Mystery Application: SeisSol

Introduction

SeisSol (https://seissol.org) is a high-performance computational seismology software to simulate complex earthquake scenarios. It supports various rheologies (i.e. material models for the solid Earth: elastic, anisotropic-elastic, acoustic, viscoelastic, poroelastic) for the simulation of propagating seismic waves, plus various boundary conditions and dynamic rupture laws (to model earthquake sources). SeisSol uses a high-order discontinuous Galerkin discretisation with Arbitrary DERivative (ADER) local time stepping on unstructured adaptive tetrahedral meshes.

Software, Documentation and Setups

You can find detailed documentation here: https://seissol.readthedocs.io/.
The code is here: https://github.com/SeisSol/SeisSol/. Use the current master branch (969f3f0) for the competition!
The setups are provided in the location designated by the SCC committee. You can also download the setups here:
https://1drv.ms/u/s!AkJiaJLtCjAXmHa9m-z4zP_FYCoh?e=8qSmrP
or:
https://syncandshare.lrz.de/getlink/fi9R3HS8YtFEJ99D5YMAjJ/scc.zip
The files in all three locations are identical.

You should verify the sha512sum:

```
sha512sum scc.zip:
89503b35d82b16375488cd95afea333d9760eeb03c165206c59795633f4c29b8e5
0ac0e79ffa586d0da70945fcf0ad07d2c5d5a122922e26ff326d21806eab94
scc.zip
```

Tasks

There are three tasks with an increasing difficulty level. The first task is a proxy application, which runs the computational kernels but does not compute a realistic earthquake scenario. The other two tasks are (simplified) versions of realistic earthquake scenarios.

Please make sure to retain the output (to standard output) of all runs, as this will be used for scoring. For the realistic scenarios, also retain the computation results in the directories called “output”. A subset of these will be used to ensure that you computed correct results. You will only get points, if these results are reasonable.

Task #1: Proxy [20 Points]

SeisSol-proxy is a simple proxy application, which runs SeisSol's computational kernels and computes their performance in GFLOP/s. It omits MPI communication and realistic mesh information.
It can be run by executing
`SeisSol_proxy_Release_s[arch]_6_elastic [number of cells] [timesteps] all`
In this command, `[arch]` defines the architecture that you are using. For example, for Intel skylake, `[arch]` equals `skx`.
The program outputs performance statistics. For us, the following line is relevant:

```
GFLOPS (hardware) for seissol proxy : [number of GFLOP/s]
```

Task:

Maximize the number of GFLOP/s that SeisSol-proxy can achieve. You may adapt `[number of cells]`. This number controls how many elements are used in the inner loop of the performance proxy. This is the number over which the program is parallelized.
`[timesteps]` controls how often the measurement loop is repeated.
The pseudocode of SeisSol-proxy looks like this:

```
for timestep from 1 to timesteps
     parallel for cell from 1 to number of cells
          kernel(cell)
```

You must not change other parts of the call, especially not "`_6_`" which gives the polynomial convergence order and not "`all`" which controls which kernels are used.

Note, that SeisSol-proxy cannot use MPI and cannot use more than one accelerator. Hence, the score will be adjusted by "available hardware". For example, if you have 8 GPUs and reach a performance of 1000 GFLOP/s, your adjusted GFLOP/s will be 8000.

Scoring

Ten base points if proxy executed successfully + (adjusted GFLOP/s reached by your team) / (maximal adjusted GFLOP/s by best team) * 10

Task #2: Northridge [20 Points]

This setup is based on a training setup for SeisSol, available here:
https://github.com/SeisSol/Training/tree/main/northridge
The setup models the 1994, M6.7 Northridge earthquake with "kinematic" point sources. This setup uses many point sources (of type "double-couple"), which taken together comprise a kinematic finite earthquake model. The earthquake source kinematics are described as a set of points, each assigned with a moment tensor and slip rate function, often derived from solving an inverse problem using seismic and/or geodetic data.

Our model uses:

- kinematic rupture model from Hartzell et al., 1996 embedded in a 3D subsurface model
- source geometry: 20 x 25 km planar fault, 40° dipping
- we smooth and interpolate the kinematic model in space and time for SeisSol

However, in contrast to the training setup, the version used for SCC uses a computational mesh with flat topography. In addition, the mesh is much larger.

Task

Compute the simulation up to t=10s. This should take slightly more than one hour.

You are also allowed to change the mesh partitioning strategy by changing `LtsWeightTypeId` in parameters.par (see, SeisSol's Documentation)

Score

Score = Base points + Performance Points
Base points: 5 if simulation runs until t=5.0s. 10 if simulation run until t=10s
Performance points: (GFLOP/s reached by your team) / (GFLOP/s by best team) * 10

SeisSol does not need to be finished to get partial credit. However, make sure that the output files are correct. The following line in the log file is relevant:
Fri Nov 11 08:55:23, Info: **77.0175 TFLOPS** (rank 0: 947.429 GFLOPS, average over ranks: 962.719 GFLOPS)

Task #3: Palu, Sulawesi [20 Points]

This setup is again based on a training setup, available here:
https://github.com/SeisSol/Training/tree/main/sulawesi
It is a simplified version of the published setup of Ulrich et al. (2019):
https://link.springer.com/article/10.1007/s00024-019-02290-5.
The setup used for the SCC is identical to the training setup, however, the size of the computational mesh is again much larger. The training setup uses an element size of 800m close to the fault, the published setup uses 200m close to the fault. Our SCC setup uses 180m close to the fault. Hence, this setup has a resolution that is fine enough for an applied geophysics publication.

