> click on both triangles.

When then apply this manual procedure on all detected small features, and we finally make the mutual intersection. Hopefully, the model once loaded in SimModeler is then free of small features!

CHAPTER

FIFTYEIGHT

OVERVIEW

This documentation is a collection of useful dynamic simulation examples to help users build models from scratch with little effort. Each example is demonstrated carefully with geometry building, parameter setup, and result visualization. Users are suggested to repeat each example in order to get a comprehensive idea of how to set up dynamic simulation models with SeisSol.

SeisSol is a part of SCEC dynamic code validation project (Harris et al. 2018) (http://strike.scec.org/cvws/). Here we show several SCEC benchmarks for beginners to quickly catch up with SeisSol workflow. Each benchmark example is composed of a short problem description, a section of *geometry, initial setups (stress, nucleation, friction, etc.)*, and *simulation results*.

Please note that the examples used here are only for demonstration purpose. For detailed benchmark tests please refer to SCEC benchmark center.

Description	Source	Faulting mechanism[Page 189, 1]	Friction law[Page 189, 2]	Number of faults	Further details
TPV5	dynamic	SS	LSW	1	3 stress asperities, see *SCEC TPV5*
TPV6	dynamic	SS	LSW	1	Bi-material fault, heterogeneous initial stress, see *SCEC TPV6*
TPV12	dynamic	N	LSW	1	depth-dependent initial stress conditions, see *SCEC TPV12*
TPV13	dynamic	N	LSW	1	Same as TPV12 with non-associative Drucker-Prager plastic with yielding in shear, see *SCEC TPV13*
TPV16	dynamic	SS	LSW	1	Randomly-generated heterogeneous initial stress conditions, see *SCEC TPV16/17*
TPV24	dynamic	SS	LSW	2	Rightward branch forming a 30 degree angle, see *SCEC TPV24*
TPV29	dynamic	SS	LSW	1	Stochastic roughness, see *SCEC TPV29*
TPV34	dynamic	SS	LSW	1	Imperial Fault model with 3D velocity structure, see *SCEC TPV34*
TPV104	dynamic	SS	fvw-RS	1	see *SCEC TPV104*
LOH.1	point	n/a	n/a	n/a	point-source benchmark, see *SISMOWINE WP2_LOH1*
Northridge	kinematic	R	n/a	1	see *Kinematic source example - 1994 Northridge earthquake*

58.1 Prerequisites

Before you begin any of the examples, you will need to install the latest SeisSol from (https://github.com/SeisSol/SeisSol). The instruction can be found at https://seissol.readthedocs.io/en/latest/compiling-seissol.html. All geometry and tetrahedral meshes are generated using free software Gmsh (http://gmsh.info/). If you do not wish to create your own mesh at this time, the meshes are also provided as part of the example. The ParaView visualization package (https://www.paraview.org/) may be used to view simulation results. You may use other visualization software, but some adaptions from what is described here will be necessary. Furthermore, you can complete a subset of the example using files provided (as described below), skipping the steps for which you do not have the proper software packages installed.

58.2 Input file resources

The files needed to work through the examples are provided. All files necessary to set up the cookbook examples can be downloaded at https://github.com/SeisSol/Examples

58.3 References

Harris, R. A., Michael Barall, B. T. Aagaard, S. Ma, and K. B. O. Daniel Roten, Benchun Duan, Dunyu Liu, Bin Luo, Kangchen Bai, Jean-Paul Ampuero, Yoshihiro Kaneko, Alice-Agnes Gabriel, Kenneth Duru, Thomas Ulrich, Stephanie Wollherr, Zheqiang Shi, Eric Dunham, Sam Bydlon, Zhenguo Zhang, Xiaofei Chen, Surendra N. Somala, Christian Pelties, Josue Tago, Victor Manuel Cruz-Atienza, Jeremy Kozdon, Eric Daub, Khurram Aslam, Yuko Kase, Kyle Withers (2018), A Suite of Exercises for Verifying Dynamic Earthquake Rupture Codes, Seismol. Res. Lett., 89(3), 1146-1162, doi:10.1785/0220170222.

[1] SS: strike-slip, N: normal, R: reverse, O: oblique

[2] LSW: linear slip-weakening friction, fvw-RS: fast-velocity weakening rate-and-state friction

CHAPTER

FIFTYNINE

SCEC TPV5

TPV5 is the first SCEC benchmark. It has spontaneous rupture on a **vertical strike-slip fault in a homogeneous halfspace**. There are slightly heterogeneous initial stress conditions. The earthquake rupture is artificially nucleated in a square zone at the center of the fault surface. The rupture then spontaneously propagates over the rest of the fault surface. As it propagates away from the nucleation zone, it encounters two square patches with initial stress conditions that are different from the rest of the fault surface.

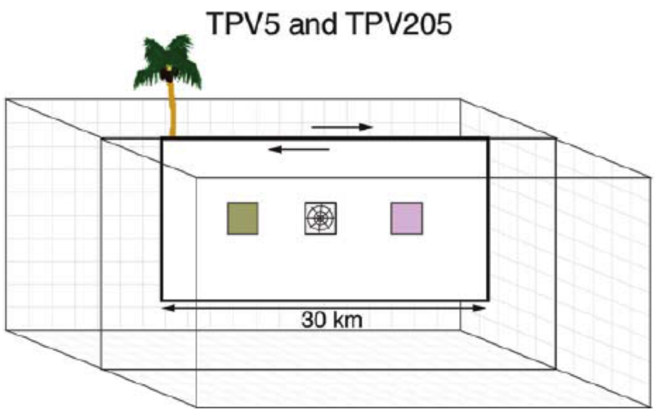


Fig. 1: Diagram of TPV5. The central square patch is the nucleation zone, while pink and green patches with higher and lower initial stress than neighbour region, respectively.

59.1 Geometry

The fault within the three-dimensional medium is a vertical right-lateral strike-slip planar fault that resides in a half-space. The fault reaches the Earth's surface. The rupture is allowed within a rectangular area that is 30000 m long × 15000 m deep. The bottom boundary of and the right and left ends of the allowed 30000 m × 15000 m rupture area are defined by a strength barrier. The nucleation point is centered both along-dip and along-strike of the 30000m × 15000m rupture area, on the fault plane, at 15000m along-strike and 7500m depth.

The mesh is generated in GMSH. All the files that are needed for the simulation are provided.

1. The tpv5.geo file contains the geometry for the fault in a cubit region.

   2. Then the .geo file can be meshed by using:

```
$ gmsh tpv5.geo -3 -optimize -o tpv5.msh
```

   3. Then convert the .msh file to 3D Gambit neutral file

```
$ gmsh2gambit -i tpv5.msh -o tpv5.neu
```

The toolbox of **gmsh2gambit** is used for converting gmsh file to Gambit neutrual file. It can be found in SeisSol GitHub https://github.com/SeisSol/SeisSol/tree/master/preprocessing/meshing

4. The 3D Gambit file can be converted to PUML format for LTS in latest version of SeisSol by:

```
$ pumgen tpv5.neu tpv5
```

The compilation and usage of PUMGen can be found in https://github.com/SeisSol/PUMGen/wiki and https://seissol.readthedocs.io/en/latest/ The geometry file (.geo) can be found at https://github.com/SeisSol/Examples/blob/master/tpv5/tpv5_f200m.geo. The mesh file can be generated using the bash file https://github.com/SeisSol/Examples/blob/master/tpv5/generating_the_mesh.sh.

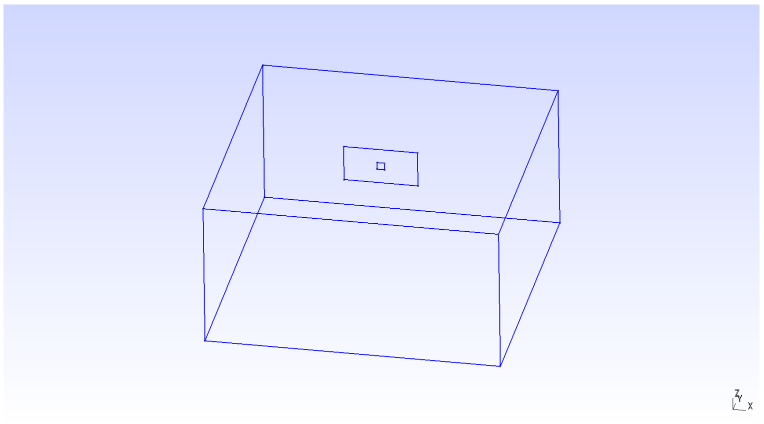


Fig. 2: Diagram of fault geometry of TPV5. The fault is 30000 m long and 15000 m wide. The square patch has a side-length of 3000m.

59.2 Parameters

59.2.1 Nucleation

Nucleation occurs because of the initial shear stress in a 3000 m $\times$ 3000 m square nucleation patch is set to be higher than the initial static yield stress in that patch. Failure occurs everywhere on the fault plane, including in the nucleation patch, following a linear slip-weakening fracture criterion.

TPV5 uses a linear-slip weakening friction everywhere on the fault. There are ten parameters associated with the friction constitutive law and fault properties in the **parameters.par**. It can be found at https://github.com/SeisSol/Examples/blob/master/tpv5/parameters.par.

Four friction constitutive parameters are: mu_s, mu_d, d_c and cohesion. Six stress parameters are: s_xx, s_yy, s_zz, s_xy, s_xz, and s_yz. All the parameters are homogeneous on the fault except for the nucleation patch in the center of the fault, where s_xy is larger compared with that elsewhere. The parameters in TPV5 are listed in Table [table:tpv5].

Parameter	Description	Value	Unit
mu_s	static friction coefficient	0.677	dimensionless
mu_d	dynamic friction coefficient	0.525	dimensionless
d_c	critical distance	0.40	m
cohesion	friction cohesion	0.0	MPa
s_yy	stress	120	MPa
s_xx,s_zz,s_yz,s_xz	stress	0	MPa
s_xy	outside the nucleation zone	70	MPa
	inside the nucleation zone	81.6	MPa

Table: Table of LSR parameters on the fault in tpv5.

Notice that there are two patches with different initial stress: the one centered at (+7.5, -7.5) has 62 MPa and (-7.5, -7.5) has 78 MPa. This initial stress is included in the fault.yaml file.

Results

All examples here generate surface and volume output files that can be visualized with ParaView. The *output* folder contains a series of files for fault dynamic rupture (hdf5 and .xdmf), wavefield (hdf5 and .xdmf), on-fault receiver (.dat) and off-fault receivers (.dat). The fault dynamic rupture and wavefield files can be loaded in Paraview. For example, open Paraview and then go through File >> import >>prefix-fault.xdmf.

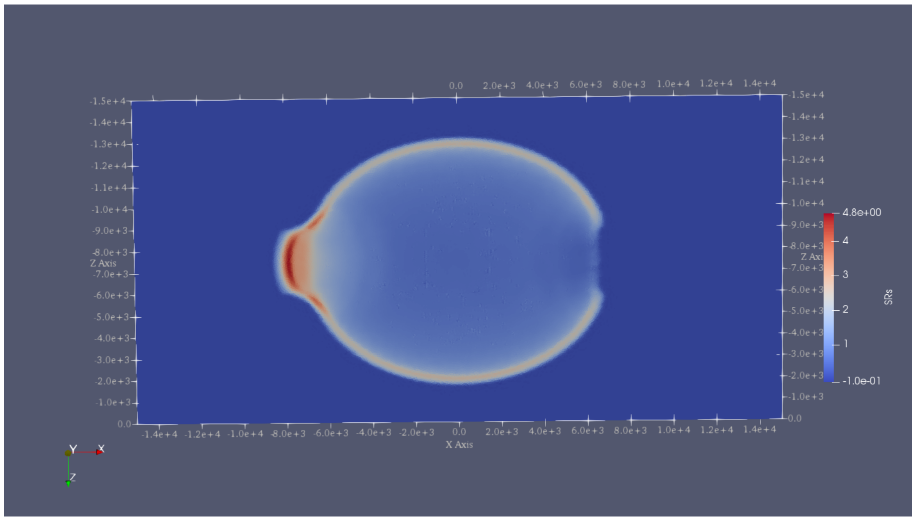


Fig. 3: Fault slip rate in the along-strike direction (SRs) at 4 seconds in TPV5, illustrated in Paraview.

In the wave filed output file (prefix.xdmf, prefix_vertex.h5 and prefix_cell.hf), the variables are shown in Table [table:wavefield]

Index	Parameter	Description
1	U	displacement in x-axis
2	V	displacement in y-axis
3	W	displacement in z-axis
4	u	particular velocity in x-axis
5	v	particular velocity in y-axis
6	w	particular velocity in z-axis

Table: Table of wavefield output in SeisSol. Index denotes the position used in *iOutputMask* in SeisSol parameter file.

