Paracousti User Manual

December 2018

CONTACTS

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

Erin Hafla
Montana State University
P.O. Box 173800
Bozeman, Montana 59717-3800

Erick Johnson
Mechanical Engineering
Montana State University
P.O. Box 173800
Bozeman, Montana 59717-3800

ACKNOWLEDGMENTS

This work was sponsored by Sandia National Laboratories Water Power Technology department and by the Department of Energies Wind and Water Power Technologies Office. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract No. DE- NA0003525.

CONTENTS

1. INTRODUCTION 7

2. INSTALLATION 8

2.1. SOFTWARE ENVIRONMENT AND SETUP 8
2.2. PRE-COMPILED EXECUTABLE 8
2.3. COMPILING THE EXECUTABLE 8

3. CREATING A PARACOUSTI SIMULATION 9

3.1. THE EARTH MODEL 11
3.2. BOUNDARY CONDITIONS 14
3.3. SOUND SOURCES 15
3.4. RECEIVER DATA 17
3.4.1. TRACE – TIME HISTORY AT A SINGLE OR ARRAY OF POINTS 17
3.4.2. SLICE – A PLANAR SNAPSHOT AT AN INSTANT IN TIME 19
3.4.3. VOLUMETRIC – A SNAPSHOT OF THE ENTIRE WAVEFIELD AT AN INSTANT IN TIME 20
3.5. ATTENUATION 20

4. RUNNING PARACOUSTI 23

4.1. ADDITIONAL COMMAND-LINE PARAMETERS 23

5. POST-PROCESSING RESULTS 25

6. EXAMPLES 26

6.1. EXAMPLE 1 26
6.2. EXAMPLE 2 26

7. THEORY 28

7.1. SOLUTION EQUATIONS 28
7.2. SOLUTION METHODOLOGY 29

8. CONCLUSIONS 34

9. REFERENCES 35

10. APPENDIX I: LIST OF COMMANDS 36

10.1. COMMANDS: 36

LIST OF FIGURES

Figure 1: General depiction of the acoustic wave propagation modeling process. 9

Figure 2: The computational domain (large box) represented by a 3-D uniformly spaced grid with nodes (grid points) indicated by black dots. 29

Figure 3: Arrangement of dependent variables and medium parameters for one cell of the standard staggered grid. 30

NOMENCLATURE

ABC	Absorbing Boundary Condition
CPML	Convolution Perfectly Matched Layer
DOE	Department of Energy
MHK	Marine Hydrokinetic
MPML	Multi-axial Perfectly Matched Layer
PML	Perfectly Matched Layer
SNL	Sandia National Laboratories
TDAAPS	Time Domain Atmospheric Acoustic Propagation Suite

1. Introduction

Paracousti is a parallelized acoustic wave propagation simulation package developed at Sandia National Laboratories and Montana State University. It solves the linearized coupled set of acousto-dynamic partial differential equations, a pair of velocity and pressure 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 due to changes in topography or bathymetry and acoustic sound speeds, including voids in the subsurface. A combined discretization of both fluid and solid domains enable varying properties in all spatial coordinates to be used, as measurements or simulation data becomes available. Sound-source profiles are evaluated as moment tensors, allowing combinations of monopole and higher-order forces. Although it assumes ideal fluid media, it can also simulate sound 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 was created as an alternative solution to the Helmholtz and wave equation methods traditionally employed to model and predict the sound propagation generated from a single or array of Marine hydrokinetic (MHK) devices in the absence of measured sound levels. MHK devices generate electricity from the motion of tidal and ocean currents as well as ocean waves to provide an additional source of renewable energy available to the United States. These devices are a source of anthropogenic noise in the marine ecosystem and must meet regulatory guidelines and address stakeholder concerns.

Example simulations and code presented in this manual are derived from this initial purpose. As Paracousti may be applied to a wide variety of environments, the applications are far more widespread. All examples in this report are presently written for MATLAB.

2. Installation

2.1. 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 and may be downloaded online.

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.

Paracousti should run on most Linux distributions. Paracousti_Linux has been built in Red Hat and is routinely tested and used on Debian, under Ubuntu. All support tutorials and input files have been written in MATLAB.

2.2. Pre-compiled executable

The current pre-compiled executable of ParAcousti_Linux was compiled with gcc 4.7.1, mpi 1.8.1, and netCDF 4.4.1.1 on 64-bit Red Hat Enterprise Linux Workstation release 6.8. This executable requires dynamically linked openmpi and netCDF libraries.

2.3. Compiling the executable

The source-code is available at the GitHub repository. A makefile is available to build Paracousti under a Linux distribution of your choice. Further details to be appended shortly.

3. Creating a Paracousti Simulation

A three-step modeling process for simulating acoustic wave propagation is depicted in general terms in Figure 1 The numerical simulation algorithm (center box) accepts various inputs (left column) and then calculates and outputs various data types (right column). Inputs into Paracousti are: 1) the "earth model", consisting of a gridded representation of the medium parameters defined on a 3-D rectangular grid, 2) a description of the recording geometry (i.e., types and positions of acoustic energy sources), and 3) the desired outputs. The size of the domain depends on the region(s) of interest and can encompass water, earth, and air domains, but as shear waves are not included; each domain is treated as a fluid. 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.

A Paracousti simulation is defined and executed through an earth model and a series of command-line flags. The earth model defines the spatial model boundaries, Cartesian grid used to solve the model, and environmental properties associated with each location within the grid. All inputs associated with the earth model are separate from any flags that define how the problem should be solved (boundary conditions for example) and must be put together in a

