Model Document: Elastic 3D Seismic Wave Modelling Example

Janaki Vamaraju

Final Project,
Tools and Techniques for Computational Sciences,
The University of Texas at Austin

December 12, 2015

Abstract

This document describes a simple numerically approximate solution to a 3D Elastic Seismic Wave Equation with a point Source.

1 Brief Problem Description

For this project I design and implement a well written,organized,parallelized and verified code that numerically approximates a solution to a 3D Elastic Seismic Wave Equation with a point Source. In a three-dimensional isotropic elastic earth, the wave equation solution consists of three velocity components and six stresses. I discretize the partial derivatives using second order in time and fourth order in space staggered finite difference operators. Realistic-sized elastic wave propagation simulations in three dimensions are extremely computationally intensive. Even assuming an isotropic elastic earth (which reduces the number of unknowns and equations to nine), reasonable sized simulations may require too much memory and too much time to run on a single processor machine. We therefore need an organized , core and disk memory optimized and verified code for further analysis. For this project I prefer a simple explicit leapfrog iterative scheme to solve for the wave equation. This application is implemented in C.

2 Physics Behind the Model

Modern computational power makes possible realistic simulations of elastic wavefields at frequencies of interest. The most general numerical methods for modelling are grid-based techniques that track the wavefield on a dense 3D grid of points, e.g. the finite-difference, finite-element and pseudospectral methods. The study of wavefield propagation with 3D finite-difference methods has contributed to a better understanding of wave-path effects and the response of complex structures. The approach using a staggered grid in finite-difference methods has the advantage of accuracy with less computational effort because it allows a larger grid size. In this project we use a velocity stress formulation of isotropic elastic wave equation to model a heterogeneous layered medium.The advantages this formulation are (1) source insertion can be expressed by velocity or stress; (2) a stable and accurate representation for a planar free-surface boundary is easily implemented; (3)

since the finite-difference operator are local, the entire model does not have to reside in memory at once; (4) it is easily extended to high-order spatial difference operators and (5) the algorithm is easily implemented on scalar, vector, or parallel computers.

Based on elastic wave theory, the equations describing elastic wave propagation within 3D, linear, isotropic elastic media can be written as :

$$\begin{aligned}
\rho\partial_{tt}u_x &= \partial_x\tau_{xx} + \partial_y\tau_{xy} + \partial_z\tau_{xz} + f_x \\
\rho\partial_{tt}u_y &= \partial_x\tau_{xy} + \partial_y\tau_{yy} + \partial_z\tau_{yz} + f_y \\
\rho\partial_{tt}u_z &= \partial_x\tau_{xz} + \partial_y\tau_{yz} + \partial_z\tau_{zz} + f_z
\end{aligned}$$

$$\begin{aligned}
\tau_{xx} &= (\lambda + 2\mu)\partial_x u_x + \lambda(\partial_y u_y + \partial_z u_z) \\
\tau_{yy} &= (\lambda + 2\mu)\partial_y u_y + \lambda(\partial_x u_x + \partial_z u_z) \\
\tau_{zz} &= (\lambda + 2\mu)\partial_z u_z + \lambda(\partial_x u_x + \partial_y u_y) \\
\tau_{xy} &= \mu(\partial_y u_x + \partial_x u_y) \\
\tau_{xz} &= \mu(\partial_z u_x + \partial_x u_z) \\
\tau_{yz} &= \mu(\partial_z u_y + \partial_y u_z)
\end{aligned}$$

(a)

Figure 1: Equations of motion and Hooke's law

In the equations above, (u_x,u_y,u_z) are the displacement components; $({}_x x,{}_y y,{}_z z,{}_x y,{}_x z,{}_y z)$ are the stress components; $(f_x,\ f_y,\ f_z)$ are the body-force components; is the density; and are Lame constants.

