Specfem3D Tutorial for 2013 CIG-QUEST-IRIS Workshop

Carl, Qinya, Elliott, Emanuele

Last compiled: January 5, 2014

Overview

Document Conventions

1. In this document, commands you should run in your shell or code blocks you should edit are indicated by an framed grey box:

```
This is a code block.
```

2. Things you should type at the prompt are prefixed by a dollar sign $, and the output from commands is not. Comments are prefixed by a hash #.

```
$ echo "A␣command␣you␣can␣run" | sed 's/can/have/g' # This is a comment.
A command you have run
```

3. Lines that have been wrapped are indicated by the backslash \ at the end of the line and the symbol ↳ at the beginning of the continued line. Most shells will understand the trailing backslash to indicate a continued command, so you should be able to copy all wrapped lines directly without any issues. Line breaks do not occur in the middle of words, so remember to include a space between each wrapped line if you are typing it out yourself.

```
This is a very long line that will wrap because it goes on and on and on and also could be \
considered a run-on sentence, too.
```

Log-in instructions

1. get account info from ARSC
2. ssh into ARSC (is X11 okay from mac? -X?)
3. Explain text editing (vi, gedit, emacs)

Step-by-step instructions, example 1: homogeneous halfspace synthetic seismograms (better call it forward simulation?)

The purpose of this example is to step through all the key steps of Specfem3D: get the code, configure the code for your cluster, generate a mesh using (Geo)Cubit, partition the mesh, generate databases, run the solver, and check the output. The example is trivial, but it is important to remember that

1. Check what modules are loaded by default:

```
$ module list
Currently Loaded Modulefiles:
  1) pgi/13.4                   3) netcdf/4.3.0.pgi-13.4
  2) openmpi-pgi-13.4/1.4.3    4) PrgEnv-pgi/13.4
```

If you do not see these modules listed, load the PGI compiler environment with `module load PrgEnv-pgi`.

2. Change to the `$CENTER` directory (as `$HOME` has a limited quota of 4 GB), and check out two copies[1] of the SPECFEM3D master branch from the code repository on GitHub (https://github.com/geodynamics/specfem3d) We will need to (a) pick a random revision (not recommended) or (b) tag a revision (more useful).

```
$ cd $CENTER
$ git clone --recursive https://github.com/geodynamics/specfem3d.git SPECFEM3D_default
$ git clone --recursive https://github.com/geodynamics/specfem3d.git SPECFEM3D_socal
```

(This may take a few minutes.)

3. check version info Commit ID will need to be fixed.

```
$ cd SPECFEM3D_default
$ git status
# On branch master
nothing to commit, working directory clean
$ git show
commit 846f6e400927b639d94bfba3a76ca132e07ef03e
Merge: 4ba0960 fa99f77
Author: Matthieu Lefebvre <ml15@princeton.edu>
Date:   Mon Dec 2 11:07:05 2013 -0800

    Merge pull request #74 from carltape/master

    added a clarification comment for running xcombine_vol_data after generating the \
    databases.
```

The *base directory* for the code is `SPECFEM3D_default`, which we will call `SPECFEM3D` in the instructions.

4. Take a quick look at the user manual, mainly as a reminder to come back to it for details.

```
$ evince doc/USER_MANUAL/manual_SPECFEM3D_Cartesian.pdf &
```

5. Check that all software is available (or that modules are loaded):

```
$ which mpirun # OpenMPI
$ which cubit  # Cubit
$ which python # Python
```

6. Configure package, using the Portland compiler, in our case:

```
$ ./configure FC=pgf90 MPIFC=mpif90
```

[1] There's no need for two copies, but they will make it a bit easier to keep track of the different exampled.

If successful, this command will generate several `Makefiles` in the `SPECFEM3D` main directory, subdirectories of `src/`, as well as `shared/constants.h` and `shared/precision.h`, among others.

7. Adapt the run scripts for your cluster and your example.

   - Copy run scripts from `utils/Cluster/` into `SPECFEM3D/`, e.g.,

```
$ cd utils/Cluster/pbs/
$ cp go_decomposer_pbs.bash go_generate_databases_pbs.bash go_solver_pbs.bash ../../../
$ cd ../../../
```

   For ARSC, set all bash scripts to run in the `standard` queue (the queue will depend on your specific cluster) by editing the line (do we need to warn people to also delete one # from the line of ##PBS -q debug?)

```
#PBS -q standard
```

   (The default simulation time and number of cores is okay for this example.)