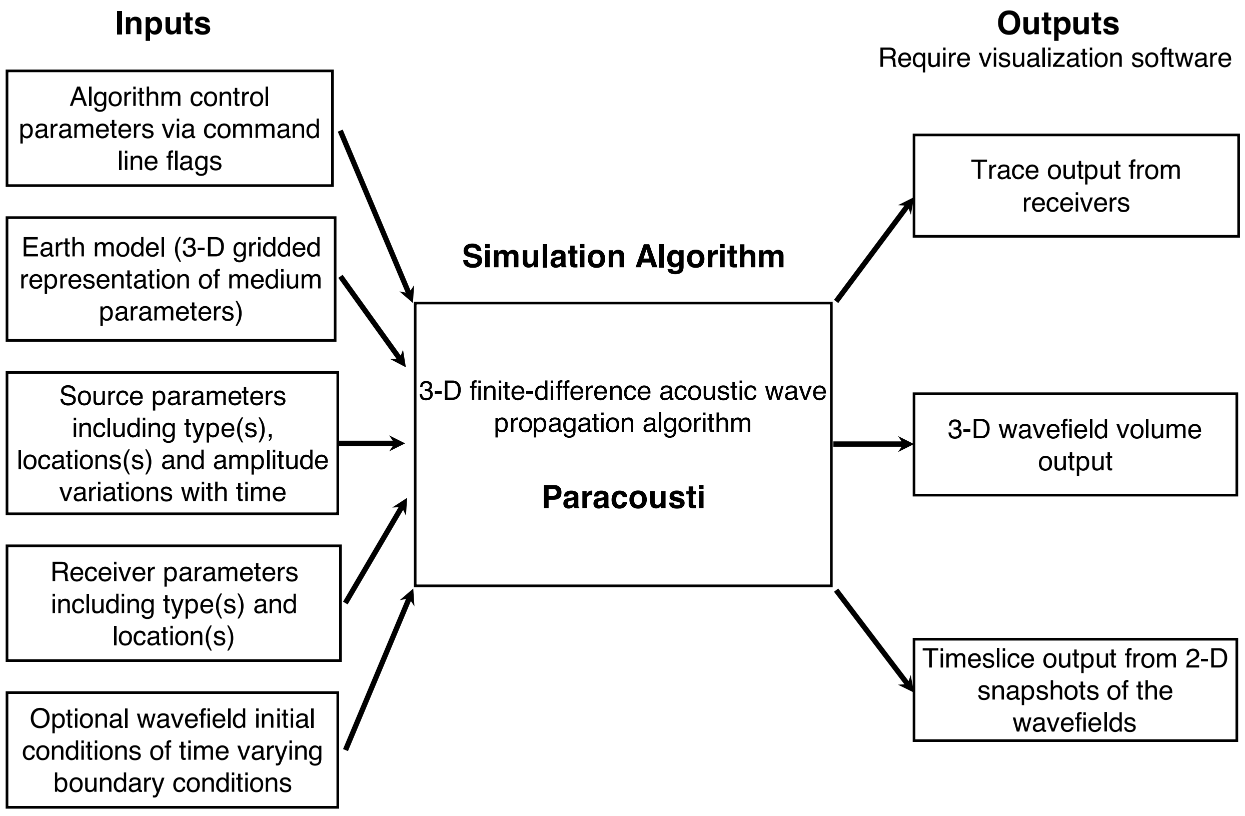
Inputs
Algorithm control parameters via command line flags
Earth model (3-D gridded representation of medium parameters)
Source parameters including type(s), locations(s) and amplitude variations with time
Receiver parameters including type(s) and location(s)
Optional wavefield initial conditions of time varying boundary conditions
Simulation Algorithm
3-D finite-difference acoustic wave propagation algorithm
Paracousti
Outputs
Require visualization software
Trace output from receivers
3-D wavefield volume output
Timeslice output from 2-D snapshots of the wavefields

***Figure 1**: General depiction of the acoustic wave propagation modeling process.*

program external to Paracousti, which runs only as an executable. The following are the properties required for constructing the earth model:

- Bathymetry: Measurement of the depth of water to the bottom of the body of water. It is required to define the individual parameter values for every grid point in the model domain as well as the overall size of the model domain
- Density: Medium mass density as a function of space over the entire 3-D model domain; individual values for every grid point
- Acoustic Sound Speed: Medium sound speed as a function of space over the entire 3-D model domain; individual values for every grid point

Once the model domain is defined along with any properties associated with every grid point location, command line flags are further used to describe any further boundary conditions, attenuative properties, energy sources, how Paracousti should accomplish the solution process, and required output. Data associated with flags other than those defined for output are defined below:

- Boundary conditions: All edges of the model domain must be constrained or described by a boundary condition. The available boundary conditions are outlined as:
    - Pressure-Free Surface: The air portion of the air-water interface on the top side of the domain is approximated as a vacuum. If this boundary condition is not used, the model assumes an infinite water depth above the model domain.
    - Convolutional Perfectly Matched Layers & Perfectly Matched Layers: These boundaries require a buffer zone around on any domain boundary that they are applied to that is used to dampen any unwanted reflections from reentering the model domain and negatively affecting the simulation solution. CPMLs and PMLs have slightly different requirements and applications dependent on the dimensions of the model domain.
- Attenuation: Additional energy dissipation may be added to the system through an attenuation factor. Paracousti assumes that the ambient medium is ideal and attenuation values are assumed to be zero unless otherwise stated. They may be applied individually to each grid point or for a range of density values.
- Sound source Information: Sources are required to provide an energy source which can then propagate through the model domain. Options for these sources are:
    - 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. Receiver locations are required to define any particular location that a type of data should be located. It is worth noting that the type of data required at the solution completion must be specified prior to running the model. Not all output types require a specific collection point required by receivers. The three general forms of output data are as follows:
    - 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. The specified location of the trace must align with the underlying Cartesian grid assigned

for the model. This is the only type of output data that requires a specific collection location.

- 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 assigned for the model. The type of output (e.g., pressure, etc.), the orientation of the plane (XY, XZ or YZ), the location of this plane, and either the time between snap shots or the total number of snap shots over a total run time is required.
- 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 or total number of 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. Several slice outputs over different orientations of the plane for a 3D run may provide similar data with a reduced output file size.

While the earth model is the only required file, more complex sound source(s), receiver locations, and a matrix used to define attenuative parameters and associated grid locations may be defined through individual text file(s) as well. The application of attenuation in the model may use a combination of program scripts and command flags or only command flags for any given model.

Note that the executable itself does not have the capability to build any models. This must be completed separately using a program like MATLAB or Python. Example command line arguments are written with respect to using MATLAB as the program required to compile the input files. Any computer program which can output netCDF files, such as Python, may be used for all required pre and post processing of files for Paracousti.

3.1. The Earth 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. The overall number of files required is driven by the size of the domain with respect to the number of grid cells necessary.

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. The values of x, y, z, and t are the initial parameters for any input file and determine the necessary location and value for every other input requirement to follow. For any file, x, y, and z define the boundaries of the model domain while values of sound speed and density must be assigned to every grid point. Matrices in you preferred program are suggested. 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. In addition to building the model or model input files externally, the following call may be used for large 3D models within program:

```
writeSubdomainSgfdModel(modelName,x,y,z,t,[domains],'vp','func',
vp,{},'rho','func',rho,{});
```

writeSubdomainSfgdModel allows the user to call out how many netCDF files the domain input file should be spread out upon and where to break the domain to reduce any memory problems associated with a large model domain. In this particular call out, x, y, z, and t are defined as stated above. Domains is a matrix defining the domain location of the interface between the number of domains required. One row of data for each domain. This time vp and rho are still a 3-D array of vp or rho values, but each is defined within a separate function file. This call may also be used in a similar fashion as writeSfgdModel for 2D files if large enough to warrant its use.

