SANDIA REPORT
SAND200X-XXXX
Unlimited Release
Printed September 2016

Paracousti User Guide

Leiph A. Preston

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND200X-XXXX
Unlimited Release
Printed September 2016

Paracousti User Guide

Leiph A. Preston
Geophysics
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0750

Abstract

Paracousti is a parallelized acoustic wave propagation simulation package developed at Sandia National Laboratories. It solves the linearized coupled set of acousto-dynamic partial differential equations using finite-difference approximations that are second order accurate in time and fourth order accurate in space. Paracousti simulates sound wave propagation within realistic 3-D earth, static atmosphere and hydroacoustic models, including 3-D variations in medium densities and acoustic sound speeds and topography or bathymetry. It can also incorporate attenuative media such as would be expected from physical mechanisms such as molecular dissipation. This report explains the usage of the Paracousti algorithm.

ACKNOWLEDGMENTS

This work was funded by the U.S. Department of Energy's Wind and Water Power Technologies Office.

CONTENTS

FIGURES

NOMENCLATURE

TDAAPS	Time Domain Atmospheric Acoustic Propagation Suite
PML	Perfectly Matched Layer
CPML	Convolution Perfectly Matched Layer
MPML	Multi-axial Perfectly Matched Layer
DOE	Department of Energy
SNL	Sandia National Laboratories

1. INTRODUCTION

Paracousti is a parallelized acoustic wave propagation simulation package developed at Sandia National Laboratories. It solves the linearized coupled set of acousto-dynamic partial differential equations using finite-difference approximations that are second order accurate in time and fourth order accurate in space. Paracousti simulates sound wave propagation within realistic 3-D earth, atmosphere and hydroacoustic models, including 3-D variations in medium densities and acoustic sound speeds and topography or bathymetry, including voids in the subsurface. Although it assumes ideal fluid media, it can also simulate wave propagation in attenuative media such as would be expected from physical mechanisms like molecular dissipation. The code assumes that the densities and sound speeds are fixed in time over the duration of a run. Paracousti uses a massively parallel design. It can run efficiently on a signal machine or on a massively parallel machine with thousands of cores.

Paracousti is similar to TDAAPS (Time Domain Atmospheric Acoustic Propagation Suite) (Symons et al., 2005; Preston, 2016) in its usage and mechanics, but, unlike TDAAPS, Paracousti does not incorporate moving media (e.g., wind or currents) into its fundamental equations.

There are a few basic pieces of information that are required as input: the earth/ atmosphere model; the source location, type and strength; and the output recording geometry. These elements will be outlined below. Note that Paracousti is not a model construction software package and it has no capability to build any models.

2. SOLUTION EQUATIONS

We solve a set of coupled first order linear partial differential equations known as the velocity-pressure system:

$$\begin{aligned} \frac{\delta v_i(\mathbf{x},t)}{\delta t} + \frac{1}{\rho(\mathbf{x})}\frac{\delta p(\mathbf{x},t)}{\delta x_i} &= \frac{1}{\rho(\mathbf{x})}\left[F_i(\mathbf{x},t) + \frac{\delta m_{ij}^a(\mathbf{x},t)}{\delta x_j} \right] \\ \frac{\delta p(\mathbf{x},t)}{\delta t} + \kappa(\mathbf{x})\frac{\delta v_i(\mathbf{x},t)}{\delta x_i} &= -\frac{1}{3}\frac{\delta m_{ii}^s(\mathbf{x},t)}{\delta t} \end{aligned} \tag{1}$$

where $v_i(\mathbf{x},t)$ and $p(\mathbf{x},t)$ are the dependent variables of particle velocity and pressure perturbations, respectively, $\rho(\mathbf{x})$ is material density, and $\kappa(\mathbf{x})$ is the material bulk modulus. Repeated indices imply summation. The right hand terms in these equations are body source terms: $F_i(\mathbf{x},t)$ is the force density vector, $m_{ij}^a(\mathbf{x},t)$ is the anti-symmetric portion of the moment density tensor and $m_{ii}^s(\mathbf{x},t)$ is the trace of the symmetric portion of the moment density tensor. A moment tensor is a 3 × 3 tensor that describes the action of various combinations of force couples applied at a point in space. The isotropic portion (representing a source of pressure) of the source is proportional to the trace of the moment tensor. Quite complex sources can be built by various combinations of these terms. Outside of the source region, the body source terms are zero, yielding a homogeneous system of partial differential equations.

An alternative method for initiating wave motion is by imposing time varying boundary conditions on the dependent variables (both velocity and pressure). These boundary conditions can be computed with another algorithm that can more accurately simulate non-linear or other near-source effects. This approach allows one to both compute the near-source effects to high accuracy and also to propagate these effects efficiently out into the far field.

Note that both the densities and bulk moduli are functions of 3-D space but not of time. Arbitrarily complex 3-D distributions of medium properties are allowed including topography, bathymetry, acoustic sound speed variations due to temperature, salinity, and pressure in the water, in addition to sub-sea bed variations in earth structure. Also, stationary atmospheric models (i.e, without wind) can be utilized. Furthermore, purely acoustic simulations within the solid earth can be made where computational speed is essential, albeit at the cost of only generating compressional waves in these cases, whereas in reality, shear waves and all their related phenomena would be created as well. Careful treatment of high contrast interfaces, such as air-earth, water-air, water-earth, using the order-switching technique outlined in Preston, et al. (2008) allows accurate and stable simulation across these boundaries. Obviously, more simplistic models can also be developed for testing and evaluation purposes.

3. SOLUTION METHODOLOGY

We utilize a finite differencing scheme in order to solve the velocity-pressure system of Equation 1. The numerical modeling domain is defined by a uniform grid of points such as shown in Figure 1. Figure 2 zooms in on one cell of the domain with a size of $d_x \times d_y \times d_z$. The eight corner nodes of this cell contain medium densities, bulk moduli and the pressure dependent variables. The three components of particle velocities reside on the twelve edges of the cell. This arrangement is known as a standard staggered grid. It allows central differencing of all the dependent variables to be used. The time axis is divided into equal segments of length d_t. Time is also staggered with pressure updates occurring on the integer time raster and velocity updating occurring on the half integer time raster. Again, this allows for more accurate central differencing.