In the fault dynamics output file (prefix-fault.xdmf, prefix-fault_vertex,h5 and prefix-fault_cell,h5), the variables are shown in Table [table:faultout]

Index	Parameter	Description
1	SRs and SRd	slip rates in strike and dip direction
2	T_s, T_d, P_n	transient shear stress in strike and dip direction, transient normal stress
3	U_n	normal velocity (note that there is no fault opening in SeisSol)
4	Mud, StV	current friction and state variable in case of RS friction
5	Ts0,Td0,Pn0	total stress, including initial stress
6	Sls and Sld	slip in strike and dip direction
7	Vr	rupture velocity, computed from the spatial derivatives of the rupture time
8	ASl	absolute slip
9	PSR	peak slip rate
10	RT	rupture time
11	DS	only with LSW, time at which ASl > d_c

Table: Table of fault dynamic output in SeisSol. Index denotes the position used in *iOutputMask* in SeisSol parameter file.

CHAPTER

SIXTY

SCEC TPV6

TPV6 is intended to reside in the "well-posed" regime for **bimaterial problems** and so uses a very high shear modulus (density*vs*vs) contrast. Material properties are homogeneous within each side of the fault, but change when one traverses to the other side of the fault.

Parameter	Description	Value	Unit
Vp	P velocity of the far side	3750	m/s
Vs	S velocity of the far side	2165	m/s
ρ	density of the far side	225	kg/m3
Vp	P velocity of the near side	6000	m/s
Vs	S velocity of the near side	3464	m/s
ρ	density of the near side	2670	kg/m3

Table: Table of bi-material parameters in Tpv6.

60.1 Geometry

TPV6 uses the same geometry as TPV5. The fault within the three-dimensional medium is a vertical right-lateral strike-slip planar fault that resides in a half-space. The fault reaches the Earth's surface. The rupture is allowed within a rectangular area that is 30000 m long $\times$ 15000 m deep. The bottom boundary of and the right and left ends of the allowed 30000 m $\times$ 15000 m rupture area are defined by a strength barrier. The nucleation point is centered both along-dip and along-strike of the 30000m $\times$ 15000m rupture area, on the fault plane, at 15000m along-strike and 7500m depth.

60.2 Parameters

TPV6 uses a similar parameter setup as TPV5 except for the bulk parameters.

Parameter	Description	Value	Unit
ρ	density of the far side	2225	kg/m3
λ	Lame parameter of the far side	10.4	GPa
μ	Lame parameter of the far side	10.4	GPa
ρ	density of the near side	2670	kg/m3
λ	Lame parameter of the near side	32	GPa
μ	Lame parameter of the near side	32	GPa

Table: Table of bi-material parameters used in SeisSol for Tpv6.

60.3 Results

Figure [fig:tpv6-4s] and [fig:tpv6-7s] show the fault slip rate at 4 s and 7 s, respectively. The slip front is asymmetric when compared with TPV5 (Figure [fig:tpv5-4s]). Figure [fig:tpv6_velocity] shows velocity recorded at two off-fault receivers. The wave picks arrives at the far-side receiver lower than those at the near-side receiver.

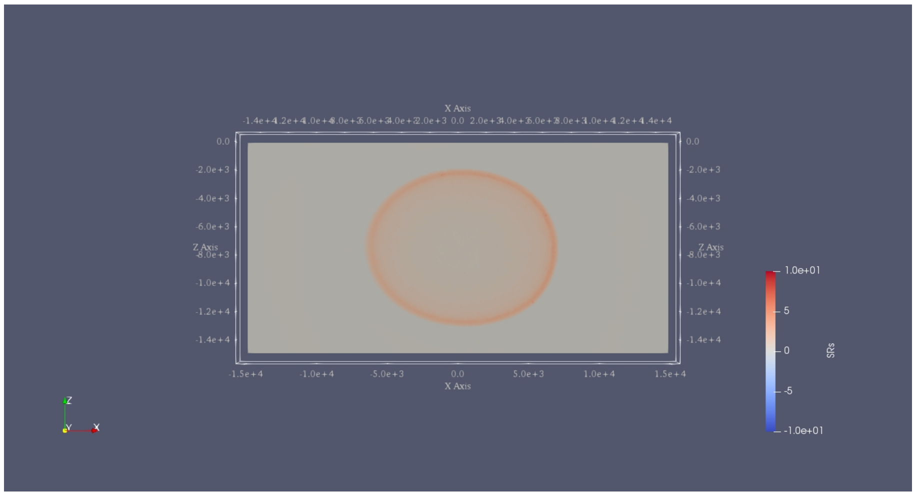


Fig. 1: Fault slip rate at 4 seconds in the along-strike direction in TPV6.

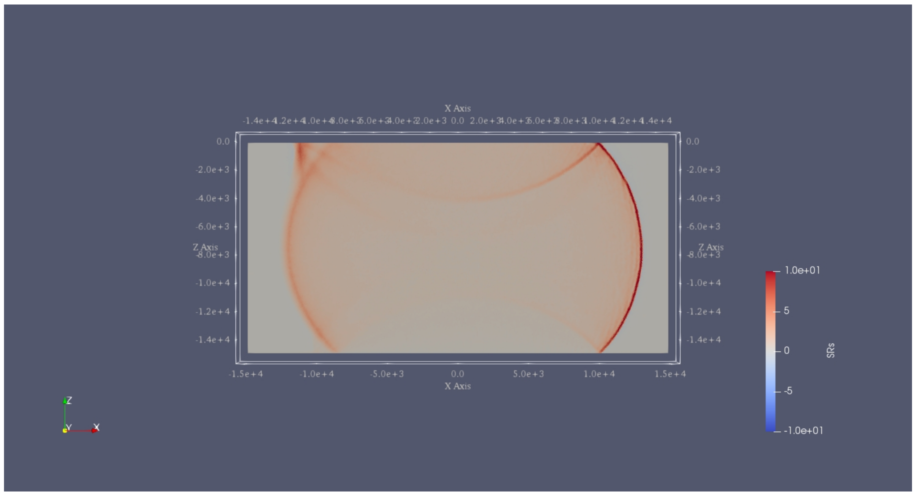
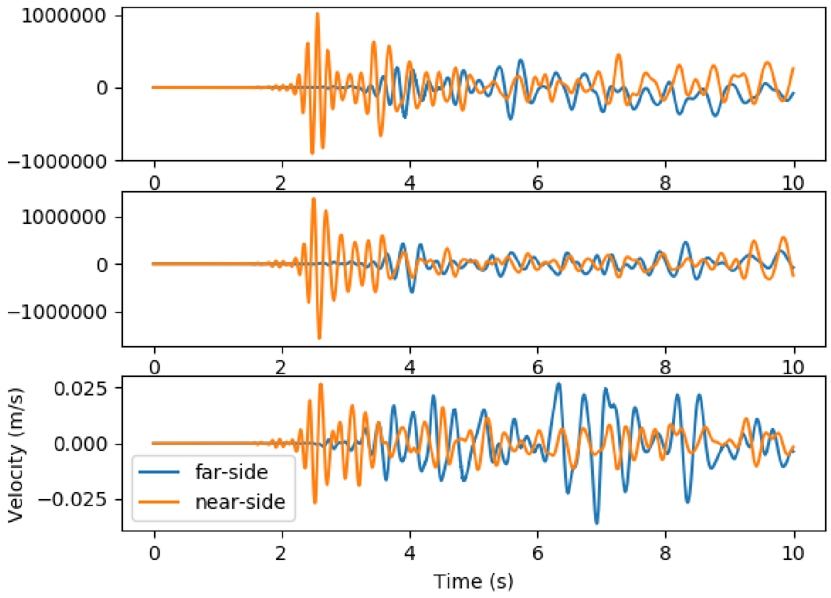


Fig. 2: Fault slip rate at 7 seconds in the along-strike direction in TPV6.


1000000
0
−1000000
0
2
4
6
8
10
0.025
0.000
−0.025
Velocity (m/s)
far-side
near-side
Time (s)

CHAPTER

SIXTYONE

SCEC TPV12

TPV12 and 13 are verification benchmarks for a 60 degree dipping normal fault embedded in a homogeneous halfspace. The rheology is linear elastic in TPV12 and non-associative Drucker-Prager visco-plastic in TPV13. Initial stress conditions are dependent on depth. Strongly super-shear rupture conditions are assumed.

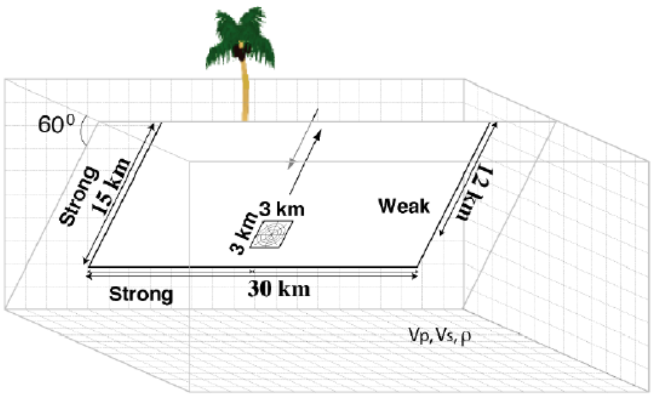


Fig. 1: Figure 1: Geometry in SCEC benchmarks TPV12/13, including a 60-degree dipping normal fault.

61.1 Geometry

A half-space domain is assumed. The fault is a 60-degree dipping, 30 km × 15 km planar, normal fault. It reaches the Earth's surface, and extends up to 12.99038 km depth. A square nucleation area of 3 × 3 km is assumed. In fault coordinates, it is centered at (0, -12) km, that is at a depth of 10.3923 km. The CAD model (see *Figure 2*) and mesh are generated with Gmsh. All the files that are needed for the simulation are provided here. The geometry and mesh generation process is similar to TPV5.

61.2 Nucleation strategy

In previous benchmarks, nucleation is achieved by imposing higher initial shear stress within the nucleation zone. In TPV12 and TPV13, nucleation is achieved by decreasing the static friction coefficient, leading the initial shear stress to exceed fault strength.

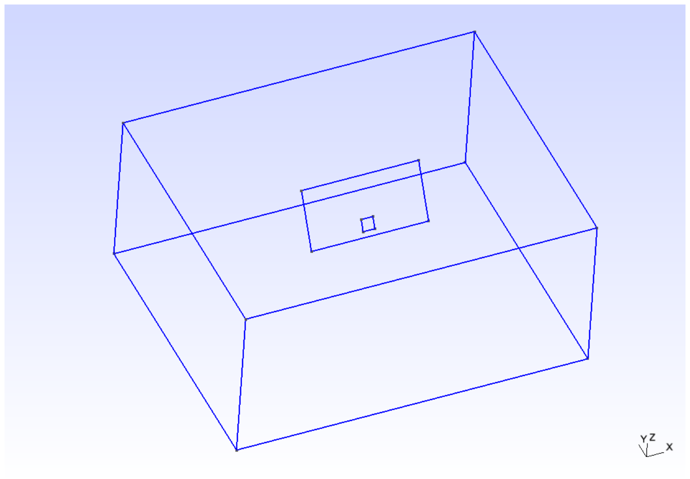


Fig. 2: Figure 2: TPV12/13 geometry modeled in Gmsh. The domain box is 500 km × 500 km × 50 km. The fault reaches the top surface.

61.3 Parameters

61.3.1 friction parameters

TPV12 uses a linear slip weakening law with different parameters inside and outside the nucleation zone. The parameters are listed in the table below:

Parameter	description	Value	Unit
d_c	critical distance	0.50	m
cohesion	shear stress cohesion	-200 000	Pa
inside the nucleation zone			
mu_s	static friction coefficient	0.54	
mu_d	dynamic friction coefficient	0.10	
outside the nucleation zone			
mu_s	static friction coefficient	0.70	
mu_d	dynamic friction coefficient	0.10	

Table 1: frictions parameters used in TPV12/13.

61.4 Initial stress

The initial stress is assumed depth-dependent in TPV12/13. Above 11951.15 m depth, deviatoric stresses are non-zero, and the stress field is well orientated for rupture of the normal fault. Below, the stress is isotropic.

Parameter	Value
above 11951.15 m depth	
σ_1	26460 Pa/m × H
σ_3	15624.3 Pa/m × H
σ_2	$(\sigma_1 + \sigma_3)/2$
P_f	$1000\mathrm{kg/m^3} \times 9.8\mathrm{m/s^2} \times H$
below 11951.15 m depth	
$\sigma_1, \sigma_2, \sigma_3$	$2700\mathrm{kg/m^3} \times 9.8\mathrm{m/s^2} \times H$

61.5 Results

SeisSol output can be visualized directly in Paraview by loading their xdmf files.