8. The rest of the instructions follow from the homogeneous halfspace example here

```
SPECFEM3D/examples/homogeneous_halfspace/README
```

   Copy the input files from examples directory into `SPECFEM3D/DATA/`

```
$ cd examples/homogeneous_halfspace/
$ cp DATA/* ../../../DATA/
```

   Note that only three input files are required: for the source (`CMTSOLUTION`), for the stations (`STATIONS`), and for the simulation parameters (`Par_file`).

9. Create mesh:

   From the directory `SPECFEM3D/examples/homogeneous_halfspace`, open the CUBIT GUI, `claro`:

```
$ claro
```

   maybe warn people that the X11 forwarding can be slow? (Close the "Tip of the Day".) To ensure that the path is local and the needed python modules and scripts are accessible, File ⟩ Set Directory, then click Choose (Choose what? examples/homogenous_halfspace/?), which is equivalent to typing in the CUBIT command window `cd 'examples/homogeneous_halfspace/'` (note the single quotes are required).

   Run the meshing script: from the Menu bar, select Tools ⟩ Play Journal File, set Files of Type to All Files, then select `block_mesh.py`. cubit2specfem3D.py needs to be fixed!

   If everything goes fine, this script creates the ten mesh files in subdirectory `MESH/`:

```
$ ls MESH
MESH/absorbing_surface_file_bottom
MESH/absorbing_surface_file_xmax
MESH/absorbing_surface_file_xmin
MESH/absorbing_surface_file_ymax
MESH/absorbing_surface_file_ymin
MESH/free_surface_file
```

```
MESH/materials_file
MESH/mesh_file
MESH/nodes_coords_file
MESH/nummaterial_velocity_file
```

You should be able to translate, rotate, and zoom on the mesh using a three-button mouse. (This can be emulated if you set X11 preferences, then (on a Mac) hold the ctrl, alt, or cmd buttons while clicking and moving the mouse.)

The CUBIT graphics window should show a mesh similar to that of Figure 1.

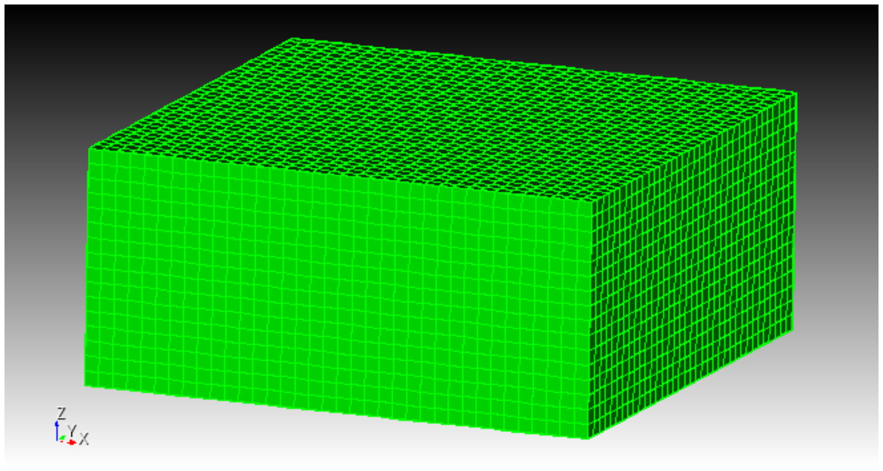

Figure 1: CUBIT mesh for example 1

10. decompose mesh files:

Compile the decomposer in directory `SPECFEM3D/`:

```
$ make xdecompose_mesh
```

This will compile the partitioner SCOTCH.

Then run the decomposer:

```
$ qsub go_decomposer_pbs.bash
```

You can check the status of the job with the command

```
$ qstat -u $USER
```

which should display something like:

```
scyld:
                                                                   Req'd  Req'd      Elap
Job ID       Username Queue       Jobname       SessID NDS TSK Memory Time     S Time
------------ -------- ----------- ------------- ------ --- --- ------ -------- - ----
690722.scyld username standard_16 go_decomposer     --   1   1     -- 01:00:00 Q   --
```

The job should take about 20 seconds. As the job progresses, the `S` column will change from `Q(ueued)` to `R(unning)` to `C(ompleted)`. The process of the parallel job is summarized in `OUTPUT_FILES/jobid.scyld.o`. If successful, it creates the four mesh partitions `proc000***_Database` in the directory `OUTPUT_FILES/DATABASES_MPI/`. The output file `OUTPUT_FILES/*.o` contains information on the partitioning.