Fourth-order accurate finite differencing is utilized to approximate the spatial derivatives in Equation 1, while second-order accurate templates are used for the temporal

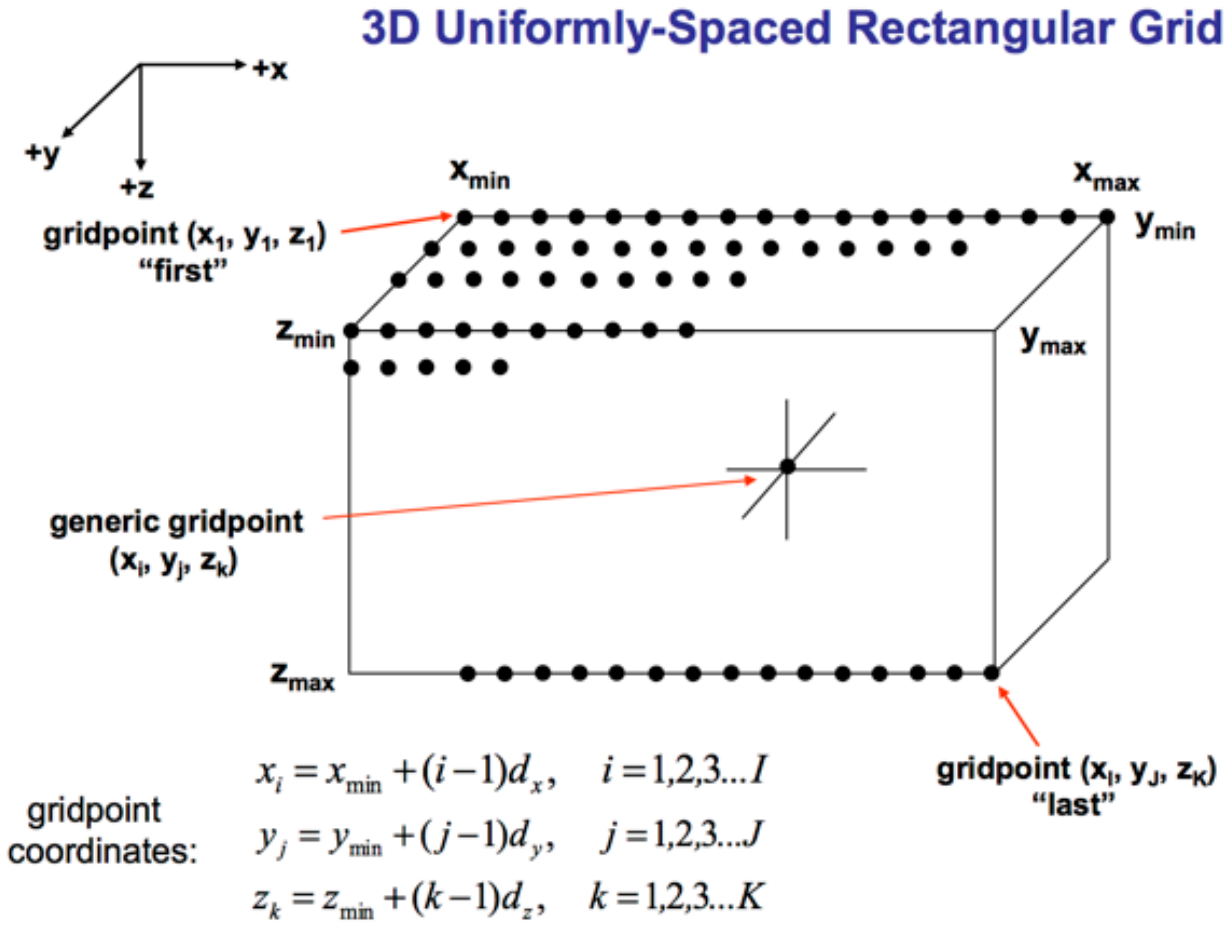

***Figure 1**: The computational domain (large box) represented by a 3-D uniformly spaced grid with nodes (gridpoints) indicated by black dots.*

