# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1.1</td>
</tr>
<tr>
<td>Installation</td>
<td>1.2</td>
</tr>
<tr>
<td>EDGE</td>
<td>1.2.1</td>
</tr>
<tr>
<td>Assets</td>
<td>1.2.2</td>
</tr>
<tr>
<td>Setup</td>
<td>1.3</td>
</tr>
<tr>
<td>Config</td>
<td>1.3.1</td>
</tr>
<tr>
<td>Meshes</td>
<td>1.3.2</td>
</tr>
<tr>
<td>Resource Requirements</td>
<td>1.3.3</td>
</tr>
<tr>
<td>Kinematic Sources</td>
<td>1.3.4</td>
</tr>
<tr>
<td>Benchmarks</td>
<td>1.4</td>
</tr>
<tr>
<td>Wave Propagation</td>
<td>1.4.1</td>
</tr>
</tbody>
</table>
Introduction

Hey there, you have reached the user guide of the Extreme-scale Discontinuous Galerkin Environment (EDGE). EDGE uses the Discontinuous Galerkin (DG-) Finite Element Method (FEM) to solve hyperbolic partial differential equations. EDGE supports different equations and element types.

Currently, the applications focus is on seismic simulations through the elastic wave equations and on unstructured tetrahedral meshes. Here, EDGE targets seismic model setups with high geometric complexity and extreme-scale ensemble simulations. The entire software stack is uniquely tailored to "fused" simulations. Fused simulations allow for different model setups within one execution of the forward solver. For example, you could share the mesh and velocity model in the fused runs, but alter the kinematic source from run to run. This approach allows the code to exploit inter-simulation parallelism and reach significantly higher simulation throughput. Typically, the speedup is around 2-5 times, depending on the configuration. In short: Fusing simulations makes the code faster.

EDGE is distributed across different resources. First of all, there is this user guide, which you are currently reading. The main purpose of the user guide is to guide you through the installation of the code and the setup of simulations. Be warned, we just started writing and important chapters might be missing. If you run into trouble, use one of the issue trackers and we’ll have a look at it together. This user guide’s issue tracker is available at https://github.com/3343/edge_usr/issues, the issue tracker of the code at https://github.com/3343/edge/issues.

The next resource is the software itself. EDGE only provides source code, which is nice because you can always look at the guts of the software, but requires you to go through the compilation yourself. Then there are the assets, which are example setups, and supporting scripts and data. For example, this is the place where setups for benchmarks and unit tests are hosted. EDGE’s developer guide describes what's going on under the hood of the code. Finally, EDGE’s homepage http://dial3343.org provides up-to-date information on the Extreme-scale Discontinuous Galerkin Environment. To reach any of EDGE’s other resources, consider using the dispatcher.

BSD 3-Clause and CC0

Disclaimer: We do not provide legal advice. The provided information is incomplete and is not a substitute for legal advice.
EDGE’s core is licensed under the BSD 3-Clause license. This does not apply to dependencies and used libraries. Libraries are either directly located in the directory submodules or included through .gitmodules. For the master branch the directory is located here and .gitmodules located here. EDGE’s automated FOSSA-reports might be helpful for further details on the licenses:

![FOSSA dashboard](image)

The sources of this user guide, and the sources of EDGE's developer guide are CC0'd. An FAQ on CC0 is provided by Creative Commons.

EDGE’s assets follow a mixed approach. Software-components, e.g., configurations for EDGE’s core, or scripts, are BSD Clause-3. Other, non-software files, e.g., generated meshes, or source-input are CC0'd to a large extent. However, if, for example, used topography of a mesh is provided under the CC BY license, a mesh might not be CC0’d.
Installation

This chapter describes how the installation of EDGE and its dependencies.

Examples

EDGE’s entire installation process is continuously tested. If you get stuck with this user guide, you might be able to find additional installation hints in the respective configurations:

- Travis CI
- GoCD
- Singularity

General Remarks

EDGE links almost all libraries statically and expects corresponding library installations. The descriptions below are adjusted accordingly and only building static versions is described. This is one of the reasons why manual library installations are recommended, for example, not using `sudo apt-get install` on your local machine. `libnuma` and compiler-provided libraries, e.g., OpenMP, are the only exception.

All of the instructions below assume that you initiate the installation of each library in EDGE’s root-directory and will put the installed library into the directory `libs`. Make sure to navigate back to the root-directory before installing the next library.

