SANDIA REPORT

SAND2000-0300
Unlimited Release
Printed March 2000

RECEIVED
MAR 20 2000
EST

Salvo

Seismic Imaging Software for Complex Geologies

Curtis Ober, Rob Gjertsen, and David Womble

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

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

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

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

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

Prices available from (703) 605-6000
Web site: <http://www.ntis.gov/ordering.htm>

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

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND2000-0300
Unlimited Release
Printed March 2000

Salvo

Seismic Imaging Software for Complex Geologies

Curtis Ober
Parallel Computational Science Department

Rob Gjertsen, David Womble
Applied Mathematics

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1111

Abstract

This report describes Salvo, a three-dimensional seismic-imaging software for complex geologies. Regions of complex geology, such as overthrusts and salt structures, can cause difficulties for many seismic-imaging algorithms used in production today. The paraxial wave equation and finite-difference methods used within Salvo can produce high-quality seismic images in these difficult regions. However this approach comes with higher computational costs which have been too expensive for standard production. Salvo uses improved numerical algorithms and methods, along with parallel computing, to produce high-quality images and to reduce the computational and the data input/output (I/O) costs.

This report documents the numerical algorithms implemented for the paraxial wave equation, including absorbing boundary conditions, phase corrections, imaging conditions, phase encoding, and reduced-source migration. This report also describes I/O algorithms for large seismic data sets and images and parallelization methods used to obtain high efficiencies for both the computations and the I/O of seismic data sets. Finally, this report describes the required steps to compile, port and optimize the Salvo software, and describes the validation data sets used to help verify a working copy of Salvo.

Salvo has successfully met the project goals of (1) demonstrating the computational feasibility of finite-difference, prestack depth migration algorithms, (2) developing a code which is scalable, easily portable to high-performance computers and is validated on synthetic and real-world data sets, and (3) developing techniques for the management of large scale I/O and data sets on massively parallel processing computers. The project partners incorporating the Salvo algorithms into their own seismic processing environments, and the selection of Salvo as a 1999 R&D 100 winner further indicates the success and usefulness of this software and project.

Acknowledgments

There have been many participants to the "3D Seismic Imaging of Complex Geologies" ACTI project. Below is a list of these people for both Sandia National Laboratories and the ACTI partners.

Sandia National Laboratories		
Curtis Ober	David Womble	Rob Gjertsen
Ron Oldfield	Louis Romero	Elaine Gorham
John Van Dyke	Greg Homicz	Roger Breeding
Mike Holzrichter	Liyang Xu	Susan Minkoff
David Roach	Lydie Prevost	Pang Chen
Andrew Bowker	Jermey Robinson	Jabed Anwar
Amerada-Hess		
Scott Morton		
ARCO		
Charles Mosher		
Conoco Inc.		
Charles C. Burch	Doug Hansen	Richard Day
Steve Whitney		
Fairfield Industries (Golden Geophysical)		
Craig Artley	Mike Sullivan	
GX Technology		
Mick Lambert		
IBM		
Billy Robinson	Tom Morgan	
Intel SSD		
Oryx Energy Co.		
Bob Fleming	Herman Chang	
PGS Tensor		
Leon Hu	Matthew Brzostowski	
Providence Technology		
Jim Manatt	Tom Nickoloff	
SGI/Cray Research		
TGS Calibre		
Sam Hussein		
University of Texas at Dallas		
George McMechan	Tong Xu	Yafei Wu
Hongbo Zhou	Houzhu Zhang	

Contents

1	Getting Started Quickly with Salvo	7
1.1	Installation and Compilation of Salvo	7
1.2	Running a Test Problem with Salvo	8
1.3	Checking Test Problem Output	9
2	Overview of Salvo	10
3	Migration Algorithm	13
3.1	Governing Equations	13
3.2	Boundary Conditions	15
3.2.1	Clayton and Engquist's Absorbing Boundary Condition . . .	16
3.2.2	Xu's Absorbing Boundary Condition	16
3.2.3	Perfectly Matched Layer	17
3.3	Phase Correction	18
3.3.1	Li Filter	19
3.3.2	Graves and Clayton Filter	19
3.4	Imaging Conditions	20
3.4.1	Poststack Imaging	21
3.4.2	Prestack Imaging	22
3.4.3	Deconvolution Imaging	23
3.4.4	Reduction of Temporal Wrap Around	24
3.5	Numerical Characteristics	24
3.5.1	Dissipation	28
3.5.2	Dispersion	28
3.5.3	Compact Finite Differences	33
3.6	Phase Encoding	35
3.7	Reduced-Source Frequency Migration	38
3.7.1	Analytic Solution	38
3.7.2	Scaling Requirements for Gaussian Source	41
3.7.3	Comparison of Actual and Model Spectra	42
3.7.4	Estimating the Source Spectrum	45
3.7.5	Parallelization and Cyclic Distribution	46
3.7.6	Numerical Results	47
3.7.7	Open Issues	50
4	Coding and Implementation	51
4.1	Initialization of Salvo	51
4.2	Reading the Trace Data	52
4.2.1	<i>READ_TRACES</i> Initialization Step	53
4.2.2	Reading Traces Step	56
4.2.3	Scattering Time Traces Step	57

4.2.4	Computing the Frequency Traces	58
4.2.5	Redistributing the Frequency Traces	58
4.2.6	<i>READ_TRACES</i> Optimal Execution Time	59
4.3	Migration	60
4.3.1	Parallelization	60
4.3.1.1	Frequency Parallelism	62
4.3.1.2	Spatial Parallelism	64
4.3.2	Reading the Velocity Model	66
4.3.3	Writing or Stacking the Image	68
5	Salvo User's Guide	71
5.1	Software Distribution	71
5.2	Compiling and Porting	72
5.2.1	Tips for Optimization	76
5.2.2	Compilation Flags and Options	76
5.3	Performance Results	79
5.3.1	I/O During Reading of Traces	79
5.3.2	I/O During Migration	79
5.3.3	Parallel Efficiencies	80
5.4	Description of Input and Output	81
5.4.1	Input Parameter Deck	81
5.4.1.1	<i>INITIAL</i> Process	82
5.4.1.2	<i>READ_TRACES</i> Process	85
5.4.1.3	<i>MIG_SPLIT</i> Process	88
5.4.2	Traces for Source and Receivers	92
5.4.3	Velocity Model	93
5.4.4	Output Data	93
6	Validation Data Sets	95
6.1	Impulse Response	95
6.2	Marmousi Model	96
6.3	SEG/EAGE Salt Model	100
	References	105
A	Memory Requirements	108
B	Migration: Miscellaneous	111
B.1	Lee and Suh Expansion Coefficients	111
B.2	Finite-Difference Equations of Tridiagonal Solves	111
B.3	Comparison of Splitting Methods	113
B.4	Discretization of Boundary Conditions	114
B.5	Discretization of the Perfectly Matched Layer	116

B.6	Graves and Clayton Filter	117
B.7	Analytic Solutions to Impulse in a Half Space	119
C	I/O: Miscellaneous	122
C.1	Velocity File Information	122
C.2	Accumulation File Information	122
C.3	Enhanced SEP File Format	123
C.4	GIF Generation of Image Plane Slices and Lines	125
D	User Manual: Miscellaneous	126
D.1	Salvo Utilities	126
D.2	Code Limitations and Assumptions	126
D.3	Pseudo Coding for Migration	129
	Index	129
	Distribution	133

1 Getting Started Quickly with Salvo

This section is for those wishing to get Salvo up and running quickly. Information on code optimization and performance are provided in Section 5. Salvo is distributed through Sandia National Laboratories, but the code is copyrighted and currently under partnership agreement; David Womble or Curtis Ober should be contacted for information on the availability of Salvo.

David Womble
Applied Mathematics
Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110
(505) 845-7471
dewombl@cs.sandia.gov
<http://www.cs.sandia.gov/~dewombl>

Curtis Ober
Parallel Computational Science
Sandia National Laboratories
P.O. Box 5800, MS 1111
Albuquerque, NM 87185-1111
(505) 844-1846
ccoer@cs.sandia.gov
<http://www.cs.sandia.gov/~ccoer>

Salvo utilizes the MPI library for communication, so the user must have a version of MPI on their platform. Most vendors provide an optimized version of the MPI library, but if the vendor does not provide a version of the MPI library then a publicly available version can be substituted. The recommended public MPI library is MPICH, which is distributed by Argonne National Laboratories. MPICH can be down loaded from <http://www.mcs.anl.gov/mpich/index.html> or via anonymous FTP at [ftp.mcs.anl.gov](ftp://ftp.mcs.anl.gov) under `/pub/mpich/mpich.tar.Z`.

1.1 Installation and Compilation of Salvo

The following is a list of the necessary steps to install and compile Salvo. These steps will produce a working version of Salvo on the desired platform. However, information is provided in Section 5 on compiling and porting Salvo to obtain better performance or for different capabilities, such as, various FFT libraries, boundary conditions, and compact finite-difference coefficients. A precompiled executable is provided for the Intel Paragon, and is located at `salvo/bin/salvo.paragon`. To produce an executable for your platform:

1. Copy the Salvo software distribution, `salvo.tar.gz`, to the desired directory location. After decompressing, `'ungzip salvo.tar.gz'`, and extracting from tar, `'tar xvf salvo.tar'`, a Salvo directory will be present with all the distribution files.
2. Change directory to the Salvo source code directory, `'cd salvo/src'`, to begin the compilation.
3. Set the environment variable, `TARGET`, to the environment file of the desired platform using the command, `'setenv TARGET platform'`. Environment files

provided with the Salvo distribution can be found in the environments directory, *'ls environment/*.env'* (e.g., *generic.env*, *sgi.env*, *paragon.env*). To set the *TARGET* variable for the SGI platform use *'setenv TARGET sgi'* (omitting the *.env* extension). Descriptions of the environment files are given in the README file in the environments directory. If an exact match is not found, begin initially with the generic settings, *'setenv TARGET generic'*.

4. Create the Salvo executable by typing *'make'*. If a problem occurs at this point, it is likely the inability to find compilers (C and Fortran are required), and libraries (*libm.a*, MPI, etc.) Consult with the local system administrator for the locations of these pieces of software. Once Salvo is successfully compiled, the Salvo executable, *salvo*, should be located in the Salvo binary directory, *salvo/bin*.
5. Compile the Salvo utilities in *salvo/utls* directory. These utilities are briefly described in Section 5.1, and provide several useful functions for processing data sets with Salvo. This step is optional and is completely independent of the Salvo imaging code. The utilities can be compiled for one of several architectures. The available architectures can be determined by typing *'make'* in the *utls* directory. Initially a generic compilation is suggested, *'make generic'*. The utility binaries are placed in the *salvo/bin* directory.

1.2 Running a Test Problem with Salvo

Once an executable of Salvo is generated, the validation problems provided in the directory *salvo/Validation_Suite* can be used to test the new executable. Of the several problems included, a 24 shot subset of the Marmousi model [30][31] are a good initial test. This two-dimensional problem is in the directory, *Marmousi24_Prestack_Freq*. The Salvo parameter input deck, *test.prm*, is located in the *Marmousi24_Prestack_Freq/prms* directory. This input deck can be passed to Salvo as a command line argument, *'salvo test.prm'*, or by standard input, *'salvo < test.prm'*. If the MPICH version of the MPI library is used, the user can launch the test job with the command *'mpirun -np1 salvo test.prm.'* The protocol for launching MPI jobs may differ for various MPI libraries and parallel platforms, so the user should determine the appropriate method on their system.

The output from this run is *test.job*, *test.acc*, and the standard output to the screen. The file *test.job* contains the *job name*, the number of processors, and the compute runtime. The file will be created in the parameter directory. The image file is *test.acc* and is created in the *Marmousi24_Prestack_Freq/images* directory. The image, *test.acc*, is stored in the accumulation file format which is described in Appendix C.2. The data printed to standard output contains information on the input deck, runtime status, and performance of the run. If changes to the input deck or Salvo executable are desired, the user should refer to the **Salvo User's Guide**, Section 5.

Figure 1: Postscript image of Marmousi model using 24 shots.

1.3 Checking Test Problem Output

The output image file, *Marmousi24_Prestack_Freq/images/test.acc*, should be checked for accuracy. The user can compare the output image, *test.acc*, with the reference image included in the distribution, *Marmousi24_Prestack_Freq/images/Reference/ref_image.acc*, using the Salvo utility *compare* (i.e., *compare test.acc Reference/ref_image.acc*). The relative norms calculated by the utility should be close to machine precision. A postscript image, *images/Reference/ref_image.ps.gz*, is provided for visual comparison and is displayed in Figure 1.

2 Overview of Salvo

Salvo is a seismic imaging software developed under the Department of Energy's Advanced Computational Technology Initiative (ACTI) project entitled, "3-D Seismic Imaging of Complex Geologies: Remote and Rapid Processing of Terabyte Data Sets." ACTI's goals attempted to foster technology transfer between the national laboratories and the oil and gas industry.

This particular project focuses on the reduction of risks and costs associated with domestic oil and gas exploration through fast and accurate imaging of complex geologies, such as thrusts in mountainous regions and salt structures in the Gulf of Mexico. Current production algorithms used today by many companies are schemes known as Kirchhoff methods. These methods are sufficient so long as the geology remains relatively simple and smooth. However new oil and gas prospects are increasingly in regions of complex geology. Imaging these structures can be computationally expensive. Data sets can be terabytes in size, and the processing time can take weeks, even with the massively parallel computers available today. To reduce these computational costs, this project developed a parallel seismic-imaging algorithm, Salvo.

The initial goals of the project were to develop (1) a scalable, finite-difference, prestack depth migration algorithm, (2) an algorithm which is easily portable to high-performance computers and is validated on synthetic and real-world data sets, (3) techniques for the management of large-scale I/O and data sets on massively parallel computers, (4) a remote seismic processing environment, and (5) a commercialization plan and cost analysis for the migration method. However due to funding restrictions, the last two goals have been reduced in scope from the initial plans.

Salvo is an implicit finite-difference frequency-space (f - x) migration algorithm of the paraxial acoustic wave equation. Approximations are used to estimate the square-root operator, and the method of fractional steps is used to produce efficient tridiagonal solves. These methods produce a highly efficient algorithm which obtains 30% to 50% of theoretical peak performance on most architectures. Additionally, Salvo is a parallel algorithm, and partitions the I/O and computational workload to decrease the overall runtime.

Salvo can migrate two- or three-dimensional, poststack or prestack seismic data sets. Conventional land, marine or ocean bottom cable surveys have been migrated using shot and receiver gathers. Because Salvo is a depth migration algorithm, it also requires an acoustic velocity model of the migration volume.

Implicit finite-difference (f - x) migration methods are computationally more expensive than Kirchhoff methods but provide higher quality images in complex geologies. Salvo, a shot-record migration, can exploit the acquisition geometry over a Kirchhoff method, a per-trace migration method. The computational costs associated with Kirchhoff methods are dependent on the number of traces. While for shot-record migrations, the computational costs are dependent on the number

Figure 2: Images of the SEG/EAGE Salt model using data sets collected by (a) conventional streamer and (b) ocean bottom cable (OBC) acquisition geometries.

of shots, which in turn is dependent on the acquisition geometry. If more traces can be acquired per shot, a shot-record migration method will reduce its computational costs relative to the trace migration method. Therefore going from two- to three-dimensional data sets, Salvo will gain relative to Kirchhoff methods, since there will be more traces per shot. This is also true for ocean bottom cable (OBC) and vertical cable data versus conventional streamer data, because again there are more traces per shot.

As an example, two data sets collected over the SEG/EAGE Salt model were migrated with Salvo. The resulting images are shown in Figure 2. These images are very similar. However there are two major differences in the generation of these images: the acquisition geometry and the computational costs. The classical streamer acquisition is from the Phase-C classic datasets (C3) of the SEG/EAGE 3D Modeling effort. This data contains approximately 2400 shots for the C3-WA dataset with each shot having approximately 800 traces. The ocean bottom cable (OBC) acquisition is a selection of shots from the forward modeling effort, containing 45 shots with each shot having 40,000 traces. However the computational costs associated with each geometry are not as similar as the images. The cost per shot is nearly equal for both the C3 and the OBC datasets, since they use the same migration volume. Therefore the C3-WA image costs about 50 times more than the OBC image based on the sheer number of shots (migrations). Thus the costs associated with Salvo may be reduced by using acquisition geometries which provide more traces per shot and still obtain high quality images.

The remainder of this document contains information on Salvo internal workings and how to run Salvo. For those in a rush, Section 1, **Getting Started Quickly with Salvo** provides the basic steps to get Salvo up and running. Additional information on compiling, testing, porting and optimizing Salvo for your system is given in Section 5, **Salvo User's Guide**. Included in this section are descriptions of the input and output data, along with code limitations and assump-

tions.

Sections 3 and 4 describe the migration and I/O algorithms, and provide details on the underlying implementations. To help identify pieces of code within Salvo with the discussions in the sections, footnotes are included. Each footnote specifies the filename(s) and routine(s) which relate to the current topic. Additionally to illustrate the capabilities of Salvo and allow the user to validate their copy of Salvo, Section 6 has several validation problems. These include the Marmousi model and the SEG/EAGE Salt model. Finally several appendices are included and provide additional documentation.

One question that is commonly asked is what does Salvo stand for? It stands for nothing; it's a name. When this ACTI project was first beginning, the definition of salvo was found – a discharge of a number of pieces of artillery or small arms in regular succession or at the same time (“parallel processing?”) either as a salute or, especially in naval battles, as a broadside [33]. A seismic survey is a regular series of “shots” over a domain of interest. The match seemed apparent and the name stuck.

3 Migration Algorithm

Migration is the process of correctly placing reflectors in the earth's interior. The term "migrate" comes from the fact that simple stacking incorrectly places reflecting layers. By applying an algorithm to improve the reflectors placement, one has migrated the reflector to its "correct" location. Thus these algorithms have been termed migration algorithms. In the following section, a basic overview of the migration method used within Salvo is presented. More detailed information on the migration algorithm for those wishing to understand the details is included in the appendices.

3.1 Governing Equations

To model wave propagation within the earth's interior, the scalar wave equation is used

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} = \frac{1}{v^2} \frac{\partial^2 P}{\partial t^2}. \quad (3.1)$$

Here $v(x, y, z)$ is the acoustic velocity of the medium, and $P(x, y, z, t)$ is the pressure wavefield. Data collected at the surface, $P(x, y, 0, t)$, will be downward propagated to illuminate reflectors within the subsurface. The scalar wave equation, Eq. (3.1), is time consuming to solve. The solution time can be reduced by transforming the hyperbolic equation in time to a parabolic equation in depth through the paraxial approximation [8, p. 54]. The reduction in runtime is basically dependent on the number of time steps for the wave equation solution, and the number of frequencies migrated for the paraxial solution, holding the problem size and the solution method (*i.e.*, computational speed) constant. The paraxial wave equation is

$$\frac{\partial P}{\partial z} = \left\{ \pm \frac{i\omega}{v} \left[1 + \frac{v^2}{\omega^2} \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \right]^{1/2} \right\} P, \quad (3.2)$$

and P is now a function of (x, y, z, ω) . The positive and negative signs correspond to upcoming and downgoing wavefields respectively. Surface pressure is used as an initial condition to solve the parabolic equation down to the next depth level. Hundreds of paraxial equations, Eq. (3.2), are solved; one for every frequency, ω , to be included in the solution.

The square-root operator of the paraxial wave equation is difficult to estimate numerically. Therefore, this operator is approximated by an optimized series which originates from a continued-fraction expansion [8, p. 84] [34, p. 513]. The continued-fraction expansion can be represented by ratios of polynomials [21], and the polynomial coefficients can be optimized for propagation angle [19]. With these approximations, the paraxial wave equation can be written as

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \left[1 + \sum_{l=1}^m \frac{\alpha_l S}{1 + \beta_l S} \right] P, \quad (3.3)$$

where

$$S = S_x + S_y = \frac{v^2}{\omega^2} \frac{\partial^2}{\partial x^2} + \frac{v^2}{\omega^2} \frac{\partial^2}{\partial y^2},$$

and α_l and β_l are the expansion coefficients derived by Lee and Suh [19]. These coefficients are listed in Appendix B.1 for reference. Commonly one term of the series is retained ($m = 1$), and propagation angles up to 65 degrees can be accurately determined. For the remainder of this discussion m is assumed to be unity, but expansions with $m = 1, \dots, 5$ are implemented within Salvo. These additional terms in the expansion are handled by the same methods described below.

A consequence of the square-root approximation is that evanescent energy (*i.e.*, waves that do not satisfy the dispersion relationship) is migrated within the solution. This energy produces artifacts that can cause difficulty interpreting seismic images. The impulse problem should produce a semi-circle (see Figure 10 on p. 35), but because of the square-root approximation, the 45 degree equation will produce a cardioid (similar to the two-dimensional poststack image without phase correction shown in Table 4 on p. 97) where the evanescent waves have looped back to the center of the image. A good discussion of this problem is provided by Bunks [7]. In Section 3.3, methods to correct this difficulty are presented.

To solve Eq. (3.3) with a finite-difference technique directly requires solving a banded matrix. The size of this matrix is on the order of $N^2 \times N^2$ where N is the number of grid points in either the x or y directions. Clearly, a direct solution of such a large matrix is prohibitive. To reduce the computational cost, each term in Eq. (3.3) is solved separately by the method of fractional steps [12]. For instance, the first equation (and term)

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} P \quad (3.4)$$

has the exact solution $P(x, y, z + \Delta z, \omega) = P(x, y, z, \omega) \exp[\pm (i\omega/v) \Delta z]$. This first term is the thin-lens term¹ and corresponds to a 5 degree equation (*i.e.*, images reflectors up to 5 degrees of dip). The second equation is given by

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \frac{\alpha_1 S}{1 + \beta_1 S} P. \quad (3.5)$$

Solving Eq. (3.5) is still computationally intensive. To reduce this expense, another operator separation is applied in the x and y directions [5] so that efficient tridiagonal solves can be performed via the Thomas algorithm [12]. To convert the operator, S , to a linear combination of S_x and S_y , Eq. (3.5) is written

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \left[\frac{\alpha_1 S_x}{1 + \beta_1 S_x} + \frac{\alpha_1 S_y}{1 + \beta_1 S_y} + \frac{-2\alpha_1 \beta_1 S_x S_y - \alpha_1 \beta_1^2 (S_x S_x S_y + S_x S_y S_y)}{(1 + \beta_1 S)(1 + \beta_1 S_x)(1 + \beta_1 S_y)} \right] P \quad (3.6)$$

¹Subroutine: thin_lens; File: thin_lens.F

or, dropping cross-product terms because of their difficulty to compute,

$$\frac{\partial P}{\partial z} \approx \pm \frac{i\omega}{v} \left[\frac{\alpha_1 S_x}{1 + \beta_1 S_x} + \frac{\alpha_1 S_y}{1 + \beta_1 S_y} \right] P. \quad (3.7)$$

These neglected terms are compensated for later by the phase correction filters. The operators in Eq. (3.7) are once again split by the method of fractional steps, and solved in sequence

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \frac{\alpha_1 S_x}{1 + \beta_1 S_x} P \quad \text{and} \quad \frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \frac{\alpha_1 S_y}{1 + \beta_1 S_y} P. \quad (3.8)$$

These two terms are the x and y diffraction terms² and details of their discretization with finite differences are shown in Appendix B.2. The method of fractional steps (MOFS) is very similar to the methods of approximate factorization (AF) or alternating direction implicit (ADI) schemes, where the x and y directions are separated to produce tridiagonal systems of equations. The MOFS only requires one evaluation of the right-hand-side terms while ADI schemes require two evaluations of the right-hand side. Also, the MOFS scheme for the paraxial wave equation only requires one level of storage ($n_x \times n_y$) for the wavefield rather than two as generally required for AF and ADI schemes. Thus to maintain low memory requirements, the MOFS is the method of choice. A brief discussion of operator-splitting methods is presented in Appendix B.3. After applying operator splitting and approximations, the paraxial wave equation reduces to a split wave equation,

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \left[1 + \frac{\alpha_1 S_x}{1 + \beta_1 S_x} + \frac{\alpha_1 S_y}{1 + \beta_1 S_y} \right] P. \quad (3.9)$$

The method of fractional steps can be shown to be equivalent to the ADI scheme if the operators, $(i\omega/v)\alpha_1 S_x/(1 + \beta_1 S_x)$ and $(i\omega/v)\alpha_1 S_y/(1 + \beta_1 S_y)$, in Eqs. (3.8) commute and a term of order Δz^2 is neglected.