The equations in (1) can be formulated into a set of first-order differential equations by first differentiating equations (2) with respect to time and then substituting the velocity components $(v_x,\ v_y,\ v_z)$ for the time-differentiated displacements. The resulting sets of equations can be written as

The staggered grid defines some of the velocity and stress components shifted from the locations of other components by half the grid length in space. One of the attractive features of the staggered-grid approach is that the various difference operators are all naturally centred at the same point in space and time. The system is not only staggered on a spatial grid but also temporally, which means that the velocity field can be updated independently from the stress field. This makes the scheme efficient and concise. The differential operators only act on the wavefield variables, differencing of the media coefficients is not necessary in this scheme, and the complexity of the media has no impact on the form of the differential terms. Therefore, the scheme can handle arbitrary media. The time updated wavefields are computed such that the velocity field at time n+(delt/2) is determined explicitly by using the velocity field at time n-(delt/2) and the stress field at time n*delt. Therefore,

$$\partial_t v_x = \frac{1}{\rho}(\partial_x \tau_{xx} + \partial_y \tau_{xy} + \partial_z \tau_{xz} + f_x)$$
$$\partial_t v_y = \frac{1}{\rho}(\partial_x \tau_{xy} + \partial_y \tau_{yy} + \partial_z \tau_{yz} + f_y)$$
$$\partial_t v_z = \frac{1}{\rho}(\partial_x \tau_{xz} + \partial_y \tau_{yz} + \partial_z \tau_{zz} + f_z)$$

$$\partial_t \tau_{xx} = (\lambda + 2\mu)\partial_x v_x + \lambda(\partial_y v_y + \partial_z v_z)$$
$$\partial_t \tau_{yy} = (\lambda + 2\mu)\partial_y v_y + \lambda(\partial_x v_x + \partial_z v_z)$$
$$\partial_t \tau_{zz} = (\lambda + 2\mu)\partial_z v_z + \lambda(\partial_x v_x + \partial_y v_y)$$
$$\partial_t \tau_{xy} = \mu(\partial_y v_x + \partial_x v_y)$$
$$\partial_t \tau_{xz} = \mu(\partial_z v_x + \partial_x v_z)$$
$$\partial_t \tau_{yz} = \mu(\partial_z v_y + \partial_y v_z)$$

(a)

Figure 2: Velocity stress formulation

the time update scheme is very straightforward, and source implementation (stress, velocity and so on) is explicit and can be accomplished by simply adding the source component to the wavefield. To represent a planar free-surface boundary in the staggered-grid finite-difference scheme, an accurate and numerically stable formulation can be implemented by explicitly satisfying the zero-stress condition at the free surface and then use an imaging condition for update of velocities. There are a number of ways to apply absorbing boundary conditions. Radiation conditions may be satisfied explicitly (Clayton and Engquist, 1977; Stacey, 1988), or the solution may be tapered over a thin strip along the boundary (Cerjan et al., 1985; Loewenthal et al., 1991). To meet radiation conditions, only two fictitious strips of nodes along the boundary for fourth order operators are required, whereas tapering generally requires more strips. Tapering is the easiest to implement, is efficient, and is used in our late simulations.

3 Requirements that were met in forming the code

To develop this code I adopted a modular coding style with a short main function and manageable chunks of code through routines provided in header files. Comments are provided in the code wherever needed to make it easily understandable. All the global variables are dynamically allocated making the memory requirements known in prior. Frequent use of SVN has helped in improving the organization of the code and better source code control.Standard and debug modes are provided to help the user analyse the code. Gnuplot and Matplotlib have been used to provide coherent figures wherever needed.HDF5 and GRVY libraries have been used to ease the input/output parts of the code. Verification analysis is done and code coverage through the gcov tool helps us analyse the weak parts of our code. Performance analysis is done through profilers such as gprof and GRVY timers. We discuss each of these topics in the rest of the document and provide illustrations as needed.

3.1 Build Procedure

The main directory contains an sbatch file "jobscript" and a Makefile. Before you begin to build the code the following modules are needed to be loaded: INTEL module for using intel compilers, GCC module for using GNU compilers, GRVY and BOOST Libraries, HDF5 library ,

On stampede (TACC COMPUTING RESOURCES) we make sure we have the specific following modules loaded: intel/13.0.2.146 , grvy/0.32.0

If u want to use the gcc compilers load module gcc and add the flags -fopenmp ,-lm for compilation and linking. Once the above modules are loaded,make command should build the executable for you. A sbatch jobscript has also been provided to automate submission jobs on stampede.

3.2 Input Options

The application derives necessary runtime options from an input le. It will always read from the same lename "input2.txt". We use the GRVY library installed on Stampede to simplify the process (API documentation is avail-able online).A sample input file has been provided in the main directory. This can be edited and used as required.The input options required are as follows and are in the same order in the input file.