11. Generate databases:

    Compile `xgenerate_databases` in the directory `SPECFEM3D/`:

```
$ make xgenerate_databases
```

Submit the job:

```
$ qsub go_generate_databases_pbs.bash
```

The job should take about a minute. It creates binary mesh files, e.g.

```
proc000***_{rho,vp,vs,x,y,z,ibool,external_mesh}.bin
```

in the directory `OUTPUT_FILES/DATABASES_MPI/`.

It is a good idea to look at the partitions of the mesh files. Load some vtk files (e.g., vs) into Paraview:

```
$ cd OUTPUT_FILES/DATABASES_MPI/
$ module load paraview
$ paraview
```

Then File ≫ Open, and select all four `proc000***_vs.vtk` files. Be sure to select them individually or Paraview will treat them as timesteps. When you are done, be sure to unload the Paraview module, since here it was compiled with the `GNU` compiler, which conflicts with the `portland` compiler we are using.

```
$ module unload paraview PrgEnv-gnu
$ cd ../../
```

12. Run simulation:

    Compile the solver, `xspecfem3D` (from `SPECFEM3D/`):

```
$ make xspecfem3D
```

Submit script to run solver:

```
$ qsub go_solver_pbs.bash
```

The simulation runs on 4 cores and should take about 30 minutes. You can track the progress with the timestamp files generated in `OUTPUT_FILES/` (type `ls -ltr` to see the most recent files). When the job is complete, you should have 3 sets (semd,semv,sema) of 12 (`ls -1 *semd | wc`) seismogram files in the directory `OUTPUT_FILES`, as well as 51 `timestamp******` files.

13. Compare your computed seismograms with the reference seismograms.

    A quick visual comparison can be done from `SPECFEM3D/` using Grace:

```
$ module load grace
$ xmgrace examples/homogeneous_halfspace/REF_SEIS/*Z.semd &
$ xmgrace OUTPUT_FILES/*Z.semd &
```

Step-by-step instructions, example 2: homogeneous halfspace sensitivity kernel

Following is an example of generating sensitivity kernels for a travel-time adjoint source at a single receiver station. Please first go through the procedures of a regular forward parallel simulation based on the instructions given in the last section. Change your directory to the top-level `SPECFEM3D/` before proceeding further.

1. First, set up the `DATA/Par_file` for a forward simulation with wavefields saved

```
$ utils/change_simulation_type.pl -F
```

   and check that the setting in `DATA/Par_file` has been modified to

```
     # forward or adjoint simulation
     SIMULATION_TYPE                   = 1
     SAVE_FORWARD                      = .true.
```

2. Provided that you have already compiled `xspecfem3D` and various database files are already in `OUTPUT_FILES/DATABASES_MPI`, run the forward simulation by submitting the job script:

```
$ qsub go_solver_pbs.bash
```

   After its completion in about 30 minutes, you should now have the calculated synthetic seismograms `X?0.BX[XYZ].semd[dva]` in the directory `OUTPUT_FILES/`:

3. Now you can create the cross-correlation traveltime adjoint sources from the forward synthetic seismograms for station `X20`. First compile the utility `xcreate_adjsrc_traveltime`:

```
$ cd utils/adjoint_sources/traveltime
$ make
$ cd -
```

   and then run in the Specfem3D directory

```
$ utils/adjoint_sources/traveltime/xcreate_adjsrc_traveltime 10. 25. 3 \
      OUTPUT_FILES/X20.DB.BX*.semd
```

   which generates traveltime adjoint sources using the Z-component synthetics signal arriving between 10 – 25 seconds. You should change the names of the adjoint source files and move them to the designated directory,

```
$ mkdir SEM/
$ mv OUTPUT_FILES/X20*.adj SEM/
$ cd SEM/
$ rename .semd.adj .adj *.adj
```

(on some linux machines, the last line may be `rename 's/.semd.adj/.adj/' *.adj`), which should create the adjoint source files `SEM/X20.DB.BX[XYZ].adj`. You can plot them with the Grace tool, and you will probably notice that only the Z-component source is non-zero, as designed.