3.2 Boundary Conditions³

To complete the tridiagonal solves associated with Eqs. (3.8), boundary conditions are required. Within Salvo five types of boundary conditions (BCs) have been implemented: zero-value Dirichlet BC, zero-slope Neumann BC, Clayton and Engquist absorbing boundary conditions (ABCs) [9], University of Texas at Dallas' ABC, and Liyang Xu's ABC. These can be specified by the compiler flag, -DBCS=n (see Section 5.2.2 for details). The first two boundary conditions have the property of being perfect reflectors, which can cause unwanted artifacts. However these two

²Subroutine: `diffract`; file: `diffract.F`

³Subroutines: `pipe_single`, `pipe_start`, `pipe_middle`, `pipe_return`; Files: `pipe_single.F`, `pipe_start.F`, `pipe_middle.F`, `pipe_return.F`

boundary conditions are useful for debugging and stability purposes. Not all of the 15, 45, 65, 80, 87 and 90 degree equations implemented within Salvo are compatible with all boundary conditions (see stability considerations on Section 3.5.1). Finite-difference approximations of the implemented boundary conditions are presented in Appendix B.4.

3.2.1 Clayton and Engquist's Absorbing Boundary Condition

Clayton and Engquist's B3 boundary condition [9] is the most robust at absorbing incident waves of those presented and is included within Salvo. In the Fourier domain, this absorbing boundary condition is

$$\frac{v}{\omega} k_z = \frac{s_w a_b - s_{lr} b_b \frac{v}{\omega} k_x}{1 - s_w s_{lr} c_b \frac{v}{\omega} k_x}. \quad (3.10)$$

where

$$s_w = \begin{cases} +1 & \text{upcoming waves} \\ -1 & \text{downgoing waves} \end{cases}, \quad s_{lr} = \begin{cases} +1 & \text{right hand boundaries} \\ -1 & \text{left hand boundaries} \end{cases}$$

and a_b , b_b and c_b are determined by matching the hyperbola of Eq. (3.10) to the dispersion relation, $k_x^2 + k_y^2 + k_z^2 = \omega^2/v^2$, for each quadrant. These coefficients are $a_b = 1$, $b_b = 1$ and $c_b = 2 - \sqrt{3}/2 = 0.845299462$, when Eq. (3.10) is solved using the points $(k_x, k_z) = (0, s_w \omega/v)$, $(s_w s_{lr} \omega/v, 0)$, $(s_w s_{lr} \omega/2v, s_w \sqrt{3}\omega/2v)$. Thus the equation to be approximated is

$$s_w i \frac{\partial P}{\partial z} - s_{lr} c_b \frac{v}{\omega} \frac{\partial^2 P}{\partial x \partial z} = -a_b \frac{\omega}{v} P - s_w s_{lr} b_b i \frac{\partial P}{\partial x}.$$

The University of Texas at Dallas' ABCs are similar, but use a suggestion from Clayton and Engquist that a_b can be zero for retarded coordinates. One difficulty with this approach is the determination of the coefficients. UT-Dallas suggests the following tuned empirical values: $b_b = 1.15$ and $c_b = -0.22$.

3.2.2 Xu's Absorbing Boundary Condition

Another set of boundary conditions were developed by Liyang Xu, a University of New Mexico student. The basic idea of this development is to determine an approximate equation for the exact boundary conditions to the diffraction equations, Eqs. (3.8). By matching the asymptotic properties of these exact boundary conditions, Xu found the following absorbing BC equation

$$c_3 v \beta_1 \frac{\partial^2 P}{\partial x \partial z} - c_4 i \omega \alpha_1 s_w \frac{\partial P}{\partial x} - c_1 s_w s_{lr} i \omega \sqrt{\beta_1} \frac{\partial P}{\partial z} - c_2 s_{lr} \frac{\omega^2 \alpha_1}{v \sqrt{\beta_1}} P = 0 \quad (3.11)$$

where $c_1 = 402$, $c_2 = 196$, $c_3 = 402$, and $c_4 = 371$. Note that the coefficients of the differential equation are dependent on α_1 and β_1 which are the expansion coefficients for the square-root operator in Eqs. (3.8). Thus this absorbing boundary condition uses the expansion coefficients to better eliminate reflected energy.

3.2.3 Perfectly Matched Layer⁴

Berenger [4] introduced a novel “sponge layer” which found wide use in electromagnetic modeling as an absorbing boundary. This sponge layer is known as the Perfectly Matched Layer (PML) because its impedance matches that of the adjacent medium (in the continuous case), resulting in a complete lack of reflections when energy enters the PML. While this technique was introduced for models governed by Maxwell’s equations, other fields have made use of it including geophysics for the paraxial wave equation.

Collino [10] showed that the PML can be interpreted as a change of variable in the complex plane. The change of variable results in a simple modification of the differential operator

$$\frac{\partial}{\partial x} \rightarrow \frac{i\omega}{i\omega + \sigma(x)} \frac{\partial}{\partial x}.$$

Applying the above to itself gives the corresponding modification for the second partial derivative

$$\frac{\partial^2}{\partial x^2} \rightarrow \frac{i\omega}{i\omega + \sigma(x)} \frac{\partial}{\partial x} \left(\frac{i\omega}{i\omega + \sigma(x)} \frac{\partial}{\partial x} \right). \quad (3.12)$$

The above transformations introduce the parameter $\sigma(x)$ which controls the attenuation of energy propagating in the x direction. The above transformations have no effect where $\sigma(x) = 0$ and energy propagates as if it were outside the PML. If $\sigma(x) > 0$, the above substitutions result in attenuation of energy propagating parallel to the x axis.

Before applying Eq. (3.12) to the paraxial equation, it is useful to apply $\partial/\partial x$ to the expression in the parenthesis. The right hand side of Eq. (3.12) becomes

$$\frac{\omega^2}{(i\omega + \sigma(x))^3} \left[\left(\frac{\partial \sigma(x)}{\partial x} \right) \frac{\partial}{\partial x} - (i\omega + \sigma(x)) \frac{\partial^2}{\partial x^2} \right]. \quad (3.13)$$

We now need expressions for the $\frac{\partial}{\partial x}\sigma(x)$ and $\sigma(x)$ terms in Eq. (3.13). In the continuous case, $\sigma(x)$ can be a step function without causing reflections. This does not hold for the discrete case. To avoid creating reflections, the value of $\sigma(x)$ is increased smoothly. A common approach is to make $\sigma(x)$ proportional to the p^{th} power of distance into the PML. Let the PML region begin at the grid point m_0 which has coordinate $x_0 = m_0\Delta x$. For $x > x_0$, the value of $\sigma(x)$ is

$$\sigma(x) \Big|_{x=m\Delta x} = \sigma_0(x - x_0)^p = \sigma_0\Delta x^p(m - m_0)^p \quad \text{where } m > m_0. \quad (3.14)$$

σ_0 is a constant greater than or equal to 0. For Salvo’s implementation of PML, the absorption increases quadratically (*i.e.*, $p = 2$).

⁴This section was implemented and documented by Mike Holzrichter.

(a) Xu's absorbing BCs

(b) PML and Xu's absorbing BCs

Figure 3: Impulse response of Salvo (a) with Xu's absorbing boundary conditions, and (b) with PML and Xu's absorbing boundary conditions.

Expression Eq. (3.13) has a term of the form $\frac{\partial}{\partial x}\sigma(x)$. When $\sigma(x)$ varies with distance into the PML according to Eq. (3.14), the derivative of $\sigma(x)$ at $x_m > x_0$ is simply

$$\left. \frac{\partial}{\partial x}\sigma(x) \right|_{x=m\Delta x} = \sigma_0 p \Delta x^{p-1} (m - m_0)^{p-1} = \frac{p}{\Delta x (m - m_0)} \sigma_m \quad (3.15)$$

where $\sigma_m = \sigma(x_m)$. Equations (3.13), (3.14), and (3.15) provide the foundations for Salvo's implementation of PML. Further details on the discretization and implementation of Salvo's PML is presented in Appendix B.5.

Results from Salvo's PML are illustrated using an impulse example. Two images were made. The first was produced using Salvo's default Liyang Xu absorbing boundary condition. The second image is identical except for a sixteen grid point thick PML region on the left and right sides.

Figure 3(a) shows that while Xu's absorbing BCs absorbed most of the energy, a small but visible amount of energy is reflected back into the interior of the migration domain. Figure 3(b) illustrates the action of the PML. The PML attenuates energy both before and after it gets reflected from the boundary. PML significantly decreases the reflected energy in the interior of the migration region.

3.3 Phase Correction

To correct for the neglected cross terms in Eq. (3.6) and for other approximations used during the development of the split wave equation, Eq. (3.9), two phase-

correction filters have been implemented within Salvo; the Li filter [20] and the Graves and Clayton filter [15]. Both filters can be used, but each has different characteristics and parallel implementations. The Li filter attempts to compensate for all the errors between the paraxial wave equation, Eq. (3.2), (assuming $v = v(z)$) and the split wave equation, Eq. (3.9). The Li filter has been implemented only for frequency parallelism; spatial parallelism would require parallel two-dimensional FFTs. The Graves and Clayton filter can be used with both frequency and spatial parallelism, however it only attempts to compensate for phase errors associated with the neglected cross-product term in Eq. (3.6). The user should carefully select the filter for their migration.

3.3.1 Li Filter⁵

To correct for the phase errors, Li [20] formed an error operator, E , which is the difference between the operators of Eqs. (3.2) and (3.9), and attempts to compensate for all the errors between them,

$$E = \sqrt{1 + S_x + S_y} - \left[1 + \sum_{l=1}^m \frac{\alpha_l S_x}{1 + \beta_l S_x} + \frac{\alpha_l S_y}{1 + \beta_l S_y} \right].$$

Applying the error operator to the wavefield, P ,

$$\frac{\partial P}{\partial z} = \left[\frac{i\omega}{v_a(z)} E \right] P,$$

the errors introduced by the approximations can be partially corrected, and the evanescent energy removed. In the expression above, $v_a(z)$ is the average acoustic velocity in any depth plane. The solution to this equation is $P(x, y, z + \Delta z, \omega) = P(x, y, z, \omega) \exp[E(i\omega/v_a) \Delta z]$ and is applied as an additional step in the method of fractional steps. If the acoustic velocity is only a function of depth, $v = v(z)$, (*i.e.*, a layered medium), the Li filter exactly corrects for the approximation errors resulting from the difference between Eqs. (3.2) and (3.9), which include the square-root approximation, the operator splitting, and numerical dispersion.

The basic implementation of the Li filter requires a two-dimensional Fourier transform to the frequency-wavenumber (f - k) domain, $P(k_x, k_y, z, \omega)$, and apply the phase correction in the f - k domain. Inverse Fourier transforms are then needed to return to the frequency-space domain, where the thin-lens and diffraction terms can be applied. Additional details are presented in Li [20].

3.3.2 Graves and Clayton Filter⁶

The Graves and Clayton filter [15] also corrects for errors associated with the cross-product term neglected in Eq. (3.6). Neglecting these cross terms introduces

⁵Subroutines: filter_li, phase_correction; File: filter_li.F

⁶Subroutines: filter_graves, f_tridag, f_pipe_*; Files: filter_graves.F, f_tridag.F, f_pipe_*.F

a phase error which is maximum along the lines $y = \pm x$. Graves and Clayton approximated the cross-product term with

$$\pm \frac{-2\alpha_1\beta_1 S_x S_y - \alpha_1\beta_1^2 (S_x S_x S_y + S_x S_y S_y)}{(1 + \beta_1 S)(1 + \beta_1 S_x)(1 + \beta_1 S_y)} \approx - \pm 2\alpha_1\beta_1 S_x S_y$$

and produced a phase-correction equation

$$\frac{\partial P}{\partial z} = \left[- \pm 2 \frac{i\omega}{v} \alpha_1 \beta_1 S_x S_y \right] P.$$

The cross-product operator, $S_x S_y$, is computed using a cascade of second-order filters. To apply this filter requires a total of 8 tridiagonal solves. This is four times more than as needed for the migration of the wavefield, P . However the number of operations required to generate the tridiagonal coefficients for the Graves and Clayton filter is fewer than for the migration of the wavefield. This phase correction operator is an additional step in the sequence of the MOFS solves. In the current implementation of Salvo, only one term of continued-fraction expansion ($m = 1$ in Eq. (3.3)) has been implemented for the Graves and Clayton filter. The finite-difference approximations and the appropriate boundary conditions are discussed in Appendix B.6.

3.4 Imaging Conditions⁷

Once the wavefields have been computed, an imaging condition must be applied to compute the migrated image. To help understand the imaging condition, it is useful to inspect an analytic solution to the impulse response problem.⁸ This problem assumes that an impulse occurs at the center of a hemispherical reflector at $t = 0$. The downgoing wave reflects off the hemisphere and the reflected upgoing wave coalesces to the point of ignition. Using the results of Appendix B.7, the following expressions for the wavefields in two and three dimensions, respectively, can be written

$$P_2(r, \theta, \omega) = \frac{A(\omega) \cos(\theta)}{\sqrt{2\pi v r}} \exp \left[\pm \frac{i\omega r}{v} \right] \left\{ \sqrt{\omega} \exp \left[\mp i \frac{\pi}{4} \right] \right\}$$

and

$$P_3(r, \theta, \omega) = \frac{A(\omega) \cos(\theta)}{2\pi v r} \exp \left[\pm \frac{i\omega r}{v} \right] \{ \mp i\omega \}$$

where r and θ are polar coordinates, and $A(\omega)$ is the amplitude spectrum of the seismic trace. The top sign of $\pm$ or $\mp$ applies to upward traveling waves, receiver wavefield, and the bottom sign applies to downward traveling waves, source wavefield. The first term in each of these expressions is the amplitude spectrum, $A(\omega)$,

⁷Subroutine: image; File: image.F

⁸This development is taken from notes provided by Louis Romero.

modified by a geometric spreading ($1/\sqrt{r}$ or $1/r$) and a directional dependence factor, $\cos \theta$. The second term is simply a time shift term related to the migration process. The last term in curly braces is a phase function which changes the shape of the original wavelet. This wavefield, P , is obtained through the normal migration process, thus the geometric spreading, directional dependence and phase function factors are all part of the wavefield before the imaging condition is applied.

3.4.1 Poststack Imaging

Poststack migration has been routinely used for inexpensive migrations and the development of velocity models. To provide this capability, a poststack imaging condition has been included within Salvo. The imaging condition used for poststack migration uses the exploding reflector concept. By halving the velocity field magnitude, the receiver wavefield is downward propagated and the real component of the wavefield at $t = 0$ is examined to determine the reflector position,

$$I = \Re \left[\int_{-\infty}^{\infty} e^{i\omega t} P_r d\omega \right]_{t=0} = \Re \left[\sum_{n=1}^{n_\omega} P_r \Delta\omega \right] \quad (3.16)$$

where I is the resulting image, P_r is the receiver wavefield, and n_ω is the number frequencies migrated. For the impulse response problem, an impulse occurs at $z = 0$ with a time signal of $a_r(t)$. The returned signal will occur at the same point at which it was sent out, and will be of the form $a_r(t - 2R_0/v)$, where R_0 is the radius of the hemispherical reflector and v is the acoustic velocity. This means that the Fourier transform of the returned signal will be of the form $A_r(\omega)e^{-2i\omega R_0/v}$. The poststack image is obtained by migrating this signal. For two-dimensional migration with initial conditions

$$P_r(0, 0, \omega) = P_r(r, \theta, \omega) = A_r(\omega) \exp[-2i\omega R_0/v],$$

the receiver wavefield is given by

$$P_r(r, \theta, \omega) = \frac{A_r(\omega) \cos(\theta)}{\sqrt{2\pi v r}} \exp \left[-\frac{i2\omega (R_0 - r)}{v} \right] \left\{ \sqrt{\omega} \exp \left[-i\frac{\pi}{4} \right] \right\}, \quad (3.17)$$

remembering that for poststack migration the velocity is halved, $v/2$. The correlation imaging condition results when P_r is substituted into Eq. (3.16) (Imaging conditions for Salvo runs are specified in the *MIG_SPLIT* parameter input deck; see Section 5.4.1.3). Multiplying I (before taking the real part) by the inverse of the phase function,

$$\left\{ \frac{1}{\sqrt{\omega}} \exp \left[i\frac{\pi}{4} \right] \right\},$$

removes any changes to the original receiver wavelet shape. The phase function is a “half” time derivative, and thus this imaging condition is termed the derivative

imaging condition. The original wavelet shape is imaged in the migrated section when the “half” derivative is removed.

Similarly in three-dimensional migration, the receiver wavefield is given by

$$P_r(r, \theta, \omega) = \frac{A_r(\omega) \cos(\theta)}{2\pi v r} \exp \left[-\frac{i2\omega (R_0 - r)}{v} \right] \{-i\omega\}. \quad (3.18)$$

The correlation imaging condition for three-dimensional migration is obtained by substituting P_r into Eq. (3.16). Multiplying P_r by the inverse of the time derivative, i/ω , will remove the phase function, and allow the original wavelet to be imaged in the migrated section (*i.e.*, derivative imaging condition).

3.4.2 Prestack Imaging

Prestack migrations can provide better images than poststack migrations, however the prestack imaging condition requires both the source and the receiver wavefields. The imaging condition for prestack migration is based on the concept that upgoing and downgoing waves correlate at reflection points and are related to the reflectivity by $\mathcal{R} = P_r/P_s$. However P_s can be small or zero which will make the reflectivity extraordinarily large. Claerbout [8] suggests the correlation relationship

$$I = \Re \left[\int_{-\infty}^{\infty} P_r P_s^* d\omega \right] = \Re \left[\sum_{n=1}^{n_\omega} P_r P_s^* \Delta\omega \right] \quad (3.19)$$

where P_s^* is the complex conjugate of the source wavefield. This relation has the same phase but does not preserve the amplitudes of the reflectivity coefficient. For the impulse response problem, the initial conditions for the prestack source and receiver wavefields are

$$P_s(0, 0, \omega) = A_s(\omega)$$

(assuming the source was set off at $t = 0$) and

$$P_r(0, 0, \omega) = P_r(r, \theta, \omega) = A_r(\omega) \exp[-2i\omega R_0/v].$$

For two-dimensional migration, the migrated wavefields for the source and receiver are

$$P_s(r, \theta, \omega) = \frac{A_s(\omega) \cos(\theta)}{\sqrt{2\pi v r}} \exp \left[-\frac{i\omega r}{v} \right] \left\{ \sqrt{\omega} \exp \left[i\frac{\pi}{4} \right] \right\}$$

and

$$P_r(r, \theta, \omega) = \frac{A_r(\omega) \cos(\theta)}{\sqrt{2\pi v r}} \exp \left[-\frac{i\omega (2R_0 - r)}{v} \right] \left\{ \sqrt{\omega} \exp \left[-i\frac{\pi}{4} \right] \right\}.$$

The correlation imaging condition is obtained by substituting P_s and P_r into Eq. (3.19),

$$I = \Re \left[\sum_{n=1}^{n_\omega} \frac{A_s(\omega) A_r(\omega) \cos^2(\theta)}{2\pi v r} \exp \left[-\frac{i2\omega (R_0 - r)}{v} \right] \{-i\omega\} \Delta\omega \right]. \quad (3.20)$$

The phase function dependence is removed when each term in the summation is multiplied by i/ω as completed in the derivative imaging condition. However to obtain the original wavelet of the receiver wavefield, the amplitude spectrum additionally needs to be set to unity (*i.e.*, $A_s(\omega) = 1$), which is simply a Dirac impulse or can be approximated by an Ormsby wavelet.

For three-dimensional migration, the migrated wavefields are

$$P_s(r, \theta, \omega) = \frac{A_s(\omega) \cos(\theta)}{2\pi v r} \exp \left[-\frac{i\omega r}{v} \right] \{i\omega\}$$

and

$$P_r(r, \theta, \omega) = \frac{A_r(\omega) \cos(\theta)}{2\pi v r} \exp \left[-\frac{i\omega (2R_0 - r)}{v} \right] \{i\omega\}.$$

Again the correlation imaging condition is obtained by substituting P_s and P_r into Eq. (3.19),

$$I = \Re \left[\sum_{n=1}^{N_\omega} \frac{A_s(\omega) A_r(\omega) \cos^2(\theta)}{(2\pi v r)^2} \exp \left[-\frac{i2\omega (R_0 - r)}{v} \right] \{-\omega^2\} \Delta\omega \right]. \quad (3.21)$$

To obtain the derivative imaging condition, the phase function (*i.e.*, two time derivatives) is removed by multiplying the terms inside the summation by $-1/\omega^2$. Once again to obtain the original wavelet of the receiver wavefield, the amplitude spectrum additionally needs to be set to unity (*i.e.*, $A_s(\omega) = 1$).

In all the imaging conditions, Eqs. (3.17), (3.18), (3.20), and (3.21), the phase shift term,

$$\exp \left[-\frac{i2\omega (R_0 - r)}{v} \right],$$

has the frequency doubled, 2ω . Thus if a migrated image is stretched back to the time domain using the velocity field, it should be stretched by half the velocity in to order match the amplitude spectrum of the original receiver data.

3.4.3 Deconvolution Imaging

The previous two imaging conditions are approximations to the imaging condition based on the reflectivity concept, $\mathcal{R} = P_r/P_s$. The obvious problem is the division by zero or a small value of the source wavefield. To avoid this problem, the complex conjugate of P_s is multiplied to the numerator and denominator, and a small value, ϵ , is added to the denominator to produce the deconvolution imaging condition,

$$I = \Re \left[\sum_{n=1}^{n_\omega} \frac{P_s^* P_r}{P_s^* P_s + \epsilon} \Delta\omega \right].$$

The deconvolution imaging condition has been implemented for two- and three-dimensional prestack migration. The difficulty with this imaging is the selection of ϵ . Since each seismic data set has its own recording amplitudes, deconvolution spiking history, *etc.*, a general value for ϵ is nearly impossible to obtain.

3.4.4 Reduction of Temporal Wrap Around

Although the reduction of temporal wrap around is not an imaging condition, it does directly use the generated image to reduce temporally wrapped events. The idea of reducing the energy in the wave propagation once it has been imaged seems to have been first introduced by Kjartansson [18] and later referenced by Li [20]. The basic method for poststack migration is to remove energy from the wavefield by subtracting out the average image from each frequency migrated

$$P_r(x, y, z, \omega) = P_r(x, y, z, \omega) - \frac{I(x, y, z)}{n_\omega}, \quad (3.22)$$

where $P_r(x, y, z, \omega)$ is the receiver wavefield, $I(x, y, z)$ is the image from the correlation imaging condition, and n_ω is the number of frequencies used in the imaging. For prestack imaging, we suggest the following method be applied to each frequency

$$P_r(x, y, z, \omega) = P_r(x, y, z, \omega) - \frac{I(x, y, z)}{n_\omega P_s^*(x, y, z, \omega)} \quad (3.23)$$

where $P_s(x, y, z, \omega)$ is the source wavefield, $I(x, y, z)$ is the image from the correlation imaging condition. The average image is only subtracted from the receiver wavefield. The source wavefield should not be reduced, since the receiver wavefield should be correlated at all depths with an unperturbed source wavefield.