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). Upon setting up a program file to build the input files for Paracousti, setting up the grid matrix is the first step.

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. Values should extend beyond the modeling area of interest by a minimum of five wavelengths dependent on the value of your maximum source frequency to comply with CPML boundary conditions. Minima may be negative counters with respect to the rest of the domain and may need to be increased to reduce interference in the model for domains with a greater amount of interactions.

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. To obtain the desired level of accuracy, a minimum of 10 grid points per desired source-wavelength are required. The 10 accounts for the number of grid points required. 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. However, it is worth noting that if memory is not a problem for your file size, a smaller dt than the calculated minima is recommended.

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. All geophysical parameters must be defined on the same domain grid, step size included, as defined by the vectors of x, y, and z axis values.

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. Velocities may be individually calculated based upon environmental conditions and location (temperature, salinity, suspended particulate count) and input into the array.

Rho: (NZ,NY,NX; float) a 3-D array of the densities in kg/m^3. Again, densities may be individually calculated based upon environmental conditions and location within the domain.

3.2. Boundary Conditions

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

-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, where Fpeak is approximately the dominant frequency of the source waveform, and *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 except specified for the location in the simulation domain.

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, thin models, an MPML can be used with the indicated command shown below.

-bpm *n R a k xfac* : multi-axial PML with a thickness of *n* nodes, and parameters *R*, *a, k,* and *xfac*. *N* is typically 10; *R* should be 0.001 or less, *a* should be pi*Fpeak where 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*.

-bpm6 *nXmin Rxmin aXmin kXmin nXmax Rxmax aXmax kXmax nYmin Rymin aYmin kYmin nYmax Rymax aYmax kYmax nZmin Rzmin aZmin kZmin nZmax Rzmax aZmax kZmax xfac* : multi-axial PML with a thickness of *n* nodes, with parameters *R*, *a* and *k* for each of the sides. Note that this means that *xfac* cannot be varied by side; only the other parameters may be varied.

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 and may be applied to the top of the model domain to approximate the air-water boundary.

-bF : 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.

3.3. Sound 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) and delta function (impulse) 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. A source function of some type is required to induce any type of perturbations within Paracousti to allow a waver function to propagate down range. As Paracousti assumes that the background medium is not moving, any ambient noise recorded from an environment would need to be modeled separately and overlaid with the sound generated from a specific source or removed from any recorded data in order to compare modeled sound propagation to experimental data.

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 three times this peak frequency.

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

-SD *0*

This adds an impulse at time zero (the 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. The trace output from a singular, unformatted delta function may be convolved after the simulation run, but not the slice data as it is taken in a point in time for a range of locations as compared to a singular location over the entire simulation time.

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

-Sw *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. For example, the source for an explosion source should be equal to the pressure twice integrated. 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. An example explosion source is provided below:

```
A = c^2/(pi()*sf^2);
amp(1:length(t)) = A*(1-cos(2*pi()*sf*t));
```

were c is the minimum sound speed for the system, sf is the sound source frequency, and t is the time vector. Note that the waveform has an amplitude of 1 Pa. The specific amplitude is specified again when the input files are written to a netCDF file, so that value may be changed to reflect any increased amplitude. It may also be left as one if the source file (text file) already includes the correct value of amplitude.

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 instance, 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.

Once a source type or function is indicated, 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. It is at this location that you may include the amplitude of the source function or leave it at one.

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. Note that the command –Sf filename.txt will still run initially thinking that the system is using a force source, but will inevitably cause the run to fail as a force source is not associated with a direct text file input.

3.4. Receiver Data

Receivers are designated locations where data will be recorded at a particular point in the model domain. They are only required for Trace type output and may be thought of as a sensor location for data collection. However, the model will run even if receivers are called and there is not a file specified to output any data. Slice output records all data for a model domain at a point in time, not a location.

3.4.1. Trace – time history at a single or array of points

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. Receiver location indicate where any trace data will be collected during the model run.

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 located on the predefined domain grid. If receiver locations lie between grid points, a cubic interpolation is by default applied to the calculated data. 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. Also note that the grid may be manipulated to have a very low number of receivers as necessary if more data was required for a particular area after an initial run. However, the grid must contain more than a singular receiver to run.

As a general note on the location of receivers, receivers may not be placed within an area in the simulation domain required to fulfil the requirements of a boundary condition. That would include any additional grid points required for a CPML, PML, pressure-free surface, etc. See Section 3.2 for a further description of the boundary conditions required for Paracousti. For that reason, it is recommended that the user start the indexing for the domain size at zero with indexing tied to any additional cells tied to boundary conditions increasing negatively on the left hand side of your domain; although this is not required.

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. Without a designated output file, Paracousti will not save and output the calculated data at each designated receiver location. Trace files are set up to record outputs from a singular designated receiver command. A separate run file will be required if output from two sets of receiver output geometries are required. Additionally, if two receiver geometries are specified with a singular output file, the second receiver geometry will overwrite the first in the output 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.

3.4.2. Slice – a planar snapshot at an instant in time

As slices do not require a spatial designation, several commands exist to define whether a set number of slices is required or which particular times throughout the simulation run data would like to be required. Again, the type of data output must still be specified.

-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", indicating particle velocities in each of the three indicated directions. *Plane* can be "XY", "XZ" or "YZ". So, for example: -En 51 Pressure XZ 0, will output 51 snapshots of the pressure field in time over the total run time of the simulation 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 a 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. Additionally, times must overlap with the original time vector and must be a multiple of the original designated time step written for the domain input files. 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 may become very large for big models with many snapshots. However, additional output files will be compiled and saved in order to collect all data required. 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 as the slice variable above, but with ‘Time’ appended. This variable is of length N and gives the time at which the snapshot was taken. All variable information may be ascertained using a MATLAB function called *ncinfo*.

-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’. If that was the only ‘xzPressure’, the –En option given then 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 variable size restrictions. The variable size restrictions result in a maximum file size that is approximately 3.9 Gigabytes.

3.4.3. Volumetric – a snapshot of the entire wavefield at an instant in time

The entire wavefield can be captured at a single or multiple time steps. While this would allow the user to capture nearly any detail they are looking for, the data requirements are likely to be extreme. We do not recommend this as a standard output solution. Further details to be appended shortly.