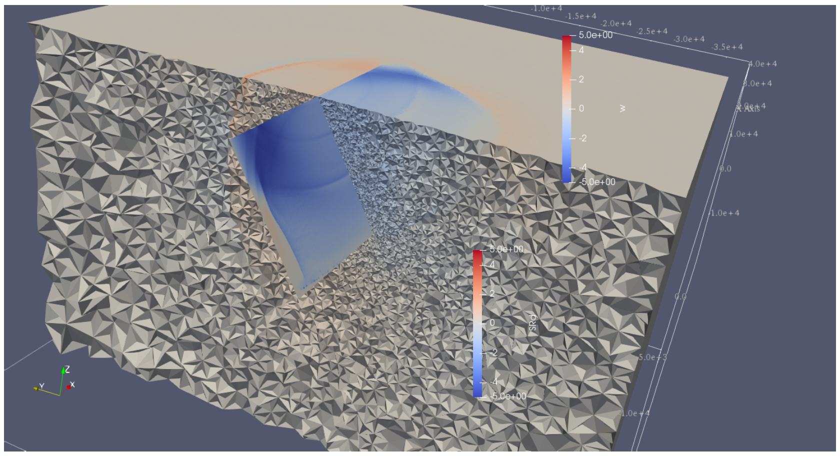


Fig. 3: Figure 3: Fault and volume output of TPV12 visualized in Paraview. Fault slip rate in dip-direction (SRd) and vertical velocity (w) in the volume. A cut-view of the volume output allows visualizing the unstructured tetrahedral mesh.

CHAPTER

SIXTYTWO

SCEC TPV13

TPV13 is similar to TPV12 except for the **non-associative Drucker-Prager visco-plastic** rheology.

The Drucker-Prager yield function is given by:

$$F(\sigma) = \sqrt{J_2(\sigma)} - Y(\sigma)$$

$Y(\sigma)$ is the Drucker-Prager yield stress, given as:

$$Y(\sigma) = \max(0, c\cos\phi - (\sigma_m + P_f)\sin\phi)$$

with $\sigma_m = (\sigma_{11} + \sigma_{22} + \sigma_{33})/3$ the mean stress,

c the bulk cohesion, ϕ the bulk friction and P_f the fluid pressure (1000 kg/m ^{3}). In TPV13 benchmark, $c = $ 5.0e+06 Pa and ϕ =0.85.

J_2 is the second invariant of the stress deviator:

$$J_2(\sigma) = 1/2\sum_{ij} s_{ij}s_{ji}$$

with $s_{ij} = \sigma_{ij} - \sigma_m\delta_{ij}$ the deviator stress components.

The yield equation has to be satisfied:

$$F(\sigma) \leq 0$$

When $F(\sigma) < 0$, the material behaves like a linear isotropic elastic material, with Lame parameters λ and μ.

Wen $F(\sigma) = 0$, if the material is subjected to a strain that tends to cause an increase in $F(\sigma)$, then the material yields and plastic strains accumulates.

62.1 Nucleation

TPV13 uses the same nucleation strategy as TPV12.

62.2 Plasticity parameters

To turn on plasticity in SeisSol, add the following lines in parameters.par:

```
&Equations
Plasticity = 1 ! default = 0
Tv = 0.03 ! Plastic relaxation
/
```

Plasticity related parameters are defined in **material.yaml**:

```
!Switch
[rho, mu, lambda, plastCo, bulkFriction]: !ConstantMap
  map:
    rho:                  2700
    mu:            2.9403e+010
    lambda:         2.941e+010
    plastCo:           5.0e+06
    bulkFriction:         0.85
[s_xx, s_yy, s_zz, s_xy, s_yz, s_xz]: !Include tpv12_13_initial_stress.yaml
```

62.3 Results

Figure 1 compares the slip-rates along strike and dip in TPV12 (elastic) and TPV13 (visco-plastic). The peak slip rate in TPV12 is higher than in TPV13. This difference can be attributed to the inelastic response of the off-fault material. See Wollherr et al. (2018) for detailed discussions.

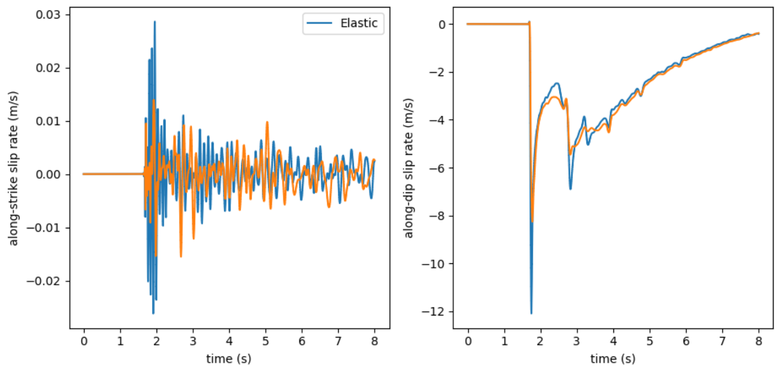


Fig. 1: Figure 1: along-strike (left) and along-dip (right) slip rate in TPV12 (blue) and 13 (orange).

CHAPTER

SIXTYTHREE

SCEC TPV16/17

TPV16/17 has spontaneous rupture on a vertical, right-lateral, strike-slip fault in a homogeneous half-space with **randomly-generated heterogeneous initial stress conditions**. The earthquake rupture is artificially nucleated in a circular zone on the fault surface. The rupture then spontaneously propagates outward on the fault surface and encounters heterogeneous stochastic initial stress conditions, some of which prevent it from propagating into certain regions on the fault surface.

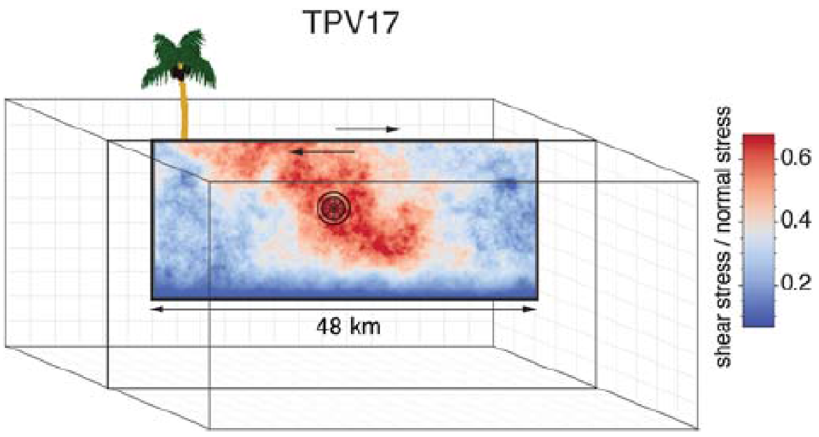


Fig. 1: Diagram of TPV16/17. The fault is 40 km long. Colors indicate the ratio of shear stress to normal stress at locations on the fault surface, at the beginning of the simulation.

63.1 Geometry

The fault is a vertical, planar, right-lateral, strike-slip fault. The fault reaches the Earth's surface. Rupture is allowed within a rectangular area measuring 48000 m along-strike and 19500 m down-dip. A node which lies exactly on the border of the 48000 m $\times$ 19500 m rectangle is considered to be inside the rectangle, and so should be permitted to slip.

The portions of the fault below, to the left of, and to the right of the 48000 m $\times$ 19500 m rectangle are a strength barrier, within which the fault is not allowed to rupture.

In the example, a vertical fault is generated with Gmsh in Figure [fig:tpv16mesh]. All the files that are needed for the simulation are provided in https://github.com/SeisSol/Examples/tree/master/tpv16.

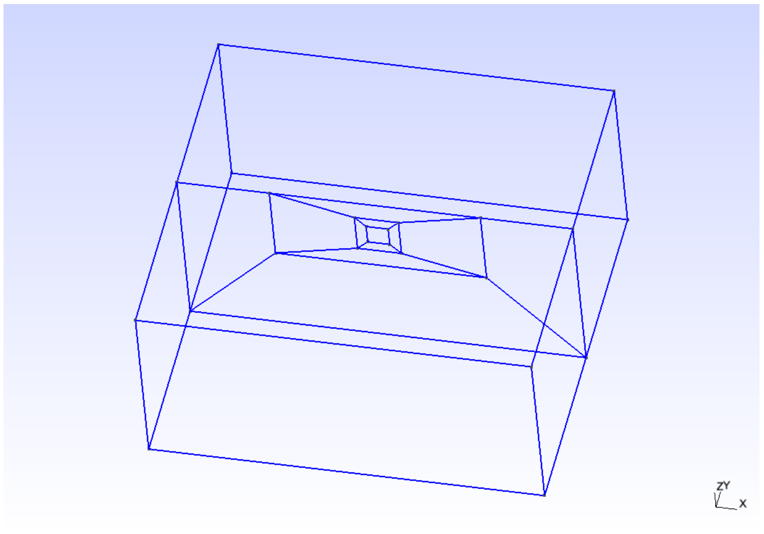


Fig. 2: Fault geometry of TPV16. Planar fault with nucleation size of 200 m.

63.2 Material parameter

Rock properties are taken to be linear elastic throughout the 3D model volume. The problem description can be found at . Table [table:tpv16material] lists all the material parameters.

Parameter	Description	Value	Unit
λ	Lame's first parameter	3.2044e10	Pa
μ	shear module	3.2038e10	Pa
ρ	density	2670	kg/m^3
Q_p	P-wave attenuation	69.3	
Q_s	S-wave attenuation	155.9	
h_{edge}	element edge length	200	m

Table: Table of bulk and material parameters in TPV16/17.

63.3 Nucleation parameters

Initial stress (Ts0) is randomly-generated in TPV16/17 (Figure [fig:tpv16ts]).

In TPV16/17, a two-stage nucleation method is used. The first stage is a circular zone of forced rupture which surrounds the hypocenter. Its radius is approximately 1 km (the exact radius is determined as part of the stochastic method that generates the initial stresses). At the hypocenter, the value of then increases with distance from the hypocenter, which creates an expanding circular region of forced rupture. The forced rupture expands at a speed of for 80% of the way, and then for the remaining 20% of the way to the edge of the zone. Outside the zone of forced rupture, is equal to 1.0E9, which means that forced rupture does not occur outside the zone.

The second stage is a circular zone of reduced which surrounds the hypocenter. Its radius is approximately 4 km (the exact radius is determined as part of the stochastic method that generates the initial stresses). In the innermost 10% of the zone, equals 0.04 m. The value of then increases linearly with distance from the hypocenter and reaches its final value of 0.4 m at the edge of the zone. Outside the zone, equals 0.4 m. The effect is to create a circular region of reduced fracture energy surrounding the hypocenter, which helps the rupture to expand during the early part of the simulation.

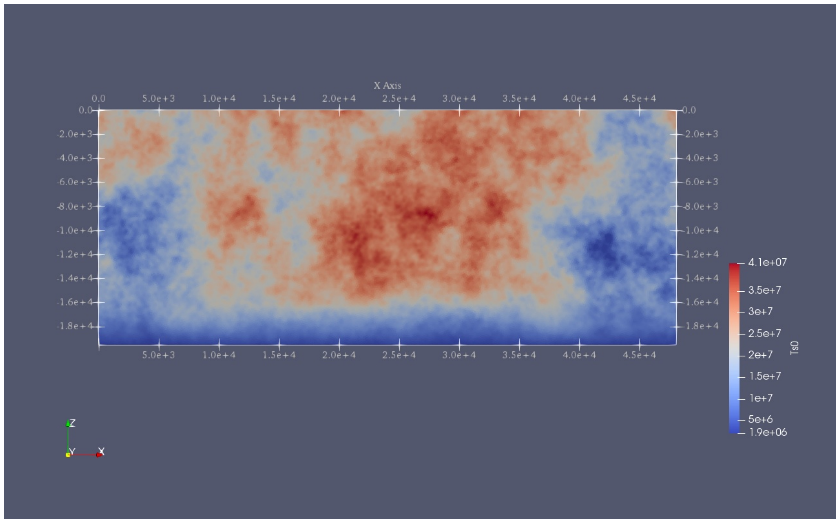


Fig. 3: Mapview of fault randomly-generated initial stress in TPV16.

63.4 Results

The earthquake nucleates and the rupture propagates on the fault surface due to the heterogenous stress ratio on the fault. Figure [fig:tpv16slip] shows the fault slip rate along strike-direction at T=5.5 s.