The reduction of temporal wrap around is a compile-time option, and is selected by the NOWRAP flag (see Section 5.2.2). Examples of the NOWRAP flag are shown in Figures 4 and 5. The first two impulses are true events, while the remaining impulses are temporally wrapped events due to a short sampling length. In Figure 4, it is shown that the aliased impulses in the poststack image are reduced very effectively. However there is some loss in amplitude for the first two impulses which are true events. In Figure 5, it is shown that the aliased impulses are removed, but additional noise is produced. As in the poststack case, there is an amplitude loss in the true events.

3.5 Numerical Characteristics

To better understand the numerical characteristics of the Salvo algorithm, a development of the amplitude ratio, G , will be useful. From the amplitude ratio, the dispersion and dissipation properties can be found, which in turn can indicate appropriate selection of the migration parameters, such as grid spacing. When employing the MOFS, an amplitude ratio for each fractional step can be determined and then G can be constructed for the entire depth step,

$$G = G_t G_x G_y,$$

where G_t , G_x , and G_y are the amplitude ratios of the thin-lens, x -diffraction, and y -diffraction terms, respectively. This development neglects phase corrections from

Figure 4: Application of reduction of temporal wrap around to two-dimensional poststack data. The left image is the original image; the center image is the image with the reduction of the temporal wrap around; the right image is the difference between the left and center images.

Figure 5: Application of reduction of temporal wrap around to two-dimensional prestack data. The left image is the original image; the center image is the image with the reduction of the temporal wrap around; the right image is the difference between the left and center images.

either the Li filter or the Graves and Clayton filter. Starting with the thin-lens term,

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} P$$

the solution is

$$P(x, y, z + \Delta z, \omega) = P(x, y, z, \omega) \exp\left(\frac{i\omega}{v} \Delta z\right),$$

indicating an amplitude ratio of

$$G_{il} = \exp\left(\frac{i\omega}{v} \Delta z\right).$$

The amplitude ratio for the diffraction term can be determined by discretizing the x -diffraction term with finite differences as shown in Appendix B.2,

$$P_{l,m}^{n+1} - P_{l,m}^n + \Delta x^2 B_x [\delta_x^2 P_{l,m}^{n+1} - \delta_x^2 P_{l,m}^n] = \Delta x^2 A_x i [\delta_x^2 P_{l,m}^{n+1} + \delta_x^2 P_{l,m}^n] \quad (3.24)$$

where

$$A_x = \pm \frac{\alpha_1}{2} \frac{v}{\omega} \frac{\Delta z}{\Delta x^2}, \quad B_x = \lambda + \frac{\beta_1}{\Delta x^2} \frac{v^2}{\omega^2},$$

$P_{l,m}^n$ is the wavefield at grid point (l, m, n) or location $(x = l\Delta x, y = m\Delta y, z = n\Delta z)$, and δ_x^2 is the central finite-difference operator for the second-partial derivative. The wavefield, $P_{l,m}^n$, can be represented with individual Fourier components and amplitudes, k_x , and $(P_{l,m}^0)_{k_x}$,

$$(P_{l,m}^n)_{k_x} = (P_{l,m}^0)_{k_x} (G_x)^n \exp(i k_x \Delta x l).$$

Additionally noting that

$$\delta_x^2 (P_{l,m}^n)_{k_x} = -4 (P_{l,m}^0)_{k_x} (G_x)^n \exp(i k_x \Delta x l) \frac{\sin^2\left(\frac{k_x \Delta x}{2}\right)}{\Delta x^2},$$

the amplitude ratio can be determined by substituting these two equations into Eq. (3.24),

$$G_x = \frac{1 - 4 [B_x + A_x i] \sin^2\left(\frac{k_x \Delta x}{2}\right)}{1 - 4 [B_x - A_x i] \sin^2\left(\frac{k_x \Delta x}{2}\right)}.$$

The amplitude ratio for the y -diffraction term, G_y , can be similarly found (replacing x 's with y 's). The amplitude ratio for an entire depth step is

$$G = \exp\left(\frac{i\omega}{v} \Delta z\right) \left[\frac{1 - 4 [B_x + A_x i] \sin^2\left(\frac{k_x \Delta x}{2}\right)}{1 - 4 [B_x - A_x i] \sin^2\left(\frac{k_x \Delta x}{2}\right)} \right] \left[\frac{1 - 4 [B_y + A_y i] \sin^2\left(\frac{k_y \Delta y}{2}\right)}{1 - 4 [B_y - A_y i] \sin^2\left(\frac{k_y \Delta y}{2}\right)} \right]$$

or

$$G = \exp\left(\frac{i\omega}{v}\Delta z\right) \begin{bmatrix} \chi_x + i\eta_x \\ \chi_x - i\eta_x \end{bmatrix} \begin{bmatrix} \chi_y + i\eta_y \\ \chi_y - i\eta_y \end{bmatrix}$$

where

$$\begin{aligned} \chi_x &= 1 - 4B_x \sin^2\left(\frac{k_x \Delta x}{2}\right), & \eta_x &= -4A_x \sin^2\left(\frac{k_x \Delta x}{2}\right), \\ \chi_y &= 1 - 4B_y \sin^2\left(\frac{k_y \Delta y}{2}\right), & \text{and } \eta_y &= -4A_y \sin^2\left(\frac{k_y \Delta y}{2}\right). \end{aligned}$$

3.5.1 Dissipation

The dissipation of the algorithm can be determined by examining the magnitude of G . Values less than unity indicate a dissipative system, where magnitudes, $(P_{l,m}^n)_{k_x}$, are decaying; values equal to unity indicate a non-dissipative system; and values greater than unity indicate an unstable system with magnitudes, $(P_{l,m}^n)_{k_x}$, increasing. The dissipation can be found by noting $\left|\exp\left(\frac{i\omega}{v}\Delta z\right)\right| = 1$.

$$|G| = \left| \frac{\chi_x + i\eta_x}{\chi_x - i\eta_x} \frac{\chi_y + i\eta_y}{\chi_y - i\eta_y} \right| = 1$$

Thus Salvo algorithm is a non-dissipative system (right on the edge of stability). One should note that this applies to the internal finite-difference points. Finite-difference equations for boundary conditions can still lead to unstable systems, and the internal grid points cannot dissipate any oscillations generated by the boundary conditions.

3.5.2 Dispersion

The dispersion of the algorithm (*i.e.*, wave numbers propagate at different speeds or have different phases) can be analyzed with a reference phase to compare against. There are two choices for this reference: the paraxial wave equation, Eq. (3.2), and the split wave equation, Eq. (3.9). The phase difference between the paraxial wave equation and the finite-difference implementation of the split wave equation will describe the total dispersion of the solution. The phase difference between the split wave equation and the finite-difference implementation of the split wave equation will describe the numerical dispersion of the solution. Some investigators have described the phase difference between the paraxial wave equation and the split wave equation as anisotropic dispersion, but this can be misleading since numerical dispersion can have anisotropic behavior. The main sources of dispersion between the paraxial wave equation and the split wave equation are the square-root approximation, and the operator splitting.

The plane-wave solution to the paraxial wave equation, assuming constant velocity, can be simply written

$$P(x, y, z, \omega) = \sum_{k_x=-\infty}^{\infty} \sum_{k_y=-\infty}^{\infty} P_{k_x, k_y} \exp(ik_x x + ik_y y + ik_z z) \quad (3.25)$$

where k_z is determined by $k_z = \pm \sqrt{\omega^2/v^2 - k_x^2 - k_y^2}$. This can be verified by plugging back into Eq. (3.2). Therefore the phase shift for the paraxial wave equation due to a depth step of Δz is

$$\phi_p = \pm \sqrt{\omega^2/v^2 - k_x^2 - k_y^2} \Delta z. \quad (3.26)$$

Similarly, the solution to the split wave equation will have the form of Eq. (3.25) where k_z is determined by

$$k_z = \pm \left(\frac{\omega}{v} - \frac{\alpha_1 \frac{v}{\omega} k_x^2}{1 - \beta_1 \frac{v^2}{\omega^2} k_x^2} - \frac{\alpha_1 \frac{v}{\omega} k_y^2}{1 - \beta_1 \frac{v^2}{\omega^2} k_y^2} \right),$$

and the phase shift for the split wave equation is

$$\phi_s = \pm \left(\frac{\omega}{v} - \frac{\alpha_1 \frac{v}{\omega} k_x^2}{1 - \beta_1 \frac{v^2}{\omega^2} k_x^2} - \frac{\alpha_1 \frac{v}{\omega} k_y^2}{1 - \beta_1 \frac{v^2}{\omega^2} k_y^2} \right) \Delta z. \quad (3.27)$$

For reference, the phase shift for the unsplit-wave equation, Eq. (3.3), is simply

$$\phi_u = \pm \left(\frac{\omega}{v} - \frac{\alpha_1 \frac{v}{\omega} (k_x^2 + k_y^2)}{1 - \beta_1 \frac{v^2}{\omega^2} (k_x^2 + k_y^2)} \right) \Delta z.$$

The phase for the finite-difference implementation of the split wave equation can be determined by finding the phase of the amplitude ratio, G ,

$$\phi_{fd} = \phi_{tl} + \tan^{-1} \left[\frac{\text{Im}(G_x)}{\text{Re}(G_x)} \right] + \tan^{-1} \left[\frac{\text{Im}(G_y)}{\text{Re}(G_y)} \right].$$

After some manipulation, one can find

$$\phi_{fd} = \left(\pm \frac{\omega}{v} \Delta z + \tan^{-1} \left[\frac{2\chi_x \eta_x}{\chi_x^2 - \eta_x^2} \right] + \tan^{-1} \left[\frac{2\chi_y \eta_y}{\chi_y^2 - \eta_y^2} \right] \right). \quad (3.28)$$

The phase error associated with the total dispersion can be determined by subtracting Eqs. (3.26) and (3.28) as shown in Figure 6 using linear compact FD coefficient. However the phase error variations with respect to depth step size, Δz ; grid spacing, Δx ; velocity, v ; and propagation angle, θ ; are difficult to extract from these equations. A series expansion for the two-dimensional propagation will allow

(a) Various depth step size

(b) Various grid spacing

(c) Various velocities

(d) Various propagation angles

Figure 6: Phase error variation (between Eqs. (3.26) and (3.28) for the 65° equation) with respect to (a) depth step size, Δz , (b) grid spacing, Δx , (c) velocity, v , and (d) propagation angle, θ . Solid black line is the reference conditions ($v = 3000$ m/s, $\Delta x = \Delta z = 15$ m, azimuth = 0° , and $\theta = 45^\circ$) in all figures.

a comparison coefficients of like terms of K_x^n where $K_x = v k_x / \omega$, and $0 < K_x < 1$. The following representations for the paraxial wave equation and finite-difference wave equation, respectively, can be found

$$\phi_{p,x} = \pm \frac{\omega \Delta z}{v} \left(1 - \frac{1}{2} K_x^2 - \frac{1}{8} K_x^4 - \frac{1}{16} K_x^6 - O(K_x^8) \right) \quad (3.29)$$

and

$$\begin{aligned} \phi_{fd,x} = \pm \frac{\omega \Delta z}{v} & \left[1 - \alpha_1 K_x^2 + \left(-\alpha_1 \beta_1 + \alpha_1 \left(\frac{\omega \Delta x}{v} \right)^2 \left(\frac{1}{12} - \lambda \right) \right) K_x^4 \right. \\ & + \left(-\alpha_1 \beta_1^2 + \frac{\alpha_1^3 \Delta z^3}{12} + \left(\frac{\omega \Delta x}{v} \right)^2 \left(\frac{\alpha_1 \beta_1}{6} - 2\alpha_1 \beta_1 \lambda \right) + \left(\frac{\omega \Delta x}{v} \right)^4 \left(\frac{-\alpha_1}{360} + \frac{\alpha_1 \lambda}{6} - \alpha_1 \lambda^2 \right) \right) K_x^6 \\ & \left. + O \left[(K_x^8), \left(\frac{\omega \Delta z}{v} \right)^5, \left(\frac{\omega \Delta x}{v} \right)^6 \right] \right]. \end{aligned} \quad (3.30)$$

The leading term in Eqs. (3.29) and (3.30) is the thin-lens term. For the paraxial equation there is a single dependence on the depth step size, Δz . The finite-difference wave equation also has this dependence but additionally has terms like, $\Delta z^3 K_x^6$, which effects the phase error with depth step size, as shown in Figure 6(a). Not surprisingly, as Δz increases so does the phase error, however too small a Δz also causes a phase error. This is due to a balance of errors between Δz and other error terms. By reducing Δz , the phase errors no longer cancel each other. Blindly reducing Δz may not decrease the phase error. As a rule of thumb, the depth step and the grid spacing should be roughly set equal, $\Delta z \approx \Delta x$.

The phase error variation with respect to grid spacing, Δx , is shown in Figure 6(b). Up to ~ 60 Hz there is only a $\pm 2\%$ phase error, however as the grid spacing increases so does the phase error. Once again this error is relatively small over the range 0-75 Hz when $\Delta x \approx \Delta z$. The velocity always appears in the denominator, and therefore the phase error decreases as the velocity increases as shown in Figure 6(c). Even for very small velocities, frequencies between 0-50 Hz are determined very accurately. The phase error dependence on propagation angle (variation of K_x between 0-1 in Eqs. (3.29) and (3.30)) is shown in Figure 6(d). Frequencies below 75 Hz are accurately propagated up to $\sim 45^\circ$. Above this angle the phase errors can be high. This is because as $K_x \rightarrow 1$, the differences in series coefficients become more significant than when $K_x \rightarrow 0$.

In Figure 7, the total phase error variation with respect to azimuthal angles is shown. Although there is an increasing phase error associated with increasing azimuthal angle, it is fairly constant over the range of 0-75 Hz. Brzostowski and Black [6] noted that constant phase error with respect to frequency leads to misplaced reflectors but does not cause dispersion (dispersion is a phase error dependence on frequency). Very little dispersion occurs over 0-75 Hz as can be seen in Figure 7, but there will be some reflector misplacement. Note: These calculations, Figures. 6, 7, 8, and 9, do not include any filter corrections (Li or Graves and Clayton).

Figure 7: Phase Error variation (between paraxial wave equation and finite-difference implementation for the 65° equation) with respect to various azimuths. Solid line is determined using the reference conditions listed in the figure.

(a) Various frequencies

(b) Various azimuths

Figure 8: Phase error due to numerical dispersion for the 65° equation with (a) various frequencies and (b) various azimuths.

In Figure 8, the phase error due to numerical dispersion (*i.e.*, the phase difference between Eqs. (3.27) and (3.28)) is shown with respect to propagation angle for several frequencies and azimuthal angles. For propagation angles below 45° , very little dispersion is present, but above 45° there are large variations in the phase error which produce substantial dispersion.

3.5.3 Compact Finite Differences⁹

Compact finite differences (compact FD) [3, p. 44] are primarily used to obtain higher-order accuracy (typically *4th* order over *2nd* order) while remaining on a small three-point stencil. Claerbout's "1/6 trick" [8, p. 262] is an application of compact FDs. The main drawback of compact FDs is that it makes the method implicit. However since the Salvo algorithm is implicit, compact FDs only add one more operation to the solves. To implement the compact FDs, the second-order spatial derivatives are replaced with

$$\frac{\partial^2}{\partial x^2} = \frac{\delta_x^2}{1 + \lambda \Delta x^2 \delta_x^2}$$

where δ_x^2 is the second-order central-difference operator,

$$\delta_x^2 u = \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}$$

and $\lambda(\omega)$ is the compact FD coefficient to increase the finite-difference accuracy ($\lambda = 1/12$ for a Taylor series expansion will produce a fourth-order scheme). The accuracy has been increased but since δ_x^2 still operates on a three point stencil the dependency on neighboring grid points has not changed. Additional details of the incorporation of compact FD into the finite-difference equations is provided in Appendix B.2.

The importance of compact FD can be seen in Figure 9. The numerical dispersion is greatly reduced with compact FD. Without compact FD, Figure 9(b), numerical dispersion can make a migrated image very difficult to interpret. Four functions for the compact FD coefficient, $\lambda(\omega)$, have been implemented. See Section 5.2.2 on how to compile with each coefficient function, -DLAMBDA. The coefficient functions are

Zero compact FD coefficient

$$\lambda(\omega) = 0$$

Constant compact FD coefficient, [8, p. 262]

$$\lambda(\omega) = \frac{1}{4} - \frac{1}{\pi^2} \approx 0.14867881$$

⁹Subroutines: pipe_*, f_pipe_*, tridag, f_tridag, filter_li; Files: pipe_*.F, f_pipe_*.F, tridag.F, f_tridag.F, filter_li.F

Figure 9: Phase error due to numerical dispersion for the 65° equation (a) with compact finite differences and (b) without compact finite differences.

Linear compact FD coefficient, [29, p. 278]

$$\lambda(\omega) = \min \left[\frac{1}{4} - \frac{1}{\pi^2}, \frac{1}{12} + \frac{\zeta}{\pi} \left(\frac{1}{4} - \frac{1}{\pi^2} - \frac{1}{12} \right) \right]$$

Quadratic compact FD coefficient

$$\lambda(\omega) = \min \left[\frac{1}{4} - \frac{1}{\pi^2}, \frac{1}{4 \sin^2 \zeta} - \frac{1}{4 \zeta^2} \right]$$

where $\zeta = \omega \Delta x \sin 65^\circ / (2v)$.

To illustrate the effectiveness of these compact FD functions, a simple post-stack Gaussian impulse response was migrated and the resulting image is shown in Figure 10. The migration domain is defined so that the impulse impinges on the left-hand boundary to test the absorbing boundary conditions. A slight reflection from the boundary can be seen. During this migration, the Li filter is applied at every depth step. As noted earlier, the Li filter corrects for the square-root approximation, the operator-splitting approximation, and numerical dispersion exactly in a constant velocity medium. Isolating the effects of numerical dispersion on the migrated section can be accomplished by (1) applying the Li filter at the first depth step (not at every step) to remove evanescent energy generated by the square-root approximation, and (2) migrating a two-dimensional data set to remove any need for the operator-splitting approximation.

The results of this experiment are shown in Figure 11 for each of the compact FD functions. When the zero compact FD coefficient is used, shown in Figure 11(a), the classic dispersion pattern (*i.e.*, ripples trailing behind a primary

Figure 10: Two-dimensional poststack Gaussian impulse response with the Li Filter applied at every depth step.

wave) can be seen around 45° propagation angle. Above this angle the primary wave amplitude decreases leading to reduced amplitudes in the dispersed waves. Below this angle, phase errors are very small as shown in Figure 9(b), and produces little dispersion. Constant compact FD coefficient, shown in Figure 11(b), improves the dispersion but over compensates and produces a leading dispersion wave. The linear compact FD coefficient, shown in Figure 11(c), has some dispersive waves around 60° propagation angle, but is a great improvement over zero compact FD coefficient. The quadratic compact FD coefficient, shown in Figure 11(d), is slightly better than the linear coefficient.

One should note that numerical dispersion has an accumulative effect with each depth step, and therefore deeper reflectors will have more dispersion than shallower reflectors with the same dip. Periodic application of the Li filter will help remove numerical dispersion.

3.6 Phase Encoding¹⁰

A summary of phase encoding¹¹ is provided here for reference, but further details are contained in Romero *et al.* [27]. The primary motivation for phase encoding is to reduce the overall computational costs associated with finite-difference, frequency-space migration for an entire survey. The image for the survey is the sum of the individual images for each shot migration, *i.e.*,

$$I(x, y, z) = \sum_n I_n(x, y, z)$$

where n is the shot index. The overall cost is clearly related to the product of the cost of an individual shot migration and the number of shot migrations to

¹⁰Phase encoding is patented under U.S. Patent 6,021,094.

¹¹Subroutines: `encode_init`, `apply_encoding`; Files: `readTraces.c`, `encode.c`, `getShotInfo.c`

(a) Zero compact FD

(b) Constant compact FD

(c) Linear compact FD

(d) Quadratic compact FD

Figure 11: Numerical dispersion of two-dimensional poststack Gaussian impulse response for various compact finite difference coefficients, $\lambda(\omega)$, with the Li filter phase correction applied only at the first depth step. Figure 10 shows the image without numerical dispersion.

complete the survey. If several shots are combined, the overall cost to migrate the survey would be reduced by roughly the number of shots combined. Shots can be combined by noting that the differential equation, Eq. (3.9), used to obtain $P_s(x, y, z, \omega)$ and $P_r(x, y, z, \omega)$ is linear, and therefore the differential equation can be solved with the summed wavefields,

$$\hat{P}_s(x, y, z, \omega) = \sum_n a_n P_{s,n}(x, y, z, \omega)$$

and

$$\hat{P}_r(x, y, z, \omega) = \sum_n a_n P_{r,n}(x, y, z, \omega)$$

in nearly the same amount of time it takes to compute any of the individual terms $P_s(x, y, z, \omega)$ or $P_r(x, y, z, \omega)$ (where a_n is some function of ω to be specified shortly). This can be completed by determining the summed values at the surface

$$\hat{P}_s(x, y, z = 0, \omega) = \sum_n a_n P_{s,n}(x, y, z = 0, \omega)$$

and

$$\hat{P}_r(x, y, z = 0, \omega) = \sum_n a_n P_{r,n}(x, y, z = 0, \omega).$$

The functions $\hat{P}_s(x, y, z, \omega)$ and $\hat{P}_r(x, y, z, \omega)$ are obtained by solving the one-way wave equations with the summed values at the surface.

However a complication arises when $\hat{P}_s(x, y, z, \omega)$ and $\hat{P}_r(x, y, z, \omega)$ are processed with the imaging condition. Cross terms between unrelated shots produce artifacts in the image. For example, when two shots are summed and the imaging condition is applied through Eq. (3.19), the image $I(x, y, z)$ is given by

$$\begin{aligned} I(x, y, z) &= I_1(x, y, z) + I_2(x, y, z) \\ &+ \sum_k \left[P_{s,1}^*(x, y, z, \omega_k) P_{r,2}(x, y, z, \omega_k) + P_{s,2}^*(x, y, z, \omega_k) P_{r,1}(x, y, z, \omega_k) \right]. \end{aligned}$$

The first two terms on the right-hand side are the same as the individual images obtained through standard migration and imaging; however, the other terms are undesirable cross terms.

These unwanted cross terms can be eliminated or greatly reduced by encoding each frequency with a different phase. Let $\gamma_1(\omega)$, and $\gamma_2(\omega)$ be two different phase functions for two shots to be combined and migrated. The two shots will be encoded with these two different phase functions in order to migrate the summed data at the surface, $\hat{P}_s(x, y, z = 0, \omega)$ and $\hat{P}_r(x, y, z = 0, \omega)$,

$$\hat{P}_s(x, y, z = 0, \omega) = e^{i\gamma_1} P_{s,1}(x, y, z = 0, \omega) + e^{i\gamma_2} P_{s,2}(x, y, z = 0, \omega)$$

and

$$\hat{P}_r(x, y, z = 0, \omega) = e^{i\gamma_1} P_{r,1}(x, y, z = 0, \omega) + e^{i\gamma_2} P_{r,2}(x, y, z = 0, \omega).$$

Due to linearity, the migrated wavefields will also have this form. Introducing these combined wavefields into the imaging condition, the image will be

$$I(x, y, z) = I_1(x, y, z) + I_2(x, y, z) + \sum_k \left[P_{s,1}^*(x, y, z, \omega_k) P_{r,2}(x, y, z, \omega_k) e^{i(\gamma_2(\omega_k) - \gamma_1(\omega_k))} + P_{s,2}^*(x, y, z, \omega_k) P_{r,1}(x, y, z, \omega_k) e^{i(\gamma_1(\omega_k) - \gamma_2(\omega_k))} \right].$$

Note that the phase functions γ_1 and γ_2 only appear in the cross terms, not in the terms producing I_1 and I_2 . The functions, $\gamma_1(\omega)$ and $\gamma_2(\omega)$, should be chosen so that when the frequencies are summed over k , the energy from the unwanted terms will be dispersed throughout the whole image or be shifted out of the region of interest. When two or more shots are summed together, a similar procedure holds, however the cross terms become more numerous and therefore more difficult to shift or disperse.