-*W minT:Dt:maxT type*: this will output snapshots of *type* for the full wavefield at the times specified by a 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. Additionally, times must overlap with the original time vector and must be a multiple of the original designated time step written for the domain input files.

3.5. Attenuation

There is no acoustic attenuation (damping) by default. However, by specifying the following flags, attenuation will be turned on. While Section 7 describes the finite difference equations required to solve the velocity-pressure equations for Paracousti (Preston, 2016) for a full description on the implementation of attenuation in TDAAPS. TDAAPS is similar to Paracousti, but allows for moving media and the equations may be simplified for a stationary 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 a specified 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 be available, but if not, the MATLAB function seawaterAtten.m may be used to compute the loss (1/m) as a function of frequency given temperature, depth, pH, salinity, and frequencies specified at which the loss factor should be computed. Once a loss vs. frequency curve is obtained, the MATLAB function acousticAttenSeek.m may be used to compute the values needed for Paracousti input for each attenuation model. Besides providing the frequencies and loss function, the sound speed at infinite frequency, the number of attenuation mechanisms desired (usually 2 is good), and optionally a reference frequency need to be described. 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]. Mechanisms are generally applied to any change in density with a two-mechanism system referencing the values required for the water column and sediment layer in the overall simulation. 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 required 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* are the relaxation frequencies for the *nR* mechanisms. *A1…anR* are the 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. Each integer value correlates with a –Q flag indicating the attenuation factor and relaxation frequency in order to assign a particular attenuation to each grid point in the domain. Note that the –Q flag only indicates that attenuation is being applied to a model as specified but not at what location a particular attenuative value is applied; the Qindex matrix is required for spatial data.

4. Running Paracousti

Now that the model geometry, receivers and sources are defined we are ready to run the program. This is a parallel code so we must use the command mpirun to tie in with the program openmpi, followed by the number of processors (-np) and the call indicating you would like to run the Paracousti executable. This is dependent on the version or location of the executable.

4.1. Additional Command-Line Parameters

Besides those already mentioned there are several command line parameters that are necessary or can be used. Each of the following commands, if selected for a particular simulation, are written together in a continuous line of code written directly on the command line prompt or in a batch file that may be run on a command line. A batch file (text file) is recommended as it may be saved with any other files created for a particular run and used to reproduce data at a later date as well as the fact that it simplifies the code required to actively start running Paracousti.

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

-p *px py pz* : (required) this gives the domain processor 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. Note that the number of processors that may be allocated include those available due to hyper-threading on a particular computer.

-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 be used 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(absI)

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 this particular example, 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. Note that Paracousti will run using the original time and spatial vector coefficients defined in the input files; it may just not be the most efficient run time available.

This is followed by all the additional commands defining the boundary conditions, and input and output parameters. These can be specified in any order and multiple sound sources and output receiver types can be used.

5. Post-Processing Results

Details to be appended shortly. Tutorials 1 and 3 provide some beginning and advanced details on accessing data and post-processing results from the output netCDF files.

6. Examples

6.1. Example 1

Considering all previous material discussed, an example acoustic run would be:

```
mpir-n -np 7 ParAcousti baseline.c-f -p 1 2-3 -T 0:0.00019:0-1 -
bp 10 .-1 -Sr -0 -Sfz 0 0 0-1 -Rg 4C -40:5:40 0:0 10:-0 -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 that were already written to an input file while calling ParAcousti as the executable. 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:

```
ncdu-p -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.

6.2. Example 2

A second example for an acoustic run would be:

```
mpir-n -np 13 ParAcousti baseline.c-f -p 1 3-4 --F -bpc6 10 1e-6
314 1 10 1e-6 314 1 10 1e-6 314 1 10 1e-6 314 1 2 1 314 1 10 1e-
6 314-1 -Sw source.t-t -Se 100 0 2-1 -Rg Pressure 100:10:4000
0:0 0:1:-0 -Ro baselineAc.trace.c-f -En 1000 Pressure XZ-0 -Eo
baselineAc.slice.cdf
```

This call starts 13 processes, with the domain decomposition: 1 processor for the x dimension, 3 processors in the y dimension and 4 processors in the z dimension. Use the model baseline.cdf for the medium parameters that were already written to an input file while calling ParAcousti as the executable. A pressure-free boundary is applied to the top of the domain and a convolutional PML boundary is applied to all six sides of the simulation domain as well. The thickness is equal to 10 nodes in the x, y, and zmax dimensions with 2 in the zmin dimension. The R parameter is set to 1e-6 in the x,y, and zmax dimensions with 1 in the zmin dimension. a and k are set to 314, which corresponds to a frequency of 100 Hz, and 1, respectively. The source waveform is set as the textfile, source.txt, and an explosive source will be applied at the model point (100,0,2) with an amplitude of 1 N. A receiver grid of 4C receivers will be placed on a line from x = 100 m to 4000 m in 10 m increments at y = 0 m and z = 0m to 40 m in 1 m increments. Traces will be output into the file baselineAc.trace.cdf. Over the simulation time designated in the input file, 1000 pressure snapshots in time will be taken in the XZ plane where y = 0. The snapshots or slices will be output into the file baselineAc.slice.cdf where the standard maximum number of data points per file are used.

7. Theory

7.1. Solution Equations

A set of coupled first order linear partial differential equation known as the velocity-pressure system were derived from a linearization of continuity, Cauchy's equations of motion, a constitutive relationship for stress, and a balance of entropy (Hafla, 2018):

$$\frac{\delta v^*}{\delta t} + \frac{1}{\rho^\circ}\nabla p^* = \frac{1}{\rho^\circ}\left[\boldsymbol{F} + \nabla \boldsymbol{m}^{dev}\right] \tag{1a}$$

$$\frac{\delta p^*}{\delta t} + \rho^\circ (c^\circ)^2 \nabla \cdot v^* = -\frac{1}{3}\frac{\delta \boldsymbol{m}^{iso}}{\delta t} \tag{1b}$$

where $v(\boldsymbol{x}, t)$ and $p(\boldsymbol{x}, t)$ are the dependent variables of particle velocity and pressure perturbations, respectively; $\rho(\boldsymbol{x})$ is material density and c is sound speed. Either a * or ° indicate whether the term is caused by a perturbation or related to the ambient conditions, respectively. The right hand terms in these equations are body source terms: $\boldsymbol{F}$ is the force density vector, $\boldsymbol{m}^{dev}$ is the anti-symmetric portion of the moment density tensor and $\boldsymbol{m}^{iso}$ is the trace of the symmetric portion of the moment density tensor. The anti-symmetric and symmetric portions of the moment density tensors may also be considered as the deviatoric and isotropic portions, respectively. 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, which represents a source of pressure and the hydrostatic portion of the moment density tensor, 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.