All current Xeon Phi supercomputers (Knights Landing) have Xeon login nodes (typically Haswell), used for compilation. Many configure-scripts check if compiled binaries are working. Thus, checks, compiled for the Xeon Phi instruction set (AVX512), tend to fail on Haswell (AVX2). Knights Landing supports all previous instructions sets (incl. AVX2): We recommend compiling all libraries, except EDGE itself, using the Xeon's instruction set. For example on the Cray machine Cori:

1. Compile all libraries for Haswell by loading the Haswell module (default) through:

   ```bash
   module load craype-haswell
   ```

2. Compile EDGE itself using the Knights Landing module:

   ```bash
   module switch craype-haswell craype-mic-knl
   ```
Getting the Code

EDGE’s sources are hosted at https://github.com/3343/edge. The repository has two branches `master` and `develop`. `master` is the most recent stable version of EDGE. The minimum acceptance requirement for `master` is passing EDGE’s continuous delivery pipeline. Periodically EDGE also provides tags, which are simply snapshots of the master branch. Typically tags passed additional, manual testing. We recommend to use the most recent tag to get started with EDGE. `develop` is the bleeding edge version of EDGE. Changes in `develop` are intended to be merged into master. However, `develop` is for ongoing development and broken from time to time.

The procedure for obtaining the code as follows:

1. Clone the git-repository and navigate to the root-directory through:

   ```
   git clone https://github.com/3343/edge.git
cd edge
   ```

   This gives you the `master` branch. If this is what you want, jump over the next step.

2. Checkout the desired tag:

   ```
   git checkout YY.MM.VV
   ```

   **Remark**: `YY.MM.VV` is a place holder. You have to replace this with your actual tag. Available tags are shown at the GitHub-homepage or directly through `git tag`.

3. Initialize and update the submodules:

   ```
   git submodule init
   git submodule update
   ```

LIBXSMM

The single core backend of EDGE’s high-performance kernels is provided through the library LIBXSMM. LIBXSMM is optional, but highly recommended due to severe performance-limitations of the vanilla kernels.

- Install libxsmm by running:

  ```
  cd submodules/libxsmm; PREFIX=../../libs make BLAS=0 install
  ```

zlib
zlib is a requirement for the HDF5 library.

   ```bash
   wget http://zlib.net/zlib-1.2.11.tar.gz -O zlib.tar.gz
   ```

2. **Extract zlib to the directory** `zlib`:
   ```bash
   mkdir zlib; tar -xzf zlib.tar.gz -C zlib --strip-components=1
   ```

3. **Configure the installation and set** `libs` **as installation directory by running**:
   ```bash
   cd zlib; ./configure --static --prefix=$(pwd)/../libs
   ```

4. **Run** `make` **to build the library and** `make install` **to put it in the** `libs` **directory.

## HDF5

HDF5 is a requirement for the NetCDF library, which is used for kinematic source description. Further, we recommend building MOAB, EDGE’s interface to unstructured meshes, with HDF5-support.

   ```bash
   wget https://www.hdfgroup.org/package/gzip/?wpdmdl=4301 -O hdf5.tar.gz
   ```

2. **Extract HDF5 to the directory** `hdf5`:
   ```bash
   mkdir hdf5; tar -xzf hdf5.tar.gz -C hdf5 --strip-components=1
   ```

3. **Configure the installation and set** `libs` **as installation directory.**
   - **Sequential:**
     ```bash
     cd hdf5; ZLIBDIR=$(pwd)/../libs ./configure --enable-shared=no --with-zlib=${ZLIBDIR} --prefix=$(pwd)/../libs
     ```
   - **Parallel:**
     ```bash
     cd hdf5; ZLIBDIR=$(pwd)/../libs ./configure --enable-shared=no --enable-parallel --with-zlib=${ZLIBDIR} --prefix=$(pwd)/../libs
     ```

     Make sure to check that the configuration, printed at the very end, matches your expectations.

4. **Finally run** `make` **to build the library and** `make install` **to put it in the** `libs` **directory.**
NetCDF

NetCDF is a requirement for kinematic source descriptions, including single point sources. The library can also be used in the installation of MOAB (unstructured mesh interface) to allow reading of exodus-meshes.

1. Download the sources ("NetCDF-C Releases") from
   https://www.unidata.ucar.edu/downloads/netcdf/index.jsp:

   wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.4.1.1.tar.gz -O netcdf.tar.gz

2. Extract NetCDF to the directory netcdf:

   mkdir netcdf; tar -xzf netcdf.tar.gz -C netcdf --strip-components=1

3. Configure the installation and set libs as installation directory. Adjust the path to HDF5, if necessary. Two examples:

   ○ Sequential:

     cd netcdf; HDF5DIR=$(pwd)/../libs CPPFLAGS=-I${HDF5DIR}/include LDFLAGS=-L${HDF5DIR}/lib ./configure --enable-shared=no --disable-dap --prefix=$(pwd)/../libs

   ○ MPI-parallel with OpenMPI-wrapper:

     cd netcdf; CC=mpicc HDF5DIR=$(pwd)/../libs CPPFLAGS=-I${HDF5DIR}/include LDFLAGS=-L${HDF5DIR}/lib ./configure --enable-shared=no --disable-dap --prefix=$(pwd)/../libs

     Check that the configuration, printed at the very end matches your expectations.