Several phase encodings are available through Salvo (see Section 5.4.1.2). As an example of phase encoding, random encoding was applied to the Marmousi model with different numbers of shots combined (or shots per migration). The results are shown in Figure 12. All images are shown at the same scale and no amplitude scaling has been applied to any of these images. As the number of shots is increased, the cross-term energy grows in strength. The computational costs for these images decreases dramatically because fewer migrations are required; the image in Figure 12(d) took about 1/240 the time to generate versus the original image. Although these images get progressively worse even migrating all 240 shots produces an image with significant structure.

3.7 Reduced-Source Frequency Migration¹¹

The goal of reduced-source frequency migration is to significantly reduce total runtime, while maintaining an image of quality similar to that provided by full migration. The strategy is to migrate a subset of the source frequencies and use an analytic solution to extrapolate the remaining frequencies via an inexpensive autoregression process¹². Close to 50% of the migration work could be eliminated with this technique. Even the more modest goal of migrating one half of the source frequencies could provide up to a 25% reduction in the total runtime. The majority of the theoretical portions of the following discussion is based on notes by Louis Romero.

3.7.1 Analytic Solution

Assuming a single arrival, the pressure function for a point source has the general form of

¹¹Subroutines: freq_reduction_calc; files: freq_reduction.c freq_reduction_ops.F

¹²Implementation and coding completed by Rob Gjertsen and Mike Holzrichter.

Figure 12: The Marmousi model using random encoding and combining (a) 4, (b) 16, (c) 60, and (d) 240 shots per migration. On the left is the encoded image and on the right is difference between the image on the left and the original image without encoding.

$$P(x, \omega) = A(\omega, x) \exp(i\omega\phi(x)). \quad (3.31)$$

Specifically for two dimensions, the solution is $P(x, \omega) = \sqrt{\omega}A(x) \exp(i\omega\phi(x))$ and for three dimensions, it is $P(x, y, \omega) = \omega A(x, y) \exp(i\omega\phi(x, y))$ (see Appendix B.7 for further details). For the 5° equation in one dimension there is excellent correspondence between the source pressure produced by Salvo and the analytic solution. Matching the two-dimensional 15° equation results from Salvo with the analytic solution is more difficult.

One difficulty is that at a finite depth z , the amplitude of the analytic solution for a Dirac delta function does not decay as the solution approaches the side boundaries of the migration domain. This causes the problem modeling an impulse which continues to infinity with a finite calculation. Thus reflections off the boundaries can cause difficulties in predicting the source wavefield. To handle this difficulty, Salvo offers the option of a spatially distributed source. The strength of the distributed source varies spatially as a Gaussian distribution and is controlled by the width parameter γ . Unlike the point source, the analytic solution for the spatial Gaussian distribution has an amplitude decay at the boundaries.

Consider the two-dimensional 15° equation

$$\frac{\partial p}{\partial z} = ik \left(p + \frac{1}{2k^2} \frac{\partial^2 p}{\partial x^2} \right) \quad (3.32)$$

with the initial spatial Gaussian source distribution

$$p(x, 0) = \frac{1}{\sqrt{2\pi\gamma}} \exp \left(-\frac{1}{2} \left(\frac{x}{\gamma} \right)^2 \right). \quad (3.33)$$

The analytic solution $p(x, z)$ has the form

$$p(x, z) = \exp(ikz) \sqrt{\frac{\gamma}{\beta}} \exp \left(-\frac{x^2}{2\beta} \right) \quad (3.34)$$

for constant velocity where $\beta = \gamma^2 + \frac{iz}{k}$ and $k = \omega/v$. The term $\exp(-x^2/2\beta)$ ensures exponential decay as x increases.

When a spatial Gaussian distribution as in Eq. (3.33) is used, undesirable decay of the envelope can occur at the grid points offset laterally from the source. To counter this, the weighted Gaussian for the source was also implemented. The weighted Gaussian uses a different Gaussian distribution for each frequency, ω , by substituting

$$\gamma(\omega) = \frac{c_0}{\omega} \epsilon_0$$

for γ in Eq. (3.33) where ϵ_0 and c_0 are user supplied constants. If c_0 is 2π times the velocity then $\frac{c_0}{\omega}$ is the wavelength and ϵ_0 expresses the width of the Gaussian in wavelengths.

3.7.2 Scaling Requirements for Gaussian Source

We hope the Gaussian solution closely approximates the point source solution with the exception that its amplitude decay at the boundaries. Based on the analytic solution, guidelines for effective use of the Gaussian source can be derived. There are three basic scaling requirements that must be satisfied in order to accurately approximate a point source.

1. The depth is large enough that the finite source region becomes small. This depth is a function of both z and γ .
2. The horizontal extent of the region is large enough that the solution decays to zero at the boundaries. This is satisfied if γ is large enough, where γ depends on z and k .
3. The grid spacing Δx is small enough to accurately model the source.

Appropriate relations for γ can be derived from the analytic solution which ensure the above requirements are satisfied. While Equation (3.34) is the analytic solution for the 15° degree paraxial equation, the scaling arguments hold for the 45° and 65° equations as well.

The solution will not mimic the phase or amplitude of a point source solution unless

$$\frac{z}{k} \gg \gamma^2.$$

In order for the solution to decay sufficiently at $x = L$, where L is the half width of the migration region, the following must hold

$$\frac{\gamma^2 L^2}{\gamma^4 + z^2/k^2} \gg 1.$$

Assuming the first constraint holds, the above simplifies to

$$\frac{k\gamma L}{z} \gg 1.$$

The spatial sampling of the source induced by the grid spacing needs to be fine enough to resolve the spatial source. If the source is not well sampled, the numerical result will differ greatly from the analytic solution. The Gaussian source will be well sampled if

$$\gamma > \Delta x.$$

These constraints are compatible provided

$$\gamma \ll L.$$

An additional relationship between γ and Δx has been observed. Grid spacing induces an effective lower bound on γ . If $\gamma \ll \Delta x$ or a point source is used, the result is similar to the result when a Gaussian source is used with $\gamma \approx \Delta x$. For sufficiently small γ almost all of the source energy is concentrated at one grid point. Due to the spatial sampling of the grid, there is very little difference in the energy distribution whether this threshold $\gamma_0 \approx \Delta x$ or the actual γ is used. Since the distributions are very similar, the outputs are very similar.

3.7.3 Comparison of Actual and Model Spectra

The effectiveness of reduced-source migration depends on how well Salvo can estimate the source spectrum given the values for a few frequencies. Salvo estimates the rest of the spectrum using the assumption that the source spectrum has the form $P(x, \omega) = \sqrt{\omega} A(x) \exp(i\omega\phi(z))$. It is useful to examine how well this model fits the actual source spectrum.

The analytic solution predicts the source spectrum will have a $\sqrt{\omega}$ envelope modulated by a complex exponential. Based on observations of actual spectra, deviations from this model generally fit into two types. High frequency deviations can be characterized as “noise” and the rest appear as changes to the envelope. It was found that there is good correspondence when $\frac{k\gamma L}{z}$ was large. Based on this expression, the following are five heuristics for predicting how well the spectrum fits the model.

1. Increasing γ reduces the noise.
2. Increasing L , the domain half-width, reduces the noise and improves the envelope.
3. Low frequencies are more susceptible to noise than high frequencies.
4. Deviation from the model increases with depth, provided $\gamma^2 \ll \frac{z}{k}$.
5. The envelope decays more quickly with lateral offset for high frequencies than for low.

Figure 13 shows the real part of the point source spectrum for an impulse problem. The spectrum clearly oscillates within an envelope which increases as $\sqrt{\omega}$. The oscillation within this envelope is a sinusoid with a noticeable amount of “noise.”

Figure 14 illustrates the first heuristic. This figure differs from Figure 13 in that a Gaussian source has been used with $\gamma = 4dx$. Figure 14 is cleaner than Figure 13. The Gaussian source can produce spectra which better match the model than those from a point source. Figure 15 is the spectrum using a Gaussian source with $\gamma = 16dx$. Figure 15, with the larger value of γ , is less noisy than Figure 14, illustrating the first heuristic. However, large values of γ cause the envelope to flatten out because the source no longer approximates a point source.

Figure 13: Real component of source wavefield for an impulse problem.

Figure 14: Real component of distributed Gaussian source ($\gamma = 4dx$).

Figure 15: Real component of distributed Gaussian source ($\gamma = 16dx$).

Figure 16: Real component of source wavefield for a large migration domain.

Figure 17: Real component of source wavefield at 150 depth steps into the earth.

Figure 16, illustrating the second heuristic, shows the result when the size of the migration domain of Figure 13 is made four times as large. The result is a spectrum which is a very good fit with the model. While increasing the size of the migration domain quickly nullifies any computational savings from reduced source, Figure 16 supports the idea behind the Gaussian source that the solution should decay before reaching the boundary. In this case the boundary was so far away from the source that the source could not interact with it and a clean spectrum was produced.

The third heuristic is most clearly illustrated by Figures 13 and 14. The noise level is higher for low frequencies than for high frequencies. Since higher frequencies are less susceptible to noise, they should be used instead of low frequencies to estimate $A(x)$ and $\phi(x)$.

Figure 17 shows the spectrum for the same model as Figure 13 except at a point deeper in the earth. The envelope has deteriorated noticeably, illustrating the fourth heuristic.

Figure 18: Real component of source wavefield offset by 125m from distributed Gaussian source.

Figure 18 illustrates the fifth heuristic. Figure 18 is produced in the same manner as Figure 14 except that it is taken from a point off to the side rather than directly below the source. The Gaussian source has exponential decay and the rate of decay increases with frequency. The analytic solution predicts decay that looks like $\exp(-x^2 k^2 \gamma^2 / z^2)$, assuming that $\gamma^2 \ll z/k$. This decay is removed in x if the point source is broadened as a function of $1/k$.

3.7.4 Estimating the Source Spectrum

In two dimensions, reduced-source migration assumes the source spectrum fits the model

$$P(\omega) = \sqrt{\omega} A(x) \exp(i\omega\phi(x)). \quad (3.35)$$

A few select source frequencies are migrated by the paraxial wave equation. These frequencies start at $\omega_0 = n_0 \Delta\omega$ and have a total of n_{ws} frequencies. The values computed for this subset are then used in the autoregression calculations to estimate $A(x)$ and $\phi(x)$ in Eq. (3.35). Equation (3.35) is then used to estimate values for the remaining frequencies. The method by which Salvo estimates $A(x)$ and $\phi(x)$ is discussed in more detail in this subsection.

First, $\phi(x)$ is estimated, and a system of equations is formed at each grid point. Let $\omega_k = k\Delta\omega$, $P_k = P(\omega_k)$ and

$$\tilde{P}_k = \frac{P_k}{A(x)\sqrt{\omega_k}} = \frac{P_k}{|P_k|} = \exp(i\omega_k\phi(x)).$$

Using $\alpha = \exp(i\Delta\omega\phi(x))$, a recurrence relation can be devised with

$$\tilde{P}_{k+1} = \tilde{P}_k \alpha.$$

Repeating the above for each pair of adjacent frequencies forms an over determined system of m equations for $m + 1 = n_{ws}$ frequencies. The resulting system has the form

$$\begin{aligned}\tilde{P}_{n_0+1} - \tilde{P}_{n_0}\alpha &= 0 \\ \tilde{P}_{n_0+2} - \tilde{P}_{n_0+1}\alpha &= 0 \\ &\vdots \\ \tilde{P}_{n_0+m} - \tilde{P}_{n_0+(m-1)}\alpha &= 0,\end{aligned}$$

which can be expressed simply as

$$\vec{v} - \vec{u}\alpha = 0.$$

The least squares solution for this system, $\vec{u}\alpha = \vec{v}$, is then

$$\alpha = \frac{u^H v}{u^H u}.$$

Since $\alpha = \exp(i\Delta\omega\phi(x))$, solving for $\phi(x)$ gives

$$\phi(x) = \frac{\ln \alpha}{i\Delta\omega}.$$

Once $\phi(x)$ is calculated for each grid point, the source pressure $P(\omega)$ is determined except for the constant multiplier $A(x)$ which can be determined with a least squares solution (or simpler technique) using the known values P_k .

3.7.5 Parallelization and Cyclic Distribution

Parallelization of reduced-source migration presents new difficulties for the Salvo algorithm. Poor load balancing can occur because migrated source frequencies are poorly distributed and how many frequencies are migrated per processor. With the Li filter, there is a nonlinear dependence of work as related to frequency. Thus the frequency distribution across processors can be selected to improve the load balance (see Section 4.3.1.1). However the frequencies selected for reduced-source migration may all reside on one processor, increasing the load imbalance, or may reside on separate processors, increasing the interprocessor communication required for the autoregression.

Two steps were taken to address the parallelization difficulty. First, the cyclic distribution was introduced (see Section 4.3.1.1). This assures that all frequencies on a processor are separated by the same number of $\delta\omega$'s, simplifying the autoregression. However this frequency distribution is still not optimal. Second, the autoregression is performed using only those frequencies local to a processor, then

the local values are averaged globally to obtain the final value. This approach was selected to obtain a fast implementation and has several drawbacks. Performing autoregression locally and averaging has some numerical accuracy questions, and the approach has poor scalability characteristics (*i.e.*, need at least two frequencies per processor for autoregression). However for relatively few processors, this implementation may be useful versus nothing. For the results shown here, no parallel processing was used with the reduced-source migration.

3.7.6 Numerical Results

Reduced-source migration was evaluated by applying it to the Marmousi dataset. All 240 shots were migrated via three different ways:

1. all source frequencies with a point source (traditional migration),
2. all source source frequencies with a weighted Gaussian spatial source distribution ($\gamma(\omega) = \lambda/2$),
3. reduced-source migration (four source frequencies) with a weighted Gaussian spatial distribution source distribution ($\gamma(\omega) = \lambda/2$).

This choice of test runs allows the effect of the weighted Gaussian source and reduced-source migration to be isolated from each other. Salvo was run in a serial mode, hence $A(x)$ and $\phi(x)$ were obtained through a global rather than local autoregression process.

The images are characterized by a small fraction of values which are large relative to the rest. Most reflections are variations in the smaller magnitude values. The data were gained and clipped aggressively to enhance the visibility of reflections in a low resolution gray scale image. The images were plotted with comparable gain and clipping. The difference images show less contrast because the magnitude of the difference is small relative to the maximum amplitudes of the differenced images.

Figure 19 shows the output when the Marmousi dataset was migrated in the traditional manner; all source frequencies were migrated and started out at a single grid point. This image is the standard against which the other images are compared.

Figure 20 shows the effect of using a weighted Gaussian source (while still migrating all source frequencies). The difference between Figure 20 and Figure 19 is shown in Figure 21. The images with and without the weighted Gaussian source are comparable. The difference between them decreases with depth because at a distance the weighted Gaussian source becomes more like a point source.

Figure 22 shows the result of reduced-source migration when only four source frequencies were migrated. Reduced-source migration produced an image which is quite similar to the image when all frequencies are migrated. The difference

Figure 19: Full migration with point source.

Figure 20: Full migration with weighted Gaussian source.

Figure 21: Difference between full migration with point source and full migration weighted Gaussian source.

Figure 22: Reduced-source migration (four frequencies) with weighted Gaussian source.

Figure 23: Difference between reduced-source migration with weighted Gaussian source and full migration with weighted Gaussian source.

between Figure 22 and Figure 20 is shown in Figure 23. There is no difference in the top fourth of the image because all source frequencies were migrated down to this point (*i.e.*, $k_{start} = 200$).

These results show that the quality of the image produced by reduced-source migration is good. From these images it is apparent that neither the weighted Gaussian source distribution nor reduced-source migration caused any reflectors to disappear or be misplaced.

The reduced-source migration images above were produced on a SGI workstation. Without reduced-source migration each depth step required 0.19 seconds, while 0.14 seconds per depth step were required for the reduced-source migration. Reduced-source yielded a 25% improvement in total execution time. This is for unoptimized code. With further code refinements, even better performance should be realized.

3.7.7 Open Issues

There are several areas related to reduced-source migration which deserve further attention but currently have not been investigated. Future work should include:

- implementing the three-dimensional solution,
- exploring parallelization issues, in particular (a) efficiently performing autoregression when frequency parallelism is used and (b) quality of images when performing autoregression on frequencies local to a processor,
- possibly handling multiple arrivals as follows

$$P(x, \omega) = A_1(\omega, x) \exp(i\omega\phi_1(x)) + A_2(\omega, x) \exp(i\omega\phi_2(x)) ,$$

- considering the effects of vertical velocity variations,
- tuning the reduced-source code for better performance, *e.g.*, using table lookups for calculations involving $\ln$ and $\exp$,
- using previous and/or neighboring estimates of $A(x)$ and $\phi(x)$ to constrain new estimates,
- improving user friendliness when deciding which source frequencies to use by making it easier to determine the total number of frequencies and source energy content for each frequency,
- using α directly and not calculating ϕ thereby removing the need for $\ln$ and $\exp$, and
- injecting analytic source solution in constant velocity media after a few depth steps to remove difficulties with finite differences and a spatial impulse.

Figure 24: Flow chart of the main driver routine.

4 Coding and Implementation

In this section, questions about the coding and implementation within Salvo are addressed. The coding can be basically broken down into corresponding processes handled by Salvo. A flowchart of the driver routine¹³ is shown in Figure 24. Salvo first reads the parameter input deck (Section 5.4.1) and determines all the processes requested for the run. Currently three processes can be completed: the initialization of Salvo (*INITIAL*), the reading of the traces data (*READ_TRACES*), and the migration of the wavefields (*MIG_SPLIT*). These processes can be repeated in the input parameter file to complete multiple migrations in a single execution. In the following sections, the coding and implementation of these processes are discussed.

4.1 Initialization of Salvo¹⁴

The initialization of Salvo by the *INITIAL* process (Section 5.4.1.1) involves the allocation and initialization of various data structures used during the migration and *READ_TRACES* process. Specifically, this involves initializing the global migration domain information and deriving the local x - y (source and receiver) domain information based on the processor x - y grid coordinates. The compute and I/O partition communicators are also formed in this stage, while the FFT communicators are formed in the *READ_TRACES* initialization phase (see Section

¹³Subroutine: main; File: driver.c

¹⁴Subroutines: initial; Files: initial.c, distribution_methods.c

4.2.1). The frequency spectrum is determined along with the frequency distribution to processors in the ω direction (processor tower); the frequency distributions include: simple linear, reverse linear, wrap, oscillate, and cyclic (see Table 1). Finally, the compute nodes allocate the data arrays/volumes for migration and traces for the *READ_TRACES* process.

4.2 Reading the Trace Data¹⁵

The objective of the *READ_TRACES* process of the Salvo algorithm is to read in the source and receiver trace data and prepare it for migration. The *READ_TRACES* process involves three major phases: (1) reading the time trace data from a parallel file system and distributing it to all the processors, (2) transforming the time traces to frequency traces via FFTs, and (3) redistributing or “transposing” the frequency traces to the array form needed by the migration. This process is applied once for the source trace and again for the receiver traces. Each of these phases is described in detail in the following subsections.

The *READ_TRACES* process is extremely I/O intensive so that reading the trace data (*i.e.*, phase (1)) completely dominates the runtime of this process. To minimize the overall runtime and reduce the critical path, it is necessary to continuously perform I/O in this process, while all computation and processing of the trace data is performed as much as possible in the background. This decoupling of the I/O and processing steps is accomplished in Salvo through the use of separate processor partitions: an I/O processor partition and a compute processor partition. Determining the best balance between the number of nodes in the I/O and compute partitions is discussed further in Section 4.2.6. The I/O processor partition handles reading the trace data, while the compute partition performs all other processing asynchronously (and in the background). The overlap of I/O and the computation is done via asynchronous communication between the I/O and compute partitions. The I/O and computation could be also effectively decoupled with asynchronous I/O calls, but many file systems do not provide this functionality or perform it poorly. Asynchronous I/O is achieved within Salvo by using an I/O partition. Another alternative is the use of threads to perform I/O, but difficulties persist in the use of threads and MPI, eliminating this option.

For the *READ_TRACES* process, code robustness and stability is ensured via strict message passing controls. Each send performed has a matching receive already posted; this avoids providing buffers for messages which cannot be processed immediately. A synchronization is performed among the compute nodes after each major phase in the *READ_TRACES* process to ensure that message buffers are clear of all messages before starting the next phase; this again is done to reduce system memory usage. A double buffering scheme with a specified block size is used by the compute nodes to allow the best possible overlap of I/O and computation,

¹⁵Subroutines: *readTraces*, *readTracesIO*; Files: *readTraces.c*, *readTracesIO.c*

which also strictly controls memory usage.

Phase encoding is also performed in the *READ_TRACES* process. If multiple shots are encoded in the migration, then all the steps of this process must be repeated for each shot encoded. Essentially, each of the encoded shots must be summed into the appropriate frequency array/volume (for both the source and receiver) that is used in the migration.

The responsibilities of the compute partition and I/O partition are as follows. The reading of trace data from the parallel file system is the sole responsibility of the I/O partition. In this process the role of the I/O partition is that of a data server, while the compute partition acts as the client. The I/O processors wait on asynchronous messages to process from the compute partition. The compute partition continuously requests blocks of traces from the I/O partition, and performs any necessary processing of the data, until all of the traces have been read in.

The processing sequence for the partitions is as follows:

1. Initialize communicators (Section 4.2.1).
2. Read a block of traces (Section 4.2.2).
3. Scatter the traces to all nodes in the compute partition (Section 4.2.3).
4. Compute frequency traces with FFTs (Section 4.2.4).
5. Scatter or distribute frequencies of each trace to correct compute node (Section 4.2.5) and perform phase encoding (if specified).
6. Repeat sequence again from step 2 until all data is read.

The data flow and execution flow of the *READ_TRACES* processing sequence is summarized in Figures 25 and 26, respectively. Each of these steps is discussed in greater detail in the following sections. The analytic execution time for each step is also presented.

4.2.1 *READ_TRACES* Initialization Step¹⁶

The primary initialization task involves generating the set of communicators required for the *READ_TRACES* process; a communicator is a subset of nodes from a set of user nodes that forms a logical grouping. The compute and I/O partition communicators are created to partition all the user nodes into those nodes that perform I/O verses those that only perform computations. The set of processors in the compute-partition communicator is further subdivided into a set of FFT communicators to facilitate performing parallel I/O. Figure 25 further clarifies this subdivision of processors with an example configuration including a nine processor

¹⁶Subroutines: *readTraces*, *readTracesInit*, *encodeInit*; Files: *readTraces.c*, *readTracesInit.c*, *encode.c*

Figure 25: Data flow diagram for the *READ_TRACES* process. The cubes on the left represent the compute partition, which is split into two FFT communicators in this example, with the two (white) FFT root nodes being responsible for receiving the trace data I/O. The FFT root nodes either read the data directly from disk (when an I/O partition is omitted), or receive the data from a node in the I/O partition/communicator. The I/O partition uses two I/O nodes and is represented by the two white cubes on the right.

Figure 26: Flow diagram of *READ_TRACES* process.

compute partition ($np = p_x p_y p_w = 9$), and a two processor I/O partition ($p_{io} = 2$). Two FFT communicators are derived from the compute partition which have five and four nodes, respectively.

When an I/O partition is not specified in the *INITIAL* section of the input deck ($p_{io} = 0$; see Section 5.4.1.1), then a subset of nodes from the compute partition are used as I/O nodes; specifically these are the FFT communicator root nodes. These nodes have an associated I/O communicator. This configuration however does not allow the overlap of I/O and computation, and a load imbalance occurs because the root nodes are overloaded by their dual role as I/O and compute nodes.