For this solution, the fluid is considered inviscid and adiabatic with the background fluid having a particle velocity of zero. It is further assumed that the background medium is stationary and that all background medium properties are constant in time until a sound perturbation is introduced into the system. Density perturbations caused by a sound wave are preserved through the derivation allowing the sound wave to propagate through a system even though the ambient fluid is considered incompressible for the velocity-pressure 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. More simplistic models may also be developed for testing and evaluation purposes. A more detailed derivation can be found in (Aldridge, 2005; Hafla, 2018).

7.2. Solution Methodology

A finite difference scheme is utilized in order to solve the velocity-pressure system of Equation 1. The numerical modeling domain is defined by a uniform Cartesian grid of points such as shown in Figure 2. Figure 3 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, where the time step remains constant. 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.

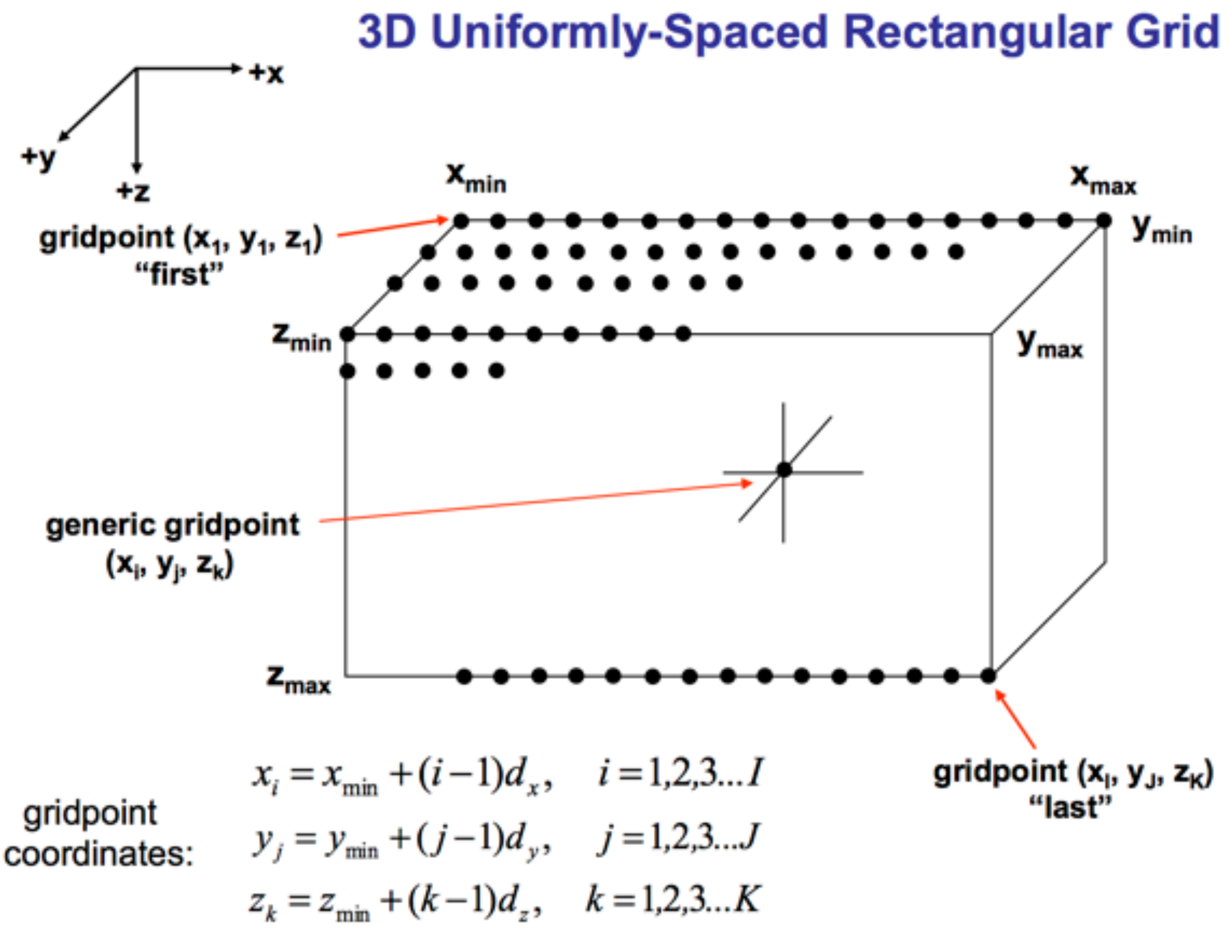
3D Uniformly-Spaced Rectangular Grid
+x
+y
+z
x_{min}
x_{max}
y_{min}
y_{max}
z_{min}
z_{max}
gridpoint (x_1, y_1, z_1) "first"
generic gridpoint (x_i, y_j, z_k)
gridpoint (x_I, y_J, z_K) "last"
gridpoint coordinates:
$x_i = x_{\min} + (i-1)d_x, \quad i = 1,2,3...I$
$y_j = y_{\min} + (j-1)d_y, \quad j = 1,2,3...J$
$z_k = z_{\min} + (k-1)d_z, \quad k = 1,2,3...K$

Figure 2*: The computational domain (large box) represented by a 3-D uniformly spaced grid with nodes (grid points) indicated by black dots.*