4. Finally run make to build the library and make install to put NetCDF in the libs directory.

MOAB

If running on unstructured meshes, you need to provide an installation of MOAB. Since ASCII-only builds of MOAB are troublesome, building with HDF5-support also for small-scale runs is recommended.

1. Generate the configure-script:
cd submodules/moab; autoreconf -fi

2. Configure the installation, three examples:

- **Sequential example using GNU compilers:**
  ```
  F77=gfortran F90=gfortran FC=gfortran CC=gcc CXX=g++ ./configure --disable-debug --enable-optimize --enable-shared=no --enable-static=yes --disable-fortran --enable-tools --enable-all-static --with-hdf5=$(pwd)/../../libs --with-netcdf=$(pwd)/../../libs --with-pnetcdf=no --with-metis=yes --download-metis --prefix=$(pwd)/lib
  ```

- **MPI-parallel example using Intel compilers:**
  ```
  F77=mpiifort F90=mpiifort FC=mpiicc CC=mpiicpc ./configure --disable-debug --enable-optimize --enable-shared=no --enable-static=yes --disable-fortran --enable-tools --enable-all-static --with-hdf5=$(pwd)/../../libs --with-netcdf=$(pwd)/../../libs --with-pnetcdf=no --with-metis=yes --download-metis --prefix=$(pwd)/lib
  ```

- **MPI-parallel example using Intel compilers and Intel MPI:**
  ```
  F77=mpiifort F90=mpiifort FC=mpiicc CC=mpiicpc ./configure --disable-debug --enable-optimize --enable-shared=no --enable-static=yes --with-mpi --disable-fortran --disable-embcslam --enable-all-static --with-hdf5=$(pwd)/../../libs --with-netcdf=$(pwd)/../../libs --with-pnetcdf=no --with-metis=yes --download-metis --prefix=$(pwd)/lib
  ```

3. Now you can build MOAB with `make` and install it through `make install`.

**EDGE**

EDGE uses **SCons** as build tool. `scons --help` returns all of EDGE's build-options. Additionally, this user guide describes all build options as part of the chapter **Config**.

You can enable the libraries in EDGE either by passing their installation directory explicitly (recommended) or by setting the environment variables `CPLUS_INCLUDE_PATH` and `LIBRARY_PATH`. For example, let's assume that you installed LIBXSMM in the directory `$(pwd)/libs`. Than we could either enable LIBXSMM by passing `xsmm=$(pwd)/libs` to EDGE's SCons-script or by using `CPLUS_INCLUDE_PATH=$(pwd)/libs/include LIBRARY_PATH=$(pwd)/lib scons [...] xsmm=yes`. 
If something goes wrong with finding a library, EDGE will tell you so. For example, if we did not install LIBXSMM in `/tmp`, but tell EDGE so anyways, we get:

```
scons equations=elastic order=4 cfr=1 element_type=tet4 xsmm=/tmp

[...]
Checking for C++ static library libxsmmnoblas..no
Warning: Could not find libxsmm, continuing without.
```

Further information on what went wrong is logged in the file `config.log`, which, in this case, shows that the compiler could not find the LIBXSMM-header:

```
[...]
scons: Configure: Checking for C++ static library libxsmmnoblas..
.sconf_temp/confest_2.cpp <-
    #include <libxsmm.h>
    int main(int i_argc, char * **i_argv) { return 0; }
g++ -o .sconf_temp/confest_2.o -c -std=c++11 -Wall -Wextra -Wno-unknown-pragmas -Wno-unused-parameter -Werror -Wextra -Wshadow -Wundef -O2 -ftree-vectorize -DPP_N_CRUNS=1 -DPP_T_EQUATIONS_ELASTIC -DPP_T_ELEMENTS_TET4 -DPP_ORDER=4 -DPP_PRECISION=64 -I. -I src -I/tmp/include .sconf_temp/confest_2.cpp
.sconf_temp/confest_2.cpp:1:21: fatal error: libxsmm.h: No such file or directory
compilation terminated.
scons: Configure: no
```

**Stack Size**

In certain settings EDGE allocates substantial amounts of data on the stack. For high-order configurations, this memory is mostly occupied by thread-private global matrix structures. To circumvent errors due to limited stacks on Linux systems use `ulimit`. `ulimit -s` shows you the current maximum, `ulimit -s unlimited` allows unlimited sized stacks. Server machines typically operate unlimited. If running CentOS, you can obtain an unlimited stack as default by adding the following line to `/etc/security/limits.conf`:

```
* stack unlimited
```