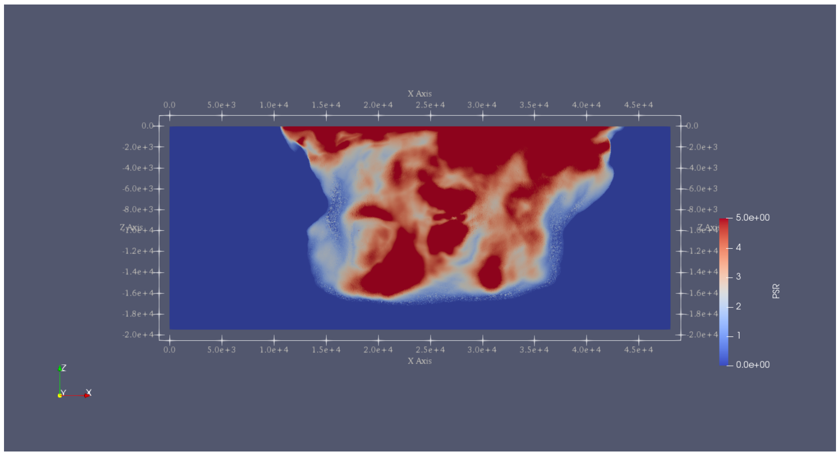


Fig. 4: Mapview of fault slip rate along strike-direction.

There are several receivers on the fault surface. Figure [fig:tpv16fault] shows slip rate along the strike- and downdip-direction on the fault at point (15 km, 0 km, -9 km).

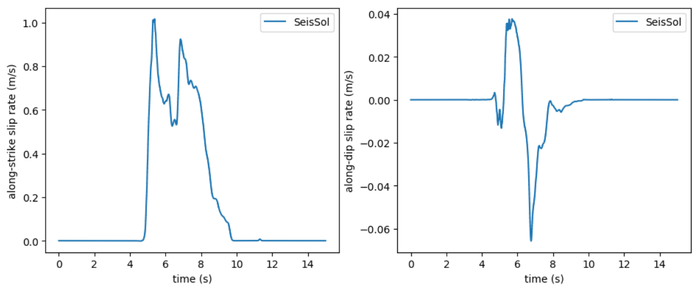


Fig. 5: Fault slip along strike- (left)and downdip- (right) direction.

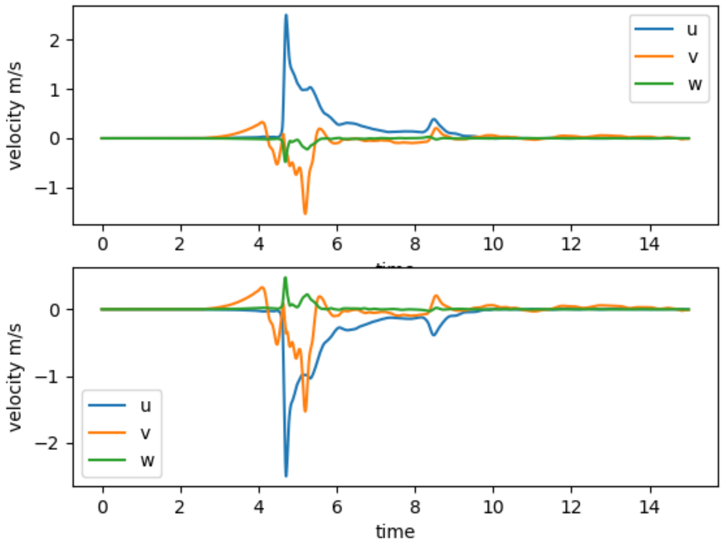


Fig. 6: Velocity at two opposite stations across the fault surface.

CHAPTER

SIXTYFOUR

SCEC TPV24

TPV24 is designed to illustrate dynamic rupture in a **fault branching** system. TPV24 contains two vertical, planar strike-slip faults; the main fault and a branch fault intersecting at an angle of 30 degrees (Figure [fig:tpv24]). The earthquake rupture is artificially nucleated in a circular zone on the main fault surface and then spontaneously propagates to the branching fault.

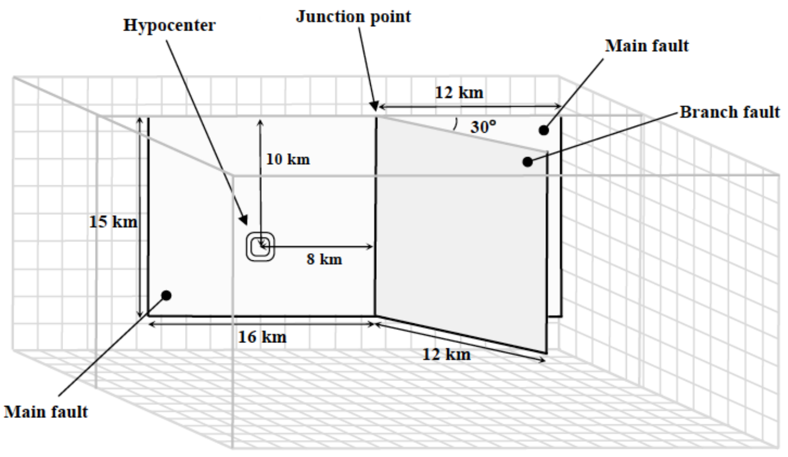


Fig. 1: Diagram of TPV24 branching fault geometry. The main fault is (16 000 + 12 000) m and the branching fault is 12 000 m. Both faults are 15 000 m wide. The intersecting angle is 30 degrees. The nucleation patch locates at 10 000 m depth and 8000 m horizontally from the joint point.

64.1 Geometry

There are two faults, called the main fault and the branch fault (Figure [fig:tpv24]). The two faults are vertical, planar, strike-slip faults. The faults reach the earth's surface.

The main fault is a rectangle measuring 28 000 m along-strike and 15 000 m deep. The branch fault is a rectangle measuring 12 000 m along-strike and 15 000 m deep. There is a junction point. It is located 12 000 m from the right edge of the main fault, and the main fault passes through it.

The branch fault makes an angle of 30 degrees to the main fault. The branch fault ends at the junction point.

The hypocenter is centered along-strike at a depth of 10 km on the left side of the main fault. That is, the hypocenter is 8000 m from the junction point, and 10 000 m deep.

Figure [fig:tpv24mesh] shows the fault model generated in Gmsh. The mesh file can be generated using https://github.com/SeisSol/Examples/blob/master/tpv24/generating_the_mesh.sh.

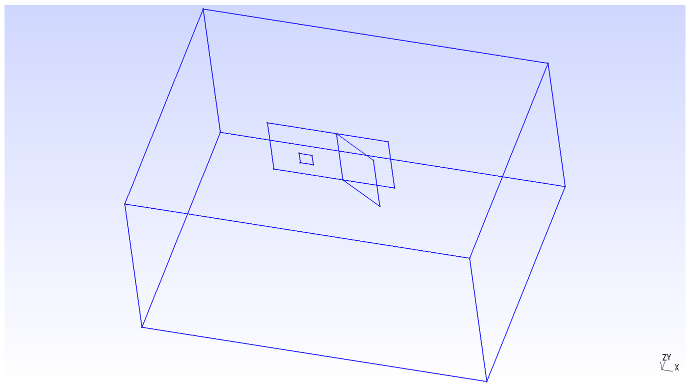


Fig. 2: Geometry generated in Gmsh of TPV24. The main fault lies in y-plane.

64.2 Initial stress

The initial stress condition is depth-dependent at the depth above 15600 m. Table [table:tpv24] summarizes the initial stress conditions in TPV24.

Parameter	Description	Value	Unit
s_zz	σ_{zz}	-2670*9.8*depth | Pa	
s_xz	σ_{xz}	0	Pa
P_f	fluid pressure	1000*9.8*depth | Pa	
s_yz	σ_{yz}	0.0	Pa
inside the nucleation zone			
s_yy	$b33 * (\sigma_{zz} + P_f) - P_f$		Pa
s_xx	$b22 * (\sigma_{zz} + P_f) - P_f$		Pa
s_xy	$b23 * (\sigma_{zz} + P_f)$		Pa
outside the nucleation zone			
s_yy	σ_{zz}		Pa
s_xx	σ_{zz}		Pa
s_xy	0		Pa

Note that the effective stress tensor is :

$$\bar{\sigma}_{effective} = \begin{bmatrix} \sigma_{xx} + P_f, & \sigma_{xy}, & \sigma_{xz} \\ \sigma_{xy}, & \sigma_{yy} + P_f, & \sigma_{yz} \\ \sigma_{xz}, & \sigma_{yz}, & \sigma_{zz} + P_f \end{bmatrix}$$

64.3 Nucleation parameters

Nucleation is performed by forcing the fault to rupture, within a circular zone surrounding the hypocenter. Forced rupture is achieved by artificially reducing the friction coefficient, beginning at a specified time. The parameter specifies how long it takes for the friction coefficient to be artificially reduced from its static value to its dynamic value. So, the friction coefficient reaches its dynamic value at time. We reduce the friction coefficient gradually, over an interval of time, in order to smooth the nucleation process and reduce unwanted oscillations.

$$T = \begin{cases} \frac{r}{0.7Vr} + \frac{0.081*r_{crit}}{0.7Vr}\left(\frac{1}{1-(r/r_{crit})^2} - 1\right), r \leq r_{crit} \\ 1E+09, r > r_{crit} \end{cases}$$

The **cohesion** zone is defined as :

$$C_0 = \begin{cases} 0.3 + 0.000675 * (4000 - depth), depth < 4000m \\ 0.3MPa, depth \geq 4000m \end{cases}$$

Note that the frictional cohesion is 3.00 MPa at the earth's surface. It is 0.30 MPa at depths greater than 4000 m, and its value is linearly tapered in the uppermost 4000 m.

The friction parameters are listed in Table [table:tpv24fric].

Parameter	Description	Value	Unit
mu_s	static friction coefficient	0.18	
mu_d	dynamic friction coefficient	0.12	
d_c	critical distance	0.30	m
C_0	fault cohesion		Pa
T	forced rupture time		s
t_0	forced rupture delay time	0.5	s

Table: Table of LSR parameters on the fault in TPV24.

64.4 Results

The model is run for 12.0 seconds after nucleation. The earthquake rupture is artificially nucleated in a circular zone on the main fault surface. The rupture then spontaneously propagates on the main fault and encounters a branching fault. The branching fault continues to rupture as well as the rest main fault. The fault slip rate is shown in Figure [fig:tpv24result1].

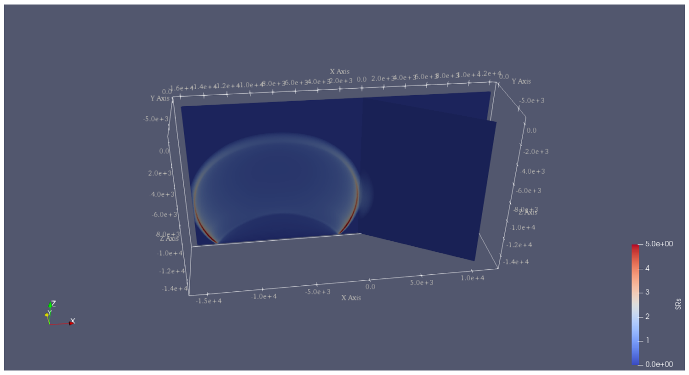


Fig. 3: Snapshot of slip rate in branching fault system. Slip rate at 2 s.

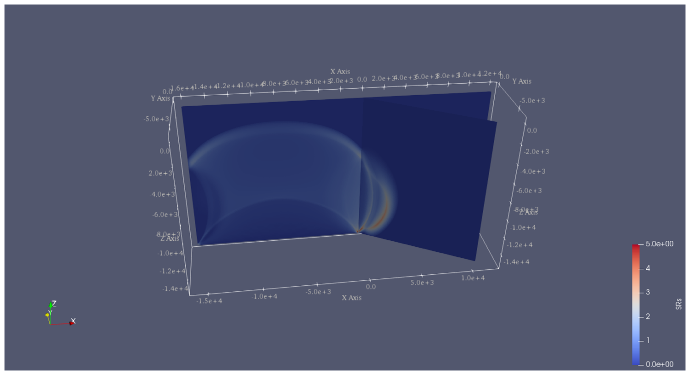


Fig. 4: Snapshot of slip rate in branching fault system. Slip rate at 3.5 s.

CHAPTER

SIXTYFIVE

SCEC TPV29

TPV 29 contains a vertical, right-lateral fault with **rough fault interface** (Figure [fig:tpv29]). The fault surface has 3D stochastic geometrical roughness (blue and red colors). In TPV 29, the surrounding rocks respond elastically.

