SORD User Guide

Contents

Summary

The Support Operator Rupture Dynamics (SORD) code simulates spontaneous rupture within a 3D isotropic viscoelastic solid. Wave motions are computed on a logically rectangular hexahedral mesh, using the generalized finite difference method of support operators. Stiffness and viscous hourglass corrections are employed to suppress suppress zero-energy grid oscillation modes. The fault surface is modeled by coupled double nodes, where the strength of the coupling is determined by a linear slip-weakening friction law. External boundaries may be reflective or absorbing, where absorbing boundaries are handled using the method of perfectly matched layers (PML). SORD is written in Fortran 95 and parallelized for multi-processor execution using Message Passing Interface (MPI).

Author

Geoffrey Ely
gely@usc.edu
University of Southern California
http://earth.usc.edu/~gely

Install, test and run

SORD has been tested on the following system configurations:

Operating systems: Linux, IBM AIX, Apple OSX, Sun Solaris
Fortran 95 compilers: GNU, IBM, Intel, Sun, Portland Group
MPI implementations: ANL MPICH, IBM, Myricom MPICH-GM

On many systems SORD will be ready to go as is, but may benefit from fine tuning. Machines with specialized parallel environments will take extra configuration. Here are the steps for installation and testing SORD:

1. SORD is distributed as a gzipped tar archive. Unpack the archive and enter the 'sord' directory:

```
tar zxvf sord.tgz
cd sord/
```

2. Configuration, compilation and execution are all handled by a wrapper script called `sord` The script takes an input file as its argument, and takes a number of optional flags (documented within the script). First, test serial compilation by running the script:

```
./sord -s in/test/tpv3test.m
```

The `-s` flag ensures that SORD operates in serial mode. Porting to a new machine may require modifying compiler options by editing the script `sh/config`.

3. Next, test serial execution:

```
./sord -si
```

The `-i` flag tells SORD to run interactively. When no input file specified, the most recent one will be used. You should receive status messages to the screen, ending with 'Finished!' when the run is complete. Each time SORD is executed, a new directory is set up for the particular run, starting with `run/01`. The directory contains the executable and scripts to run the code, and will contain all of the generated output and metadata.

4. The parallel version requires MPI. If MPI is not already installed, MPICH2 is recommended. It is available from Argonne National Lab:

http://www.mcs.anl.gov/research/projects/mpich2

Example install commands for MPICH2:

```
./configure -prefix=$HOME/local --with-device=ch3:shm --enable-f90 --
disable-cxx
make install
```

5. Test parallel compilation:

```
./sord -pd
```

The `-p` flag specifies parallel mode, and the `-d` flag deletes previous run derctories that we don't need anymore.

6. Launching MPI jobs varies among parallel environments, such as MPD, PBS or LoadLeveler. If SORD is already properly configured for your environment, you will be able to enter the run directory and start the job interactively with the `run` script, or submit the job to the batch system with the `que` script:

```
cd out/01
./run   # (or ./que)
```

If you are porting to a new environment, you may have to create a custom configuration script. See `sh/datastar` and `sh/teragrid` for examples. The configuration script should be named exactly as is returned from the `hostname` command and placed in the `sh/` directory, and it will be automatically executed on that system. If the hostname is not consistent between logins, you may be to devise another means to identify the system. Look at the `sh/config` script for examples.

Input and Output

Input is specified as key value pairs in a text file. As a convenience, the input file may be valid Python, MATLAB, or Bash shell code. This practice facilitates using the input parameters in other pre- or post-processing tasks. Large data sets such as the material model are stored separately in floating-point

binary files. Annotated example input files are located in the `in/` directory. The input file `defaults.m`, that is read before any other input, contains a short description of each SORD parameter.

Output is saved in binary formant. Associated metadata files `meta.py` and 'meta.m' contain a summary of parameters for the run, and a structured description of the binary output. MATLAB utilities for manipulating and visualizing SORD output are included in the `m/` directory. Using the `read4d` function for accessing SORD binary output correctly accounts for byte order when moving data between big-endian and little-endian architectures. Separate Fortran utilities are also included in the `util/` directory for converting the byte order of binary files, and for converting to and from ASCII text format.

Statistic, such as peak acceleration and peak velocity, are computed periodically during each run and stored in the directory `stats/`. Additionally, internal code timings, for benchmarking performance, are collected and saved to the `prof/` directory. Inspecting these files during a run is a good way to check that it is proceeding correctly. The raw binary files can be examined with the standard UNIX command `od -f` or with the included Fortran utilities. The `stats` utility computes the minimum, maximum, and mean values for binary files.