**Singularity Bootstrap**

Singularity is software, which allows container-based execution of HPC-codes at close-to-native performance. EDGE provides a Debian-bootstrap for automated installation of different configurations:
<table>
<thead>
<tr>
<th>Build Option</th>
<th>Enabled Bootstrap Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>element_type</td>
<td>tet4 (4-node tetrahedral elements)</td>
</tr>
<tr>
<td>equations</td>
<td>elastic (elastic wave equations)</td>
</tr>
<tr>
<td>order</td>
<td>1 (FV), 2-6 (ADER-DG)</td>
</tr>
<tr>
<td>cfr</td>
<td>1 (non-fused), 8</td>
</tr>
<tr>
<td>arch</td>
<td>hsw (Haswell), knl (KnightsLanding)</td>
</tr>
<tr>
<td>xsmm</td>
<td>yes (LIBXSMM enabled except for FV)</td>
</tr>
<tr>
<td>zlib</td>
<td>yes</td>
</tr>
<tr>
<td>hdf5</td>
<td>yes</td>
</tr>
<tr>
<td>netcdf</td>
<td>yes (enables kinematic sources)</td>
</tr>
<tr>
<td>moab</td>
<td>yes (unstructured meshes), no (regular meshes)</td>
</tr>
<tr>
<td>parallel</td>
<td>omp (shared memory parallelization)</td>
</tr>
</tbody>
</table>

Once a container is generated, you can run it on systems with Singularity installed, without installing any further dependencies. Example systems with Singularity support are the XSEDE-resources Stampede and Comet. If you have root-access to a system with Singularity and debootstrap installed, you can generate a container containing EDGE and all its dependencies.

1. Clone a clean copy of EDGE, including all submodules.
2. Generate an archive of EDGE's root-directory:
   ```bash
tar -czf /tmp/edge.tar.gz edge
   ```
3. Create a Singularity container image of with a maximum size of 8 GiB (8192 MiB):
   ```bash
   sudo singularity create -s 8192 /tmp/edge.img
   ```
4. Import EDGE's source code:
   ```bash
   gunzip -c /tmp/edge.tar.gz | sudo singularity import /tmp/edge.img
   ```
5. Run the bootstrap to install the dependencies and EDGE-configurations:
   ```bash
   sudo singularity bootstrap /tmp/edge.img ./debian.def
   ```
6. The bootstrap might run for several hours, maybe grab a coffee.
Assets

EDGE provides assets, for example, binary input, in a separate repository. Due to limitations of git, when it comes to large files, we use Git Large File Storage (Git LFS) in this repository. To use EDGE's assets repository, install the Git LFS command-line client by following the linked instructions. Further background information and documentation on Git LFS is available at https://www.atlassian.com/git/tutorials/git-lfs.

Obtaining Data

- The best performance, when cloning EDGE's asset repository, is obtained by using `git lfs clone`. This will download all, possibly large, files stored in the Git LFS store.
- If you are only interested in certain files or directories, you can, by using `git lfs clone --exclude=*`, initialize the assets repository with non-LFS files and Git LFS pointers only. Now, to obtain only a certain file or directories, use `git lfs fetch` with the arguments `-I` and `-X`. For example, `git lfs fetch -I test/*` would download all files and directories in the directory `test`. After downloading the files from the remote Git LFS store, you can replace the Git LFS pointers in your local Git repository with the actual files through `git lfs checkout test/*`. 
Configuration

EDGE uses XML-files for build- and runtime-configurations. Using XML-input for building EDGE through `--xml=[...]` is optional. All build arguments might also be passed to SCons directly. In contrast, passing XML-input for the runtime configuration through `-x [...]` is mandatory. Except for the verbose flag `-v`, no other command-line arguments are accepted.

XML-tree

The following shows an overview of all XML-nodes in EDGE. Dependent on the build-type, only a subset of the nodes is active. For example, a given velocity model `<velocity_model> </velocity_model>` is only parsed for seismic problems. The comment `<!-- [ ] -->` indicates, that a node is allowed to appear multiple times in the XML-tree.

```xml
<edge>
  <build>
    <cfr/>
    <equations/>
    <element_type/>
    <order/>
    <mode/>
    <arch/>
    <precision/>
    <parallel/>
    <cov/>
    <tests/>
    <build_dir/>
    <xsd>/
    <zlib/>
    <hdf5/>
    <netcdf/>
    <moab/>
    <inst/>
  </build>
  <cfr>
    <mesh>
      <files>
        <in/>
      </files>
      <boundary>
        <free_surface/>
        <outflow/>
    </boundary>
  </mesh>
</edge>
```
<boundary>
</mesh>