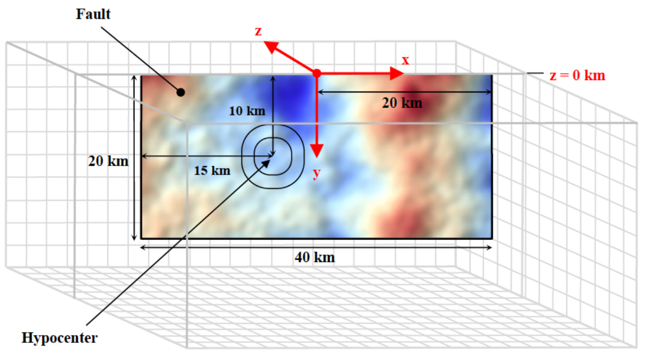


Fig. 1: Diagram of TPV 29. The fault is 40 km long along the strike. There is a circular nucleation zone on the right-lateral fault surface. The fault surface has 3D stochastic geometrical roughness (blue and red colors). The hypocenter is located 15 km from the left edge of the fault, at a depth of 10 km.

65.1 Geometry

The roughed fault interface model is generated with Gmsh is complicated than planar faults in previous sections. There are 5 steps to generate the model.

1.Download fault topography data from SCEC. There are 2001 nodes
along the strike and 1201 nodes along the downdip. The node files should contain:

```
Line 1: nx, ny
Line 2 to nx: positions of nodes along the strike (in meters)
Line nx+3 to ny+nx+3: positions of nodes along the downdip (in meters)
Line to the end: fault topography of each nodes (nx\*ny, in meters)
```

Using generate_mytopo_tpv29.py to create the topography data.

2.Make a model with plane fault as Figure [fig:tpv29geo]. The Gmsh tpv29.geo file can be found at https://github.com/SeisSol/Examples/blob/master/tpv29/tpv29.geo.

3.Use *gmsh_plane2topo.f90* and interpol_topo.in* to shift the planar fault according to positions given in *mytopo_tpv29*.

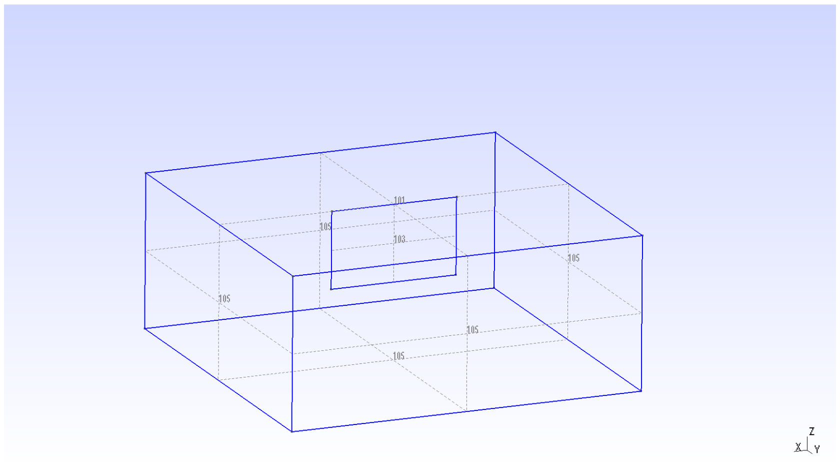


Fig. 2: Diagram showing the geometry of TPV 29. The center of nucleation is at (-8, 0, -10) km on the main fault.

```
$ ./gmsh_plane2topo interpol_topo.in
```

This will generate a step1_modified.msh file which containing rough fault surface.

4.Make a new step2.geo file that contains the new rough fault and mesh following general Gmsh process.

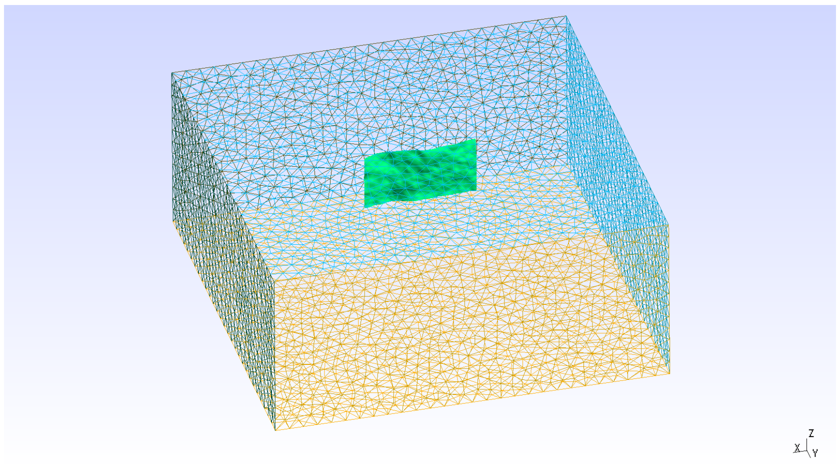


Fig. 3: Diagram showing the geometry of TPV 29. The center of nucleation is at (-8, 0, -10) km on the main fault.

5. Generate MSH mesh with the command line:

```
& gmsh tpv29_step2.geo -3 -optimize_netgen -o tpv29_step2.msh
```

option optimize_netgen is necessary for optimizing meshing with good quality.

6.Then convert the .msh file to 3D Gambit neutral file and PUML format as same as shown in TPV5

::

$ gmsh2gambit -i tpv29_step2.msh -o tpv29.neu $ pumgen tpv29.neu tpv29

The mesh can be created by using the bash script https://github.com/SeisSol/Examples/blob/master/tpv29/generating_the_mesh.sh.

Here we show a fully opensource workflow which allows generating a mesh accounting for tpv29 rough fault geometry. This yields a mesh that does not properly account for the intersection between fault and the free-surface. We note that it is here not an important issue, as the tpv29 benchmark does not feature surface rupturing. Another drawback of this workflow is that the rate of mesh size coarsening is not easy parametrizable. A more straightforward and accurate way to generate a mesh would be to use simModeler.

65.2 Material parameters

In TPV29, the entire model volume is a linear elastic material, with the following parameters listed in Table [table:tpv29material].

Parameter	Description	Value	Unit
ρ	density	2670	kg/m^3
λ	Lame's first parameter	3.2044e10	Pa
μ	shear module	3.2038e10	Pa
h_{edge}	element edge length	200	m
V_p	P wave velocity	6000	m/s
V_s	S wave velocity	3464	m/s

[table:tpv29material]

65.3 Initial stress

The initial stress are listed in Table [table:tpv29fault].

Parameter	Description	Value	Unit
mu_s	static friction coefficient	0.12	
mu_d	dynamic friction coefficient	0.18	
d_c	critical distance	0.30	m
s_zz	σ_{zz}	-2670*9.8*depth	Pa
Pf	fluid pressure	1000*9.8*depth	Pa
s_xz,s_yz	σ_{xz}, σ_{yz}	0	Pa
s_yy		$\Omega * b33 * (\sigma_{zz} + P_f) - P_f$	Pa
s_xx		$\Omega * b11 * (\sigma_{zz} + P_f) - P_f$	Pa
s_xy		$\Omega * b13 * (\sigma_{zz} + P_f)$	Pa

Table: Table of initial stress in TPV 29. $b11, b33, b13$ are 1.025837, 0.974162, 0.158649, respectively.

Note that the effective stress tensor is :

$$\bar{\sigma}_{effective} = \begin{bmatrix} \sigma_{xx} + P_f, & \sigma_{xy}, & \sigma_{xz} \\ \sigma_{xy}, & \sigma_{yy} + P_f, & \sigma_{yz} \\ \sigma_{xz}, & \sigma_{yz}, & \sigma_{zz} + P_f \end{bmatrix}$$

where Ω is defined as:

$$\Omega = \begin{cases} 1, depth \leq 17000m \\ (22000 - depth)/5000m, 17000 < depth < 22000m \\ 0, depth \geq 22000m \end{cases}$$

65.4 Nucleation parameters

TPV29 uses a similar strategy for dynamic rupture nucleation.

$$T = \begin{cases} \frac{r}{0.7Vr} + \frac{0.081*r_{crit}}{0.7Vr}(\frac{1}{1-(r/r_{crit})^2} - 1), r \leq r_{crit} \\ 1E+09, r > r_{crit} \end{cases}$$

The cohesion zone is defined as :

$$C_0 = \begin{cases} 0.4MPa + 0.000675MPa * (4000 - depth), depth < 4000m \\ 0.4MPa, depth \geq 4000m \end{cases}$$

The friction parameters on the fault are listed in Table [table:tpv29fric].

Parameter	Description	Value	Unit
mu_s	static friction coefficient	0.12	
mu_d	dynamic friction coefficient	0.18	
d_c	critical distance	0.30	m
t_0	forced rupture delay time	0.5	s

Table: Table of friction parameters in TPV 29.

65.5 Results

The earthquake rupture is artificially nucleated in a circular zone on the fault surface.

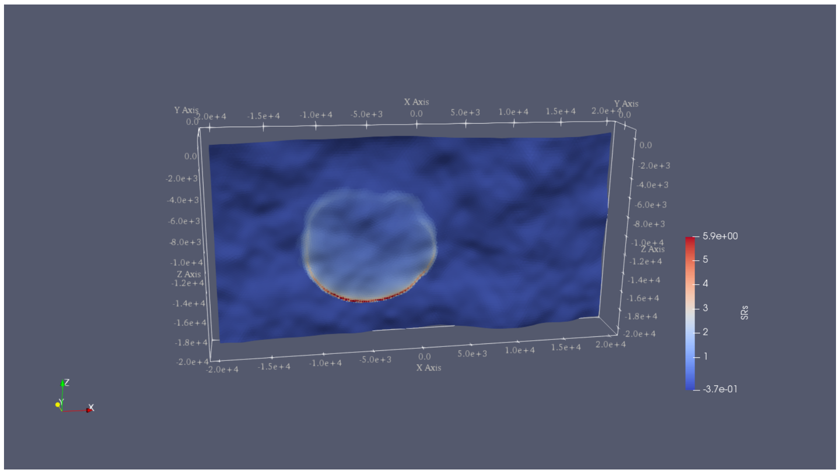


Fig. 4: Snapshot of slip rate along the strike at T=3 s in TPV 29. The fault has a rough surface.

CHAPTER

SIXTYSIX

SCEC TPV34

The TPV34 benchmark features a right-lateral, planar, vertical, strike-slip fault set in a half-space. The velocity structure is the actual 3D velocity structure surrounding the Imperial Fault, as given by the SCEC Community Velocity Model CVM-H.