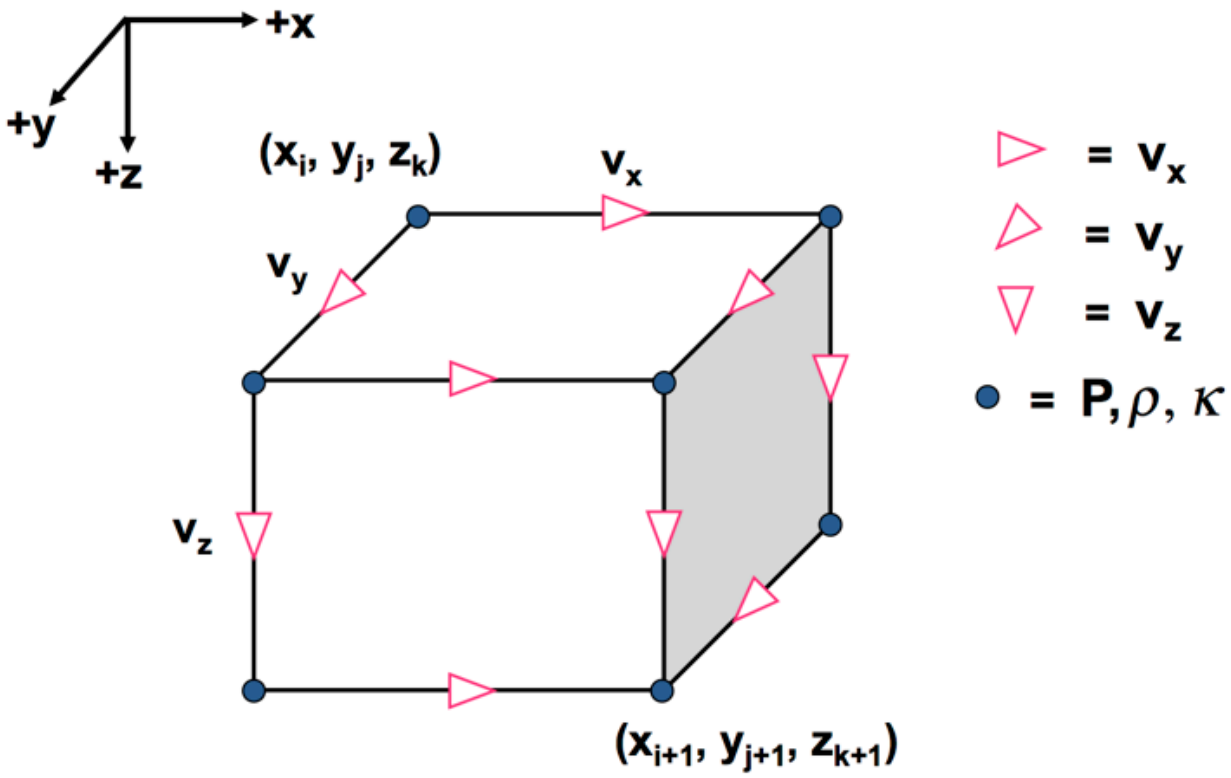

***Figure 2**: Arrangement of dependent variables and medium parameters for one cell of the standard staggered grid.*

derivatives. However, near high contrasts in medium parameters, such as at the air-water, air-earth or water-earth interface, we use second-order spatial accuracy in the immediate vicinity of the interface to increase accuracy and preserve numerical stability (Preston et al., 2008). Time evolution uses an explicit in-time leap-frog method.

In order to simulate an unbounded domain, we impose "absorbing boundary conditions" (ABC) on the flanks of the 3-D grid in order to suppress reflected energy. Two basic choices of absorbing boundary conditions are available: sponge (wavefield taper) or perfectly matched layers (PMLs). A sponge boundary condition (Cerjan et al., 1985) is a very simple ABC in which the dependent variables within the boundary layer are multiplied by a factor less than one. Although it is simple and fast, these ABCs perform poorly for grazing incidence and the optimal parameters are problem-dependent. A much better performing, but more computationally expensive per node ABC, is the PML. First introduced in Beringer (1994), this ABC is theoretically perfect, meaning that at the onset of the PML zone, no reflection back into the computational domain would occur. In practice this is not true, but it does provide much superior performance over a thinner ABC zone, which makes it comparable in computational speed to the sponge and is recommended in virtually all cases. We provide three PML options: traditional PML, convolutional PML (CPML), and multi-axial PML (MPML). The MPML is the most general form of the three, with the others just being special cases. In

a MPML, unlike the traditional or CPML, the wavefield is damped both perpendicular and parallel to the domain face (Meza-Fajardo et al., 2008). This will produce some small reflection back into the computational domain, but in certain instances, such as long-propagating grazing incidence waves, the MPML outperforms the other PML types. The CPML (Komatitsche and Martin, 2007) was designed to greatly reduce boundary and grazing incidence effects that hampered the traditional PML with the result that only a negligible amount of energy is reflected back inside the domain from the boundaries of the computational domain. For most problems, the CPML is the best choice, but if grazing incidence waves contaminate the solution, the MPML would be the next best option. Only for very simple problems where grazing incidence wave will not be a concern should the traditional PML be considered.

We have the ability to impose a pressure-free surface at the water-air or earth-air interface. Although not strictly true at these interfaces, it is a very good approximation. An alternative that we commonly use, however, is to simply assign air or even vacuum properties above the water or earth. This latter approach indeed must be used when there is topography since the pressure-free surface implementation requires the air-earth and/or air-water interface to be planar.

4. INSTALLATION AND SOLUTION PROCESS

A three-step modeling process for simulating acoustic wave propagation is depicted in general terms in Figure 3. The numerical simulation algorithm (center box) accepts various inputs (left column) and then calculates and outputs various data types (right column). A crucial input is the "earth model", consisting of a gridded representation of the medium parameters defined on a 3-D rectangular grid, as in Figure 1. Another input file contains a description of the recording geometry (i.e., types and positions of acoustic energy sources and receivers). Calculated data are of three types: 1) traces, or time-series of particle velocities or pressure at designated receiver locations in the 3-D grid; 2) "time slices" or 2-D pictures of time-evolving acoustic wavefields; and 3) 3-D wavefield volumes. Visualization software (often user-specific) is needed to display these data types.

4.1 Software Environment and Setup

Paracousti is a collection of C, C++ and Fortran source files. As such, to compile Paracousti a compiler capable of compiling all these languages is required. A netcdf 3+ library must also be on your library path. MPI (openMPI) is an additional requirement for compilation.