4. Setup corresponding adjoint stations file `STATIONS_ADJOINT` for those receivers with adjoint source files already setup in `SEM/` (e.g., `X20`):

```
$ cp examples/homogeneous_halfspace/DATA/STATIONS_ADJOINT DATA/
```

5. Now you can finally run the kernel simulation that interacts adjoint wavefields with the restored forward wavefields. In the `SPECFEM3D/` directory, first change the simulation type in `DATA/Par_file` again:

```
$ utils/change_simulation_type.pl -b
```

which sets in `DATA/Par_file`:

```
     # forward or adjoint simulation
     SIMULATION_TYPE                 = 3
     SAVE_FORWARD                    = .false.
```

Because the kernel simulation is in essence two simulations at the same time, you will need to increase the requested runtime to $\sim$ 1.5 hours to ensure that there is enough leeway to complete the run. In `go_solver_pbs.bash`,

```
#PBS -l nodes=1:ppn=4,walltime=1:30:00
```

Then run the kernel simulation by submitting the job script:

```
$ qsub go_solver_pbs.bash
```

If successful after $\sim$ 1.5 hours runtime, it will create travel-time kernel files as defined in Eq (17-20) of Tromp et al. (2005) as

```
OUTPUT_FILES/DATABASES_MPI/proc000***_{alpha,beta,kappa,mu,rho,rhop}_kernel.bin
```

6. The kernel files can be visualized with Paraview. First compile the program that combines selected slices of kernels and produces vtk files for Paraview. In the `SPECFEM3D/` directory,

```
$ make xcombine_vol_data
$ cd bin/
$ ./xcombine_vol_data 0 3 alpha_kernel ../OUTPUT_FILES/DATABASES_MPI/ ../OUTPUT_FILES/ 1
```

where id 0 to 3 indicates that we use all 4 partitions of the processors. By default, it produces the vtk file `OUTPUT_FILES/alpha_kernel.vtk`, which can be imported into Paraview.

7. You can then visualize the vtk kernel file in Paraview. After opening up the Paraview GUI, on the top menu, click File › Open, navigate to the `OUTPUT_FILES/` directory, choose the `alpha_kernel.vtk` file and click Apply in the Properties tab. Go to the Display tab, and in the Style section, choose Representation › Surface, and then in the Color section, choose Color by › alpha_kernel. Now the model block can be rotated by the left button of the mouse, translated by the middle button and zoomed in/out by the right button (the binding of the keys may be different on Mac).

   Similarly we can visualize the source/receiver positions. Click File › Open, choose `OUTPUT_FILES/sr.vtk` and click Apply. While it is highlighted in the Pipeline Browser tab, select menu Filters › Alphabetical › Glyph, and click Apply. In the Properties tab, select Glyph Type › Sphere, and hit Apply again. The little sphere representing the receiver will appear on the free surface. To see both the source and receiver, click on the eye-like icon in front of `alpha_kernel.vtk` in the Pipeline Browser to hide the view of `alpha_kernel`. You will find the source sphere at the center of the block.

   Now reactivate `alpha_kernel.vtk`, and while it is highlighted, choose menu Filters › Alphabetical › Clip, and in the Properties tab, type in the Normal of a y-plane $[0, 1, 0]$ (the default origin is fine), click Apply to accept, and on the Display tab, Representation › Surface and Color by › alpha_kernel. Then click on Edit color map button to work with the Color Scale Editor pop-up menu. In this window, Choose Preset › Red to blue HSV › OK (In this dialog, you may also import your own favourite color scales). Then unclick the Automatically Rescale to Fit Data Range checkbox, click Rescale Range to choose a minimum of $-1\mathrm{e}{-12}$ and maximum $1\mathrm{e}{-12}$, and click Rescale. You may then click the Color Legend tab and choose to Show Color Legend. Close the popup window, and now the P sensitivity kernel linking the source and receiver can be nicely seen. The visualization results can be saved by menu File › Save screenshot, which produces a PNG file that is similar to Figure 2.