- x0 and x (starting and ending x coordinates of the domain)
- y0 and y (starting and ending y coordinates of the domain)
- z0 and z (starting and ending z coordinates of the domain)
- h (grid size)
- total time of the simulation and step size
- width of the Absorbing boundary
- peak frequency of ricker source function
- Number of layers in the medium
- Density (kg/m^3) S-Wave velocity and P-wave velocity (m/s) for each layer
- Order of the finite difference spatial discretion (2nd order or 4th)
- 0 to turn off the verification mode and 1 to turn on
- Receiver locations. Stating and ending x y and z coordinates
- one source location (x,y,z coordinate respectively)
- 0 to turn off debug mode 1 to turn on debug mode

```
 #-------------------------------------------
# Example global input variable definitions
#-------------------------------------------

x0 = -200               # grid location starting on x axis (all in m)
x = 200                 # grid location end on x axis
yy0 = -5                # grid location start on y axis
y = 5                # grid location end on y axis
z0 = 0                # grid location start on z axis
z = 400              # grid location end on z axis
h = 2                # grid size (m)
totaltime = 0.6               #total simulation time(secs)
delt = 0.00015          # time step(secs)
PML = 40                # absorbing boundary width(m)
fc = 20               # freq of the ricker source(Hz)
nol = 2     # number of layers
Du = '1000 2000'                # density gridfilefor each layer(Kg/m3)
Vs = '2500 2750'        # Vs grid file (Shear velocity m/s)
Vp = '3000 3000'        # Vp grid file (Compression wave velocity m/s)
order          = 4                # finite difference order in space (2/4)
modev = 0 #verification mode off = 0 on = 1
rec = '22 221 24 24 4 4'       # receiver grid locations(starting x location,ending x location

                                         # input grid location accorindg to formula ((xlocation-
src = '23 24 10'       # source locations (1 grid point x y and z coordinate)

debug = 0 # set to 0 to off debug mode 1 to on debug mode;
```

3.3 Verification and Code Coverage

For a two layered model in the sample input file the seismogram collected at the surface are shown below:

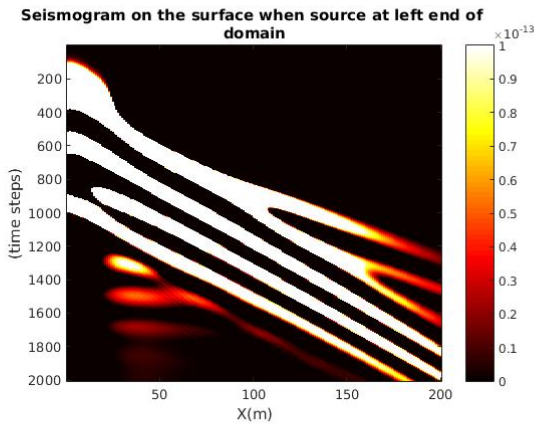
Because of the change in the properties of the two layers the reflected waves are clearly noticeable. We now perform a Verification our application. The finite difference scheme implemented is very much conditionally stable. Based on the stability and grid dispersions performed the following criteria must be satisfied when choosing a grid size and a step size.

$h < (\frac{Vs}{n*Fmax})$ where Vs is the minimum Shear velocity of the medium, Fmax is maximum source frequency and n is a Holberg or Taylor coefficient based on the order of the FD scheme. This is a check for grid dispersion.

$delt < (\frac{h}{H*\sqrt{3}*Vpmax})$ where H is a constant and Vp max is maximum P wave velocity.

In our application we have an option for verification where various grid sizes are selected and tested for stability or grid dispersion problems. Error message is displayed if the code becomes unstable.

We also us the code coverage tool to help us aid in our analysis of the code. Using the –coverage flag and gcov the following results have been generated.