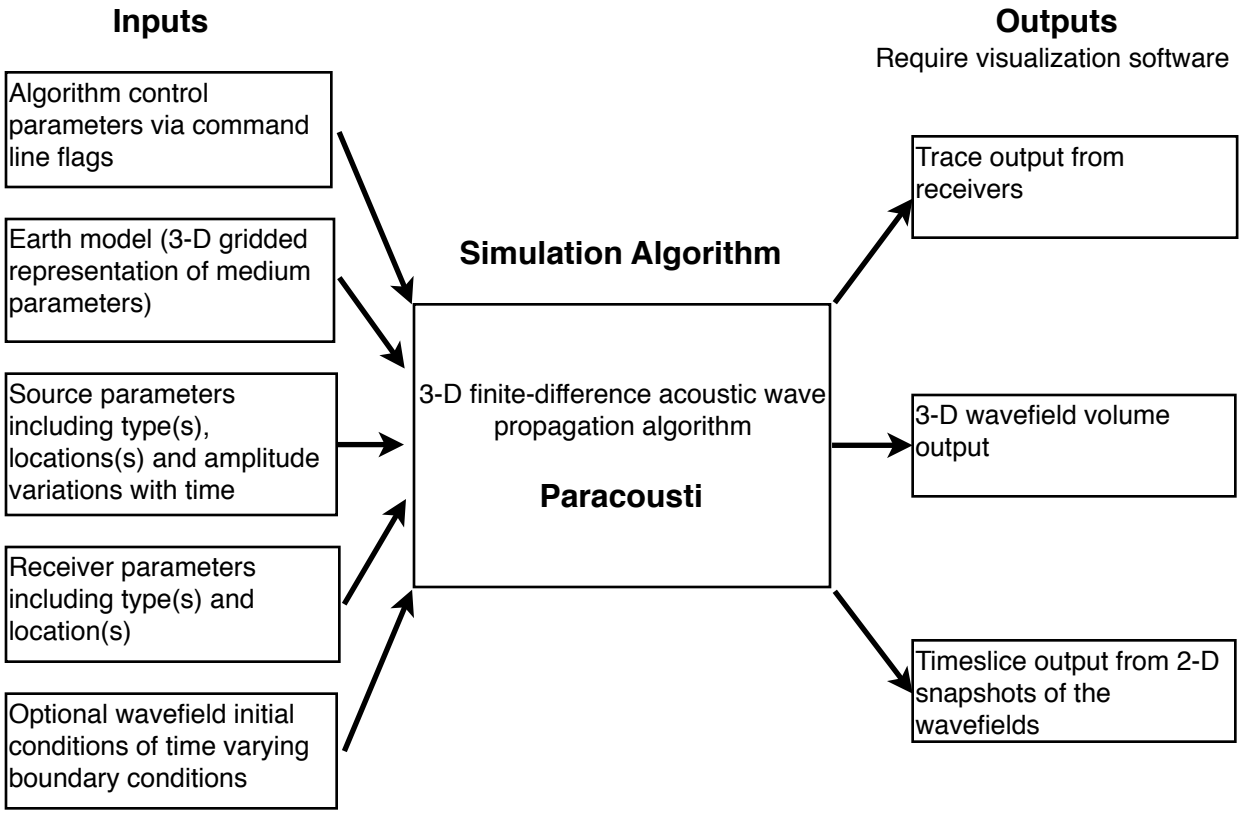

***Figure 3**: General depiction of the acoustic wave propagation modeling process.*

If you have an executable of Paracousti, then the netcdf requirement depends on whether the netcdf library was statically linked internally to the executable. If it was bundled as a static internal library, then the netcdf library need not be on the machine. In all cases, it is required that standard C, C++, and Fortran libraries reside on your runtime library path. MPI is needed to run Paracousti, even on a single machine. Paracousti must always be executed through mpirun or mpiexec.

The next four sections will describe the basic elements required by Paracousti to run as well as some of the most common command line argument flags used to control its operation.

4.2 Input Parameters

The following information is needed to initialize an algorithm run:

- Density: Medium mass density as a function of space over the entire 3-D model domain
- Acoustic Sound Speed: Medium sound speed as a function of space over the entire 3-D model domain
- Source Information: This will consist of either:
  - Body force and/or moment tensor sources. Each source requires a location in 3-D space and how the source amplitude varies as a function of time.
  - Time dependent boundary conditions: Pressure and the three components of particle velocity as required on a surface as a function of time. More detail is given below.
- Output type and parameters: Several types of data output are available. Depending on the data type, different parameters are required for definition.

4.3 Output Parameters

The following types of output data are available. They are not mutually exclusive and multiple outputs of a single type are also allowed.

- Trace data: These represent point receivers (like a hydrophone) in the model. They record pressures or particle velocities at a specific location as a function of time. The 3-D location and type of trace (pressure, x-component of velocity, etc.) are required to define a trace.
- Time slices: This output type captures pressures or velocities on a 2-D slice through the model as a function of time. The slice plane must be aligned with the underlying Cartesian grid. The type of output (e.g., pressure, etc.), the orientation of the plane (XY, XZ or YZ), the location of this plane, and the time between snap shots is required. These make nice movies to view the evolution of the wavefield with time.
- Volume output: This captures pressures or velocities on the entire 3-D grid as a function of time. The type of output and the time between snap shots is required. This gives one the ability to look at any view of the evolving wavefield as a function of time, but the output files are enormous and this output type is generally not used unless absolutely required.

4.4 Model

The earth/atmospheric model is stored in one or more NetCDF format binary files (http://www.unidata.ucar.edu/software/netcdf/). Libraries and routines for reading and writing NetCDF files in a variety of programming and scripting languages, including C, C++ and Matlab, are available through the above website. A NetCDF file is a self-contained unit that defines dimensions, variables and attributes within a single file that is machine independent. Multiple variables may be contained within a single file and all the dimensions associated with those variables are contained within that same file.

A simple model building code for Matlab is writeSgfdModel.m, a function that writes files in the correct format for Paracousti. writeSgfdModel.m can write simple 1-D models, or full 3-D models with existing 3-D Matlab arrays. The 3-D arrays must be built elsewhere and writeSgfdModel.m will simply write out the correct format model file for that 3-D data.

For 1-D models, the Matlab call would be:

```
writeSgfdModel(modelName,x,y,z,t,'1d',vpz,vsz,rhoz);
```

with modelName containing the name of the model file you want to create as a single quoted string, x gives the x-axis values in standard Matlab vector format, i.e., x0:dx:xf with x0 the starting x-axis value, dx the increment (see below for how to determine dx) and xf is the final x-axis value, and y, z and t provide the y, z and time axis vectors respectively (see below on how to define dt for the time-axis), vpz is a vector of length(z) containing the vp values in m/s, vsz and rhoz are the same except for vs (m/s) and density (kg/m^3). Note that z is defined positive pointing downwards. Also, note that vs is required for the above 1-D model call even with acoustics (this is only true for the writeSgfdModel function for 1-D models); however, vs is ignored for acoustic runs. For 3-D models, use the call:

```
writeSgfdModel(modelName,x,y,z,t,'vp',vp,'rho',rho);
```

with the first five arguments the same as above. This time vp is a 3-D array of vp values (m/s) with dimensions [nz, ny, nx]. rho is the same except for representing density (kg/m^3). These Matlab calls build the model file as described below.