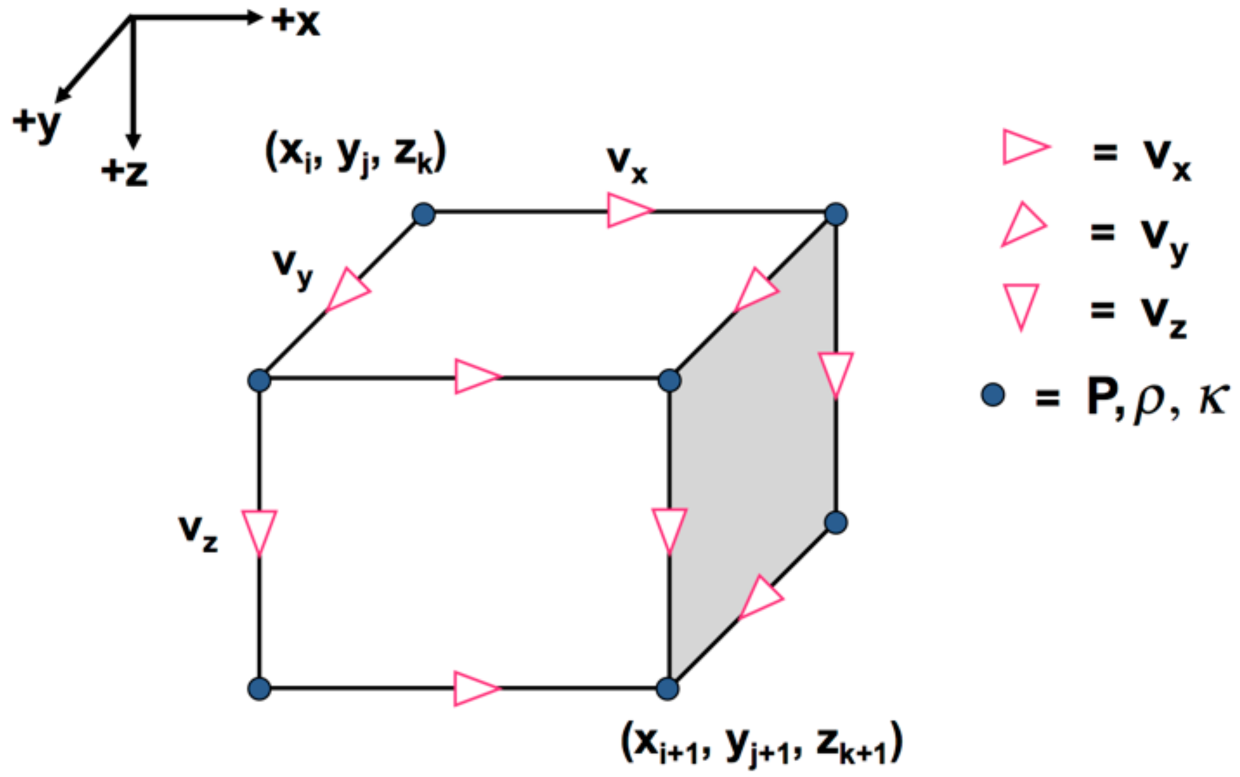

***Figure 3**: Arrangement of dependent variables and medium parameters for one cell of the standard staggered grid.*

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 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. The staggered grid and solution-step removes spurious oscillations within the solution. Due to the nature of a finite grid, the water-sediment layer interface of the domain is not continuously smooth, but rather steps at grid points. However, the grid size is dictated by the highest contributing frequency and so the interactions at the interface are well captured. The discretized equations for the finite difference solution with a full derivation are provided in (Preston, 2016); contributions from a moving ambient filed are included. For Paracousti, equations were simplified to omit background velocities, as they are significantly smaller than the speed of sound in underwater systems. The finite difference equations are shown below.

$$\widehat{v_x}\left(x_i + \frac{d_x}{2}, y_j, z_k, t_l + \frac{d_t}{2}\right) = \widehat{v_x}\left(x_i + \frac{d_x}{2}, y_j, z_k, t_l - \frac{3d_t}{2}\right) \quad (2a)$$

$$
\begin{gathered}
-\frac{1}{\hat{\rho}\left(x_i+\frac{d_x}{2},y_l,z_k\right)}\{p_x[\hat{P}(x_i+d_x,y_j,z_k,t_l)+\hat{P}(x_i+d_x,y_j,z_k,t_l-d_t) \\
-\hat{P}(x_i,y_j,z_k,t_l)-\hat{P}(x_i,y_j,z_k,t_l-d_t)]+q_x[\hat{P}(x_i+2d_x,y_j,z_k,t_l) \\
+\hat{P}(x_i+2d_x,y_j,z_k,t_l-d_t)-\hat{P}(x_i-d_x,y_j,z_k,t_l)-\hat{P}(x_i-d_x,y_j,z_k,t_l-d_t)]\} \\
+\frac{1}{\hat{\rho}(x_i+\frac{d_x}{2},y_l,z_k)}\widehat{f_x}(x_i+\frac{d_x}{2},y_j,z_k,t_l-\frac{d_t}{2})
\end{gathered}
$$

$$
\begin{gathered}
\widehat{v_y}\left(x_i,y_j+\frac{d_y}{2},z_k,t_l+\frac{d_t}{2}\right)=\widehat{v_y}\left(x_i,y_j+\frac{d_y}{2},z_k,t_l-\frac{3d_t}{2}\right) \\
-\frac{1}{\hat{\rho}\left(x_i,y_l+\frac{d_y}{2},z_k\right)}\{p_x[\hat{P}(x_i,y_j+d_y,z_k,t_l)+\hat{P}(x_i,y_j+d_y,z_k,t_l-d_t) \\
-\hat{P}(x_i,y_j,z_k,t_l)-\hat{P}(x_i,y_j,z_k,t_l-d_t)]+q_x[\hat{P}(x_i,y_j+2d_y,z_k,t_l) \\
+\hat{P}(x_i,y_j+2d_y,z_k,t_l-d_t)-\hat{P}(x_i,y_j-d_y,z_k,t_l)-\hat{P}(x_i,y_j-d_y,z_k,t_l-d_t)]\} \\
+\frac{1}{\hat{\rho}(x_i,y_l+\frac{d_y}{2},z_k)}\widehat{f_y}(x_i,y_j+\frac{d_y}{2},z_k,t_l-\frac{d_t}{2})
\end{gathered} \tag{2b}
$$

$$
\begin{gathered}
\widehat{v_z}\left(x_i,y_j,z_k+\frac{d_z}{2},t_l+\frac{d_t}{2}\right)=\widehat{v_z}\left(x_i,y_j,z_k+\frac{d_z}{2},t_l-\frac{3d_t}{2}\right) \\
-\frac{1}{\hat{\rho}\left(x_i,y_l,z_k+\frac{d_z}{2}\right)}\{p_x[\hat{P}(x_i,y_j,z_k+d_z,t_l)+\hat{P}(x_i,y_j,z_k+d_z,t_l-d_t) \\
-\hat{P}(x_i,y_j,z_k,t_l)-\hat{P}(x_i,y_j,z_k,t_l-d_t)]+q_x[\hat{P}(x_i,y_j,z_k+2d_z,t_l) \\
+\hat{P}(x_i,y_j,z_k+2d_z,t_l-d_t)-\hat{P}(x_i,y_j,z_k-d_z,t_l)-\hat{P}(x_i,y_j,z_k-d_z,t_l-d_t)]\} \\
+\frac{1}{\hat{\rho}(x_i,y_l,z_k+\frac{d_z}{2})}\widehat{f_z}(x_i,y_j,z_k+\frac{d_z}{2},t_l-\frac{d_t}{2})
\end{gathered} \tag{2c}
$$