(a)

Figure 3: Seismogram on the surface of the domain

```
-:    0:Source:forward.c
-:    0:Graph:forward.gcno
-:    0:Data:forward.gcda
-:    0:Runs:1
-:    0:Programs:1
-:    1:#include<stdio.h>
-:    2:#include<stdlib.h>
-:    3:#include<math.h>
-:    4:#include <string.h>
-:    5:#include<unistd.h>
-:    6:#include<grvy.h>
-:    7:#include<sys/time.h>
-:    8:#include<time.h>
-:    9:#include"omp.h"
-:   10:#include"globalvar.h"
-:   11:#include"damping3d.h"
-:   12:#include"routines.h"
-:   13:#include"write.h"
-:   14:
1:   15:int main(int argc, char **argv)
-:   16:{
-:   17:int count,i;
-:   18:double t,time1,time2;
1:   19:allocations();  //Dynamic allocatins of all variables
1:   20:grvy_timer_init("GRVY Example Timing");
1:   21:for(ml=1;ml<=1;ml++)  // loop for 6 shots
-:   22:{
1:   23:grvy_timer_reset();
```

```
        1:   24:grvy_timer_begin("Main Program");
        1:   25:time1 = omp_get_wtime();
        1:   26:count2 = 0;
        -:   27:count =0;
     4002:   28:for(t=0;t<totaltime;t=t+delt)
        -:   29:{
     4001:   30:printf("%d Current time step is %lf\n",ml,t);
     4001:   31:forcing(t);   // Enter forcing function
     4001:   32:stressupdate();  // update stresses
     4001:   33:velocityupdate(); // update velocities
     4001:   34:boundarycns();  // apply boundary conditions
     4001:   35:storewave();// storing recorded signals
        -:   36:count = count+1;
        -:   37://printf("%3.50lf \n",U3[mat(23,24,4)]);
        -:   38:}
        1:   39:printf("code is good %d \n",count2);
        1:   40:time2 = omp_get_wtime();
        1:   41:printf("time is %3.50lf \n",time2-time1); //prints time take for one shot
        1:   42:write_to_file(forward,count2); // writing to disk the stored velocities
        1:   43:grvy_timer_end("Main Program");
        1:   44:grvy_timer_finalize();
        1:   45:grvy_timer_summarize();
        -:   46:const int num_procs = 1;
        -:   47:
        1:   48:  grvy_timer_save_hist("C-Example1","My clever comment",num_procs,"hist.h5");
    #####:   49:exit(1);
        -:   50:}
        -:   51:}
```

The above results clearly show us the maximum usage of lines takes place while updating stresses ,velocities and applying boundary conditions for each time step. Using the analysis files for other parts of codes we can clearly check for thos parts which do not contribute much to the code. This will help us in making our code efficient and finding procedures which take a lot of time.

3.4 Runtime Performance Analysis

In this section we provide runtime performance measurements of the application using refinements of mesh sizes.The total runtime is then delineated into at least 3 sections which account for at least 90% of the total application runtime. To help analyse runtime various tools are available. We choose to do the following

1. Use time ./executable to measure total runtime
2. Use openmp timer to measure the runtime
3. Use gprof for basic profiling and runtime measurement

4. Lastly make use of the GRVY library timers to give us a detailed analysis as we refine our meshes.

For the analysis we first choose a simple heterogeneous 2 layered medium that is provided as the sample input file in the main directory. With a mesh size of 2m the domain covers an area of 400m by 10m by 400m. The simulation is run for 0.6 seconds with a 0.00015 time step. Results are as follows:

Table 1: runtime measurements

	real,4m46.083s
time ./forward.out	user,50m15.868s
	sys,20m19.938s
openmp timer	284.40933s

Using gprof to profile and measure the runtime of the application we have:

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 40.74    807.95   807.95     4001     0.20     0.20  velocityupdate
 38.49   1571.20   763.25     4001     0.19     0.19  stressupdate
 20.70   1981.57   410.38     4001     0.10     0.10  boundarycns
  0.05   1982.47     0.90        1     0.90     1.04  allocations
  0.02   1982.80     0.33                             exp.L
  0.01   1983.01     0.21     4001     0.00     0.00  storewave
  0.01   1983.15     0.14 19293750     0.00     0.00  damping
  0.00   1983.17     0.02                             exp
  0.00   1983.17     0.00     4001     0.00     0.00  forcing
  0.00   1983.17     0.00        1     0.00     0.00  write_to_file