For complex and especially large models, it is often more convenient to build the model in C, C++, etc. To accomplish this, one must know what is required of the netCDF model file. Paracousti requires that several variables (and their associated dimensions) be defined within a model file. The required dimensions are 'NX', 'NY', 'NZ', 'NT' and 'numCoord'. Besides 'numCoord' which is always equal to 4, these dimensions are determined by the user based on the desired model size and length of time of simulation. 'NX', 'NY' and 'NZ' define the number of nodes in the x, y and z directions respectively. 'NT' gives the number of time steps to run the algorithm (this parameter may be overridden on the command line).

Several variables are required and are outlined below (dimension followed by variable type in parenthesis):

minima: (numCoord; float) a four element array containing [x0, y0, z0, t0], which are the minimum values of x, y, z and t, respectively. Note that if the x-axis is oriented along the east-west direction and the y-axis is along the north-south direction, then x increases towards the east, y increases toward the north and z increases down into the earth.

increments: (numCoord; float) a four element array containing [dx, dy, dz, dt], which are the spacing between x, y, z and t nodes. It is best if dx, dy and dz are equal. dt is also known as the time step. The model extents will be (NX-1)*dx, (NY-1)*dy and (NZ-1)*dz and the total simulation time will be (NT-1)*dt. These values must be computed from characteristics of the acoustic sound speeds in the model and on desired frequency content. The dx should be defined as:

dx = min(Vp)/max(Freq)/10

for good wave simulation (see -hc flag below which could allow less stringent dx criteria). Vp is the acoustic sound speed. So, for example, for good wave simulation with a minimum Vp of 500 m/s and a maximum frequency of 100 Hz, dx = 500/100/5 = 1 m. Note that the maximum frequency is defined as the frequency where the far-field amplitude spectrum is 1% of its peak value. The far-field spectrum is the doubly differentiated source waveform if an explosion source is utilized and the singly differentiated source waveform if a force source is utilized. The maximum dt is defined as (approximately) (if you use the -hc flag, see requirements for dt listed there):

dt = dx/max(Vp)/2.04

It is recommended that you use the Matlab function cflDt() in order to find the optimal dt, especially if non-standard FD coefficients are used (-hc flag, below). To use this function, use:

dt = CFLFraction*cflDt(dx,max(Vp),[c0 c1])

where CFLFraction should be between 0 and 1 (exclusive), and c0 and c1 are the inner and outer coefficients, respectively, for the FD operator. [c0 c1] is an optional argument. If it is not provided, standard Taylor Series coefficients, c0=9/8 and c1=-1/24 are assumed.

x: (NX; float) a vector of x-axis values
y: (NY; float) a vector of y-axis values
z: (NZ; float) a vector of z-axis values
time: (NT; float) a vector of time values

The actual geophysical parameters are given in the variables below:

vp: (NZ,NY,NX; float) a 3-D array of the compressional-wave velocities in m/s. In C, the array is defined as a packed 1-D array with x varying the fastest, then y, then z.
rho: (NZ,NY,NX; float) a 3-D array of the densities in kg/m^3.

4.5 Receiver Geometry

The receiver geometry can either be supplied on the command line for simple layouts or in a plain text file for more complicated geometries. The command line allows additions of single receivers or of a uniform grid of receivers. Using a file allows completely arbitrary receiver placements for thousands of receivers if desired.

For the file method, simply define a flat text file with three columns: x, y, and z. Each line will be a new receiver and the x, y and z values will be in the model coordinate system. To include the file, add the following to the command line:

`-Rf3 type filename.txt`

where *type* is either “Pressure”, “3C”, “4C”, “Vx”, “Vy” or “Vz”, where 3C gives all three velocity components per receiver line and 4C also includes pressure.

To add individual receivers, add the following to the command line:

`-R type x y z`

This adds one receiver of type (see above) at position x, y, z in model coordinates.

To add a uniform grid of receivers, add the following to the command line:

`-Rg type x0:dx:xf y0:dy:yf z0:dz:zf`

This adds a uniform grid of receivers of *type* (see above) on the grid defined by the Matlab style vectors. For example, `x0:dx:xf` means x ranging from x0 to xf at an increment of dx (note that dx here is independent of the model dx).

Other receiver command line options that may be useful are:

-Rl : use trilinear interpolation instead of the default cubic interpolation for receiver points. This is important to do for receivers within about 2 grid nodes of any major model interface (such as the sea surface or sea bottom) because a cubic interpolator will reach across the interface to obtain interpolated values, whereas trilinear interpolation is more localized.

-Ra : make acceleration traces instead of the default velocity traces

-Rd : make displacement traces instead of the default velocity traces

-Ro *traceOutputFile.cdf* : the trace output from the receivers will be output into this NetCDF file. There are several dimensions and variables in this file, but we will discuss only those most pertinent to reading the file.

The 'numReceivers' dimension gives the number of receivers in the file. Note that this number is the total number of components and receivers, so, for example, if you added 100 3C receivers, numReceivers would equal 300.

The following are pertinent variables:
receiverX: (numReceivers) receiver X position
receiverY: (numReceivers) receiver Y position
receiverZ: (numReceivers) receiver Z position

receiverBx: (numReceivers) x-component of receiver, between 0 and 1.
receiverBy: (numReceivers) y-component of receiver.
receiverBz: (numReceivers) z-component of receiver.

Note that a Vx receiver will have receiverBx=1.0 and receiverBy and Bz equal to zero, while a Vz receiver will have receiverBz=1.0 and the others equal to zero. Of course, for pressure receivers, these variables will be equal to zero and are not used.

receiverType: (numReceivers) coded type of receiver. A pressure receiver will have a value of 2 here, whereas other types will have a 1.

receiverData: (numReceivers,NT) a 2-D array containing all of the trace data. Each row is a full timeseries. Output units are in MKS units, so velocities are in m/s and pressures are in Pascals.

4.6 Sources

Sources are added to the command line. Both explosion and arbitrarily oriented force sources are available. First, a source time function must be defined. Ricker wavelet (doubly differentiated gaussian) and delta function (spike) source time functions can be added with command line flags or the user can specify any arbitrary wavelet. A Ricker wavelet is nice for visualization since it is compact in both time and frequency, but it is not a very realistic source. A delta function source makes time slice visualization impossible, but the output is very flexible, since the output of one model run can then be convolved with any number of source time functions, instead of having to run a new model for each source time function.