Equations 2a – 2c represent the x, y, and z components of the particle velocity on a half step as defined previously with steps of i, j, k, and l correlating to x, y, z, and time, t, respectively. v is the particle velocity, P the perturbation pressure, ρ the density, x, y, and z are the spatial locations, d the grid spacing, f is a force source vector, and both p and q acting as memory variable containing staggered fourth-order accurate non-dimensional finite-difference coefficients. The pressure equation (4) is as follows along with required expansions (3a – 3c) for equations (2). Full descriptions of the force source vector and memory variables are provided in (Preston, 2016; Ostashev, 2005). With respect to the pressure equation, κ is the bulk modulus, ω the relaxation frequency, R and r are memory variables, and e is the energy density source.

$$
+\frac{1}{\hat{\rho}(x_i+\frac{d_x}{2},y_l,z_k)}\widehat{f_x}(x_i+\frac{d_x}{2},y_j,z_k,t_l-\frac{d_t}{2}) \tag{3a}
$$

$$+\frac{1}{\hat{\rho}(x_i, y_l + \frac{d_y}{2}, z_k)} \hat{f}_y(x_i, y_j + \frac{d_y}{2}, z_k, t_l - \frac{d_t}{2}) \tag{3b}$$

$$+\frac{1}{\hat{\rho}(x_i, y_l, z_k + \frac{d_z}{2})} \hat{f}_z(x_i, y_j, z_k + \frac{d_z}{2}, t_l - \frac{d_t}{2}) \tag{3c}$$

$$\begin{gathered}
\hat{P}\left(x_i, y_j, z_k, t_l + d_t\right) = \hat{P}\left(x_i, y_j, z_k, t_l\right) \\
-\hat{\kappa}(x_i, y_j, z_k)\{p_x[\widehat{v_x}(x_i + \frac{d_x}{2}, y_j, z_k, t_l + \frac{d_t}{2}) + \widehat{v_x}(x_i + \frac{d_x}{2}, y_j, z_k, t_l - \frac{d_t}{2}) \\
-\widehat{v_x}(x_i - \frac{d_x}{2}, y_j, z_k, t_l + \frac{d_t}{2}) - \widehat{v_x}(x_i - \frac{d_x}{2}, y_j, z_k, t_l - \frac{d_t}{2})] \\
+q_x[\widehat{v_x}(x_i + \frac{3d_x}{2}, y_j, z_k, t_l + \frac{d_t}{2}) + \widehat{v_x}(x_i + \frac{3d_x}{2}, y_j, z_k, t_l - \frac{d_t}{2}) \\
-\widehat{v_x}(x_i - \frac{3d_x}{2}, y_j, z_k, t_l + \frac{d_t}{2}) - \widehat{v_x}(x_i - \frac{3d_x}{2}, y_j, z_k, t_l - \frac{d_t}{2})] \\
p_y[\widehat{v_y}(x_i, y_j + \frac{d_y}{2}, z_k, t_l + \frac{d_t}{2}) + \widehat{v_y}(x_i, y_j + \frac{d_y}{2}, z_k, t_l - \frac{d_t}{2}) \\
-\widehat{v_y}(x_i, y_j - \frac{d_y}{2}, z_k, t_l + \frac{d_t}{2}) - \widehat{v_y}(x_i, y_j - \frac{d_y}{2}, z_k, t_l - \frac{d_t}{2})] \\
+q_y[\widehat{v_y}(x_i, y_j + \frac{3d_y}{2}, z_k, t_l + \frac{d_t}{2}) + \widehat{v_y}(x_i, y_j + \frac{3d_y}{2}, z_k, t_l - \frac{d_t}{2}) \\
-\widehat{v_y}(x_i, y_j - \frac{3d_y}{2}, z_k, t_l + \frac{d_t}{2}) - \widehat{v_y}(x_i, y_j - \frac{3d_y}{2}, z_k, t_l - \frac{d_t}{2})] \\
+p_z[\widehat{v_z}(x_i, y_j, z_k + \frac{d_z}{2}, t_l + \frac{d_t}{2}) + \widehat{v_z}(x_i, y_j, z_k + \frac{d_z}{2}, t_l - \frac{d_t}{2}) \\
-\widehat{v_z}(x_i, y_j, z_k - \frac{d_z}{2}, t_l + \frac{d_t}{2}) - \widehat{v_z}(x_i, y_j, z_k - \frac{d_x}{2}, t_l - \frac{d_t}{2})] \\
+q_z[\widehat{v_z}(x_i, y_j, z_k + \frac{3d_z}{2}, t_l + \frac{d_t}{2}) + \widehat{v_z}(x_i, y_j, z_k + \frac{3d_z}{2}, t_l - \frac{d_t}{2}) \\
-\widehat{v_z}(x_i, y_j, z_k - \frac{3d_z}{2}, t_l + \frac{d_t}{2}) - \widehat{v_z}(x_i, y_j, z_k - \frac{3d_z}{2}, t_l - \frac{d_t}{2})]\} \\
-\hat{\kappa}\left(x_i, y_j, z_k\right) \sum_{r=1}^{R} \frac{1}{2} d_t \omega_r \left[\widehat{p_r}\left(x_i, y_j, z_k, t_l + d_t\right) + \widehat{p_r}\left(x_i, y_j, z_k, t_l\right)\right] \\
+\hat{e}(x_i, y_j, z_k, t_l + d_t) - \hat{e}(x_i, y_j, z_k, t_l)
\end{gathered} \tag{4}$$

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 choices of absorbing boundary conditions are available: sponge (wavefield taper) or perfectly matched layers (PMLs). PMLs have been implemented instead of the sponge boundary as the performance is better of a layer of a similar thickness. The PML is an ABC that is theoretically perfect in that at the start of the PML zone, there is zero reflection back into the computational domain. This is not absolute in practice, but provides an adequate boundary condition for Paracousti (Cerjan et al., 1985; Beringer, 1994). Three PML options are provided: traditional PML, convolutional PML (CPML), and multi-axial PML (MPML). The MPML is the most general form, with the others 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 troubled the traditional PML. This results in a negligible amount of energy that is reflected back inside the domain from the boundaries of the computational domain. For most problems, the CPML is recommended, but if grazing incidence waves appear in the solution, the MPML would be second. Only for very simple problems where grazing incidence wave will not be a concern should the traditional PML be considered. The grazing angle may also be manually reduced by increasing the size of the zone along the boundary of the domain dedicated to the boundary condition.