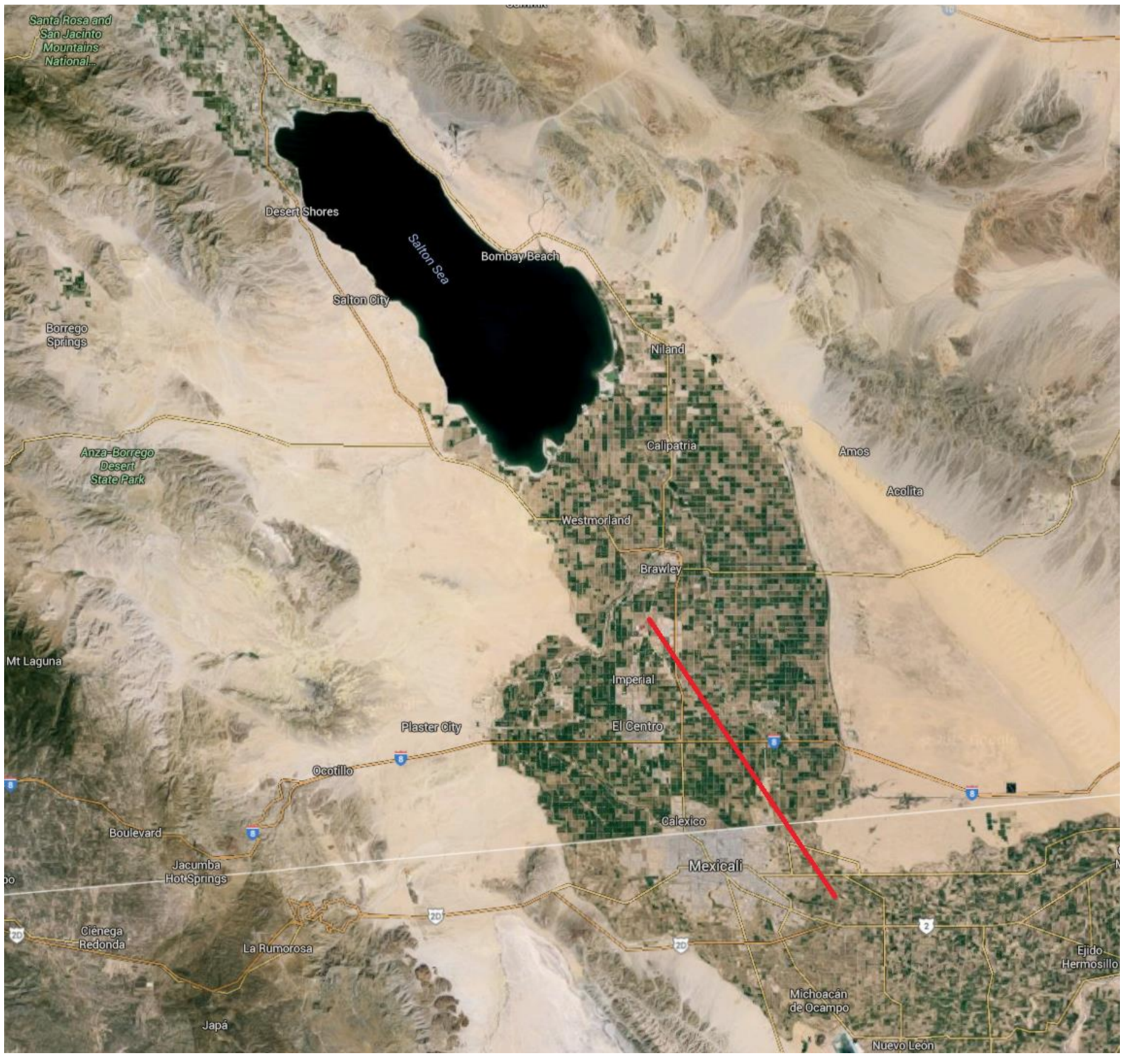


Fig. 1: The Imperial Fault. The red line marks the Imperial Fault. It straddles the California-Mexico border, south of the Salton Sea. The Imperial Fault is approximately 45 km long and 15 km deep, with a nearly vertical dip angle ranging from 81 to 90 degrees according to the SCEC Community Fault Model CFM-4.

66.1 Geometry

The model volume is a half-space. The fault is a vertical, planar, strike-slip fault. The fault reaches the Earth's surface. Rupture is allowed within a rectangular area measuring 30000 m along-strike and 15000 m down-dip.

There is a circular nucleation zone on the fault surface. The hypocenter is located 15 km from the left edge of the fault, at a depth of 7.5 km.

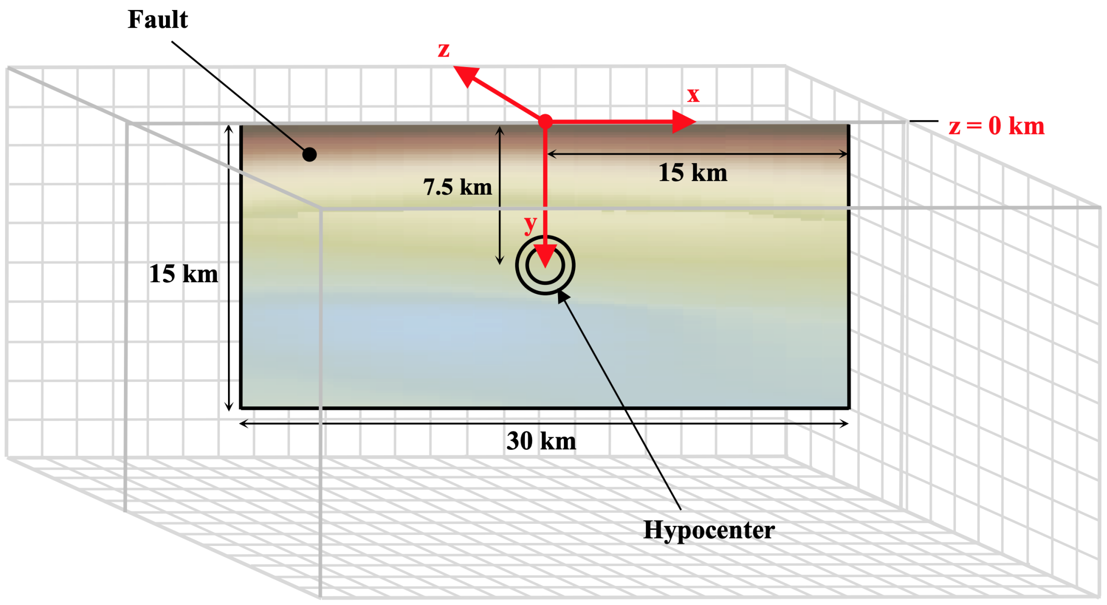


Fig. 2: TPV34 overview.

The geometry is generated with Gmsh. All the files that are needed for the simulation are provided at https://github.com/SeisSol/Examples/tree/master/tpv34.

66.2 Material

To obtain the velocity structure for TPV34, we need to install and run the SCEC Community Velocity Model software distribution CVM-H. For TPV34, we are using CVM-H version 15.1.0 and we use ASAGI to map the material properties variations onto the mesh. Detailed explanations are provided at https://github.com/SeisSol/Examples/blob/master/tpv34/generate_ASAGI_file.sh. We generate 2 netcdf files of different spatial resolution to map more finely the 3D material properties close to the fault. The domain of validity of each netcdf files read by ASAGI is defined by the yaml tpv34_material.yaml file:

```
[rho, mu, lambda]: !IdentityMap
  components:
    # apply a finer CVM-H data inside the refinement zone
    - !AxisAlignedCuboidalDomainFilter
      limits:
        x: [-25000.0, 25000.0]
        y: [-25000.0, 25000.0]
        z: [-15000.0, 0.0]
      components: !ASAGI
```

(continues on next page)

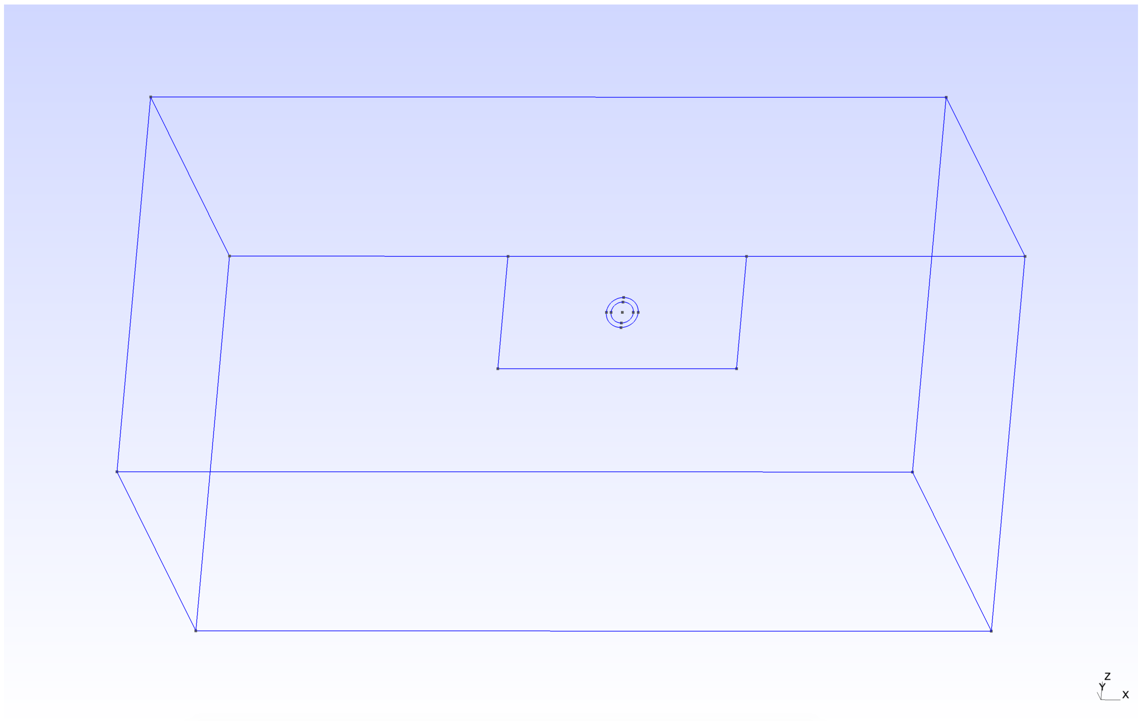


Fig. 3: Modeled TPV34 fault in Gmsh.

(continued from previous page)

```
        file: tpv34_rhomulambda-inner.nc
        parameters: [rho, mu, lambda]
        var: data
    # apply a coarser CVM-H data outside the refinement zone
    - !ASAGI
        file: tpv34_rhomulambda-outer.nc
        parameters: [rho, mu, lambda]
        var: data
```

66.3 Parameters

TPV34 uses a linear slip weakening law on the fault. The parameters are listed in Table below.

Parameter	inside the nucleation zone	Value	Unit
mu_s	static friction coefficient	0.58	
mu_d	dynamic friction coefficient	0.45	
d_c	critical distance	0.18	m

The cohesion is 1.02 MPa at the earth's surface. It is 0 MPa at depths greater than 2400 m, and is linearly tapered in the uppermost 2400 m. The spatial dependence of the cohesion is straightforwardly mapped using Easi.

```
[mu_d, mu_s, d_c]: !ConstantMap
  map:
```

(continues on next page)

(continued from previous page)

```
    mu_d: 0.45
    mu_s: 0.58
    d_c: 0.18
[cohesion]: !FunctionMap
  map:
    cohesion: |
      return -425.0*max(z+2400.0,0.0);
```

66.4 Initial stress

The initial shear stress on the fault is pure right-lateral.

The initial shear stress is τ_0 = (30 MPa)(μ_x)

The initial normal stress on the fault is σ_0 = (60 MPa)(μ_x).

In above formulas, we define $\mu_x = \mu/\mu_0$, where μ is shear modulus and μ_0 = 32.03812032 GPa.

```
[s_xx, s_yy, s_zz, s_xy, s_yz, s_xz]: !EvalModel
  parameters: [radius,mux]
  model: !Switch
    [mux]: !AffineMap
      matrix:
        x: [1.0,0.0,0.0]
        z: [0.0,0.0,1.0]
      translation:
        x: 0.0
        z: 0.0
      components: !ASAGI
        file: tpv34_mux-fault.nc
        parameters: [mux]
        var: data
    [radius]: !FunctionMap
      map:
        radius: |
          xHypo =     0.0;
          zHypo = -7500.0;
          return sqrt(((x+xHypo)*(x+xHypo))+((z-zHypo)*(z-zHypo)));
  components: !FunctionMap
    map:
      s_xx:     return -60000000.0*mux;
      s_yy:     return -60000000.0*mux;
      s_zz:     return 0.0;
      s_xy: |
        pi = 4.0 * atan (1.0);
        s_xy0 = 30000000.0*mux;
        s_xy1 = 0.0;
        if (radius<=1400.0) {
          s_xy1 = 4950000.0*mux;
        } else {
          if (radius<=2000.0) {
```

(continues on next page)

(continued from previous page)

```
          s_xy1 = 2475000.0*(1.0+cos(pi*(radius-1400.0)/600.0))*mux;
        }
      }
      return s_xy0 + s_xy1;
    s_yz:     return 0.0;
    s_xz:     return 0.0;
```

66.5 Results

All examples here can be visualized in Paraview. The *output* folder contains a series of files for fault dynamic rupture (hdf5 and .xdmf), wavefield (hdf5 and .xdmf), on-fault receiver (.dat) and off-fault receivers (.dat). The fault dynamic rupture and wavefield files can be loaded in Paraview. For example, open Paraview and then go through File > import > 'prefix'-fault.xdmf.