To add a Ricker wavelet, add the following to the command line:

`-Sr Fpeak`

where *Fpeak* is the peak frequency of the desired Ricker wavelet. Note that the 1% level is about 3 times this peak frequency.

To add a delta function wavelet, add the following to the command line:

`-SD 0`

This adds a spike at time zero (first time sample). The output from a delta function wavelet is useless without convolution with a reasonable source time function. A reasonable source time function is one that has its once- (force) or twice- (explosion) differentiated waveform at 1% of the peak amplitude spectrum at or below the maximum frequency that the model was designed for. Note: make sure that the model dt is multiplied into the convolution to obtain accurate amplitudes.

To add an arbitrary waveform, add the following to the command line:

`-Sf` *`filename.txt`*

filename.txt is a plain text file containing two columns: t and amp. t is the time starting at t0 with samples every dt. amp is the amplitude of the source time function at that time. The amplitudes of the source time function are usually normalized so amp varies between -1 and +1. The length of file should be <=NT. If the file length is < NT, the source time function will be padded with zeros out to NT samples. Just as for any source time function convolved with a delta function, this source time function must be reasonable. In the far field, the source time function will be once-differentiated for a force source and twice-differentiated for an explosion source. These far field wavelets should have their 1% of peak amplitude spectrum at or below the maximum frequency that the model was designed for. Too much higher frequency energy leads to large numerical dispersion and inaccurate results.

In order to test the frequency content of your source time function, the following procedure is recommended. First, prepend and append a few zeros (say 3) to your discrete source time function. This simulates the implied initial and final conditions assumed by Paracousti for source time functions. If an explosion source is used, numerically differentiate this extended source time function twice; if a force source is used, do the differentiation once. Now, look at the Fourier Transform of this signal. Find the **maximum** frequency at which the amplitude spectrum is greater than 1% of the peak of the spectrum.

The default interpretation of any source time function is that it is a time series of force for force sources, or of moment for moment sources. This means that the far field wavelet will be proportional to the single differentiation of a force source waveform or the double differentiation of the moment source waveform. In some cases, however, it may be more convenient to specify the moment rate waveform instead of the moment itself for moment sources. In this case the far field waveform would be the single differentiation of the input moment rate waveform. In order to specify that all following moment source waveforms are moment rate waveforms use the flag:

`-Smr`

It is important to specify this flag before defining any of the source waveforms with `-Sr`, `-SD` or `-Sf` flags to have them interpreted as moment rate waveforms.

Now, the type and location of the source must be specified. To add an explosion source, add the following to the command line:

`-Se` *x y z amp*

This adds an explosion source of amplitude *amp* (N-m) at position x, y, z.

To add a force type source, add the following to the command line:

`-Sfz` *x y z amp*

This adds a vertical force source of amplitude *amp* (N) at position x, y, z. To specify an x-directed or y-directed for source use -Sfx or -Sfy, respectively.

4.7 Attenuation

There is no acoustic attenuation (damping) by default. However, by specifying the following flags, attenuation will be turned on (see Preston, 2016, for details on the implementation of attenuation in TDAAPS, which is like Paracousti, but allows for moving media). The attenuation model is allowed to have 3-D variations, but it is designed to work with a finite (and relatively small) number of unique attenuation models. Internally, an index keeps track of which of the attenuation models a particular grid point belongs to. The index method is also a way one can specify the full 3-D attenuation parameterization, but there is also another method that is based on ranges of sound speed in the 3-D model. For example, you can specify that one attenuation model applies to sound speeds between 1490 and 1500 m/s, and another attenuation model applies to sound speeds above 1500 m/s. This would mean that all nodes between 1490 and 1500 m/s would have the same attenuation model, model1, and all nodes with sound speeds above 1500 m/s would have the same attenuation model, model2. Each attenuation model is specified by a number of attenuation mechanisms, each of which gives the attenuation factor and relaxation frequency. Also, the adjustment factor that adjusts the input sound speeds to the sound speed at infinite frequency is unique to each attenuation model. However, for typical seawater conditions, this factor is very close to 1.0 and can be chosen as 1.0 if desired.

In order to run Paracousti with attenuation, you must convert attenuation loss (1/m = neper/m) as a function of frequency to the attenuation factors and relaxation frequencies for each attenuation model. The loss versus frequency curve may already be available to you, but if not, you can use the Matlab function seawaterAtten.m to compute the loss (1/m) as a function of frequency given temperature, depth, pH, salinity, and frequencies at which you want the loss factor computed. Once you have a loss vs. frequency curve you use the Matlab function acousticAttenSeek.m to compute the

values needed for Paracousti input for each attenuation model. Besides giving it the frequencies and loss function, you also provide the sound speed at infinite frequency, the number of attenuation mechanisms you desire (usually 2 is good), and optionally a reference frequency. The primary output is an array of length 2*number of mechanisms, with adjacent terms being attenuation factor (a) followed by relaxation frequency (w). Thus, for a 2-mechanism attenuation model, the output would be [a1,w1,a2,w2]. The second output (if the reference frequency is input) is the factor that the sound speed at that reference frequency must be multiplied by in order to reach the infinite frequency sound speed. All these parameters are needed in order to specify the attenuation model.

In Paracousti an attenuation model is given by the following flag:

-Q *nR cFac w1 a1 w2 a2 … wnR anR*: *nR* is the number of mechanisms, *cFac* is the factor that the sound speed must be multiplied by to go from the input sound speeds to infinite frequency sound speed. If the input sound speeds are for infinite frequency then *cFac* should be 1.0. *w1…wnR* relaxation frequencies for the *nR* mechanisms. *a1…anR* attenuation factors for the *nR* mechanisms. Note that if all *a1…anR* are 0.0 then it is equivalent to a non-attenuating medium.

-QC *minC maxC*: The preceding attenuation model (-Q flag) applies to nodes whose sound speeds are between *minC* and *maxC*. If either *minC* or *maxC* is '--' (two dashes in a row) then it means there is no limit in that direction. This flag is optional and the last given -Q flag will apply to all nodes not assigned thus far. Therefore, if you want every point in the model to use the same attenuation model, then all you need to give is the -Q flag.