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

8. 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. Furthermore, Paracousti has the capability to calculate the propagated sound field from multiple sound sources with unique profiles and these sources can have monopole and/or dipole contributions. As each finite-difference grid point allows for separate density and sound speed values, real world domains can be easily represented, evaluated and compared to analytical solutions and literature. As this program was developed for modeling MHK deployments, examples previously presented in this report and available for Paracousti are for modeling any underwater sound-source and its propagation. Additional modeling information is available at:

https://github.com/SNL-WaterPower/Paracousti

9. References

1. Aldridge, D.F., S.L. Collier, D.H. Merlin, V.E. Ostashev, N.P. Symons, and D.K. Wilson, Staggered-grid finite-difference acoustic modeling with the Time-Domain Atmospheric Acoustic Propagation Suite (TDAAPS), Technical Report, Sandia National Laboratories, Albuquerque, NM, pp. 1-107, 2005.
2. Berenger, J-P., A Perfectly Matched Layer for the Absorption of Electromagnetic Waves, *J. Comp. Phys.*, 114, 185-200, 1994.
3. 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.
4. Hafla, E., E. Johnson, C.N. Johnson, L. Preston, D. Aldridge, and J.D. Roberts, Modeling underwater noise propagation from marine hydrokinetic power devices through a time-domain, velocity-pressure solution, *J. Acoust. Soc. Am.*, 143 (6), 1-12, 2018.
5. 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.
6. 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.
7. Ostashev, V.E., D.K. Wilson, L. Liu, D.F. Aldridge, N.P. Symons, and D. Marlin, Equations for finite-difference, time-domain simulation of sound propagation in moving inhomogeneous media and numerical implementation, *J. Acoust. Soc. Am.,* 117 (2), 503–517, 2005.
8. Preston, L.A., TDAAPS 2: Acoustic Wave Propagation in Attenuative Moving Media, SAND2016-7859, Sandia National Laboratories, Albuquerque, NM, August 2016.
9. 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.
10. 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.

10. Appendix I: List of Commands

For the user’s convenience, the following section contains an alphabetical list of all the commands described within this document along with the page number where the original description of the command may be found. It also includes a brief description of each component. For further detail or suggestions on implementation, please reference the appropriate page.

10.1. Commands:

	Page
-bF: a pressure-free surface boundary condition that is used to simulate an air-water and/or air-earth interface that is flat 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.	15
-bp *n R* : traditional PML boundary condition with a thickness of *n* nodes and parameter *R*. *n* is typically 10; *R* should be 0.01 or 0.001 in general.	15
-bpc *n R a k* : convolutional PML boundary condition 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, where Fpeak is approximately the dominant frequency of the source waveform, and *k* should be 1.	14
-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 boundary condition 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 except specified for the location in the simulation domain.	14
-bpm *n R a k xfac* : multi-axial PML boundary condition with a thickness of *n* nodes, and parameters *R*, *a, k,* and *xfac*. *n* is typically 10; *R* should be 0.001 or less, *a* should be pi*Fpeak where 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.	14
-bpm6 *nXmin RXmin aXmin kXmin nXmax RXmax aXmax kXmax nYmin RYmin aYmin kYmin nYmax RYmax aYmax kYmax nZmin RZmin aZmin kZmin nZmax RZmax aZmax kZmax xfac*: multi-axial PML boundary condition with a thickness of *n* nodes, with parameters *R*, *a* and *k* for each of the sides. See –bpm command.	15
-Ef *maxPointsPerSliceFile* : sets 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	20

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

-En *N type plane pos* : outputs 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", indicating particle velocities in each of the three indicated directions. *plane* can be "XY", "XZ" or "YZ". 26

-Eo *sliceFile.cdf* : output the slices (snapshots) to the netCDF file *sliceFile.cdf.* 26

-Et *minT:Dt:maxT type plane pos*: outputs snapshots of *type* ground motion on the given *plane* at position *pos* at the times specified by a MATLAB-style vector starting at time *minT*, stopping at time *maxT*, every *Dt* seconds. Remaining parameters are as in -En. 20

-hc 4 *c0 c1* : define the 4th order spatial finite-difference coefficients to be used 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. 23

-np : number of processors. 23

modelFile.cdf : the model name provided directly after the executable with no flag preceding it. 23

-p *px py pz* : gives the domain processor 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. 23

-R *type x y z :* This adds one receiver of *type* is either "Pressure", "3C", "4C", "Vx", "Vy" or "Vz" at position x, y, z in model coordinates. A 3C gives all three velocity components per receiver line and 4C also includes pressure. 18

-Ra : make acceleration traces instead of the default velocity traces. 19

-Rd : make displacement traces instead of the default velocity traces. 19

-Rf3 *type filename.txt :* where *type* is either "Pressure", "3C", "4C", "Vx", "Vy" or "Vz"; A 3C gives all three velocity components per receiver line and 4C also includes pressure. *filename.txt* represents the text file containing three columns with the x, y, and z reciever position. 18

-Rg *type x0:dx:xf y0:dy:yf z0:dz:zf :* Adds a uniform grid of receiver of *type* "Pressure", "3C", "4C", "Vx", "Vy" or "Vz", defined by MATLAB style vectors. A 3C gives all three velocity components per receiver line and 4C also includes pressure. For example, *x0:dx:xf* means x ranging from x0 to xf at an increment of dx. 18

-Rl : use trilinear interpolation instead of the default cubic interpolation for receiver points. 18

-Ro *traceOutputFile.cdf* : output the trace data from the receivers to this netCDF file. 19

-SD *0 :* add a delta function wavelet as a spike at time zero. 16

-Se *x y z amp* : add an explosion source of amplitude *amp* (N-m) at position x, y, z. 17

-Sfz *x y z amp :* add a force type source of amplitude *amp* (N-m) at position x, y, z. To specify an x-directed or y-directed for source use -Sfx or -Sfy, respectively. 17

-Smr : specifies that the following moment source waveforms are moment rate waveforms. 17

-Sr *Fpeak :* add a Ricker wavelet as a source, where *Fpeak* is the peak frequency of the desired Ricker wavelet. 15

-Sw *filename.txt :* add an arbitrary waveform, where *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. 16

-T *t0:dt:tf* : redefine the time vector for the simulation in MATLAB vector notation. 23

-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* are the relaxation frequencies for the *nR* mechanisms. *a1…anR* are the attenuation factors for the *nR* mechanisms. Note that if all *a1…anR* are 0.0 then it is equivalent to a non-attenuating medium. 21

-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. 22

Sandia National Laboratories