The number of FFT communicators is generally based on the number of input streams or disks as indicated in Figure 25. Using an I/O partition, the number of FFT communicators either equals the number of I/O nodes specified, p_{io} , or the number of disk files specified, whichever is smaller. If there is no I/O partition, then the number of FFT communicators equals the number of disk files specified in the input deck; the root FFT nodes in this situation read the data directly from disk and perform these tasks. The FFT communicators also divide the compute partition nodes as evenly as possible for load balancing.

In addition to setting up the necessary communicators for the *READ_TRACES* process, there is some initialization and set up for the phase encoding, and all large memory allocations are also performed in the initialization. Other initialization is performed by the *INITIAL* process, refer to Section 5.4.1.1 and Section 4.1 for full details.

4.2.2 Reading Traces Step¹⁷

The primary function of this step is to read in blocks of traces, where each block of traces is a set of traces that requires at most *blocksize* bytes. *Blocksize* is a user supplied parameter which assists in realizing optimal I/O performance for the platform's file system. A *blocksize* of 64 Kb works well for the Intel Paragon. In the case with an I/O partition, traces are read by I/O nodes in *blocksize* chunks and the root node of each FFT communicator waits for and receives data from its designated I/O node. If there is still additional data to read, then a prefetch request is initiated for another block of trace data (allowing computation to overlap with I/O). Specifically, the root node in each FFT communicator asynchronously requests from its designated I/O node (in the I/O partition) the next set of traces. In the case of no I/O partition, the data is read directly from disk by the root node in each FFT communicator.

As mentioned earlier, I/O nodes in their role as servers wait on asynchronous requests from the compute partition. Each I/O node has a designated node in the compute partition that it interacts with; specifically, a root node in an FFT communicator. Currently the requests recognized by the I/O nodes are a request

¹⁷Subroutines: `readTraces`, `requestTraceData`, `readTraceData`, `readTracesIO`, `generate_src_distribution`; Files: `readTraces.c`, `readTracesIO.c`, `readTraceData.c`, `gaussian_dist.c`

for additional input traces or a quit signal. A quit signal is sent to the I/O partition when all of the compute nodes have finished reading the set of traces.

The model for the execution time is broken up into two phases. The first phase, reading a block of traces from disk, requires

$$t = \alpha_d + \beta_d b \quad (4.1)$$

seconds, where α_d is the disk latency, $1/\beta_d$ is the bandwidth of the disk, and b is the number of bytes requested. The second phase (with an I/O partition) is the time to send the data from an I/O node and uses

$$t = \alpha_c + \beta_c n_t g_t \quad (4.2)$$

seconds with α_c as the communication latency, $1/\beta_c$ as the communication bandwidth, n_t is the number of time samples, and g_t is the size in bytes of a time sample.

When a spatial Gaussian source distribution is selected in the *READ_TRACES* process (Section 5.4.1.2), the source signal is introduced over a range of grid points centered at (x_s, y_s) . Normally, the source signal is introduced into the migration domain at a single grid point at (x_s, y_s) . For a given frequency, the relative amplitudes vary **spatially** as a Gaussian. However, the phase of each frequency are the same as in the point source case.

Salvo has two options for spatial Gaussian source distributions: (1) using a distribution where the width parameter, γ , is the same for all frequencies, or (2) a weighted distribution where the width parameter, γ , is inversely proportional to frequency, *i.e.*, proportional to wavelength. In the case where a fixed value of γ is used, the relative amplitudes vary **spatially** as

$$e^{-\frac{1}{2} \frac{(x-x_s)^2}{\gamma^2}}$$

independent of ω . For the weighted Gaussian, there is a different width parameter, $\gamma_n = \gamma(\omega_n)$, for each frequency, ω_n , and the relative amplitudes for a given frequency vary as

$$e^{-\frac{1}{2} \frac{(x-x_s)^2}{\gamma_n^2}}$$

The width parameter varies with frequency as $\gamma_n = \frac{c}{\omega_n} \epsilon$, where c is the velocity and ϵ is a width parameter. If c is the velocity of the medium at the surface, $\epsilon/2\pi$ expresses the width parameter in wavelengths.

4.2.3 Scattering Time Traces Step¹⁸

Once a block of traces is in memory, the traces are scattered to all nodes in the compute partition. Specifically, each root node in an FFT communicator evenly distributes traces to the other nodes in the FFT communicator.

¹⁸Subroutines: scatterData; Files: scatterData.c

The execution time for this one-to-all communication operation using a standard recursive bisection algorithm is

$$t = p_f(\alpha_c + \beta_c \frac{b n_t g_t}{2i}) \quad (4.3)$$

seconds, where p_f is the number of nodes in the FFT communicator.

4.2.4 Computing the Frequency Traces¹⁹

The computation of the frequency traces is performed by all the nodes in the compute partition (in all the FFT communicators). Each processor performs a one-dimensional FFT of each trace that it holds. The execution time for the standard real-to-complex FFT is

$$T_{FFT} = \gamma [K n_t \log n_t + C] \quad (4.4)$$

seconds with constants K and C . Whenever possible, optimized vendor FFT routines are used for the Fourier transform, although the FFTPACK routines from NetLib (<http://netlib.cs.utk.edu/>) are provided as a default method for maximum code portability.

4.2.5 Redistributing the Frequency Traces²⁰

In this phase, the compute partition must reorder the data so that it is in the correct format for the migration stage. This is accomplished by having every processor in the compute partition scatter or distribute frequencies for its traces to the other compute nodes, and copy received frequency data to its own frequency array volume.

The frequency array volume is partitioned with respect to space and/or frequency as indicated in Figure 28, where each processor has approximately $\frac{n_x n_y n_\omega}{p_x p_y p_\omega}$ grid points. Based on the processor grid partitioning in the x , y , and ω directions, the x domain interval and the y domain interval of any processor can be determined. With this information, the processors globally determine the number of traces that each processor should send and receive based on the (x, y) coordinates of the traces and the selected frequency distribution. The frequency distribution indicates how a frequency ω is mapped onto p_ω processors in the ω dimension of the processor grid (see Table 1 for information on frequency distributions).

For the runtime communication analysis, this step involves an “almost” all-to-all communication phase. Each compute processor in a specific tower communicator determines the number of trace messages it should receive and posts the asynchronous receives. A compute processor then sends a subset of each frequency

¹⁹Subroutines: `fftData`; Files: `fftData.c`

²⁰Subroutines: `scatterFrequencies`; Files: `scatterFreq.c`

trace to the appropriate processor based on the (x, y) location, which determines the tower, and the frequency mapping that provides the specific processor level in the tower. This is performed for each trace that a compute processor owns. Finally, each subset frequency trace is received and inserted (added into the frequency array volume, if phase encoding is used). A lower bound estimate of the execution time (assuming only frequency parallelism) is

$$t = \sum_{i=1}^{\log p_w} (\alpha_c + \beta_c \frac{b n_w g_w}{2^i}) \quad (4.5)$$

seconds, and to simplify later analysis the following linear scatter formula is used

$$t = b p_w (\alpha_c + \beta_c \frac{n_w g_w}{p_w}). \quad (4.6)$$

A slight improvement for some architectures may involve sending the entire frequency trace to a specific base node in a tower (for each trace) for the initial step, and performing an one to all redistribution in each tower independently as the second step; this two step process can be used to provide an estimate of the runtime when there is spatial parallelism.

4.2.6 *READ_TRACES* Optimal Execution Time

The *READ_TRACES* process is dominated by the I/O operations to read in the trace data, so the run time is minimized when the communication and computations are performed entirely in the background by the compute nodes; this is most efficiently performed when the time spent by an I/O node equals the time spent by an associated set of compute nodes in an FFT communicator. The processor configuration required to achieve this situation is then determined by finding the optimal ratio of compute nodes to I/O nodes (or the number of I/O nodes) given a set of specific architectural and problem dependent parameters [26].

This configuration is found by equating the time spent by the I/O nodes to read the traces, Eq. (4.1), and transfer the traces to the compute partition, Eq. (4.2), with the time spent by the compute partition to distribute the traces, Eq. (4.3), FFT the traces, Eq. (4.4), and scatter the frequencies, Eq. (4.6). The resulting system is

$$\alpha_d + \beta_d b(n_t g_t) + \alpha_c + \beta_c b(n_t g_t) = \quad (4.7)$$

$$p_f(\alpha_c + \beta_c \frac{b n_t g_t}{p_f}) + \frac{b}{p_f} (T_{FFT} + p_w(\alpha_c + \beta_c \frac{n_w g_w}{p_w})) \quad ,$$

and is simplified by using T_{FFT} for the time to FFT the traces. The optimal number of compute nodes, C_{opt} , is then found by solving the quadratic equation for p_f :

$$\begin{aligned}
C_{opt} &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \\
a &= \alpha_c + \beta_c \frac{b n_t g_t}{p_f} \\
b &= -(\alpha_d + \beta_d b n_t g_t + \alpha_c + \beta_c b n_t g_t) \\
c &= b(T_{FFT} + p_\omega(\alpha_c + \beta_c \frac{n_\omega g_\omega}{p_\omega})).
\end{aligned} \tag{4.8}$$

This model for the optimal ratio of compute to I/O nodes, C_{opt} , is closely matched by experimental results obtained on an Intel Paragon, and additional information on the model and validation experiments can be found in Oldfield *et al.* [26].

4.3 Migration

The main computational kernel of the Salvo algorithm is the migration which consumes the majority of the runtime. The downward propagation of the source and receiver wavefields is completed through the method of fractional steps where separate terms are calculated such as the thin lens term, the x and y diffraction terms, and the phase correction filter. These terms are computed during each depth step and require the reading of the velocity model and writing of the seismic image. However these I/O operations slow down the migration. To eliminate this cost, the use of an I/O partition can hide the I/O through asynchronous reads and writes. The overall computational flow is shown in Figure 27. In the following sections, more details are supplied for the parallelization employed in the computational kernel, and the velocity reading and image writing of the I/O partition.

4.3.1 Parallelization

There are two types of parallelism in Salvo: frequency parallelism and spatial parallelism. These two parallelisms are illustrated by the processor cube in Figure 28, and can be used individually or in combination. In frequency parallelism, each processor migrates a subset of the frequencies, and very little communication is required. This communication is limited to distributing the velocity model to all processors at the beginning of a migration step, and constructing the image by summing images across the processors. These messages are passed through tower communicators where every processor has a common spatial domain. The limitations of frequency parallelism are the number of processors must be smaller than the number of frequencies being processed, and each processor must store in memory the full velocity plane and at least one frequency. These problems can be mitigated when necessary by spatial parallelism in which the x - y domain is distributed among processors. However a new problem arises – the tridiagonal solves in the x and y directions must be parallelized. This introduces additional

Figure 27: Flow chart for the migration process, *MIG_SPLIT*, and the relationship between the compute and I/O partitions. Possible communication to the I/O partition is indicated by (*), and the letter indicate the source code for that operation.

Figure 28: Processor decomposition of the computational domain (n_x, n_y, n_ω) for (a) frequency parallelism with p_ω processors, and for (b) spatial parallelism with p_x and p_y processors.

communication and therefore produces lower parallel efficiencies than frequency parallelism. Column and row communicators provide a means to pass tridiagonal information between processors which have frequencies in common.

4.3.1.1 Frequency Parallelism²¹ Because very little communication is required with frequency parallelism, very high parallel efficiencies are possible. Load imbalances can be reduced by properly distributing the frequencies to each processor. Figure 28(a) suggests that the easiest decomposition of frequencies to each processor is a simple linear distribution (the first n_ω/p_ω frequencies go to the first processor, the second n_ω/p_ω frequencies go to the second processor, *etc.*). This simple linear distribution assumes that each frequency, ω , requires the same amount of work (*i.e.*, there are no load-balancing issues). This is true for most of the Salvo algorithm; the exception is the Li filter.

Within the Li filter, evanescent energy is simply zeroed out to remove it from the calculation, while the remaining energy is phase corrected, requiring additional work. This disparity in work load is frequency dependent, and is roughly linear with respect to frequency, however it does have some non-linear dependence. To better load balance the processors, several frequency distribution schemes have been implemented, which are listed in Table 1. The frequency distribution is specified at compile time through `-DFREQDIST` (see Section 5.2.2).

Reverse Linear distribution places high frequencies on low numbered processors and low frequencies on high numbered processors. Wrap distribution matches the

²¹Subroutine: `simple_distribution`, `reverse_distribution`, `freq_wrap`, `freq_osc`; file: `distribution_methods.c`

Table 1: Frequency distribution schemes for frequency parallelism.

Distribution Scheme	Description
Simple Linear	The first n_w/p_w frequencies go to the first processor, the second n_w/p_w frequencies go to the second processor, <i>etc.</i>
Reverse Linear	The reverse ordering of Simple Linear
Wrap	The first processor gets the first and last frequencies, the second processor gets the second and next to last frequencies, <i>etc.</i> Once the last processor has its frequencies, the cycle starts again with the first processor.
Oscillate	The last processor gets the last frequency, the next to last processor gets the next to last frequency, <i>etc.</i> Once the first processor has its frequency, the first processor gets the next frequency and the second processor gets the next frequency, <i>etc.</i>
Cyclic	First processor receives frequencies $0, p_w, 2p_w, \text{etc.}$, while the i th processor receives frequencies $i, p_w + i, 2p_w + i, \text{etc.}$

least and most work intensive frequencies on one processor and the next pair of least and most work intensive frequencies on the next processor, and so on. This is continued and each processor may get several pairings. This pairing is optimal if the work load is a linear function of frequency and there are no remaining frequencies. If there are remaining frequencies, they are distributed so that no processor has more than one additional frequency compared to the other processors. A problem with this distribution is that any remaining frequencies (load imbalances) are the middle frequencies, and are relatively expensive compared to low frequencies.

The Oscillate distribution tries to have any remaining frequencies be the relatively inexpensive low frequencies. By distributing the high frequencies first, the last frequencies to be distributed are the low frequencies. The distribution is a simple v-cycle starting with the highest frequency to the last processor, and continues to the first processor (the down cycle). The first processor gets the next frequency and continues to the last processor (the up cycle). This up and down cycle continues until all the frequencies are distributed.

The Cyclic distribution maps the frequencies in a cyclic manner among the processor levels; for this mapping, processor p_0 has frequencies $0, p_w, 2p_w, \dots$, while processor p_i has frequencies $i, p_w + i, 2p_w + i, \dots$, *etc.*, when there are p_w processors. This mapping scheme is not as efficient as the wrap or oscillate mappings, but is designed specifically for use with the reduced-source frequency migration (see Section 3.7).

If the Li filter is not used, any of the distribution schemes work equally well. However if the Li filter is used, either the Wrap or Oscillate distribution schemes should be used for efficiency reasons. Both of these methods have performed well during normal runs.

Figure 29: Schematic of pipeline tridiagonal solves.

4.3.1.2 Spatial Parallelism²² Spatial parallelism introduces a new problem related to the tridiagonal solves. Since tridiagonal solves begin at one boundary and forward sweep to the other (and backward sweep across the domain again), spatial decomposition requires communication between the processors during the serial process of tridiagonal solves. The parallelization of a single tridiagonal solve can not achieve high parallel efficiencies because of this serial nature. However in the paraxial solution, this difficulty can be circumvented because there are many tridiagonal systems to solve. Not only are there one tridiagonal solve for each grid line ($n_x + n_y$ solves), but there are also tridiagonal solves for each frequency, ω , for a total of $n_\omega(n_x + n_y)$ tridiagonal solves. To obtain higher parallel efficiencies, a pipeline is setup as schematically shown in Figure 29.

In the first stage of the pipeline, processor zero starts on a set of tridiagonal solves. Once it completes its portion of the forward sweep, it passes the necessary coefficients to processor one. In the second stage of the pipeline, processor one continues the first set of tridiagonal solves, while processor zero starts a second set of tridiagonal solves. This process continues until all of the processors are busy computing forward sweeps. Once processor three completes its first forward sweep, it begins the backward sweep, and passes back through the pipeline the necessary coefficients for the backward sweep. This continues until all the tridiagonal solves have been completed. These four processors in Figure 29 are an example of a column or row communicator for spatial parallelism.

²²Subroutines: tridag, pipe_single, pipe_start, pipe_middle, pipe_return; files: tridag.F, pipe_single.F, pipe_start.F, pipe_middle.F, pipe_return.F

There are two major sources of inefficiencies in the pipelined tridiagonal solves: idle time associated with filling and emptying the pipeline (the end processors are sitting idle waiting for their first forward sweep), and communication time associated with the message-passing between processors (a single tridiagonal solve has a very short message (1 or 2 variables), causing too many message startups). The optimal number of tridiagonal solves in a set is determined by minimizing the overhead time associated with communication and idle time.

To reduce the overhead time, the message sizes can be increased (and therefore reduce the number of startups) by blocking together many tridiagonal solves and passing several forward sweeps and several backward sweeps at once [32]. If too many solves are blocked, the time to fill and empty the pipeline is too long. The overhead time, $T_{over} = T_{comm} + T_{pipe}$, (where T_{comm} is communication time, and T_{pipe} is the idle pipeline time) can be minimized by selecting the number of tridiagonal solves per block, b . First expressions for both T_{comm} and T_{pipe} need to be constructed.

Assuming the problem is decomposed by $p_x \times p_y$ processors in the spatial directions and p_ω processors in the frequency direction, the number of tridiagonal solves completed by each processor is

$$N_{solves} = \left(\frac{n_x}{p_x} + \frac{n_y}{p_y} \right) \frac{n_\omega}{p_\omega}.$$

The time to send forward and backward messages for b solves is $(2\alpha + 24b\beta)$ (2 complex variables, or 16 bytes, for the forward sweep, and one complex variable, or 8 bytes, for the backward sweep), where α is the message latency (seconds) and β is the inverse of the bandwidth (seconds/byte). Thus communicating N_{solves}/b messages, the communication time is

$$T_{comm} = N_{solves} \left(2\frac{\alpha}{b} + 24\beta \right).$$

To characterize the pipeline time, half the time for the end processor to receive its first message will be used,

$$T_{pipe} = [N_{work}b(n_x + n_y)\gamma + (p_x + p_y)(2\alpha + 24b\beta)] / 2,$$

where N_{work} is the number of operations required for one grid point during the tridiagonal solve and γ is the time to complete a floating point operation. The optimal block size, b_{min} , can be determined by constructing T_{over} and solving $\partial T_{over} / \partial b = 0$;

$$b_{min} = \left[\frac{4N_{solves}\alpha}{N_{work}(n_x + n_y)\gamma + 24(p_x + p_y)\beta} \right]^{1/2}.$$

By assuming $n_x = n_y$, $p_x = p_y$ and $N_{work}n_x\gamma \gg 24p_x\beta$, b_{min} simplifies to

$$b_{min} = \left[\frac{4}{N_{work}} \frac{n_\omega}{p_x p_\omega} \frac{\alpha}{\gamma} \right]^{1/2}.$$

Figure 30: Comparison of predicted optimum block size, b_{min} , and computer run times for three sample problems.

It should be noted that b_{min} is not dependent on n_x or β in this expression, thus the blocking size is weakly dependent on the problem size or the computer bandwidth. However it is dependent on the length of the pipeline, p_x , the number of frequencies on each processor, n_w/p_w , and the ratio of the message latency and the time for a floating point operation, α/γ . (The machine dependent parameters α , β , and γ in Salvo are set in *salvo/src/include/machine_speed.inc*).

In Figure 30, three samples problems are shown with varying block size, b . The predicted optimum, b_{min} , compares very well with the minimum determined by computer experiment. One important fact from Figure 30 is that pipelining a single tridiagonal solve is quite expensive and therefore should be avoided. This model for b_{min} has been incorporated into Salvo²³ and may be overridden by a user-supplied value for b_{min} (see Section 5.4.1.3).

4.3.2 Reading the Velocity Model²⁴

The migration portion of the algorithm is dominated by the computation of the wavefield propagation. However the velocity model must read and interpolated for each depth step. This I/O should not hinder the computations, so it is desirable to use an I/O partition to perform I/O in the background maximizing the efficiency

²³Subroutine: *mig_split*; file *mig_split.c*

²⁴Subroutines: *migrate*, *init_vel*, *read_vel*, *process_io_requests*, *extract*, *prepare_plane*; files: *migrate.F*, *init_vel.c*, *read_vel.c*, *process_io_requests.c*, *extract.c*, *prefetch.c*

of the compute partition. Reading the velocity model is divided into four distinct steps:

1. Read the velocity plane for the next depth step, z , from the parallel file system (if necessary).
2. Construct the full velocity plane on each I/O node.
3. Interpolate and send the requested portion of the velocity model to the base node of each tower in the compute partition.
4. Broadcasting the velocity plane up each tower in the compute partition.

A non-blocking request from the base node of each tower in the compute partition initiates the reading of velocity plane. Once the I/O partition receives all the combined individual requests for the next velocity plane, the data for the velocity plane is read from the disks by the I/O partition (step 1). Each depth plane of the velocity model is distributed evenly to each disk in the parallel file system (the x - y plane is partitioned along the y axis as shown in Appendix C.2), so the I/O nodes evenly read from the disks in parallel. The time to read the velocity plane from the parallel file system is

$$t = \alpha_d + \beta_d \frac{n_{v,x} n_{v,y} g}{p_{io}}$$

where α_d is the disk latency, $1/\beta_d$ is the disk bandwidth, $n_{v,x}$ and $n_{v,y}$ are the x and y dimensions of the velocity model plane, g is the number of bytes per velocity element, and p_{io} is the number of nodes performing I/O. If the required depth z is located between two currently stored velocity planes, z_{top} and z_{bottom} , then no read is required from the velocity model. In other words, two planes from the velocity model are maintained so that $z_{bottom} \leq z \leq z_{top}$, which facilitates interpolation in the z direction if the velocity model and migration domain do not have common depth levels. The interpolation in the z direction is a simple linear interpolation on the “slowness” (*i.e.*, the inverse of the acoustic velocity). This produces a velocity plane at the desired depth, z , in the migration algorithm, but still has not been interpolated in the x and y dimensions.

The second step is to combine the planes from each of the I/O nodes to form the full velocity plane. The velocity plane is divided evenly (along the y axis) among each of the I/O nodes in contiguous planes and is combined via an all-to-one gather, *i.e.*, MPI_Allgather [23]. The standard implementation of the all-gather routine is simply a gather followed by a broadcast. Once this operation is complete each I/O node has a copy of the full velocity plane. The time to perform this operation is

$$t = (p_{io} - 1)(\alpha_c + \beta_c \frac{n_{vx} n_{vy} g}{p_{io}}) + (\log_2 p_{io})(\alpha_c + \beta_c n_{vx} n_{vy} g)$$

with α_c representing the network communication latency and $1/\beta_c$ representing the network communication bandwidth.

The third step in the velocity reading stage requires sending each compute node at the base of a tower the requested portion of the velocity plane. Embedded in the original velocity model request is information about which section of the velocity plane is required by that compute node and the grid to place it in. Generally the velocity model is on a much coarser grid than the migration grid. So in this case the velocity plane is interpolated to the correct grid before it is sent to a compute node. For grid points on the interior of the velocity plane, a simple linear interpolation is performed on the “slowness” to obtain the acoustic velocity at the desired migration points. For grid points outside the velocity plane, an extrapolation from the nearest grid points is used. The interpolation of a velocity plane requires

$$t = \frac{1}{p_{io}}(5n_{vx} n_{vy} + 7n_y + 24n_x n_y)\gamma$$

seconds with $1/\gamma$ as the processing rate measured in floating point operations per second and n_x and n_y corresponding to the migration grid dimensions. The time needed to send the extracted velocity plane to a compute node is

$$t = \alpha_c + \beta_c \frac{n_x n_y g}{p_x p_y}$$

seconds, where p_x and p_y are the x and y dimensions of the processor grid.