<velocity_model>
  <domain>
    <half_space>
      <origin>
        <x/>
        <y/>
        <z/>
      </origin>
      <normal>
        <x/>
        <y/>
        <z/>
      </normal>
    </half_space>
  <!-- [... ] -->
</domain>
</velocity_model>

<setups>
  <kinematic_sources>
    <file/>
  <!-- [... ] -->
</kinematic_sources>

<end_time/>
</setups>

<output>
  <receivers>
    <path_to_dir/>
    <freq/>
  <receiver>
    <name/>
    <coords>
      <x/>
      <y/>
      <z/>
    </coords>
  </receiver>
  <!-- [... ] -->
</receivers>
</wave_field>
</type/>
The node `<edge>` is the root of both, the runtime- and the build-configuration.

The node `<build>` describes the build-configuration and is only used by SCons. EDGE also parses `<build>` at runtime, however the information is only logged and does not influence runtime behavior.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Allowed Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfr</td>
<td>1, 2, 4, 8, 12, 16</td>
<td>Number of concurrent/fused forward runs. 1, 4, 8, and 16 are typically used.</td>
</tr>
<tr>
<td>element_type</td>
<td>line, quad4r, tria3, hex8r, tet4</td>
<td>Element type used for spatial discretization. line: line elements, quad4r: 4-node, rectangular quadrilaterals, tria3: 3-node triangles, hex8r: 8-node, rectangular hexahedrons, tet4: 4-node tetrahedrons.</td>
</tr>
<tr>
<td>order</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
<td>Convergence rate of the solver. 1: Finite volume solver (P0 elements), 2-9: ADER-DG solver (P1-P8 elements).</td>
</tr>
<tr>
<td>mode</td>
<td>release, debug, release+san, debug+san</td>
<td>Compile mode. release: fastest option for use in production configurations, debug: debug flags and disabled optimizations, release+san (gnu and clang): same as release, but with enabled undefined behavior and address sanitizers, debug+san (gnu and clang): same as debug, but with enable undefined behavior and address sanitizers.</td>
</tr>
<tr>
<td>arch</td>
<td>host, snb, hsw, knl</td>
<td>Targeted architecture. host: uses the architecture of the machine compiling the code, snb:</td>
</tr>
</tbody>
</table>
### Config

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>knl</td>
<td>SandyBridge, hsw: Haswell, knl: KnightsLanding.</td>
<td></td>
</tr>
<tr>
<td>precision</td>
<td>32, 64</td>
<td>Floating point precision in bit. 32: single precision arithmetic, 64: double precision arithmetic.</td>
</tr>
<tr>
<td>parallel</td>
<td>none, omp, mpi, mpi+omp</td>
<td>Shared and distributed memory parallelization. none: disabled, omp: OpenMP only, mpi: MPI only, mpi+omp: hybrid parallelization with MPI and OpenMP.</td>
</tr>
<tr>
<td>cov</td>
<td>yes, no</td>
<td>Support for code coverage reports.</td>
</tr>
<tr>
<td>tests</td>
<td>yes, no</td>
<td>Unit tests. yes: builds unit tests in the separate binary tests.</td>
</tr>
<tr>
<td>build_dir</td>
<td>/path/to/build_dir</td>
<td>Path to the build-directory. Temporary files and the final executable(s) are stored in the build-directory.</td>
</tr>
<tr>
<td>xsmm</td>
<td>yes, no, path/to/xsmm</td>
<td>LIBXSMM support. Available only for ADER-DG and elastics.</td>
</tr>
<tr>
<td>zlib</td>
<td>yes, no, path/to/zlib</td>
<td>zlib support.</td>
</tr>
<tr>
<td>hdf5</td>
<td>yes, no, path/to/hdf5</td>
<td>hdf5 support.</td>
</tr>
<tr>
<td>netcdf</td>
<td>yes, no, path/to/netcdf</td>
<td>NetCDF support.</td>
</tr>
<tr>
<td>moab</td>
<td>yes, no, path/to/netcdf</td>
<td>MOAB support. If MOAB is enabled, EDGE is build with support for unstructured meshes. If disabled, EDGE is build with support for regular meshes.</td>
</tr>
<tr>
<td>inst</td>
<td>yes, no</td>
<td>EDGE’s high-level code instrumentation through the Score-P library.</td>
</tr>
</tbody>
</table>
Meshes

This section describes how to use meshes in EDGE.