The earthquake is driven by a dynamic rupture internal boundary condition, which simulates the earthquake source using a physics-based method. This is relatively expensive (and full GPU support is in active development). The SCC setup therefore uses a different friction law ("linear slip weakening") than what was used in the published paper.

Task

Run the Palu scenario for the 14.x million element mesh up to a simulation time of t=10.0s.

You can also use `MeshFile = 'meshing/training'` (in parametersLSW.par) for trial runs using a very small mesh; however, submissions must be done with the large setup.

You are not allowed to change parameters of the simulation with the exception of introducing and fine tuning the settings `vertexWeightElement` and `vertexWeightDynamicRupture` as described in https://seissol.readthedocs.io/en/latest/parameter-file.html and in https://seissol.readthedocs.io/en/latest/memory-requirements.html.
This sets the weights for partitioning the mesh with parMETIS, such that each element has a base cost of `vertexWeightElement`. If an element has a dynamic rupture boundary condition, it gets an additional weight of `vertexWeightDynamicRupture`. The default value for both variables is 100. Fine-Tuning may or may not bring a small (!) increase in performance.
You are also allowed to change the mesh partitioning strategy by changing `LtsWeightTypeId`, as described in https://seissol.readthedocs.io/en/latest/memory-requirements.html.

This setup should take roughly 3-4 hours for the full simulation time.

Score

Score = Base points + Performance Points
Base points: 5 if simulation runs until t=5.0s. 10 if simulation run until t=10s
Performance points: (GFLOP/s reached by your team) / (GFLOP/s by best team) * 10

SeisSol does not need to be finished to get partial credit. However, make sure that the output files are correct. The following line in the log file is relevant:
Fri Nov 11 08:55:23, Info: **77.0175 TFLOPS** (rank 0: 947.429 GFLOPS, average over ranks: 962.719 GFLOPS)

Compilation

Required compile options/libraries

There are detailed instructions on how to install SeisSol in our wiki. Here are some relevant pages:

- https://seissol.readthedocs.io/en/latest/installing-dependencies.html for installing dependencies
- https://seissol.readthedocs.io/en/latest/compiling-seissol.html for compiling SeisSol
- https://seissol.readthedocs.io/en/latest/gpus.html for installing on GPUs

You can use spack to install SeisSol's dependencies (https://seissol.readthedocs.io/en/latest/installing-dependencies.html#spack-installation), however this does not include HIPSycl. You need easi+lua and Asagi for some scenarios, see table below.

We also have some examples for the installation on certain clusters, which can provide some additional information. Especially for GPU based systems, the description of the Marconi cluster can be relevant https://seissol.readthedocs.io/en/latest/marconi.html as it includes the necessary MPI configuration.

This is an overview of the easi/SeisSol features required for the scenarios:

Task	Target	MPI	HDF5/METIS	NETCDF	ASAGI	Lua
SeisSol-proxy	seissol-proxy	✗	✗	✗	✗	✗
Northridge	seissol-bin	✓	✓	✓	✗	✗
Palu	seissol-bin	✓	✓	✓	✓	✓

Hint: For GPUs you need LLVM, which requires some time. Even unpacking the tar archive takes a long time - you can do it in the background.
For AMD GPUs, you probably need to recompile HIP with clang version 16.0.0 (from https://github.com/RadeonOpenCompute/llvm-project.git 2c769dcf9f307f8576382c4ad680b13b2465c870). This is due to performance regressions in recent official builds of HIP. You need to use the following CMake flags when configuring SeisSol: `-DDEVICE_BACKEND=hip -DDEVICE_ARCH=gfx90a`. Device arch needs to be chosen according to your GPU.

For all setups and architectures: Use the CMake flags `-DORDER=6 -DPRECISION=single -DCOMMTRHEAD=ON` when configuring SeisSol. Other CMake flags can be changed.

You can compile SeisSol-proxy with the command `make SeisSol-proxy` and the main SeisSol application with `make SeisSol-bin`.

Possible traps

- Make sure metis uses `IDXTYPEWIDTH=64`.
- Linking of hdf5 etc needs to be consistent for asagi/easi/SeisSol otherwise this may lead to very strange errors.
- Sometimes, you need to compile yaml-cpp/easi (and possibly other libraries) with position independent code. You can do this with: `-DCMAKE_POSITION_INDEPENDENT_CODE:BOOL=ON`

Things to tune when running

Pinning

You need to pin OpenMP threads
Use one fewer OpenMP threads than cores for running. (When using hyperthreading/SMT you need to leave one physical core free.) This is because SeisSol manually pins the communication/output threads to all free cores.

You can use numactl. Beware that the pinning needs to be GPU aware.

Hints for running on GPUs are discussed here:
https://seissol.readthedocs.io/en/latest/gpus.html

Environmental variables

https://seissol.readthedocs.io/en/latest/environment-variables.html lists some common environmental variables and their choice. You can of course set the variables freely.

For the latest AMD GPUs (MI250), the following setting is necessary/leads to enable managed memory: `export HSA_XNACK=1`. The old ones do not support managed memory at all and thus this type of memory falls back to zero-copy memory which significantly degradates GPU performance.

`DEVICE_STACK_MEM_SIZE` changes the amount of GPU stack memory SeisSol allocates for intermediate computations. You may need to change it, as documented here:
https://seissol.readthedocs.io/en/latest/gpus.html?highlight=DEVICE_STACK_MEM_SIZE#execution

Miscellaneous

For AMD, the linux kernel may need to support HMM (Heterogeneous Memory Management). You can check this by running:

```
$ cat /boot/config-<kernel version>-generic

...

CONFIG_ARCH_HAS_HMM=y

CONFIG_HMM=y

CONFIG_HMM_MIRROR=y
```