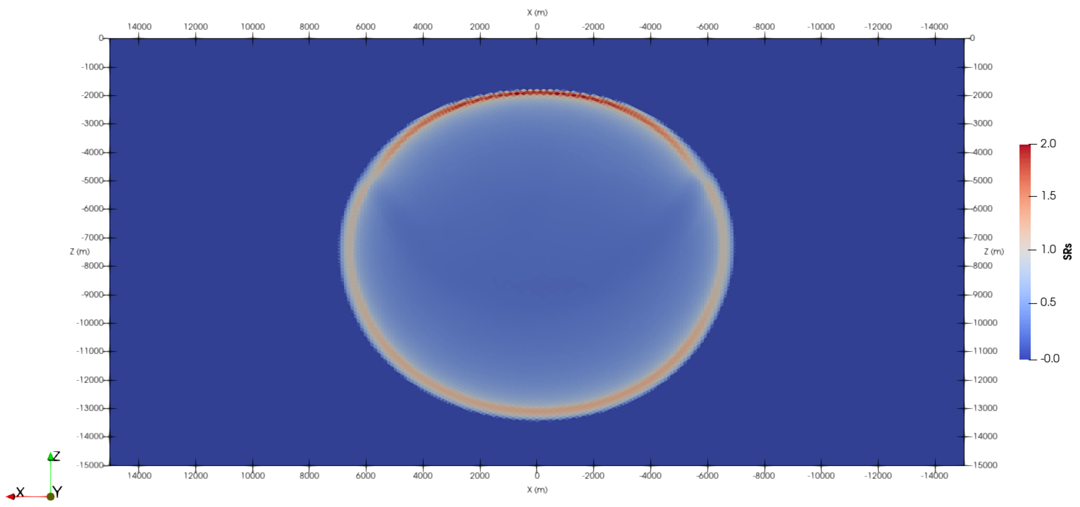


Fig. 4: Fault slip rate in the along-strike direction (SRs) at a rupture time of 3 seconds in TPV34, visualized using Paraview.

CHAPTER

SIXTYSEVEN

SCEC TPV104

In this example, we illustrate how to implement **rate-state friction law** using a slip law with strong rate weakening (RS-SL-SRW) and setup parameters in SeisSol.

TPV104 has a planar rectangular vertical strike-slip fault with the main rupture region of velocity-weakening friction, a zone on the fault surface with transitional friction surrounds the main fault rupture region, and the outer regions on the fault surface have velocity-strengthening friction (Figure [fig:tpv104]).

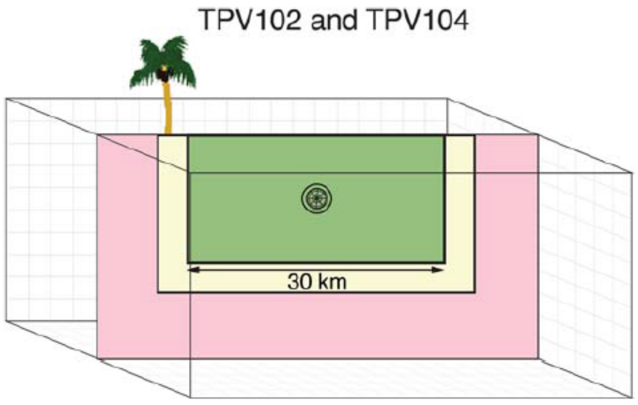


Fig. 1: Diagram of TPV104. The vertical planar fault that has velocity-weakening friction region (green), which is surrounded by velocity-strengthening region (pink). It encounters a finite width transitional region (yellow) where the friction smoothly changes from velocity weakening (green) on the inside to velocity strengthening (red) on the outside.

67.1 Geometry

TPV104 uses the same vertical fault as TPV5. We use the mesh file of TPV5 directly.

67.2 RSL parameters

TPV104 uses rate-state friction where shear stress follows:

$$\tau = f(V,\psi)\sigma$$

The friction coefficient is a function of slip rate V and state ψ:

$$f(V,\psi) = a * arcsinh[\frac{V}{2V_0}\exp(\frac{\psi}{a})]$$

The state variable evolves according to the equation:

$$\frac{d\psi}{dt} = -\frac{V}{L}[\psi - \psi_{ss}(V)]$$

$$and$$

$$\psi_{ss}(V) = a\ln[\frac{2V_0}{V}\sinh(\frac{f_{ss}(V)}{a})]$$

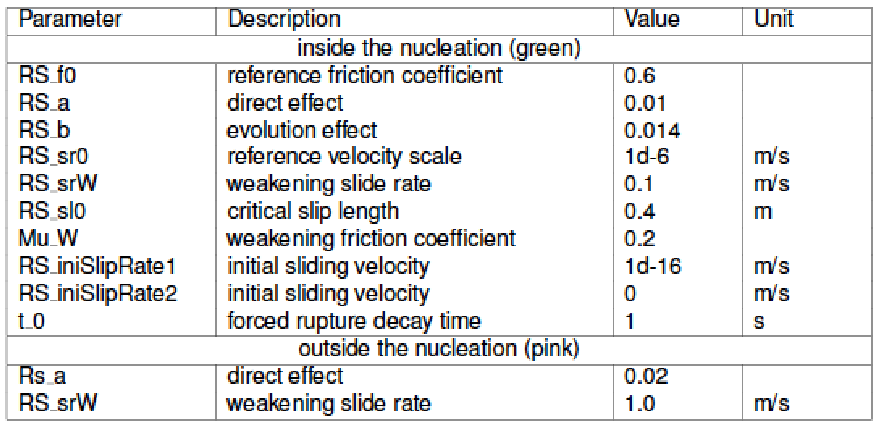
$f_{ss}(V)$ is the stead state friction coefficient that depends on V and the friction parameters $f_0, V_0, a, b, f_w and V_w$.

$$f_{ss}(V) = f_w + \frac{f_{LV}(V) - f_w}{[1 + (V/V_w)^8]^{1/8}}$$

with a low-velocity steady state friction coefficient:

$$f_{LV}(V) = f_0 + (b - a) * \ln(V/V_0)$$

In SeisSol input file, Rate-state friction law can be used by choosing *FL=103* in *parameter.par*. The friction parameters of RS-SL-SRW are shown in Table [table:tpv104rsl].



Parameter	Description	Value	Unit
	inside the nucleation (green)		
RS_f0	reference friction coefficient	0.6	
RS_a	direct effect	0.01	
RS_b	evolution effect	0.014	
RS_sr0	reference velocity scale	1d-6	m/s
RS_srW	weakening slide rate	0.1	m/s
RS_sl0	critical slip length	0.4	m
Mu_W	weakening friction coefficient	0.2	
RS_iniSlipRate1	initial sliding velocity	1d-16	m/s
RS_iniSlipRate2	initial sliding velocity	0	m/s
t_0	forced rupture decay time	1	s
	outside the nucleation (pink)		
Rs_a	direct effect	0.02	
RS_srW	weakening slide rate	1.0	m/s

Table 14: RS-SL-SRW parameters of TPV 104 .

Fig. 2: Table of rate-state friction used in tpv104.

To stop the rupture, the friction law changes from velocity-weakening in the rectangular interior region of the fault to velocity-strengthening sufficiently far outside this region. The transition occurs smoothly within a transition layer of width w = 3 km. Outside the transition layer, the fault is made velocity-strengthening by increasing a by $\triangle a = 0.01$ and V_w by $\triangle V_{w0} = 0.9$.

The input files of TPV104 can be found at https://github.com/SeisSol/Examples/tree/master/tpv104.

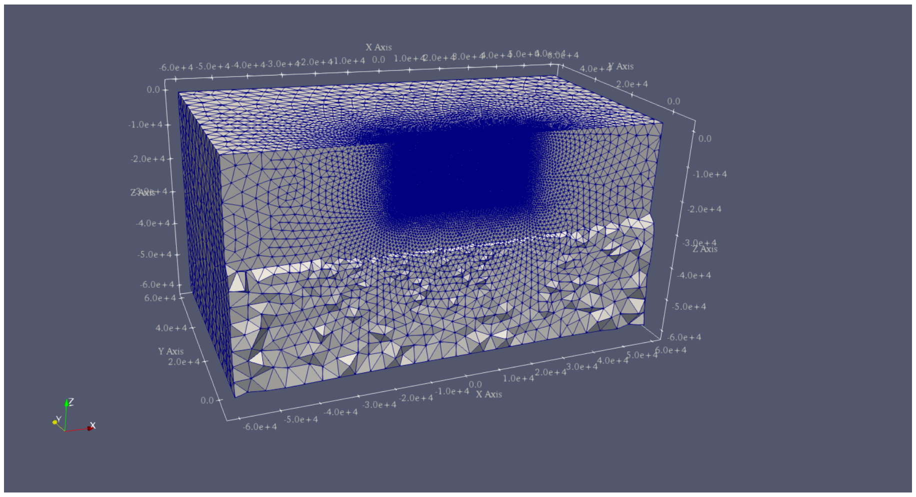


Fig. 3: Diagram shows the tetrahedral meshing of TPV 104 shown in the ParaView panel.

67.3 Results

The earthquake nucleates in the velocity-weakening zone spontaneously. The rupture propagates through the transition zone into the velocity-strengthening region, where it smoothly and spontaneously arrests. Nucleation is done by imposing additional shear stress in a circular patch surrounding the hypocenter.

Figure [fig:tpv104sr] shows the slip rate on the fault along the downdip direction at T=5s.

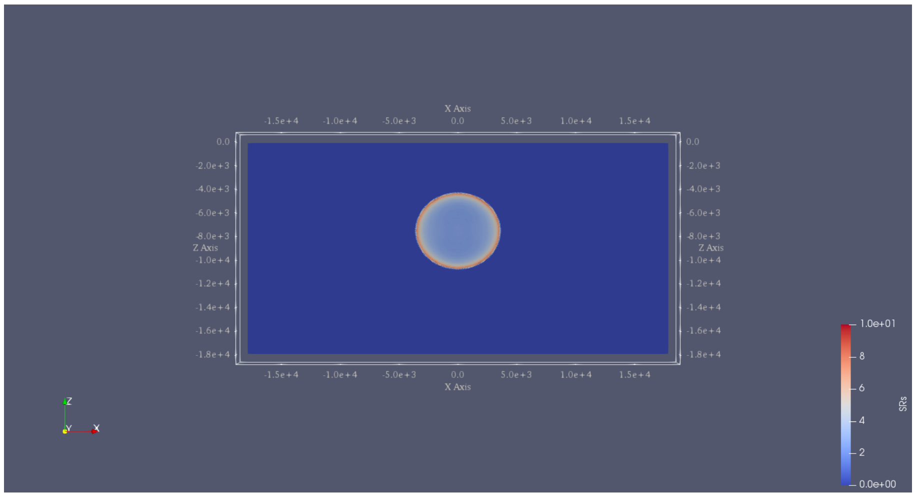


Fig. 4: Slip rate along-strike on the fault at 2 s of TPV104.

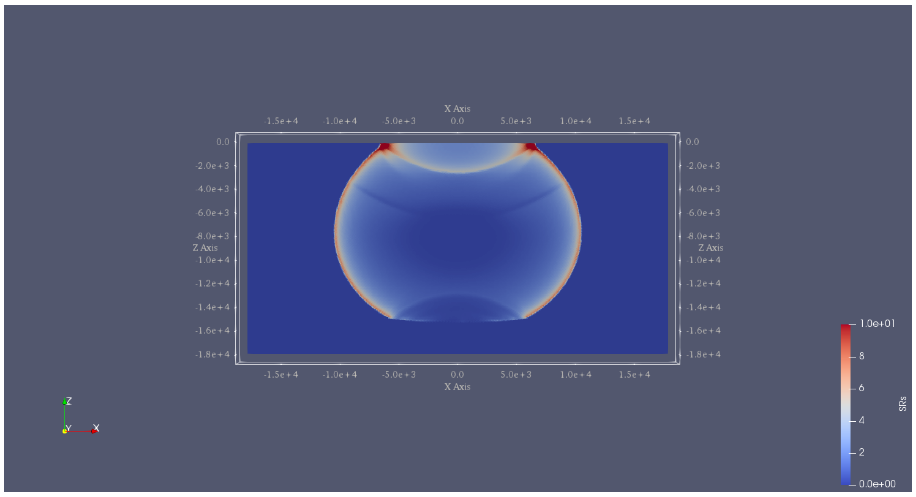


Fig. 5: Slip rate along-strike on the fault at 5 s of TPV 104.

CHAPTER

SIXTYEIGHT

POINT SOURCE

68.1 SISMOWINE WP2_LOH1

SISMOWINE is intended as a long-term interactive web interface for verifying numerical modeling methods in seismology. Numerical-method developers and numerical modelers may compare their solutions with other solutions. SISMOWINE is a continuation of the original SPICE Code Validation interface established within the 6th Framework Programme project.

LOH1 is used as an example here to illustrate the implementation of source point for earthquake nucleation in SeisSol. The details of the LOH1 model can also be found at the point source example.

The model uses Right-handed Cartesian, x positive North, y positive East, z positive downward, all coordinates in meters. The source is buried at 2000 m in a half-space Earth (Figure [fig:loh1]. The top layer is 1000 m thick and the bottom layer is 33000 m. The material parameters are listed in Table [table:loh1].

	Vp (m/s)	Vs(m/s)	density	Qp	Qs
layer	4000	2000	2600	Inf	Inf
half-space	6000	3464	2700	Inf	Inf

Table: Material properties in LOH1.

68.2 Geometry

The mesh is generated using Gmsh.