Note that the order of the -Q flags is important and that any -QC flag must be given before a new -Q flag is given. If a variable called ***Qindex*** is found in the netCDF model file, then the first -Q flag will correspond to index 0 nodes, the second to the index 1 nodes and so on. The ***Qindex*** variable is a 3-D array the same size as vp or rho each node with an integer between 0 and nQmodels-1 (where nQmodels is how many -Q flags there will be on the command line).

5. RUNNING PARCOUSTI

Now that the model, receivers and sources are defined we are ready to run the program. This is a parallel code so we must use mpirun, followed by the number of processors (np). Besides those already mentioned there are several command line parameters that must be or can be used.

modelFile.cdf : The model name is provided directly after the executable with no flag preceding it.

-p *px py pz*: (required) this gives the domain decomposition of the model. There will be px processors in the x-direction, py in the y, and pz in the z. The code uses a master-slave node approach to decomposition, so the total number of processors requested on the mpirun is px*py*pz+1 = np. For numerical efficiency it is best if px is as small as possible.

-T *t0:dt:tf* : (optional) redefine the time vector for the simulation in Matlab vector notation

-hc 4 *c0 c1*: (optional) define the 4th order spatial finite-difference coefficients to use instead of the default coefficients. *c0* is the inner coefficient and *c1* is the outer coefficient. Defaults values for these are from the Taylor series expansion coefficients for a 4th order accurate difference and have the values *c0*=9/8 and *c1*=-1/24. For correctly chosen values of these coefficients and time steps the run time for a given model can be greatly reduced for a given accuracy. For example, for a maximum phase speed error of 0.375%, coefficients *c0* = 1.14337598613568, *c1* = -0.0490462530034956 run at 0.5 times the CFL limit provides the minimum run time. With this combination of parameters, dx can then be defined as:

dx = min(Vp)/max(Freq)/gnpw

where gnpw is the number of grid nodes per minimum wavelength, which in this case is 4.46 instead of 10. This allows a much larger dx than defined above in the model section. Note that the CFL limit (maximum time step allowed for stable execution) does depend on these coefficients and dx. The CFL limit is:

dt_{CFL} = dx/max(Vp)/sqrt(3)/sum(abs(c))

where c is [*c0 c1*]. To achieve the desired accuracy and optimal runtime, the dt used in the algorithm should be 0.5*dt_{CFL} in the example stated above. For ease, it is recommended that one use the Matlab function *cflDt()* as described above under ***increments***. In the example given here CFLFraction in the description of *cflDt()* would be 0.5. For other levels of desired accuracy, the Matlab function *optimSpeedTest*.m* can be used to find the coefficients and fraction of the CFL limit (CFLFraction) that minimizes run time.

The following are boundary conditions to damp unwanted reflections from the computational domain boundary and at least one should be given.

-bpc *n R a k* : convolutional PML with a thickness of *n* nodes, with parameters *R*, *a* and *k*. *n* is typically 10; *R* should be 0.001 or less, *a* should be pi*Fpeak (Fpeak is approximately the dominant frequency of the source waveform), *k* should be 1.

The above command applies the same CPML parameters on all 6 sides of the model. Sometimes different CPML zones are desired for each side. This can be accomplished with the following command:

-bpc6 *nXmin RXmin aXmin kXmin nXmax RXmax aXmax kXmax nYmin RYmin aYmin kYmin nYmax RYmax aYmax kYmax nZmin RZmin aZmin kZmin nZmax RZmax aZmax kZmax* : convolutional PML with a thickness of *n* nodes, with parameters *R*, *a* and *k* for each of the sides. All parameters have the same meaning as in the -bpc option.

The convolutional PML does a better job of damping unwanted reflections than the traditional PML, especially if the model is skinny in one dimension compared to other. This is the recommended boundary condition for most cases. If domain boundary reflections are still problematic, especially for very long, skinny models, an MPML can be used.

-bpm *n R a k xfac*: multi-axial PML with a thickness of *n* nodes, with parameters *R*, *a, k,* and *xfac*. *n* is typically 10; *R* should be 0.001 or less, *a* should be pi*Fpeak (Fpeak is approximately the dominant frequency of the source waveform), *k* should be 1, and *xfac*, the cross-factor, should be between 0.01 and 0.05.

Similar to -bpc6, each side of the model can be specified for an MPML using the -bpm6 flag. It has the same form as -bpc6, except with *xfac* added following *kZmax*. Note that this means that *xfac* cannot be varied by side, only the other parameters.

For simple models the traditional PML can also be used:

-bp *n R* : traditional PML with a thickness of *n* nodes and parameter *R*. *n* is typically 10; *R* should be 0.01 or 0.001 in general.

A pressure-free surface is a physical boundary condition that is used to simulate an air-water and/or air-earth interface that is flat. Precisely, it is the physical boundary condition that would occur if a vacuum replaced the air; however, it is a very good approximation for air.

-bF: Use a pressure-free boundary condition for the top (minimum Z) flank of the model. The actual interface is placed at z = zmin+2*dz. Typically for these models, the z-axis is defined such that z = 0 is coincident with the pressure-free surface; thus, zmin would typically be set to -2*dz. The boundary condition is enforced by

forcing pressure at the interface to be zero at all time steps as well as all other conditions implied by this imposition. When the topography and sea surface are flat, this boundary condition provides the most accurate response. It can be used in combination with the CPML boundary, but **not** with the traditional PML boundary.

For visualization purposes, there are a few optional flags that can be used:

-En *N type plane pos* : This will output N snapshots of *type* ground motion on the given *plane* at position *pos* evenly spaced in time. *type* can be “Pressure”, “Vx”, “Vy” or “Vz”. *plane* can be “XY”, “XZ” or “YZ”. So, for example, `-En 51 Pressure XZ 0`, will output 51 snapshots of the pressure field on the XZ plane at y=0. Multiple -En lines are allowed per command line.