Finally, the base node of each tower broadcasts its portion of the velocity plane up the tower. The time for this step is

$$t = (\log p_w)(\alpha_c + \beta_c \frac{n_x n_y g}{p_x p_y}).$$

In the situation without an I/O partition, a set of compute nodes are designated as I/O nodes using as many nodes as there are velocity files. These I/O nodes perform the same steps in reading the velocity as in the I/O partition case except that the first two steps are no longer performed in the background (*i.e.*, no longer independently prefetching/reading the data for the next depth step).

4.3.3 Writing or Stacking the Image²⁵

As with the read velocity operation, the writing or stacking of the image plane is performed during the migration of the seismic data. Specifically, after each depth step, the new x - y image plane is written to disk or is stacked (or added) with a previously created image file and then written to disk. The objective again is to perform the I/O in the background as much as possible.

²⁵Subroutines: `migrate`, `write_acc`, `stack_img`, `process_io_request`, `ioWriteImg`, `perform_io_stack`; files: `migrate.F`, `write_acc.c`, `stack.c`, `process_io_request.c`, `write.c`

Writing the image plane to disk is similar to the inverse of the read velocity operation. After the image is summed over all frequencies (*i.e.*, summation to the base node of each processor tower), the base node sends its portion of the image plane to the I/O partition through its designated I/O node. The I/O partition then constructs the full image plane on the root I/O node via an all-to-one gather operation, scatters the full image plane to each node in the I/O partition with a broadcast, and finally writes the image to the file system, with each I/O node independently writing to disk its contiguous subset of the image plane.

The communication time for gathering the image plane from the base node of each tower is

$$t = \alpha_c + \beta_c \left(\frac{n_x n_y g}{p_x p_y} \right).$$

The time required to scatter the entire image plane to each node in the I/O partition is

$$t = (p_{io} - 1) \left(\alpha_c + \beta_c \frac{n_x n_y g}{p_{io}} \right).$$

And finally, the time for each I/O node to write the data to disk is

$$t = \alpha_d + \beta_d \frac{n_x n_y g}{p_{io}}.$$

Through the use of an I/O partition, the I/O can be performed concurrently with the migration, and the only overhead arises through sending the image plane from the compute partition to the I/O partition. Writing the image file is essentially the same for both frequency and spatial parallelism, except that in the case of spatial parallelism multiple tower base nodes will send their portions of the image to the I/O partition instead of a single node sending the entire image plane. For the case without an I/O partition, each base node in the tower communicator writes their individual images to a separate file (resulting in $p_x p_y$ files of size $\frac{n_x n_y}{p_x p_y}$ that partition the spatial domain); this is the only exception to the default partitioning of the file data along the y -axis of the migration domain (Section C.2).

The output image can be stacked on the fly with a previously migrated image. This stacking process involves a similar procedure with the addition that the specified stack file image plane must be read in and stacked or added to the current migration image plane. The stacking process is currently restricted to frequency parallelism (spatial parallelism defaults to writing the image instead of stacking). When using an I/O partition, the base node in the tower communicator sends the full migration image plane to the I/O partition. In the background, the I/O partition reads in the stack file image plane and determines the new domain that results from the stack file and the current migration shot. Using the new domain information, the I/O nodes request the currently migrated image plane from the compute partition, so that it conforms to the file partitioning (along the y -axis of the new file domain) used to write out the new stacked image plane. The I/O

partition reads in the stack file in the optimal manner from disk (to achieve maximum parallel I/O bandwidth), but this data may require redistribution to align it with the new file partitioning for the new domain. This redistribution is designed to obtain maximal I/O bandwidth and to conserve memory usage, but some parallelism is sacrificed in the process; the possibly large memory requirements are given primary emphasis in this situation. For the case without an I/O partition, the tower base node in the compute partition performs the necessary steps for stacking.

The runtime t for performing the stacking operation with an I/O partition of size p_{io} (ignoring the addition operations) is approximately

$$t = [\alpha_c + \beta_c(\frac{n_x n_y g}{p_x p_y})] + [\alpha_d + \beta_d \frac{\tilde{n}_x \tilde{n}_y g}{p_{io}}] + [(p_{io} - 1)(\alpha_c + \beta_c \frac{\tilde{n}_x \tilde{n}_y g}{p_{io}})] + [\alpha_d + \beta_d \frac{\hat{n}_x \hat{n}_y g}{p_{io}}]$$

with $\tilde{n}$ representing the dimensions from the stack file domain and $\hat{n}$ representing the dimensions for the new file domain after stacking; the four terms represent: (1) transfer of data from compute to I/O partition, (2) reading the stack file image plane, (3) redistribution of the stack file data plane (worst case), and (4) writing the new image plane to disk. The first term for the time t is the only one that cannot be performed in the background of the migration computation. Without an I/O partition the runtime t only includes the second and fourth terms (let $p_{io} = 1$).

Figure 31: The Salvo directory tree for the software distribution.

5 Salvo User's Guide

The general purpose of this section is to provide documentation on how to setup, compile and run Salvo. In the following sections, details of the software distribution and Salvo directory structure are provided, steps for compiling and porting the Salvo software are presented, performance results are discussed, and Salvo's input and output data are explained.

5.1 Software Distribution

As documented in Section 1, the Salvo software may be available for distribution through Sandia National Laboratories. The contact information for David Womble and Curtis Ober is in Section 1 and they should be contacted for the current status of the Salvo software.

The Salvo software distribution is contained in a single file which is a compressed tar file, *salvo.tar.gz*, and is approximately ~ 290 Mb in size. The uncompressed files are ~ 325 Mb in size and additional disk space will be needed for seismic data sets. The Salvo distribution should be moved to the desired directory for decompression, '*ungzip salvo.tar.gz*', and extraction, '*tar xvf salvo.tar*'. Figure 31 shows the resulting Salvo directory tree. Most directories include a README file that provides useful information on the contents of the directory.

The *bin/* directory contains the Salvo and utility executables after compilation.

Access to the Salvo executable and utility programs is most convenient if the binary directory are included in the user's path which is set in the *.cshrc* file (e.g., '*set path = (\$path /usr/gjertsen/salvo/bin)*'). The *docs/* directory contains postscript copies of various conference papers and this document, Salvo: Seismic Imaging Software for Complex Geologies.

Detailed documentation on all of the Salvo routines and utilities is provided via man page documentation in the *man/* directory. To utilize the man page facilities the *MANPATH* environment variable (usually set in the *.cshrc* file) should be set to include the directory *salvo/man* (e.g., '*setenv MANPATH \${MANPATH}:/usr/gjertsen/salvo/man*'). Information on a specific Salvo routine or utility is simply obtained on the Unix command line with the command '*man routine_name.*' The man pages are fully installed with the Salvo distribution, but information on regenerating or further customizing the man pages is located in the directory *salvo/man/src*.

The *src/* directory contains the Salvo source code and the Makefiles to compile the code. Information on compiling Salvo is contained in subsequent sections. Directories in *src/* contain source code for initialization and execution control (*driver/* and *init/*), for reading and distributing seismic data (*readTraces/* and *seggio/*), for migration (*mig_split/*), for I/O during migration (*io_partition/* and *read_vel/*), and for miscellaneous tasks and information (*gd1.2/* and *include/*).

The Salvo utilities, located in the *utils/* directory, are used for preprocessing input files to Salvo and post processing the output files from Salvo. These utilities are quite useful but are not meant for industrial processing. All of the utilities are listed in Appendix D.1 with a brief summary of their function. Additional information on each utility can be obtained from the man pages by using '*man utility_name*' on the Unix command line.

Lastly a set of validation problems are provided in the directory, *Validation_Suite/*. This suite of problems contains all the input data (e.g., parameter files, source traces, receiver traces, and velocity model) required by Salvo to run the impulse and Marmousi problems discussed in the validation section, Section 6. Additionally the suite contains reference images to verify a working version of Salvo. The SEG/EAGE Salt model data sets, discussed in Section 6, are not part of the software distribution. Although these data sets are too large to be included, interested people should contact David Womble or Curtis Ober (see pg. 7) for availability of these data sets.

5.2 Compiling and Porting

Before beginning to port and compile Salvo, three other pieces of software are required: a Fortran compiler, a C compiler and MPI. Most computer vendors supply Fortran and C compilers, however a few vendors do not supply a version MPI for their computers. If one is not available, MPICH is a recommended public version which is distributed by Argonne National Laboratories. MPICH can be

downloaded from <http://www.mcs.anl.gov/mpi/index.html> or via anonymous FTP at [ftp.mcs.anl.gov](ftp://ftp.mcs.anl.gov/pub/mpi/mpich.tar.Z) under `/pub/mpi/mpich.tar.Z`.

Once the compilers and MPI are located and setup, Salvo can be compiled. Although the total algorithm is over 48,000 lines of C and Fortran code, compiling Salvo is very simple. Salvo has been ported to a variety of platforms: Intel Paragon, IBM SP2, Cray T3E and T3D, SGI Origin 2000, SUN Ultras, Sandia's CPlant(tm), DEC 8400, and a network of workstations. These ports only took hours to perform and generally obtained very high performance. Generating a Salvo executable is a two step process: setting up the environment file and compiling the Salvo source code with `'make'`.

Environment files, located in `salvo/src/environment/*.env`, define the compilation flags and libraries for the architecture. Before trying to make the Salvo executable, one environment file must be selected for compilation by setting the environment variable, `TARGET`. For example, if an executable for an SGI platform is desired, `'setenv TARGET sgi'` (note: omit the `.env` extension) will set the correct compilation flags and libraries definitions for the Makefile. If the environment variable, `TARGET`, is not set, an error similar to the following will occur when `'make'` is executed.

```
Makefile:33: environments/.env: No such file or directory
make: *** No rule to make target 'environments/.env'. Stop.
```

An example environment file is shown in Figure 32. The `DEFINES` variable contains Salvo compilation flags, while most of the machine dependent compiler flags are specified by the `OPT#` variables. The `OPT#` variables are used for different levels of optimization for different computational kernels. The remainder of the environment file specifies the compilers, libraries, and include (header) files. The last set of statements provides the general compiler commands. This is used unless specific compilation commands are provided for individual files.

If an environment file for the desired computing platform does not exist in `salvo/src/environments`, then the following steps should be followed for the new architecture.

1. Copy an environment file from a similar architecture to the new environment file, e.g., `'cp arch.env new_arch.env.'` Remember to set `TARGET` to the new architecture, `'setenv TARGET new_arch'`.
2. Edit the environment file to configure the platform compilers, optimization flags, include flags, libraries, and desired configuration defines from Section 5.2.2. Changes may be necessary to use the platform specific version of MPI. In addition, the new architecture should be defined in the environment file, i.e., `'DEFINES = -Dnew_arch.'`


```

# =====
#
# Copyright 1998 Sandia Corporation. Under the terms of Contract
# DE-AC04-94AL85000, there is a non-exclusive license for use of
# this work by or on behalf of the U.S. Government.
#
# =====
#
# A listing of DEFINES, their definitions, and settings is provided
# in the environments/README file.
#
# -----
# Salvo Compiler Flags -----
DEFINES1 =
DEFINES2 = $(DEFINES1) -n32 -DSGI -DFORTRAN_UNDERSCORE -DEXPTL
DEFINES = $(DEFINES2) -DEQN=65 -DBCS=4 -DLAMBDA=2 -DDEBUG=0 -DOUTPUT=3
# -----
# Compiler Optimizations -----
OPT8 = -O3 -OPT:div_split=ON:fast_complex=ON:IEEE_arithmetic=2:swp=ON \
      -TARG:platform=ip28:processor=r10000
# These are set for mig_split/Makefile
OPT7 = -O3 -OPT:div_split=ON:fast_complex=ON:IEEE_arithmetic=2:swp=ON \
      -TARG:platform=ip28:processor=r10000
OPT9 = -O3 -OPT:div_split=ON:fast_complex=ON:IEEE_arithmetic=2:swp=ON \
      -TARG:platform=ip28:processor=r10000
# -----
# Includes, Libraries, and Linker -----
# Link in MPICH version of MPI (-32) =====
# INCLUDES = -I./include -I/Net/pecos/wave/acti/mpich/include \
#           -I/Net/pecos/wave/acti/mpich/lib/IRIX32/ch_p4 \
# LIBS      = -L/Net/pecos/wave/acti/mpich/lib/IRIX32/ch_p4 \
#           /usr/lib32/libcomplib.sgimath.a -lmpich -lftn -lm \
#           -lcomplib.sgimath -lmalloc_ss
# LIBS      = -L/Net/pecos/wave/acti/mpich/lib/IRIX32/ch_p4 \
#           /usr/lib32/libcomplib.sgimath.a -lmpich -lfe -lc -lm -lftn
# LINK      = cc -32 -v
# =====
# Link in MPICH version of MPI (-n32) =====
# INCLUDES = -I./include -I/Net/pecos/wave/acti/mpich/include \
#           -I/Net/pecos/wave/acti/mpich/lib/IRIXN32/ch_p4 \
# LIBS      = -L/Net/pecos/wave/acti/mpich/lib/IRIXN32/ch_p4 \
#           /usr/lib32/libcomplib.sgimath.a -lmpich -lfe -lc -lm -lftn
# LIBS      = -L/Net/pecos/wave/acti/mpich/lib/IRIXN32/ch_p4 \
#           /usr/lib32/libcomplib.sgimath.so -lmpich -lfe -lc -lm -lftn
# LINK      = cc -n32 -v
# =====
# Link in SGI's version of MPI =====
# INCLUDES = -I./include
# LIBS      = /usr/lib32/mips4/libmpi.so /usr/lib32/mips4/libcomplib.sgimath.a \
#           -lfe -lc -lm -lftn
# LINK      = cc -n32 -v
# =====
#
# -----
# Archiver -----
ARFLAGS = -cr
AR = ar
# -----
EXIT_CMD = cp salvo ../bin/salvo; rm -f salvo; echo "Done!"
LIBRARY = $(DIRECTORY).a
#####
.SUFFIXES : .c .C .f .F
.c.o :
$(CC) $(CFLAGS) $(INCLUDES) -c $<
.f.o :
$(F77) $(OPT8) $(FFLAGS) $(INCLUDES) -c $<
.F.o :
$(F77) $(OPT8) $(FFLAGS) $(INCLUDES) -c $<

```

Figure 32: A environment file for an SGI Indigo2 with R10000 processor.

3. The native platform FFT libraries can be utilized by configuring the environment file properly and editing the files *salvo/src/readTraces/fftData.c* and *salvo/src/mig_split/filter_li.F*. In *fftData.c* the routine *fftData* uses one-dimensional real-to-complex FFTs, while for *filter_li.F* the routine *phase_correction* uses one-dimensional complex-to-complex FFTs. The native FFT routines should be included so that they are only invoked when '*new_arch*' is defined for the preprocessor.

To compile Salvo, the '*make*' command is invoked in the *salvo/src* directory. The Makefile structure for Salvo is a simple two-level recursive make call. The top-level Makefile includes the *TARGET* environment file, and recursively calls the Makefiles in each of the source directories below *salvo/src*. During the compilation, the dependence of each file is checked to determine which files need recompilation because of their dependence on other files (e.g., include and library files). Each lower-level Makefile creates directory library files, **.a*, for all the object files within its directory. The object files are left in each directory for later recompilation. The linking of all the libraries occurs in the top-level Makefile, which additionally places the Salvo executable in the *salvo/bin* directory. If a complete recompilation is desired, the '*make clean*' command will remove all the object and library files, and the '*make*' command will compile Salvo again.

The syntax checkers, *lint* (for C) and *ftnchek* (for Fortran), were used to improve portability through enforcing standard syntax usage and variable initialization. In general, the I/O tasks are written in C and the computational tasks are written in Fortran. Therefore *lint* generally checks the I/O portions of Salvo and *ftnchek* checks the computational portions of Salvo. The C source code can be checked by setting the environment variable, *TARGET*, to *lint* (i.e., '*setenv TARGET lint*') and recompiling with the '*make*' command. Files with the *.lint* extension will be created for each C file and can be reviewed for any inconsistencies. The Fortran source code can be checked by using the *ftnchek* software (<http://netlib2.cs.utk.edu/fortran/ftnchek.tgz>). A script, *run_ftnchek*, is provided in the *salvo/src/mig_split* directory. This directory contains almost all the Fortran source code. The output from *ftnchek* is placed in *ftnchek.output*.

The following is a summary check list for compiling Salvo: (This list is shown for the environment file, *generic.env*, but others can be substituted.)

1. Set the environment variable, *TARGET*. '*setenv TARGET generic*'
2. Modify the environment file, *generic.env*, for the desired architecture, the native FFTs (if needed) and Salvo capabilities.
3. Edit the Salvo files, *salvo/src/readTraces/fftData.c* and *salvo/src/mig_split/filter_li.F*, to incorporate native FFT libraries (if needed).
4. Compile Salvo with '*make*'.

5. Validate Salvo executable. (see Section 6).

5.2.1 Tips for Optimization

Salvo can obtain very high performance because of the numerical algorithms selected, careful implementation, and the ability to use the highest optimization flags, such as *-O4* and *-fast*. For the C code, the high optimization flags are not required to obtain good performance. However for the Fortran coding, the use of high optimization can make a drastic difference in performance. In general, low optimization flags should be set in the environment file initially. Once a working version of Salvo is confirmed using the validation suite, higher optimization flags should be tried. It should be noted that often only one or two routines prevent all of Salvo to be compiled at the highest optimizations. These routines should be compiled at lower optimizations while the remainder compiled at high levels. An example of this is shown in Figure 32. The OPT9 is used for a few files in the *mig_split/* directory (see also the *mig_split/Makefile*).

When the Li filter is used, a major portion of the runtime is spent on fast Fourier transforms (FFTs). Thus it is important to have a good FFT library. As a default, the NetLib FFTPACK is provided however many vendors supply optimized FFT libraries for their computers. These should be used when possible.

The user should also look at the Salvo compiler flags specified in the their environment file for possible improvements. One important Salvo compiler flag should be invoked if possible, EXPTL. This flag replaces the Fortran complex exponential provided by the compiler with a table lookup of the sine and/or cosine function with one Newton iteration to calculate the complex exponential. This replacement saves 90% of the runtime for the thin-lens calculations of Eq. (3.4). It is also used in the Li filter for additional savings.

5.2.2 Compilation Flags and Options

The following Salvo compiler flags can be set or defined in the environment file to customize Salvo performance and capabilities. The compiler flags are defined for both the Fortran and C code sections unless otherwise noted.

BCS=type Defines the type of boundary conditions to be used on the domain's side boundaries. Valid types are: (0) Dirichlet with zero value, (1) Neumann with zero slope, (2) Clayton and Engquist absorbing conditions, (3) University of Texas at Dallas absorbing boundary condition which are only implemented for two-dimensional problems and the 65 degree equation, and (4) Liyang Xu's absorbing boundary condition. The default type for BCS is 4. Not all combinations of BCS and EQN work together. BCS=2 and BCS=4 work with all values of EQN. However, Xu's boundary conditions (BCS=4) are tuned for EQN=65. For spatial parallelism only BCS=2 and BCS=4 are defined. This is defined for the Fortran coding.

COEF_SEP Perform the coefficient generation and tridiagonal solve in two separate steps. The default operation is to generate the coefficients and perform the forward sweep of the tridiagonal solve in one pass through the data. This is defined for the Fortran coding. Useful if vendor supplies an optimized tridiagonal solve, see SP2.

CPLANT Specifies coding for Sandia CPlant(tm) (*e.g.*, the FFT routines).

DEBUG=n Sets Salvo debugging level, using $0 \leq n \leq 5$, where all debugging output is turned off at $n = 0$ and the most debugging output is at $n = 5$. If **DEBUG** is undefined, then the default is $n = 0$.

DEC_FFTS Use the DXML FFT library routines provided by DEC.

DEC8400 Specifies coding for the DEC8400.

DEC8400_ERR_OFF Useful flag for DEC8400 difficulties with MPI's Wait-some in scatterFreq.c. Allows code to continue execution even though MPI error flags are generated in MPI_Wait-some, which would normally halt execution immediately (with the default MPI error handler). This is defined in the C code.

EQN=num Defines the migration equation with the propagation angle. Valid values are: 5, 15, 45, 60, 65, or 75. If **EQN** is not defined with one of these values or is not defined at all, the 65 degree equation is used. This is defined for the Fortran coding.

EXPTL Use the complex exponential calculation provided by Salvo. Salvo uses a table of the sine function and one step of a Newton iteration to determine the sine and cosine values of the complex exponential. This is defined for the Fortran coding.

FFTPACK Indicates use of the FFT package from NetLib, FFFPACK. This overrides use of all other architecture defines for the FFT routines and is the default selection.

FORTTRAN_UNDERSCORE Allows C routines called from Fortran (or vice versa) to have an underscore added to the end of the subroutine name. Machines currently known to need this include the DEC Alpha, Sun, Paragon, SGI, and ASCI Red TFLOP.

FREQDIST=n Determines the type of frequency distribution or mapping to use for the migration. The types of mappings include: (0) simple linear, (1) reverse of simple linear, (2) frequency wrap, (3) oscillatory, and (4) cyclic distribution. If **FREQDIST** is not defined, then the oscillatory mapping ($n=3$) is the default. Refer to Table 1 for additional information on the currently defined mappings.

GAUSSIAN Indicates that a spatial Gaussian source distribution should be generated for the source pressure field instead of a single point source. See Section 5.4.1.2 for additional information on this option.

LAMBDA=n Specifies the compact finite difference coefficient function to be used. Valid values include: (n=0) standard discretization without any compact finite differences, (n=1) constant coefficient, $\lambda = 0.14867881$, (n=2) Linear variation with respect to frequency, and (n=3) quadratic variation with respect to frequency. The default LAMBDA is 1, constant coefficient. If the Li filter is used, LAMBDA can be set to 1, 2 or 3. However if LAMBDA=2 or 3, it is possible to generate small oscillations along the boundaries in some problems. With the Graves and Clayton filter, one should use LAMBDA = 2 or 3 to reduce dispersion.

LI_NOCOPY If LI_NOCOPY is defined, the computational domain is extended to prevent copying of the wavefield to another two-dimensional temporary array of size $n_{x2} \times n_{y2}$. The domain is first padded by 20% and then additionally padded to the next radix of 2, 3, and 5. This additional padding prevents wrap-around (in the x and y directions) in the Li filter. Wrap-around is due to the assumption of periodic functions within the FFTs. The costs here are that the computational domain is increased and additional work is performed in the thin-lens and diffraction routines. However the Li filter correction is applied to the wavefield in place thus avoiding costly copying of the wavefield. The domain must be defined large enough so that reflections do not impinge on the boundary. If LI_NOCOPY is not defined, each frequency is copied to a temporary array of size $n_{x2} \times n_{y2}$, where the domain sizes n_{x2} and n_{y2} are padded by 20% and then additionally padded to the next radix of 2, 3, and 5. These copies can be expensive. Because part of the domain is clipped during the copy back to the space domain, there is little concern about oscillatory growth at the boundaries

L_ENDIAN Little Endian (DEC, Intel). If this environment flag is defined, the *segvio* library byteswaps any SEG Y data during the reads.

NOWRAP Performs the reduction of temporal wrap-around calculation, which removes the wavefield signal after it has been imaged once. This is only needed if the trace signals are very short or "smiles" are expected to interfere with deeper events. This is defined for the Fortran coding.

OUTPUT=n The amount of output providing statistical information and various information on the processes is controlled by the compilation define `-DOUTPUT=n` for $0 \leq n \leq 5$. All output is suppressed at level $n = 0$ and all output is generated at the highest output level $n = 5$.

PARA Specifies coding for the Intel Paragon (*e.g.*, FFT routines).