68.3 Point source input

The point source needs to be turned on in *parameter.par* file.

```
&SourceType
Type = 50
FileName='LOH1_source.dat'
/
```

The source input file can be generated using the script. Duration of the source is 4 seconds.

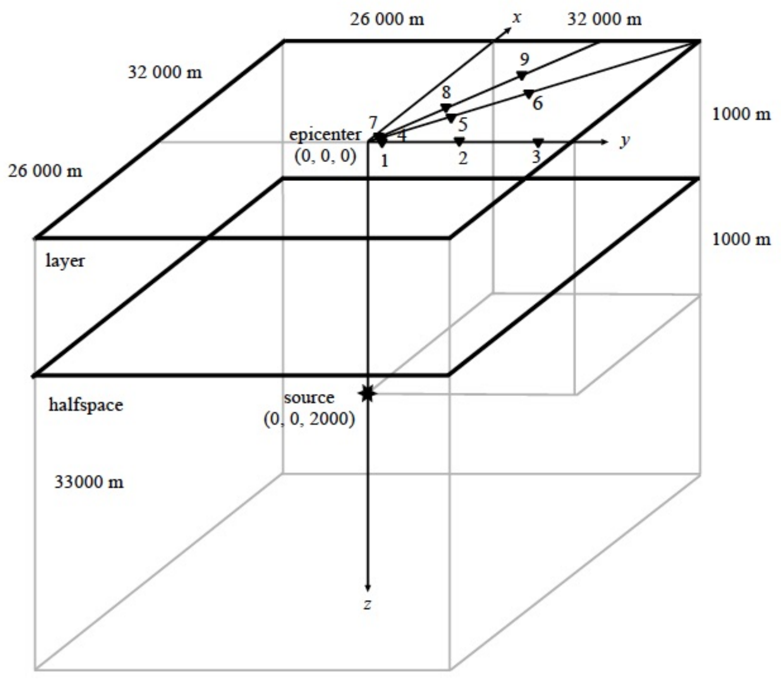


Fig. 1: Geometry of LOH1.

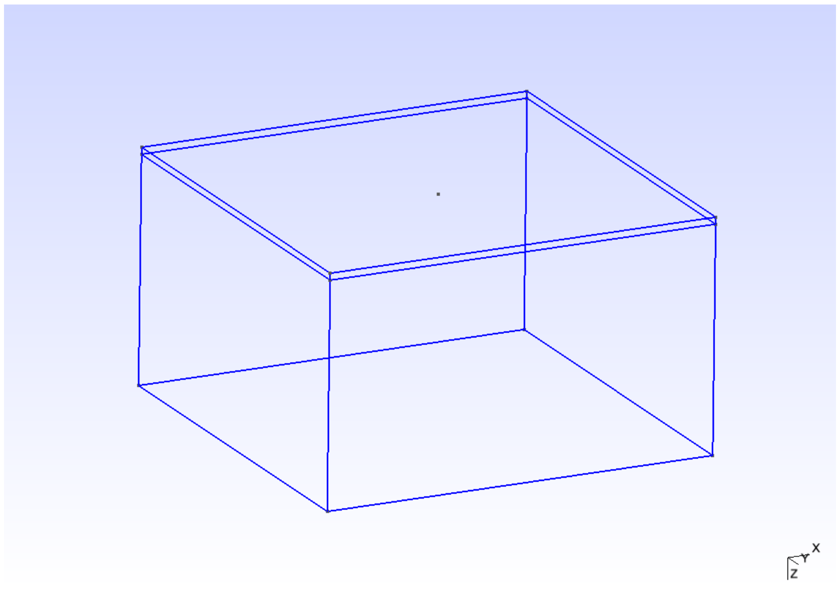


Fig. 2: Geometry of LOH1 model (Gmsh). A 1 km layer of low velocity (Vp=4000 m/s, vs=2000 m/s) is at the top of high velocity (vp=6000 m/s, vs=3464 m/s).

68.4 Results

The comparison with solution is shown in Figure [fig:compare_loh1].

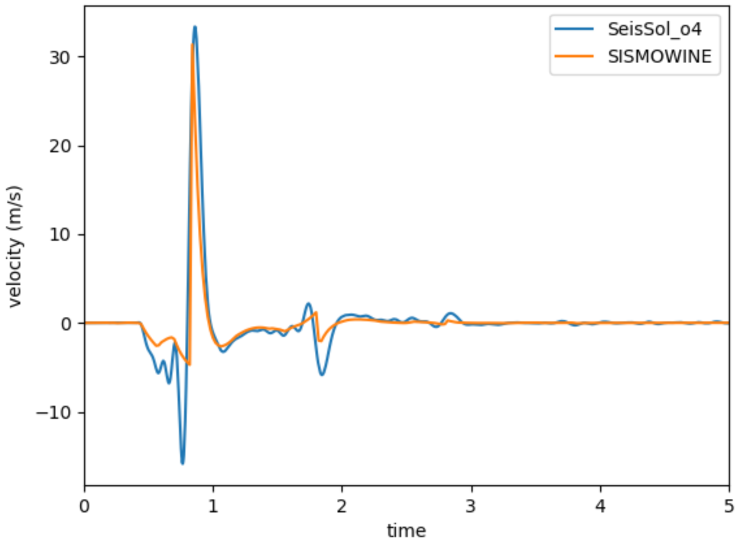


Fig. 3: Benchmark of x-component particle velocity at receiver point 1 (0.0, 693.0,0.1). Bule is 4-order SeisSol and orange is SISMOWINE result.

CHAPTER

SIXTYNINE

KINEMATIC SOURCE EXAMPLE - 1994 NORTHRIDGE EARTHQUAKE

We use a model of the 1994 Northridge earthquake to illustrate how to set up kinematic models in SeisSol.

This Mw6.7 earthquake occurred in the San Fernando Valley region of Los Angeles, California, USA on January 17. The estimated duration of this typical reverse-slip earthquake ranges between 10 and 20s. The ruptured fault strikes N122°E and dips at 40°.

69.1 Geometry

The structural model is built with Gmsh. It features a planar fault of 20 km $\times$ 25 km dipping 40 °, within a half-space region of 100 km $\times$ 100 km $\times$ 60 km

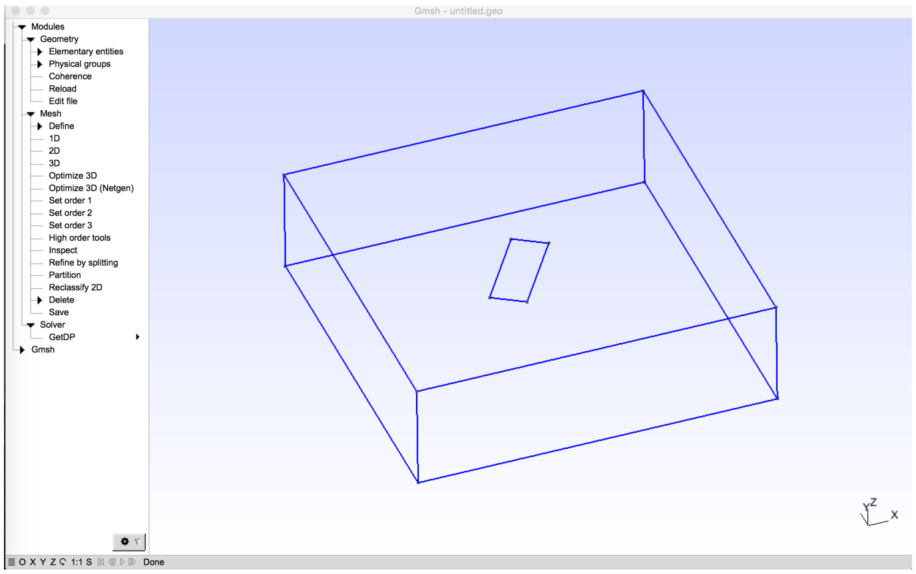


Fig. 1: Geometry assumed for the 1994 Northridge earthquake scenario.

69.2 Kinematic rupture Source

Kinematic models in the *standard rupture format* can be used directly in SeisSol, after converting them to nrf with the rconv tool (see next and *Multiple point-sources* for more details), with the following lines in the parameter.par file:

```
&SourceType
Type = 42
FileName='northridge.nrf'
/
```

A description of the standard rupture format can be found at https://strike.scec.org/scecpedia/Standard_Rupture_Format. Please note that some variables describing the kinematic model are given in non SI units.

The fault of the Northridge model is divided into 20 point-sources along strike times 25 point-sources along dip. This can be seen in the SRF file header, whose format is the following:

```
version (1.0)
PLANE 1
ELON ELAT NSTK NDIP LEN WID STK DIP DTOP SHYP DHYP
POINTS NP
```

With:

ELON top center longitude ELAT top center latitude NSTK number of point sources (subfaults) along strike NDIP number of point sources (subfaults) down-dip LEN segment length (km) WID segment width (km) STK segment strike DIP segment dip DTOP depth to top of fault segment (km) SHYP along strike location (from top center) of hypocenter for this segment (km) DHYP down-dip location (from top edge) of hypocenter for this segment (km) NP Number of points in fault plane

Each point source (subfault) is described in the file in a data block, with the following format:

```
LON LAT DEP STK DIP AREA TINIT DT
RAKE SLIP1 NT1 SLIP2 NT2 SLIP3 NT3
SR1[1] SR1[2] SR1[3] . . . SR1[NT1]
SR2[1] SR2[2] SR2[3] . . . SR2[NT3]
SR3[1] SR3[2] SR3[3] . . . SR3[NT3]
...
```

With:

LON: longitude of subfault center LAT: latitude of subfault center DEP: depth of subfault center (km) STK: strike DIP: dip AREA: area of subfault (cm^2) TINIT: initiation time when rupture reaches subfault center (sec) DT: time step in slip velocity function (sec) RAKE: direction of u1 axis (rake direction) SLIP1: total slip in u1 direction (cm) NT1: number of time points in slip rate function for u1 direction SLIP2: total slip in u2 direction (cm) NT2: number of time points in slip rate function for u2 direction SLIP3: total slip in u3 (surface normal) direction (cm) NT3: number of time points in slip rate function for u3 direction SR1[1],. . . ,SR1[NT1] slip rate at each time step for u1 direction (cm/sec) SR2[1],. . . ,SR2[NT2] slip rate at each time step for u2 direction (cm/sec) SR3[1],. . . ,SR3[NT3] slip rate at each time step for u3 direction (cm/sec)

69.2.1 Project geographic coordinates

The geographic coordinates of the source model are projected to Cartesian coordinates with the pre-processing tool rconv.

```
rconv -i northridge.srf -o northridge.nrf -m "+proj=tmerc +datum=WGS84 +k=0.9996 +lon_0=-
↪118.5150 +lat_0=34.3440" -x visualization.xdmf
```

69.3 Results

The fault is ruptured over 7s, leading to a MW6.7 earthquake. A snapshot of the vertical ground-surface velocity at 7s after rupture onset in shown below.

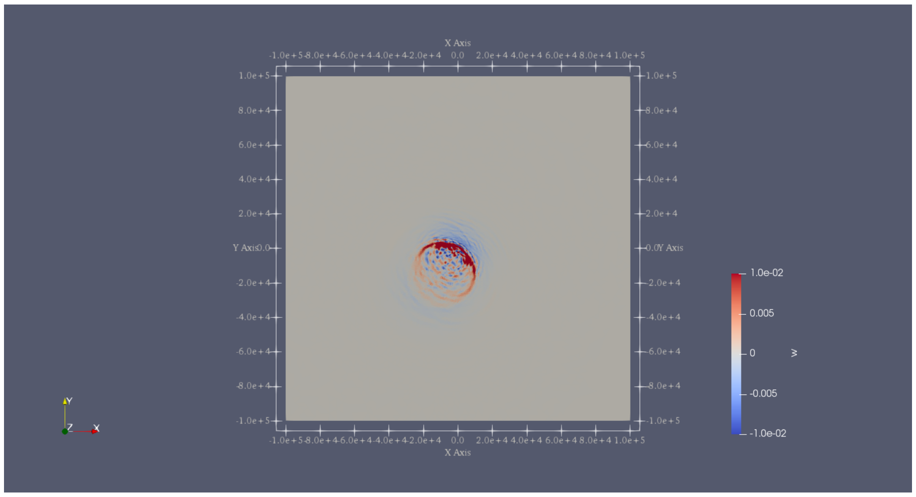


Fig. 2: vertical ground-surface velocity at 7 s after rupture onset. Note the unsmooth velocity field due to the coarse resolution of the kinematic model used.

CHAPTER

SEVENTY

COPYRIGHTS