     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.00% of 1983.17 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00 1982.82                 main [1]
              807.95    0.00    4001/4001        velocityupdate [2]
              763.25    0.00    4001/4001        stressupdate [3]
              410.38    0.00    4001/4001        boundarycns [4]
                0.90    0.14       1/1           allocations [5]
                0.21    0.00    4001/4001        storewave [7]
                0.00    0.00    4001/4001        forcing [10]
                0.00    0.00       1/1           write_to_file [11]
-----------------------------------------------
              807.95    0.00    4001/4001        main [1]
[2]     40.7  807.95    0.00    4001         velocityupdate [2]
-----------------------------------------------
              763.25    0.00    4001/4001        main [1]
[3]     38.5  763.25    0.00    4001         stressupdate [3]
-----------------------------------------------
              410.38    0.00    4001/4001        main [1]
[4]     20.7  410.38    0.00    4001         boundarycns [4]
-----------------------------------------------
                0.90    0.14       1/1           main [1]
[5]      0.1    0.90    0.14       1         allocations [5]
                0.14    0.00 19293750/19293750     damping [8]
-----------------------------------------------
                                                 <spontaneous>
[6]      0.0    0.33    0.00                 exp.L [6]
-----------------------------------------------
                0.21    0.00    4001/4001        main [1]
[7]      0.0    0.21    0.00    4001         storewave [7]
-----------------------------------------------
                0.14    0.00 19293750/19293750     allocations [5]
[8]      0.0    0.14    0.00 19293750         damping [8]
-----------------------------------------------
                                                 <spontaneous>
[9]      0.0    0.02    0.00                 exp [9]
-----------------------------------------------
                0.00    0.00    4001/4001        main [1]
[10]     0.0    0.00    0.00    4001         forcing [10]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[11]     0.0    0.00    0.00       1         write_to_file [11]
-----------------------------------------------
```

Using GRVY Library timers :

```
-------------------------------------------------------------------------------------------------------------
GRVY Example Timing - Performance Timings:            |      Mean      Variance       Count
--> velocityupdate    : 1.15282e+02 secs ( 40.2063 %) | [2.88133e-02  4.80174e-05       4001]
--> stressupdate      : 1.01561e+02 secs ( 35.4210 %) | [2.53840e-02  4.86655e-05       4001]
--> boundarycns       : 4.86667e+01 secs ( 16.9732 %) | [1.21636e-02  4.40085e-05       4001]
--> write_to_file     : 2.09174e+01 secs (  7.2952 %) | [2.09174e+01  0.00000e+00          1]
--> storewave         : 2.17522e-01 secs (  0.0759 %) | [5.43670e-05  9.06918e-08       4001]
--> Main Program      : 2.32978e-02 secs (  0.0081 %) | [2.32978e-02  0.00000e+00          1]
--> forcing           : 4.13728e-03 secs (  0.0014 %) | [1.03406e-06  2.45192e-13       4001]
--> allocations       : 0.00000e+00 secs (  0.0000 %) | [    N/A     0.00000e+00          0]
--> GRVY_Unassigned   : 5.37720e-02 secs (  0.0188 %)

    Total Measured Time = 2.86726e+02 secs (100.0000 %)
-----------------------------------------------------------------------------------------------
```

As we refine our mesh size the results from GRVY timers have been as follows:

Table 2: My caption

GRID SIZE	TOTAL TIME (s)	three functions with longest runtime		
2m	284.4s	Velocity update-40.44%	stress update-35.76%	BC-17%
3m	179.7 s	Velocity update-40.45%	stress update-22.69%	BC-21%
5m	97.602s	Write to file-72 %	Main program-10 %	Velocity-7.6%
8m		Write to file-82%	Main program-12%	Velocity-2%

All the timers show us approximately the same time. With help of gprof and grvy libraries we can further analyse our code. We can see from the last results that as the mesh size increase computation time increases. With smaller meshes sizes the functions that take the maximum time are that perform update of velocities,stresses and apply boundary conditions. With larger mesh sizes the overall time reduces but major chunk of time is being taken by the function which writes the results to files. This tells us that our writing to files procedure can be a lot more efficient.

4 Conclusions

This project has helped me to learn various tools and technique to improvise my code structure making it more organized, well structured and parallelised. Tools like SVN,GRVY llibrary,HDF5 library have helped me to store all the changes I make to the code,record their performance and perform an efficient way of input parsing data to the code. Since our application has been relatively small I chose a simple makefile to build the code. Introducing complexities in my code can be make it unstructured. This is where GNU autotools is the perfect option to build my code. I would further like to implement it and make a better build structure for the application. Profiler and timers have helped me in making the application faster. I understand that MPI parallelism would make the application further efficient and less time consuming compared to Openmp.