- REDUCED_SRC** Enable reduced-source migration. When this flag is set, an additional line of parameters are read at the end the *MIG_SPLIT* section of the input deck. Section 5.4.1.3 provides additional details on this technique.
- SP2** Specifies coding for the IBM SP2 (*e.g.*, FFT routines).
- SGI** Specifies coding for SGIs (*e.g.*, FFT routines).
- SGI_64** Specifies coding for SGI 64 bit architecture such as with the FFT routines and defines for *d_real* and others.
- SUN** Specifies coding for SUNs (*e.g.*, FFT routines).
- TIMELINE** Outputs timeline stamps with each node writing to its own file.
- T3D (or T3E)** Specific coding for the Cray T3D and T3E, *e.g.*, FFT routines.
- VELOCITY** Output the interpolated velocity model used by Salvo during migration. This is defined for the Fortran coding.
- WRITE_GIF** Indicates that GIF image slices should be generated during the migration using the *gd1.2* package.
- XTRA_SYNC** Helps prevent Salvo from hanging when running with extremely large poststack problems (in *diffract.F*). This is defined for the Fortran coding.

5.3 Performance Results

5.3.1 I/O During Reading of Traces

Parallel I/O bandwidth depends on the number of independent disks or RAIDs. If there are multiple independent disks or RAIDs, then the user can utilize them by spreading the files across these disks and using the same number of I/O nodes as disks; essentially the files are being striped across the disks manually. This technique accelerates Salvo's trace input phase especially for large poststack migration runs, and may be useful for writing or stacking the image and reading the velocity model when using large numbers of processors. The user should consult Section 4.2.6 for determining the optimal number of compute to I/O processors for the *READ_TRACES* process.

5.3.2 I/O During Migration

The overall runtime during migration is dominated by the computations, but the I/O can noticeably impact the overall performance when using a large number of frequency processors ($p_w \gg 1$) without an I/O partition. This effect is demonstrated in Figure 33. This figure plots the predicted run times for both the I/O and

Figure 33: Calculated execution time for one migration depth step of the North Sea data set. The solid line is the execution time for the compute partition while the dashed line is the overlapped execution time for the I/O partition.

compute partitions, which closely correspond to the actual run times on an Intel Paragon [26]. Normally an I/O partition is utilized and this would hide or overlap the execution time required by the I/O nodes. The impact of an I/O partition becomes significant (for this problem) when $p_\omega \geq 64$, and would approximately increase the total run time 30% if an I/O partition was not used with $p_\omega = 128$.

5.3.3 Parallel Efficiencies

To test the computational performance of Salvo, a simple impulse problem similar to those shown in Tables 5 and 6 was run on the Intel Paragon. Spatial parallelism was tested by setting $p_\omega = 1$ and varying the number of processors in the x and y directions, $p_x \times p_y$, and the number of grid points, $n_x \times n_y$. The total number of grid points was selected to remain roughly the same number of grid points on each processor (*i.e.*, scaled-problem size). For frequency parallelism, the number of processors in the ω direction, p_ω , was varied while keeping $p_x \times p_y = 1$ and the total number of frequencies, $n_\omega = 128$, constant (*i.e.*, fixed-problem size).

Efficiencies for the impulse run are shown in Table 2. For spatial parallelism, the parallel efficiency drops to 71% for a 3×3 processor mesh, which is primarily due to the overhead associated with the pipelined tridiagonal solves. However for larger processor meshes, the efficiencies remain relatively flat and only decrease to 66% for a 8×8 processor mesh. Thus once the penalty of setting up the pipeline tridiagonal solves has been incurred, lengthening the pipeline costs very little. The frequency parallelism is very efficient, remaining above 90% for most problems. The drop off in efficiency for the 64-node run is due to too little work for each processor (*i.e.*, Amdahl's Law).

Table 2: Efficiencies for a sample impulse problem for spatial and frequency parallelism on the Intel Paragon.

Spatial Parallelism			Frequency Parallelism		
$p_x \times p_y$	$n_x \times n_y \times n_w$	Efficiency	p_w	$n_x \times n_y \times n_w$	Efficiency
1 × 1	101 × 101 × 32	100.0	1	101 × 101 × 128	100.0
3 × 3	301 × 301 × 32	71.6	2	101 × 101 × 64	100.0
4 × 4	401 × 401 × 32	71.3	4	101 × 101 × 32	101.6
5 × 5	501 × 501 × 32	69.5	8	101 × 101 × 16	98.6
6 × 6	601 × 601 × 32	68.7	16	101 × 101 × 8	98.6
7 × 7	701 × 701 × 32	67.2	32	101 × 101 × 4	92.1
8 × 8	801 × 801 × 32	66.7	64	101 × 101 × 2	85.9

5.4 Description of Input and Output

To run Salvo, three types of input data are required: the parameter input deck, traces for the receiver and source, and a velocity model for the migrated region. The parameter input deck defines the Salvo run by listing the processes (and their input) to be executed. Currently there are three processes recognized by Salvo: *INITIAL*, *READ_TRACES*, and *MIG_SPLIT*. These processes are described in more detail in Sections 5.4.1.1, 5.4.1.2, and 5.4.1.3. Trace data is needed for the migration, and is either receiver data for poststack runs or receiver and source data for prestack runs (discussed in Section 5.4.2). Finally, a velocity model is required for depth migration (more details in Section 5.4.3).

5.4.1 Input Parameter Deck²⁶

The input deck file (or prm file) controls how Salvo performs the migration. As mentioned earlier, each prm file contains a set of processes and associated input parameters that are to be read and executed by Salvo for the migration. The set of processes available is specified by the following table.

process name	read parameters routine	process data routine
<i>INITIAL</i>	get_initial_params	initial
<i>READ_TRACES</i>	get_readTracesParams	readTraces
<i>MIG_SPLIT</i>	get_mig_split_params	mig_split

The essential data structures for specifying the processes are in the file *salvo/src/driver/read_deck.c*. The maximum number of current processes is specified in this file with the arrays and tables containing the process names, read parameter routines, and process data routines, which are respectively defined as *process_name*,

²⁶Subroutines: main, read_deck_driver, read_deck; files: driver.c, read_deck.c

read_params, and *process_data*. The user can incorporate additional processes by expanding the arrays and tables with additional entries for the new process name, and routines to read the input parameters and process the data. These routines must conform to the respective function definitions, *GET_PARAM_FN* and *PROCESS_DATA_FN* in *read_deck.h*.

The main driver program constructs a process list while reading in one or more prm files. The process list contains the set of specified processes with the appropriate data that is read in from the prm file(s). Once the process list is formed, each list entry is evaluated in order using the function and data associated with the current list entry (or current process). The overall format of this driver provides the user with a large amount of flexibility in specifying how migration runs are to be performed and makes the code easily extensible. An example prm file is shown in Figure 34 for a single shot of the Marmousi model. Results of the full migration are shown in Section 6.2. Finally, comment lines can be added to any process (at any line) by specifying a '#' in the first column of line.

5.4.1.1 INITIAL Process The *INITIAL* process includes input parameters defining the processor cube, migration volume, some general trace information, and the frequency range for migration. The first section in Figure 34 is an example input for the *INITIAL* process.

Line 1 The first line of the process is denoted by the process name, *INITIAL*.

Line 2 The second line lists the name of the job.

job name The job name (*marm_3000-0.out*) is used to generate an output file of this name that contains information on the run including: the *job name*, the number of processors, and the runtime.

Line 3 The third line describes the allocation of processors to the compute partition and the I/O partition that are used during the *READ_TRACES* and *MIG_SPLIT* processes.

px, py, pw, pio The processor cube is specified by the number of nodes used in the *x*-direction, *px*; *y*-direction, *py*; and the frequency direction, *pw*. For the example in Figure 34, a cube of dimension $1 \times 1 \times 1$ for a total of 1 node is requested. Since no I/O nodes are specified, this run requires a total of 1 node. The formula for the total number of processors is $p = (p_x \times p_y \times p_w) + p_{io}$.

Line 4 The fourth line describes the *x* domain of the migration volume.

```

# -----
# Marmousi 24 Shots -- marm_3000-0
#

INITIAL
marm_3000-0.out  jobname
1 1 1 0          px py pw pio
0 9200           xmin xmax
0 0              ymin ymax
25 1 4           dx dy dz
369 1 751        nx ny nz
0.004 1024 1024  dt nt nt_pad
0.00 62.5        fmin fmax

READ_TRACES
seggy            trace type
../sources/ormsby.segy 0.000 source trace, time shift
1 0 1.0          # shots, enc., enc. parm
1                # trace files
../traces/marm_3000-0.segy      trace file 1
0.0 0.0 0.0 0.0 coordinate transform
65536            I/O blocksize
1 0 1.0 2000.0 1.57 gaussian params

MIG_SPLIT
1                number of velocity files
../model/marm25.vel velocity filename
25.0 25.0 4.0   vdx vdy vdz
369 1 751       vnz vny vnz
0.0 0.0 0.0     vxmin vymin vzmin
1500            vmin velocity
2               number of freq decay points
0.0 70.0         depth and freq of decay pt. 1
50000 70.0       depth and freq of decay pt. 2
prestack         stack type (pre or poststack data)
1                filter type (0=G&C, 1=Li)
0 1 1            bmin nterms filter_inc
2 0             img type: 0=Cor 1=Dec 2=Der, eps
1               number of output file
../stack.04.acc  output filename
0               stacking flag (0=no, 1=yes)
0 0.00          PML parameters
1 200 4 96      reduced src parameters

```

Figure 34: Parameter (.prm) file for one shot of the Marmousi model.

xmin, xmax It is important to remember that Salvo is insensitive to the unit of length, thus either feet or meters (miles, kilometers, *etc.*) can be used as long as the same unit of length is consistently used throughout the problem (for trace locations, the velocity model, *etc.*).

Line 5 The fifth line describes the y domain of the migration volume.

ymin, ymax For the example shown, the x - y domain extends from 0-9200 m in the x direction and 0 m in the y direction (*i.e.*, *ymin* will equal *ymax* for two-dimensional problems).

Line 6 The sixth line specifies the grid spacing.

dx, dy, dz The units of the grid spacing should match the units of Lines 4 and 5.

Line 7 The seventh line specifies the number of grid points of the migration domain in each direction.

nx, ny, nz The number of grid points should be consistent with the migration domain and the grid spacing, such that $n_x = (x_{max} - x_{min})/dx + 1$ and $n_y = (y_{max} - y_{min})/dy + 1$. The migration depth is determined by $z_{max} = dz \times n_z$. For the sample problem, the x - y domain will have an 369×1 grid with a grid spacing of 25.0 m by 1.0 m. This grid will be downward continued for 751 steps with a depth step of 4.0 m.

Line 8 Line eight specifies information on the time traces including the sampling rate (in seconds), the number of samples per trace, and the length to pad each trace.

dt, nt, nt_pad The receiver traces have 1024 samples at a sample rate of 0.004 seconds with a padded trace length of 1024 samples (*i.e.*, the total trace length is *nt_pad*, 1024 samples, NOT $1024+1024=2048$). *nt_pad* can be smaller than, equal to, or larger than *nt*. The trace will be truncated or padded with zeros as necessary.

Line 9 The final line of the *INITIAL* process description provides the frequency range for migration.

fmin, fmax The values are given in Hertz. The maximum range which Salvo will migrate is 0 to the Nyquist frequency. Although 0 Hz can be specified, it is not migrated, but the next frequency, $d\omega$, will be. The range shown is 0 to 62.5 Hertz in the example.

5.4.1.2 *READ_TRACES* Process The *READ_TRACES* process reads the input trace data. This process can read data in SEG Y [28], SEP (Appendix C.3), or SALVO (Section 5.4.2) format. Additional information on these trace formats is found in Section 5.4.2.

Line 1 The first line specifies the process name, *READ_TRACES*.

Line 2 The second line indicates the format of the receiver and source data as either SEG Y, SEP, or SALVO format.

file type The source and receiver traces must have the same file format. Valid strings for the file type are “seg y”, “sep”, or “salvo”.

Line 3 The third line specifies the source traces file.

source file, time shift *source file* is the name of the file with the source trace. It may include a path if necessary. If the run is a poststack run, the source filename should be set to the string “null”. *time shift* is a time shift (in seconds) to apply to the source trace. A positive time shift moves the impulse later in time and a negative time shift moves the impulse earlier in time. The number of samples per trace and the sampling rate are specified in the *INITIAL* process. Both the receiver and source traces must have the same number of samples and sampling rate.

Line 4 The fourth line specifies the phase-encoding parameters. Phase encoding is only useful for prestack migration.

of shots This number specifies how many shots will be encoded. It does not necessarily need to match the number of receiver files in Line 5 of the *READ_TRACES* (*i.e.*, there may be one shot spread across several receiver files).

encoding type Table 3 provides the various encoding types and parameters available. The types of encoding are

(0) trivial sums traces without any encoding, thus completing a simple stack before migration.

- (1) *random* applies a random phase shift to the frequencies.
- (2) *linear* (based on shot) applies linear phase shifts to multiple shots by equally spacing the time shifts between each shot.
- (3) *chirp* applies a frequency-modulation chirp function as phase shift.
- (4) *+/-i* applies a random + or - i phase shift to each and every frequency.
- (5) *linear phase shift* applies a linear phase shift to two shots.
- (6) *modified chirp* applies an optimal phase shift for the known source function (*i.e.*, currently implemented for a derivative of a Gaussian).

encoding parm Chirp encoding requires a stretching parameter which may vary between the approximate range [2-10]. Linear phase encoding requires the desired time shift between the shots, t_0 , in seconds. Modified Chirp requires a stretching parameter which can vary between the range [0-1]. For all other encoding types the encoding parameter is ignored.

Line 5 The fifth line specifies the number of receiver trace files.

receiver files The number of receiver trace files to read in. The number of samples per trace and the sampling rate are specified in the *INITIAL* process. Both the receiver and source traces must have the same number of samples and sampling rate.

Line 6 The sixth line sets the receiver filename(s).

trace file #1

:

trace file #n One line for each receiver trace file and one line per filename. There should be no more than one shot or partial shot per file (*i.e.*, no multiple shots in a single file). Each trace in the receiver file is binned to the nearest migration grid point. If multiple traces are binned to the same grid point, the last trace over writes the previous trace at that location. The exception to this is when phase encoding is invoked. If a trace falls outside the migration grid, it is dropped from the migration and ignored.

Table 3: Phase Encoding selection within Salvo.

encoding number	encoding type	encoding parameter
0	Trivial	not used
1	Random	not used
2	Linear	not used
3	Chirp	stretch factor
4	+/- i	not used
5	Linear Phase	t_0
6	Modified Chirp	stretch factor

Line 7 The seventh line describes the coordinate translation and rotation of the trace data.

$x0, y0, z0, theta$ This can be used to align the input seismic data along the x and y axis, or rotate two-dimensional data which is in the y - z plane to an x - z plane which is required by Salvo. The untransformed coordinates, $(x0, y0, z0)$, are the new origin for trace locations (*i.e.*, traces locations (x, y, z) will be translated to $(x0, y0, z0)$ by $x-x0, y-y0$, and $z-z0$). After translation, the traces will be rotated about the new origin by $theta$ degrees. Positive values of $theta$ are a counter clockwise rotation, and negative values are clockwise rotation.

Line 8 The eighth line specifies the reading block size for the trace data.

blocksize This is machine dependent, but for the Intel Paragon 64 Kb (*i.e.*, 65536 bytes) is the optimal value.

Spatial Gaussian Source Distribution A two-dimensional Gaussian spatial source distribution, instead of a standard point source, is available in Salvo by defining the flag -DGAUSSIAN for compilation. This feature is useful with the reduced-source migration (Section 5.4.1.3) and may be useful in other contexts. If this feature is invoked, then the user must include an additional input line in the *READ_TRACES* process for the parameters: *perform_gaussian*, *fixed_gamma*, γ , c_0 , and ϵ_0 .

Line 9 (Required only if the flag -DGAUSSIAN is defined during compilation.) This should follow the last standard line in the *READ_TRACES* process that specifies the blocksize for reading traces.

- perform_gaussian* The parameter *perform_gaussian* is a boolean flag (value of 0/1) specifying whether the source traces is distributed spatially using a Gaussian weighting.
- fixed_gamma* The parameter *fixed_gamma* is another boolean flag specifying whether the value for γ is constant (*fixed_gamma* = 1), or if γ is a function of frequency ω (*fixed_gamma* = 0).
- γ The parameter γ controls the width of the spatial Gaussian source if *fixed_gamma* is set to 1.
- c_0, ϵ_0 The last two parameters, c_0 and ϵ_0 , are used when the weighted Gaussian, $\gamma(\omega)$, is indicated (*fixed_gamma* set to 0). The two-dimensional Gaussian generated is centered at the source location, (x_s, y_s) , which is specified by the source trace/file in the *READ_TRACES* process. Section 3.7 provides additional information on the spatial Gaussian distribution.

5.4.1.3 MIG_SPLIT Process The *MIG_SPLIT* process executes the finite-difference depth migration algorithm. In Figure 34, an example of the *MIG_SPLIT* process is shown.

Line 1 The first line for this process simply lists the process, *MIG_SPLIT*.

Line 2 The number of files that the velocity model is spread across.

of velocity files

Line 3 The file name(s) of the velocity model.

velocity file #1

:

velocity file #n One line for each velocity file name.

Line 4 The grid spacing of the velocity model.

dx, dy, dz The velocity model grid spacing should match the units of migration domain specified in the *INITIAL* process. The velocity model grid spacing does not need to match the migration grid spacing. Salvo will interpolate the velocity model to the needed migration grid “on the fly”.

Line 5 The number of grid points in the velocity model.

nx, ny, nz Two-dimensional models will need the *y* dimension set to one.

Line 6 The velocity model origin.

xmin, ymin, zmin The velocity model origin (*i.e.*, the grid point location that has the minimum *x*, *y* and *z* values). The velocity model is just a set of binary numbers without any headers, thus no rotation should be required.

Line 7 The minimum velocity within the velocity model.

min. velocity Only an estimate is necessary and is used for guidelines within Salvo.

Line 8 The eighth line specifies the number of frequency decay points.

of points The number of frequency decay points that should be used in the migration.

Line 9 and 10 The ninth and tenth lines specify a depth and frequency (Hertz) pair. One line is needed for each frequency decay point specified in Line 8.

depth, freq. pair #1

:

depth, freq. pair #n A depth (length) and frequency (Hertz) pair is specified; this pair indicates the maximum frequency to be migrated at the specified depth. The maximum frequency is linear interpolated for migration depths between specified frequency decay points. As the frequency decays with depth, higher frequencies may not be required for migration. By eliminating these frequencies and not migrating them, a runtime savings is possible.

Line 11 The eleventh line sets the migration type.

migration type The migration type is specified by either of the strings "poststack" or "prestack."

Line 12 The twelfth line specifies the filter type.

filter type There are two filters currently implemented: the Graves and Clayton filter, and the Li filter; these are specified with zero or one respectively. If spatial parallelism is used, the Graves and Clayton filter is currently the only possible choice. However, for frequency parallelism either filter can be used, but the Li filter currently has higher parallel efficiencies and has lower computational costs. If no filter is desired, use *filter_inc* = 0 but still specify one of these filters.

Line 13 The thirteenth line specifies miscellaneous parameters.

bmin The first parameter, *bmin*, is used to set the number of tridiagonal solves in a pipeline block when spatial parallelism is used. If *bmin* is set to zero, Salvo automatically determines the optimal block size, otherwise Salvo will use the value specified.

nterms The second parameter is the number of terms used in the series expansion for the square-root operator approximation, *nterms*. All the hooks are here for multiple terms, however, Salvo is not routinely tested for bugs using this option, so care should be exercised when using values other than one. Other valid values for *nterms* are 2, 3, 4 and 5, when EQN=65 only.

filter_inc The third value is *filter_inc*, the filter increment, which is used to apply the Li or Graves and Clayton filter at regular specified depth steps. If *filter_inc* equals zero, the filter will not be applied during the run. If *filter_inc* is one, the filter will be applied at every depth step. If *filter_inc* is two the filter will be applied at every other depth step, and so on. For values of *filter_inc* greater than zero, the Li filter is always applied on the first depth step.

Line 14 The fourteenth line specifies the imaging condition and the deconvolution epsilon.

image type, epsilon Three types of imaging conditions are available with Salvo: correlation imaging (*image type* = 0), deconvolution imaging (*image type* = 1), and derivative imaging (*image type* = 2). The deconvolution imaging condition also requires a threshold value, *epsilon*. The other imaging conditions do not use this value.

Line 15 The fifteenth line specifies the number of image files for the output image.

of image files

Line 16 The sixteenth line sets the filename(s) for the image file(s). The path may be included if necessary.

image file #1

:

image file #n If multiple image files are specified, an additional line is added for each additional image file name.

Line 17 The seventeenth line specifies whether the image file should be stacked with a previous image file on disk (=1) or not (=0).

stack flag If *stack_flag* is set to one, the current image being generated will be stacked with the file specified on Line 16 of the *MIG_SPLIT* process. The domains of these two images need not match, because the resulting image file will encompass both domains.

Line 18 The PML parameters for the absorbing layer boundary conditions.

PML_thick, PML_absorption *PML_thick* specifies the thickness of the PML in grid cells. Supplying a value of 0 results in no PML. The recommended value is 16. *PML_absorption* is the absorption value, and larger values increase the absorption. Supplying a value of 0.0 results in no absorption (although PML calculations are still performed). The recommended value is 0.01. Supplying too large of a value for absorption results in numerical instability. Thicker layers require smaller values of *PML_absorption*. The PML is placed within the $n_x \times n_y$ migration region (as opposed to augmenting each side of this region with *PML_thick* grid cells). Because of this one may want to increase n_x and n_y when using the PML in order to ensure the region being imaged is undisturbed by the PML. Use of the PML is complementary to Salvo's other boundary conditions. The PML is used in conjunction with the boundary condition specified by the compilation flag -DBCS. The grid point on the boundary implements the absorbing boundary condition specified by BCS independent of the PML. Collino

and Monk [11] found that traditional absorbing boundary conditions could be interfaced with PMLs beneficially and not have an adverse effect.

Reduced-Source Frequency Migration The ability to migrate a subset or subrange of source frequencies, and calculate the remaining source frequencies through an “inexpensive” extrapolation process, is available through reduced-source frequency migration. This feature is invoked by defining the compilation flag `-DREDUCED_SRC`. If this feature is used, then the user must include an additional input line in the *MIG_SPLIT* process for the parameters: *perform_freq_reduction*, *k_start*, *n_ws*, and ω_0 .

Line 19 (Required only if the flag `-DREDUCED_SRC` is defined during compilation.) This should follow the last standard line in the *MIG_SPLIT* process that specifies the PML parameters.

perform_freq_reduction The parameter *perform_freq_reduction* is a boolean flag (value of 0/1) specifying whether the reduced-source frequency migration should be employed.

k_start The parameter *k_start* provides the depth step level for starting this process, so that some full migration steps may be initially employed before commencing the reduced migration

n_ws, ω_0 The reduced-source migration frequency range is provided by the parameters, *n_ws* and ω_0 ; *n_ws* is the number of source frequencies to migrate and ω_0 is the starting frequency index, so the range of source frequencies migrated at each depth step is ($\omega_0 \leq \omega_i \leq \omega_0 + n_{ws} - 1$), while the remaining frequencies are extrapolated each step from the subset of migrated source frequencies. The details of this technique are given in Section 3.7.

5.4.2 Traces for Source and Receivers

Salvo currently accepts traces in the SEG-Y, SEP, and SALVO format. Salvo can process traces in the “industrial standard” SEG-Y format by using separate files for the source and receiver traces; the source file contains a single source trace. The standardized format, SEG-Y [28], is notoriously abused, however Salvo assumes that all header information is consistent and correct. Please refer to [28] for information on the SEG-Y file format.