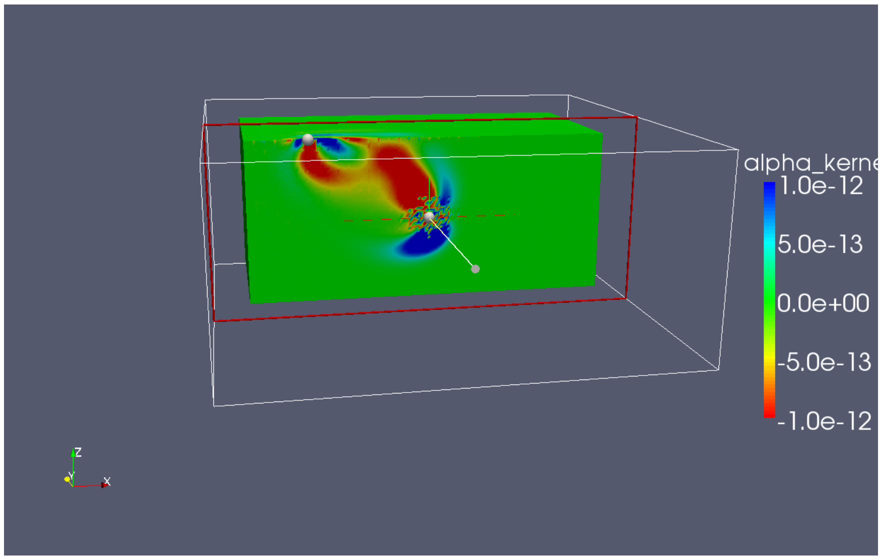

Figure 2: P travel-time sensitivity kernel for example 2

Note: The rescaling of the colour bar is necessary because the magnitude of the kernel is very large at the source and receiver locations due to the unrealistic assumption of point sources. To be able to distinguish the kernel itself, it is sometimes necessary to reduce the maximum values of the colour scale by four orders of magnitude. Smoothing these kernels to get rid of the spuriously large values at the sources and receivers is prudent before using them in tomographic inversions.

Again be sure to run `module unload paraview PrgEnv-gnu` before proceeding to other examples.

note that there are some new scripts added (by Daniel?)

Step-by-step instructions, example 3: southern California

This example is at a scale that will likely not run on a single laptop or desktop computer. In other words, the required memory must be distributed over a number of different machines in order to run the simulation. Here we use 96 cores of the cluster.

1. Copy bash scripts into the base directory:

```
$ cd SPECFEM3D_socal
$ cp ../SPECFEM3D_default/*bash .
```

2. configure

```
$ ./configure FC=pgf90 MPIFC=mpif90
```

3. compile all

```
$ make all
```

4. Link the mesh directory as an example

```
$ cd examples
$ ln -s /import/c/d/ERTHQUAK/GEOCUBIT_MESH/socal_med400km .
```

5. Modify `go_decomposer_pbs.bash` to point to the new directory:

```
MESHDIR=examples/socal_med400km/MESH/
```

6. Link the tomography file and copy input files

```
$ cd DATA
$ ln -s /import/c/d/ERTHQUAK/MODEL/cvm119_1000_1000_0250_741_549.xyz tomography_model.xyz
$ cp ../examples/socal_med400km/in_data_files/* .
```

7. modify `go_generate_databases_pbs.bash` and `go_solver_pbs.bash` to have the proper time limits and number of cores

```
#PBS -l nodes=6:ppn=16,walltime=1:00:00
#PBS -q standard
```

8. Follow the same steps as in Example 1: decompose, generate databases, solver. Here the programs have all been compiled, so only submitting the run scripts is needed. (But wait for each one to finish, and check the output before proceeding to the next step.)

   - The decomposer takes about 3 minutes.

- The generate databases takes about 15 minutes.
- The solver takes about 25 minutes.

The simulation runs on 96 cores and should take about 25 minutes. You can track the progress with the timestamp files generated in `OUTPUT_FILES/` (type `ls -ltr` to see the most recent files). When the job is complete, you should have 3 sets (semd,semv,sema) of 1107 (`ls -1 *semd | wc`) seismogram files in the directory `OUTPUT_FILES`, as well as 4 `timestamp******` files.