-Et: *minT:Dt:maxT type plane pos*: This will output snapshots of *type* ground motion on the given *plane* at position *pos* at the times specified by the Matlab-style vector starting at time *minT*, stopping at time *maxT*, every *Dt* seconds. Note that output will be written from the nearest time-step to the specified times, i.e, there is no temporal interpolation performed. Times outside the max and min simulation time will not be written. Remaining parameters are as in -En.

-Eo *sliceFile.cdf* : output the slices (snapshots) to the NetCDF file *sliceFile.cdf*. All slices are stored in this file, so this file can become very large for big models with many snapshots. Each slice is stored in an appropriately named variable in the file. The variable names are given as ‘*planeType*’, so the variable named ‘xzPressure’ would refer to pressure on the xz plane. These variables are 3-D arrays of dimension (N,planeDim1,planeDim2), where N is the number of slices, planeDim1 is the size of first of the plane dimensions and planeDim2 is the size of the second plane dimension. So, ‘xzPressure’ from the -En example above would have dimension (51,NZ,NX). Note that the ordering of the dimension sizes are the same as for the 3-D geophysical parameters. A second useful variable in the cdf file has the same name at the slice variable above, but with ‘Time’ appended. This variable is of length N and gives the time at which the snapshot was taken.

-Ef *maxPointsPerSliceFile*: Set the maximum number of points per variable name that can be written to a single file. The number of points is the size of the plane times the number of slices times the number of positions for that variable. Multiple slice output files will be created if the total number of points exceeds *maxPointsPerSliceFile*. It will alter the filename given by the -Eo flag by appending “_#” just prior to the “.cdf” ending, where # starts at 0 and is incremented until all slice variable points are in files with less than or equal to *maxPointsPerSliceFile*. For example, given the -En line above, the variable is ‘xzPressure’ and if that was the only ‘xzPressure’ -En option given then there would be only 1 position, at y=0. The total points for ‘xzPressure’ would be computed as NX*NZ*51*1 (where 51=N from the -En example, and 1 is the number of positions). If this total points exceeds *maxPointsPerSliceFile* then the slice output will be divided up among multiple output slice files, none having its number of points exceeding *maxPointsPerSliceFile*. The

default is 1000000000 (one billion) and *maxPointsPerSliceFile* should not exceed this number due to netCDF variable size restrictions.

6. EXAMPLES

So, putting it all together, an example acoustic run would be:

```
mpirun -np 7 ParAcousti baseline.cdf -p 1 2 3 -T 0:0.00019:0.1 -
bp 10 .01 -Sr 50 -Sfz 0 0 0 1 -Rg 4C -40:5:40 0:0 10:10 -Ro
baselineAc.trace.cdf
```

This call starts 7 processes, with the domain decomposition: 1 processor for the x dimension, 2 processors in the y dimension and 3 processors in the z dimension. Use the model baseline.cdf for the medium parameters. The timing given in this file is overridden so that the simulation time starts at 0 and goes to 0.1 seconds at a time step of 0.00019 s. A PML boundary 10 nodes wide with a theoretical reflection coefficient of 0.01 will be applied to all 6 sides of the model. The source waveform is set to a 50 Hz Ricker wavelet and a vertically down-directed force source will be applied at the model point (0,0,0) with an amplitude of 1 N. A receiver grid of 4C receivers will be placed on a line from x = -40 m to 40 m in 5 m increments at y = 0 m and z = 10 m. Traces will be output into the file baselineAc.trace.cdf.

A useful tool for finding out how a certain trace or slice file was created is to use ncdump. This is a utility program provided as part of the standard NetCDF C/C++/ fortran distribution. Using the call:

```
ncdump -h filename.cdf
```

This command will print out the dimensions, variables and attributes of the file. For trace and slice files, there will be an attribute called “history” followed by the command line call that created the file.

7. CONCLUSIONS

This brief report outlines the processes needed to use the 3-D massively parallel acoustic simulation code Paracousti. This code has the ability to perform 3-D full waveform acoustic simulations in solid, fluid, and (ideal) gaseous media with support for accurate high contrast interfaces between varying media types, including realistic topography, bathymetry, and subterranean voids. Although ideal, fixed fluid and gaseous media are assumed, it can incorporate attenuative losses that would be expected from physical mechanisms such as molecular dissipation.

8. REFERENCES

1. Berenger, J-P., A Perfectly Matched Layer for the Absorption of Electromagnetic Waves, *J. Comp. Phys.*, 114, 185-200, 1994.
2. Cerjan, C., D. Kosloff, R. Kosloff, and M. Reshef, A Non-Reflecting Boundary Condition for Discrete Elastic and Acoustic Wave Equations, *Geophys.*, 50 (4), 705-708, 1985.
3. Komatitsch, D., and R. Martin, An Unsplit Convolutional Perfectly Matched Layer Improved at Grazing Incidence for the Seismic Wave Equation, *Geophys.*, 72 (5), SM155-SM167, doi 10.1190/1.2757586, 2007.
4. Meza-Fajardo, K.C., and A.S. Papageorgiou, A Nonconvolutional, Split-Field, Perfectly Matched Layer for Wave Propagation in Isotropic and Anisotropic Elastic Media: Stability Analysis, *Bull. Seis. Soc. Am.*, 98 (4), 1811-1836, doi: 10.1785/0120070223, 2008.
5. Preston, L.A., TDAAPS 2: Acoustic Wave Propagation in Attenuative Moving Media, SAND2016-7859, Sandia National Laboratories, Albuquerque, NM, August 2016.
6. Preston, L.A., D.F. Aldridge, N.P. Symons, Finite-Difference Modeling of 3D Seismic Wave Propagation in High-Contrast Media, *Soc. Expl. Geophys. 2008 Annual Meeting Extended Abstracts*, 2008.
7. Symons, N.P., D.F. Aldridge, D.H. Marlin, S.L. Collier, D.K. Wilson, V.E. Ostashev, *Modeling with the Time-Domain Atmospheric Acoustic Propagation Suite (TDAAPS)*, SAND2006-2540, Sandia National Laboratories, Albuquerque, NM, May 2006.

DISTRIBUTION

1	MS0750	Leiph Preston	6911
1	MS1124	Jesse Roberts	6122
1	MS0899	Technical Library	9536 (electronic copy)


Sandia National Laboratories