Unstructured Meshes

Assume that you created a mesh `loh1_ext_small_16.msh` in gmsh with a total of 16 partitions and would like to use it in EDGE. While you could provide the gmsh-mesh directly to EDGE, this would result in considerable overhead when MOAB parses the mesh. Therefore, we recommend converting the mesh to the MOAB-native HDF5-format before running simulations.

mbconvert

You can use MOAB's `mbconvert` for this purpose, e.g.:

```
mbconvert -t loh1_ext_small_16.msh loh1_ext_small_16.h5m
```

mbpart

Similar, if your mesh `loh1_ext_small.msh` is not partitioned already, you can use the tool `mbpart` to do both steps:

```
mbpart 4 loh1_ext_small.msh loh1_ext_small_4.h5m -m ML_KWAY
```

MOAB's native format alone works for moderate sizes of your mesh. However, for large-scale setups, we have to consider how the mesh is parsed. EDGE's default is given by MOAB's `PARALLEL=BCAST_DELETE` option, which means that one rank reads the entire mesh from disk and broadcasts it to all other rank. All of the ranks then extract their required information and delete the remainder. This default mode has two severe drawbacks at scale: 1) The file-input is sequential, and 2) the entire mesh has to fit on a single rank.

Therefore, at scale, we use a different approach, which allows for parallel file-input and ensures that every rank only reads what is required. For partitioning, we use the option `--reorder` in `mbpart`, e.g.:

```
mbpart 4 loh1_ext_small.msh loh1_ext_small_4.h5m -m ML_KWAY --reorder
```
This will order the entities by their owning rank, thus it is sufficient for every rank to read the corresponding part of the file.

After reordering the entities, we have to make EDGE aware of this by overwriting the default behavior for mesh-input. This is accomplished by forwarding \texttt{READ\_PART} to MOAB through the \texttt{READ/} attribute in the mesh’s runtime configuration:

\begin{verbatim}
<edge>
  <build><!-- build options --></build>
  <cfr>
    <mesh>
      <options>
        <read>PARALLEL=READ\_PART;PARALLEL\_RESOLVE\_SHARED\_ENTS;PARTITION=PARALLEL\_PARTITION;</read>
      </options>
      <!-- additional mesh parameters -->
    </mesh>
    <!-- additional runtime options -->
  </cfr>
</edge>
\end{verbatim}

\textbf{Issues and hints:}

- MOAB 4.9.2 segfaults when using \texttt{mbpart} if compiled sequentially as described in section MOAB. Compile an MPI-parallel version for this, sequential execution is ok though.
- Writing partitioned meshes ( *h5m * ) to file systems with disabled locking fails for HDF5 1.10.1 , e.g. NERSC’s Cori:

  HDF5\text{-}DIAG: Error detected in HDF5 (1.10.1) MPI-process 0:
  [...]
  #003: H5FDsec2.c line 940 in H5FD\_sec2\_lock(): unable to lock file, errno = 524, error message = 'Unknown error 524'
    major: File accessibility
    minor: Bad file ID accessed
  [...]

  As a workaround, execute \texttt{mbpart} with the environment variable

  \texttt{HDF5\_USE\_FILE\_LOCKING} set to \texttt{FALSE} . Details are in HDF5 1.10.1’s release notes.

- EDGE looses bit-reproducibility for different rank-counts in the reordering step, because reordering leads to different mappings to the reference element.

\textbf{Entity Types}
EDGE parses the entity types (integer) specified in the mesh, e.g. you could use integer 105 to encode free-surface boundary conditions of faces.
Resource Requirements

This chapter summarizes EDGE's resource requirements for seismic simulations.

Memory

EDGE's memory requirements depend on the chosen convergence rate and the number of fused runs.

In this section we only consider required memory for 4-node tetrahedral elements, the elastic wave equations (9 quantities), double precision arithmetic (64-bit per value) and the following data structures in every element:

- Degrees Of Freedom (DOFs)
- Time Integrated DOFs
- Each of the eight Riemann solvers (sometimes called flux solvers)
- Each of the three Jacobians (sometimes called star matrices)

Therefore, the memory requirements for the mesh, kinematic sources, internal dynamic rupture boundaries, etc. are neglected.

Increasing the convergence rate, increases the number of modes per element. An increase in the number of fused runs, also increases the memory footprint of every element. However, data is shared for fused simulations, which reduces the relative memory footprint per run. For example, a second order simulation without fused runs (C1) requires 6,336 bytes in theory. By fusing eight runs, the per-element footprint increases to 10,368 bytes. This is equivalent to an increase by \( \frac{10,368}{6,336} \approx 1.64 \) in required memory. However, the memory footprint per element and forward run decreases to \( \frac{10,368}{8} = 1,296 \). The corresponding improvement per forward run is therefore: \( \frac{6,336}{1,296} \approx 4.9 \).