As expected, the seismograms contain numerical noise, since the source half duration in `CMTSOLUTION` was set to 0 s. This allows for maximal flexibility in post-processing, since the seismograms can be convolved with any source time function (see manual). However, it is important to know what the minimum resolving period of a particular mesh and model is, since these periods provide a guide for how to filter the seismograms in post-processing.

9. Now make a change to one of the input files in `SPECFEM3D/DATA/`, either `Par_file`, `CMTSOLUTION`, or `STATIONS`. Rename the `OUTPUT_FILES` directory if you do not want to over-write your previous output. Then submit the new job. (Note that no recompilation is needed.)

Step-by-step instructions, example 4: GPU

In this example, we show how SPECFEM3D can be used on a GPU cluster. In general, there are only two *required* steps to use GPU computing:

1. Enable CUDA during configuration:

```
$ ./configure --enable-cuda MPI_INC=-I$MPI_DIR/include
```

2. Enable CUDA at runtime in `DATA/Par_file`:

```
GPU_MODE    =   .true.
```

In practice, you may need to do several other steps depending on how your cluster has been set up. When running on your own cluster, you should consult their documentation in case there are additional steps that must be taken. The following example shows the extra steps that are required on the FISH cluster at ARSC.

1. *TODO*: start with login to fish
2. Check out version XXX again:

```
$ git clone --recursive https://github.com/geodynamics/specfem3d.git SPECFEM3D_GPU
$ cd SPECFEM3D_GPU
```

3. Load the CUDA toolkit:

```
$ module load cudatoolkit
```

4. The compilers on fish work a little differently than on pacman. There are wrappers that automatically run the real compilers and apply MPI or CUDA options once the modules are loaded.

   When compiling for CUDA, the CUDA compiler (which is separate from the C or Fortran compilers) needs to know where the MPI headers are installed. This information must be set using the `MPI_INC` variable when running `./configure`. On fish, loading the MPI module sets the `MPICH_DIR` environment variable, which is used in the instructions below.

```
$ ./configure MPIFC=ftn CC=cc FC=ftn --with-cuda MPI_INC=-I$MPICH_DIR/include
```

5. *TODO*: Copy example config stuff...

6. The GPU mode only affects the solver (`xspecfem3D`), but it is a good idea to remember to enable it now. Edit `DATA/Par_file` and set this option:

```
GPU_MODE    =   .true.
```

7. Now, you can compile things as usual:

```
$ make xdecompose_mesh
$ make xgenerate_databases
$ make xspecfem3D
```

8. The scripts to run jobs on fish are similar to the ones on pacman. Fish uses a slightly different method to run MPI programs compared to pacman. Instead of using `mpirun -np <#procs>`, you use `aprun -n <#procs>`, and to use GPUs, you must use the `gpu` queue.

```
$ cd utils/Cluster/pbs/
$ cp go_decomposer_pbs.bash go_generate_databases_pbs.bash go_solver_pbs.bash ../../../
$ cd ../../../
$ vim *.bash
  # Change 'mpirun -np' to 'aprun -n'
  # Change '#PBS -q standard' to '#PBS -q gpu' (only necessary for go_solver_pbs.bash)
```

   TODO: Add note about one process per node for GPU runs. This probably depends on which example we're using.

9. Now, you can submit your jobs as usual:

```
$ qsub go_decomposer_pbs.bash
# Wait for it to finish

$ qsub go_generate_databases_pbs.bash
# Wait for it to finish

$ qsub go_solver_pbs.bash
# Wait for it to finish
```

10. Depending on how many GPUs you use, you may see up to 20 or 30 times faster execution. If everything ran correctly on the GPU, you should see some additional files in `OUTPUT_FILES`:

```
$ ls OUTPUT_FILES/gpu*.txt
gpu_device_info_proc_000000.txt
gpu_device_mem_usage_proc_000000.txt
gpu_device_info_proc_000001.txt
gpu_device_mem_usage_proc_000001.txt
gpu_device_info_proc_000002.txt
gpu_device_mem_usage_proc_000002.txt
```

```
gpu_device_info_proc_000003.txt
gpu_device_mem_usage_proc_000003.txt
```

These files contain statistics of the run on the GPU devices. If the jobs crash, you should check these files to ensure that you did not use too much memory per GPU.