The following table gives the memory footprint per element in dependency of the order for a non-fused run (C1), four (C4), and eight fused runs (C8):
### Element Throughput

Analogue, to the discussed memory requirements, all considerations in this section are limited to 4-node tetrahedral elements, the elastic wave equations (9 quantities), and double precision arithmetic (64-bit per value). Further, the reported times per element and time step were measured simulating the LOH.1 benchmark with a total of 350,264 tetrahedral elements. Architecture was a single node of Cori Phase II (Intel Xeon Phi 7250 68-core processors at 1.4 GHz with Intel Turbo Boost enabled) and all data allocated in High Bandwidth Memory (HBM/MCDRAM).

The following table shows the required time per element and per time step in dependency of the order for non-fused configurations (C1) and eight fused runs (C8):

<table>
<thead>
<tr>
<th>Order</th>
<th>C1-Seconds</th>
<th>C8-Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6.06E-08</td>
<td>1.30E-07</td>
</tr>
<tr>
<td>3</td>
<td>1.14E-07</td>
<td>2.88E-07</td>
</tr>
<tr>
<td>4</td>
<td>2.08E-07</td>
<td>7.30E-07</td>
</tr>
<tr>
<td>5</td>
<td>4.41E-07</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>6.93E-07</td>
<td>-</td>
</tr>
</tbody>
</table>
Kinematic Sources

EDGE has two one types of kinematic source implementations:

1. Each of the fused runs has a completely independent, non-fused kinematic source description. This approach is the most flexible since the source description can be arbitrary. However, no data is shared in the storage of the point sources. Further, this leads to less efficient code for source updates.
2. Oops.. fused sources are not implemented for the time being. Please let us know if you would like to have them, and we'll bump up the priority of the implementation.

The number of fused kinematic sources equals the number of fused runs. In this case all possible data of the point sources will be shared in their storage. The two exceptions are the slip-rates per point source and the coefficients of the slip rates (encoding slip-direction, subfault area, etc.). Thus, the following parameters are shared:

- The number of point sources (or subfaults) the kinematic source descriptions has to be identical.
- The locations of the point sources have to be identical (up to a numerical tolerance).
- The number of slip rates samples has to be identical.
- The onset times and sampling intervals (→ duration together with the number of samples) of the point sources have to be identical.

Given your input, EDGE will automatically determine the appropriate implementation. Here, EDGE will fuse the source descriptions if the slip-parameters cover the fuse runs. If only one set of slip parameters per point source is given, non-fused sources will be used. In general, we recommend to use both, fused kinematic sources and fused runs, for higher performance.

Standard Rupture Format

EDGE implements the "Standard Rupture Format" (SRF) for kinematic sources. Here, we use an intermediate format, which converts converting the ASCII-SRF to an intermediate binary netCDF-format. You can use the tool rconv for the conversion from SRF to netCDF.
Benchmarks

Wave Propagation Benchmarks

Most of EDGE's wave propagation benchmarks are based on the benchmark descriptions at http://www.sismowine.org.

Currently EDGE provides setups for the following benchmarks:

- HSP1a: Homogeneous space, elastic, near receivers, single point source
- HHS1: Homogeneous halfspace, elastic, single point source
- LOH1: Layer over homogeneous halfspace, elastic, single point source
- LOH2: Layer over homogeneous halfspace, elastic, kinematic rupture
- Can4: Three thin layers in a halfspace, reaching the surface at a low angle, elastic, single point source

Input: Kinematic Sources

EDGE provides pre-generated kinematic source descriptions as assets. However, most are easily generated, since the moment rate time history of many wave propagation benchmarks is simply given by an analytic function. You might generate the sources through the script `kinematic_bench.py`.

HSP1a (regular) example:

```
python tools/processing/kinematic_bench.py --xml examples/bench/elastic/wp1/hsp1a/reg_src.xml
```

The template for the wave propagation benchmarks is located at `tools/processing/kinematic_bench.cdl`. Sources for the other wave propagation setups can be generated by calling the respective XML-configurations, provided as assets.

LOH.2

The Layer Over Halfspace benchmark (LOH.2) is a purely elastic setup, consisting of a 1000m thick layer, having different material parameters \( v_s = 2000 \text{ m/s}, v_p = 4000 \text{ m/s}, \rho = 2600 \text{ kg/m}^3 \) than the half-space below \( (v_s = 3464 \text{ m/s}, v_p = 6000 \text{ m/s}, \rho = 2700 \text{ kg/m}^3) \). Free-surface boundary conditions at the top of the layer are used, and outflow boundaries everywhere.
else. Source is a right-lateral strike-slip finite fault with a constant rupture velocity of $3000 \text{ m/s}$. The source-time function is similar everywhere, since only the onset time is variable. A detailed description of the LOH.2 problem is given in the "Final Report to Pacific Earthquake Engineering Research Center, Lifelines Program Task 1A01, Tests of 3D Elastodynamic Codes".

EDGE provides two setups solving the LOH.2 problem. This allows us to check consistency of the results, when using different features of the code.

- The first setup, our "reference" solution, simply uses 40 isolated configurations, each with a single point source. The 40 point sources are located at the epicenter in y-direction and their associated patches cover the z-dimension of the entire finite fault. Each of the square patches has a size of $100 \text{ m} \times 100 \text{ m}$. The effect of the remaining fault, with respect to the seismic receivers, is computed by simply shifting and adding the obtained solutions accordingly.

![Fig. 1: Illustration of the reference setup solving the LOH.2 benchmark. The 1000m thick layer is shown in gray, the finite fault in red, and the location of the hypocenter as a yellow star. 40 point sources (black and blue squares) with a finite fault size of 100m in y- and z-direction are used to derive EDGE's "reference" solution. The patches are centered at the epicenter in y-direction, and cover the entire fault in z-direction.](image)

- The second setup uses a single simulation with a single kinematic source description. The kinematic source consists of a total of $80 \times 40 = 3,200$ patches covering the entire
finite fault. Each of the square patches has a size of 100m × 100m.

Fig. 2: Illustration of the kinematic source description for the LOH.2 benchmark. The 1000m thick layer is shown in gray and the location of the hypocenter as a yellow star. 3200 point sources (black and blue squares) with a finite-fault size of 100m in y- and z-direction are used to derive EDGE’s kinematic solution. The patches cover the entire finite fault.
Fig. 3: Illustration showing the receiver-placement at the surface in gray, which is required to obtain the LOH.2 solution from isolated simulations. The fault is shown in red, the epicenter s1, indicating the location of the 40 point sources, is located at the yellow star. Three additional sources s2-s4 (extending in depth) are illustrated through yellow squares. We obtain the solution at receiver r1 with respect to the waves of s2-s4 by adding r2-r4 with respect to the waves generated by s1. Analogue, m1 is given by adding m2-m4.

By using identical meshes and convergence rates for both setup, we obtain almost identical numerical setups. Small differences, however, exist:

1. The seismic waves, originating from the point sources in the second setup, propagate through different elements before reaching the considered receivers. This effect is greater for unstructured meshes.
2. The 40 point sources of the reference setup do not exist in the kinematic setup: The centers of the 40 fault patches are located on the boundaries of the second setup’s patches.
3. We only executed EDGE five times for the provided solution of the first setup, and thus fused eight simulations per run. For the second setup, only one non-fused forward simulation was used. Since EDGE uses different kernels for the seismic wave propagation component in the two cases, errors, resulting from machine precision, are present.

Can4

The layers of the benchmark are shallow and reach the surface at a low dipping angle (wedge). This poses a modeling challenge to numerical software. We model the layers explicitly by using a tetrahedral mesh and aligning the faces to the material contrasts. Further, we avoid ill-shaped elements in the spatial discretization, by vertically cutting off the last dipping part of the layers. Here, the cut-off is chosen, such that the resulting height of the first layer is not smaller than the characteristic length of the elements in the wedge. Despite not explicitly meshing the remainder of the wedge, we still used appropriate material parameters for elements after the cut-off. This results in an increased scattering of the seismic waves, since the material interface now follows the unstructured mesh.
Fig. 4: Illustration showing the three layers of the Can4 benchmark. The red, dashed line shows the cut-off in EDGE’s assumed geometry, avoiding ill-shaped elements in the mesh. The result is a minimum thickness (blue) of the first layer, equal to the characteristic length of the elements in the wedge.

We mitigate the extreme ratio of the computational domain with respect to the depth of the layers, by using a problem-adapted mesh-refinement. Here, we use the highest refinement in the wedge of the three layers, which reduces the negative impact of the normalization through the cut-off. The remainder of the three layers and our region of interest, given by \([-5000 \text{ m}, 5000 \text{ m}] \times [-5000 \text{ m}, 5000 \text{ m}] \times [0, 5000 \text{ m}]\) use tetrahedral element sizes, matching the desired frequency content. The location of the point source is additionally refined by an attractor. This allows for sharper a discretization of discontinuities, and thus reduces errors, which might be introduced by insufficient source discretization through large element-sizes in the region of interest. In x-direction (south-north) and z-direction (depth), our region of interest is surrounded by a sponge layer with a coarse resolution.
Fig. 5: Illustration showing the problem-adapted mesh refinement of our Can4 setup. The highest resolution is used for the dipping parts of the layers (red), followed by decreasing resolution in the three layers (darker to lighter gray). Further, the point source (yellow star) is refined with a distance-dependent, linear gradient of decreasing refinement (blue sphere), reaching the coarsest resolution at the boundary of the sphere. The resolution in the region of interest (light gray) is chosen to match our desired frequency content, while the remainder (white) is coarse and acts as a sponge layer.