For SEP files, Salvo assumes that the Enhanced SEP format is used. The SEP file contains the receiver traces with the source trace being the last trace in a file. See Appendix C.3 for more details on the Enhanced SEP file format. Salvo ignores

the final trace in an SEP file, if it is a source trace, and expects a separate source file that only contains a source trace, which allows the user more flexibility in using different types of sources without having to manipulate large receiver files.

The SALVO format, which was designed for initial testing of Salvo, is similar to the enhanced SEP format in that the file contains the receiver traces with the source trace as the last trace in the file. Unlike when using the SEP format files, the source trace in the SALVO receiver file(s) is utilized and the separate source file name is essentially ignored. The SALVO trace files have a simple format with no file header and five word (IEEE 4 byte float) trace headers. The five words for the trace header consist of (1) a 0 or 1 value designating a receiver or source trace respectively, (2) the number of samples, (3) the x location of the trace, (4) the y location of the trace, and (5) the z location of the trace.

5.4.3 Velocity Model

The velocity model is a three-dimensional volume stored in a binary file. The model does not need to match the migration volume or the trace data because Salvo automatically interpolates and/or extrapolates the velocity model to the needed migration volume. This can cause a problem if there is a user input error; the velocity model will be interpolated whether it is correct or not. For debugging the velocity model the flag `-DVELOCITY` should be used during compilation. The ordering of the velocity file is simply xyz ordering where the fastest changing direction is x then y and finally z .

Velocity models which are in SEG Y format can be converted to the correct binary ordering by using the Salvo utilities, *seg y2bin* and *transpose*. The *seg y2bin* utility just strips out the the SEG Y header information, and writes the trace data. To get the correct ordering, the traces in the SEG Y file must be sorted to xy ordering (*i.e.*, the fastest changing direction is x then y) prior to running *seg y2bin*, and every x and y location on the velocity model grid must have a trace. The output from *seg y2bin* is ordered xy . Using *transpose*, the correct ordering can be obtained.

5.4.4 Output Data

There are four types of output: the accumulation file for the seismic image, the *job name* file, the standard output, and the timeline files. The primary output of Salvo is the migrated image file, which is in accumulation format (see Section C.2). The final image can be output into multiple files instead of a single file by changing the input parameters for the *MIG_SPLIT* process. The accumulation file can be converted to the industry standard format, SEG Y, by using the Salvo utility, *acc2seg y*. The *job name* output file is also generated for each Salvo run. This file is specified in the *INITIAL* process for an input deck, and the contents of this file are the job name, the number of processors used and the runtime.

Besides the image file and the *job name* file, the standard program output from Salvo also can be adjusted by the user. General output of Salvo providing statistical information and various information on the processes is controlled by the compilation flag, `-DOUTPUT=n` for $0 \leq n \leq 5$, where no output is provided at level $n = 0$ and all output is generated at the highest level $n = 5$. The debugging output is controlled by a compilation flag `-DDEBUG=n` for $0 \leq n \leq 5$, where all debugging output is turned off at level $n = 0$ and the highest amount of output is at level $n = 5$.

The final type of output, which is useful for performance evaluation between the compute and I/O partition, is the *timeline generation* option. This option is activated by defining `-DTIMELINE` during compilation. When activated this causes each processor to output time stamps for various portions of the code in its own file. (The number of timeline files is equal to the number of processors.) After running Salvo, this information is used for post processing and can help identify possible bottlenecks in the code (refer to the Salvo utility *tl2tec* for post processing information).

6 Validation Data Sets

Comprehensive testing of Salvo can be performed for several validation problems by executing the script, *run_tests.pl*, in the directory *salvo/Validation_Suite*; in some cases the command '*perl run_tests.pl*' is necessary for execution. Running this script without any command line arguments provides the user with information on all of the possible script arguments and options. The script can have one to three command line arguments which are:

1. the type of testing to perform (an integer value).
2. an indication of how the MPI job should be launched (default is *mpirun*).
3. the threshold tolerance level for image comparison (default is $1.0e - 05$).

Default values are used if either of the last two arguments is omitted. As an example, the command '*run_tests.pl 0*' would perform a validation with all of the test cases, use *mpirun* to launch the jobs, and use $1.0e - 05$ as the threshold comparison tolerance.

The script performs the validation by launching a set of Salvo test runs and comparing the image produced by each run to a reference image. For each test case performed, the script indicates which test directory is used, and whether the test was successful (error tolerance less than the threshold). The success of a test case is determined by comparing the L_2 norm of the relative difference between the generated image, *images/test.acc*, and the reference image, *images/Reference/ref_image.acc*. In the following sections, three validation problems are shown. These runs will allow the user to see the general capabilities and to validate their copy of the Salvo code.

6.1 Impulse Response

An impulse response in a constant velocity medium is one of the simplest tests of a migration method, and it provides images which are not complicated by other considerations. For these tests, a single Ricker wavelet is placed at the quarter point location in the x direction. The central frequency for the wavelet is 30 Hz, and a sampling rate of 4 ms for 2 seconds of data. For prestack migration, the source trace is an Ormsby wavelet with a square frequency spectrum over the range 0-62.5 Hz. The velocity model has a constant acoustic speed of 2000 m/s. These values are used for all the following impulse tests.

In Table 4, images for the two-dimensional poststack and prestack migrations are shown for various combinations of boundary conditions, and Li filter. The poststack image produced with Neumann BCs and without phase correction, Table 4(a), is a good image, but does contain the cardioids from the square-root approximation and has ripples in the image from numerical dispersion. The prestack

image is substantially worse, complicated by reflections from the side boundaries and by aliased cardioids. In Table 4(b), the poststack and prestack images have improved with the application of the absorbing boundary conditions. Although the cardioids (aliased cardioids for the prestack image) and numerical dispersion are still present, the reflected energy has been reduced and the image quality is greatly improved.

The Li filter corrects for numerical dispersion and cardioids from the square-root approximation, as can be seen in Table 4(c). For the prestack image there are two reflections from the left-hand boundary. The upper reflection is similar to the poststack reflections, and is expected. However the lower reflection is slightly unexpected. This reflection is caused by a correlation between a direct wave from the surface and a reflected wave from the left-hand boundary. The strength of the lower reflection is comparable to the true event because two signals are overlaid: a direct source wave with a reflected receiver wave, and a direct receiver wave with a reflected source wave. In Table 4(d), images produced from the Salvo migration using absorbing boundary conditions with the Li phase correction filter are shown. The reflections from the left-hand boundary have been reduced. The lower reflection in the prestack image has been reduced but is still present. This is due to the fact that half of the energy contributing to this event is from direct waves which does not get absorbed by the boundary, which then correlates with the very little energy that is reflected. This reflection can be further reduced by using the PML.

In Tables 5 and 6, vertical and horizontal sections for the three-dimensional impulse response are shown. When neither absorbing boundary conditions or phase correction filters are applied the resulting images are very difficult to interpret, as seen in Tables 5(a) and 6(a). However the horizontal section for the poststack image clearly shows the non-circular cross-section of the impulse response caused by the operator splitting. With the application of the Li filter, Tables 5(b) and 6(b), numerical dispersion and aliased cardioids are removed. The Graves and Clayton filter, Tables 5(c) and 6(c), produces similar results but the image still contains the cardioid from the square-root approximation. The horizontal sections show the effectiveness of the absorbing BCs and the improvement in the circular cross-sections from the application of the Graves and Clayton filter. Finally in Tables 5(d) and 6(d), the images from the application of the absorbing boundary conditions and the Li filter are shown. The vertical sections are very similar to the two-dimensional images, and the horizontal sections for the poststack and prestack migrations show circular cross-sections of the impulse response.

6.2 Marmousi Model

A slightly more difficult but more realistic problem, the Marmousi model provides the additional problem of strong lateral velocity variation. This model has a realistic velocity model which is similar to a true geological region. However synthetic

Table 4: Image Matrix for two-dimensional Impulse Response with (a) Neumann BCs without phase correction; (b) absorbing BCs without phase correction; (c) Neumann BCs with Li filter; and (d) absorbing BCs with Li filter.

	2D Poststack	2D Prestack
(a)
(b)
(c)
(d)

Table 5: Image Matrix for 3D Impulse Response - Vertical Sections with (a) Neumann BCs without phase correction; (b) Neumann BCs with Li filter; (c) absorbing BCs with Graves and Clayton filter; and (d) absorbing BCs with Li filter.

	3D Poststack	3D Prestack
(a)
(b)
(c)
(d)

Table 6: Image Matrix for 3D Impulse Response - Horizontal sections with (a) Neumann BCs without phase correction; (b) Neumann BCs with Li filter; (c) absorbing BCs with Graves and Clayton filter; and (d) absorbing BCs with Li filter.

	3D Poststack	3D Prestack
(a)
(b)
(c)
(d)

Figure 35: Acoustic velocity model for the Marmousi data set.

data have been generated over this two-dimensional model. In Figure 35, the velocity model is shown, and illustrates the difficulty of imaging the target in the lower right-hand corner which is beneath a highly faulted region.

Two different sets of synthetic trace data were obtained during this project: the original data [30] [31] and new trace data generated by a separate forward modeling algorithm (details on this data set are unknown). Substantial differences appeared in the resulting images. The original data produced images with ringing beneath various reflectors, while the new data did not. It has been suggested by one of our partners that the forward modeling algorithm, which produced the original data had a bug. Figures 36 and 37 were generated using the new trace data.

The trace data is comprised of 240 streamer shots each with 104 traces. Each shot was migrated over the entire velocity model (*i.e.*, 0-9200 m by 0-3000 m). The migration (and velocity) grid had a spacing of 25 m in the x -direction and 4 m in the z -direction, and the frequency range was 0-62.5 Hz. Both derivative imaging and the Li filter were used during the migration. In Figure 36, the final image is shown. Not only is the faulted region clearly imaged but the target is very well defined; even the top and bottom of the reservoir can be seen!

During the testing of the migration, it was found that 24 shots (every 10th shot) produced a very good image as shown in Figure 37. This provided a data set which was fairly inexpensive to run and still produced a very good image. This also indicates the possibility that fewer migrations may be needed with f - x migration.

6.3 SEG/EAGE Salt Model

The SEG/EAGE Salt model data is not provided with the Salvo software distribution due to its sheer size. However the 45 shot ocean bottom cable (OBC) acquisition (see below for description) may be available through Sandia by contacting David Womble or Curtis Ober (contact information is given in Section 1). For the Phase C3 data set, Paul Docherty of Fairfield Industries should be contacted for copies of the SEG-Y tapes.

Figure 36: Image of the Marmousi model using all 240 shots.

Figure 37: Image of Marmousi model using 24 equally spaced shots.

Figure 38: Velocity model of the SEG/EAGE Salt model with the salt structure shown in white.

The data for the Salt model was generated with a forward modeling code from IFP [2] which subsequently was parallelized by several national laboratories: Sandia National Laboratories [25], Oak Ridge National Laboratories, Los Alamos National Laboratories, and Lawrence Livermore National Laboratories. The velocity model used in the forward modeling is shown in Figure 38, and is comprised of 676 by 676 by 210 grid points with a grid spacing of 20 m in each direction. A salt structure is located within the velocity model and is shown as a white structure in Figure 38. The salt structure is surrounded by faulted and layered media. Reflecting surfaces within the model are introduced by strong velocity spikes along the surface. These surfaces are dipping structures that have stair step variations. This stair stepping is caused by the transfer of these reflecting surfaces to the discrete velocity grid. Lastly the velocity model has a flat horizontal reference reflector located near the bottom of the model.

Five seconds of trace data with 8 ms sampling interval was collected at the model surface on a 200 by 200 grid centered on the shot location. The forward modeling volume was 321 by 321 by 210 but had a sponge region of 15 to 20 grid points around the sides of the volume. This reduces the effective modeling region to approximately 281 by 281 by 210. This brings up the possible aperture problems for the forward modeling, especially beneath the salt structure.

Two data sets were obtained for migration with Salvo: Phase C3 data set and 45 receivers of OBC data. The Phase C3 data set is a streamer acquisition with 100 receivers on 8 streamers. Approximately 2400 shots for a total of 1,812,000 traces are contained in the wide azimuth data set. A migration volume of 271 by

271 by 210 centered around the shot was used for this data set. A larger volume would be useless since the forward modeling domain was only 281 by 281 by 210.

The OBC data set was extracted from the SEG/EAGE Salt model archive at Lawrence Livermore National Laboratories. Currently this data set is unavailable, but may become accessible through SEG. Forty-five receiver locations were collected over the Salt model. Each receiver recorded data from shots located on a 200 by 200 grid centered around the receiver. The total number of traces is approximately 1,800,000 and the migration volume is similar to the Phase C3 data set.

In Figures 39 and 40, the images using the Phase C3 and OBC data sets are shown respectively. The bottom of each cutaway is along the flat reference reflector to check for its correct placement. The black region indicates very good placement of the reference reflector. The gray hole is not a misplacement but a shadow zone near and beneath the root of the salt structure. Since there was no illumination of this zone by the shots, a shadow zone is produced.

Differences in the images on the left and right of the images are due to the acquisition footprint and not to any migration or data differences. The top and bottom of the salt structure are very well imaged, as well as the surrounding layers in both migrations. However as noted in Section 2, the computational costs are not as similar. The OBC costs approximately 50 times less based on the number of migrations; 2400 migrations for the C3 data set and 45 for the OBC data set (assuming the migration volumes are equal). Lastly in each image ripples (or periodic waves can be seen in on the right of each image. Although this is a classic sign of numerical dispersion, it is believed that these are due to point diffractors located at the end of each stair step in the velocity model.

Figure 39: Image of the SEG/EAGE Salt model using Phase-C3 data sets.

Figure 40: Image of the SEG/EAGE Salt model using original forward migration data sets of 45 shots (*i.e.*, OBC data set).

References

- [1] Amdahl, G.M., *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*, Proceeding of the American Federation of Information Processing Societies, Washington, D.C., Vol. 30, 1967, pp. 483-485.
- [2] Aminzadeh, F., Burkhard, N., Nicoletis, L., Rocca, F. and Wyatt, K., *SEG/EAEG 3-D Modeling Project: 2nd Update*, Leading Edge, September, 1994.
- [3] Anderson, D.A, Tannehill, J.C., and Pletcher, R.H., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corp., New York, 1984.
- [4] Berenger, J.P., "A Perfectly Matched Layer for the Absorption of Electromagnetic Waves," *Journal of Computational Physics*, Vol. 114, 1994, pp. 185-200.
- [5] Brown, D. L., *Applications of Operator Separation in Reflection Seismology*, *Geophysics*, Vol. 48, 1983, pp. 288-94.
- [6] Brzostowski, M.A. and Black, J.L., *Frequency dispersion in finite-difference migration*, *Geophysics*, Vol. 54, No. 11, November 1989, pp. 435-1447.
- [7] Bunks, C., *Effective Filtering of Artifacts for Implicit Finite-Difference Paraxial Wave Equation Migration*, *Geophysical Prospecting*, Vol. 43, 1995, pp. 203-220.
- [8] Claerbout, Jon F., *Imaging the Earth's Interior*, Blackwell Scientific Publications, Boston, 1985.
- [9] Clayton, R.W. and Engquist, B., *Absorbing Boundary Conditions for Wave-Equation Migration*, *Geophysics*, Vol. 45, No. 5, May, 1980, pp. 895-904.
- [10] Collino, F., "Perfectly Matched Absorbing Layers for the Paraxial Equations," Technical Report 2964, INRIA, 1996.
- [11] Collino, F., Monk, P., "Optimizing the Perfectly Match Layer," Preprint, 1998.
- [12] Fletcher, C.A.J., *Computational Techniques for Fluid Dynamics Vol. I*, Springer-Verlag, Berlin, 1988.
- [13] French, W. S., *Two-Dimensional and Three-Dimensional Migration of Model-Experiment Reflection Profiles*, *Geophysics*, Vol. 39, 1974, pp. 265-277.
- [14] Gibson, B., Larner, K. and Levin, S., *Efficient 3-D Migration in Two Steps*, *Geophysical Prospecting*, Vol. 31, 1983, pp. 1-33.
- [15] Graves, R.W. and Clayton, R.W., *Modeling Acoustic Waves with Paraxial Extrapolators*, *Geophysics*, Vol. 55, No. 3, March, 1990, pp. 306-319.

- [16] Gustafson, J.L., *Reevaluating Amdahl's Law*, Communications of the ACM, Vol. 31, No. 5, May, 1988, pp. 532-533.
- [17] Jakubowicz, H. and Levin, S., *A Simple Theory of 3-D Migration*, Geophysical Prospecting, Vol. 31, 1983, pp. 34-56.
- [18] Kjartansson, E., *Attenuation of seismic waves in rocks and applications in energy exploration*, Ph.D thesis, Stanford University, 1979.
- [19] Lee, Myung W. and Suh, Sang Y., *Optimization of One-Way Wave Equations*, Geophysics, Vol. 50, No. 10, October, 1985, pp. 1634-1637.
- [20] Li, Zhiming, *Compensating Finite-Difference Errors in 3-D Migration and Modeling*, Geophysics, Vol. 56, No. 10, October, 1991, pp. 1650-1660.
- [21] Ma, Z., *Finite-Difference Migration with Higher Order Approximation*, 1981 Joint Mtg. Chinese Geophys. Soc. and Society of Exploration Geophysicists, 1981.
- [22] Marchuk, G.I., *On the Theory of the Splitting-Up Method*, Proceedings of the 2nd Symposium on Numerical Solution of Partial Differential Equations, SVNPADE, 1970, pp. 469-500.
- [23] Message-Passing Interface Forum. *The MPI Message-Passing Interface Standard*, <http://www.mcs.anl.gov/mpi/standard.html>, May 1995.
- [24] Ober, C.C., Oldfield, R.A., Womble, D.E. and Van Dyke, J.P., *Seismic Imaging on the Intel Paragon*, J. Computers & Math. with Applic., Vol. 35, No. 7, 1998, pp. 65-72.
- [25] Oldfield, R.A., Semeraro, B. D. and Van Dyke, J.P., *Parallel Acoustic Wave Propagation and Generation of a Seismic Dataset*, Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, 1995, pp. 243-244.
- [26] Oldfield, R.A., Womble, D.E., Ober, C.C., *Efficient Parallel I/O in Seismic Imaging*, High Performance Computing Applications, Vol. 12, No. 3, 1998, pp. 333-344.
- [27] Romero, L.A., Ghilgia, D.C., Ober, C.C., and Morton, S.A., *Phase encoding of shot records in prestack migration*, Geophysics, In press, 2000. See also <http://www.geo-online.org/eman/emanlist.html>.
- [28] Society of Exploration Geophysicists, *Digital Tape Standards*.
- [29] Stolt, R.H. and Benson, A.K., *Seismic Migration Theory and Practice, Handbook of Geophysical Exploration*, Section I. Seismic Exploration, Vol. 5, Geophysical Press, London, 1986.

- [30] Versteeg, R., and Grau, G., *Practical aspects of seismic data inversion, the Marmousi experience*, European Association of Exploration Geophysicists, Proc. 1990 EAEG Workshop, 52nd EAEG Meeting, pp. 1-194, 1990.
- [31] Versteeg, R., *Sensitivity of prestack depth migration to the velocity model*, Geophysics, Vol. 58, No. 6, June, 1993, pp. 873-882.
- [32] Womble, D.E., Dosanjh, S.S., Van Dyke, J.P., Oldfield, R.A. and Greenberg, D.S., *3-D Seismic Imaging of Complex Geologies*, Proceedings of the '95 Simulation MultiConference, 1995.
- [33] Webster's New World Dictionary, Third college Edition, Editor in Chief Victoria Neufeldt, Cleveland and New York, 1988.
- [34] Yilmaz, O., *Seismic Data Processing, Investigations in Geophysics No. 2*, Society of Exploration Geophysicists, P.O. Box 702740, Tulsa, OK 74170-2740, 1987.

A Memory Requirements

The major data structures in Salvo are documented in Tables 7, 8, 9 and 10. The approximate number of array elements are listed on a per-node basis along with the element type (complex or float) and a brief description. A complex variable is assumed to be two floats. Below is a list of definitions for the variables used in the tables.

$blocksize$ the I/O read block size specified by the user in the *READ_TRACES* process.

b_{min} the block size of tridiagonal solves in the pipelining for spatial parallelism.

n_f the number of output image files.

n_s the number of shots to phase encode per migration.

n_t the number of samples in each trace.

n_x the number of global grid points in the x direction.

n_y the number of global grid points in the y direction.

$n_{2,x}$ the next power of 2 greater than n_x .

$n_{2,y}$ the next power of 2 greater than n_y .

$n_{v,x}$ the number of grid points in the velocity model for the x direction.

$n_{v,y}$ the number of grid points in the velocity model for the y direction.

n_ω the number of global grid points in the frequency, ω , direction.

$\tilde{n}$ the dimensions for the stack file from a previous migration(s).

$\hat{n}$ the dimensions for the new file domain after stacking.

p_{io} the number of processors in the I/O partition.

p_x the number of processors in the x direction.

p_y the number of processors in the y direction.

p_ω the number of processors in the frequency, ω , direction.

Table 7: Memory Requirements for the *READ_TRACES* process.

Number of Elements	Element Type	Description
$2 \frac{n_x}{p_x} \frac{n_y}{p_y} \frac{n_\omega}{p_\omega}$	complex	Memory for source and receiver wavefields
6 blocksize	float	Three memory blocks for double buffering
$2 n_s n_t$	complex	Phase encoding arrays to encode traces

Table 8: Memory requirements for the compute partition of the *MIG_SPLIT* process.

Number of Elements	Element Type	Description
$\frac{n_x}{p_x} \frac{n_y}{p_y} \frac{n_\omega}{p_\omega}$	complex	Memory for Graves and Clayton filter, if used.
$n_{2,x} n_{2,y}$	complex	Memory for Li filter, if used. If LI_NOCOPY is defined, then this is not required.
$\max \left[2 \frac{n_x}{p_x} \frac{n_y}{p_y}, \max(n_x n_y, 2 n_{v,x} n_{v,y}) \right]$	float	Memory for image and working arrays, thus needs to be maximum for all needs.
$4 \max \left(\frac{n_x}{p_x}, \frac{n_y}{p_y} \right) b_{min}(\max(p_x, p_y) + 1)$	complex	Memory for tridiagonal solves.
$6 \max \left(\frac{n_x}{p_x}, \frac{n_y}{p_y} \right)$	complex	Memory for pipelined tridiagonal solves in the Graves and Clayton filter.
$4 \max(n_x, n_y)$	complex	Memory required for the coefficient separation, if COEF_SEP is defined.
$2 \frac{n_x+2}{p_x} \frac{n_y+2}{p_y}$	float	Memory for the velocity planes

Table 9: Memory requirements for reading the velocity model on I/O nodes (if I/O partition is present) or on compute nodes (if I/O partition is not present). * denotes memory required on I/O nodes when running with an I/O partition, but this memory is not required on compute nodes when running without an I/O partition because it has been allocated in Table 8.

Number of Elements	Element Type	Description
$2n_{v,x}n_{v,y}$	float	* Memory for two depth planes of the velocity model.
$n_{v,x}n_{v,y}$	float	Memory for intermediate interpolated depth plane.
$\frac{n_x+2}{p_x} \frac{n_y+2}{p_y}$	float	* Memory for the final interpolated velocity plane for local domain including ghost points.
$n_{v,x}n_{v,y}$	float	Memory for communication buffer.

Table 10: Memory requirements for writing and stacking the image.

Number of Elements	Element Type	Description
Compute Partition (Memory not required if I/O partition present)		
$\frac{n_x}{p_x} \frac{n_y}{p_y}$	float	Memory for writing image.
$\widetilde{n_x} \widetilde{n_y} + \widehat{n_x} \widehat{n_y}$	float	Memory for stacking the image.
I/O Partition		
$2n_x n_y$	float	Memory for two global copies of the image plane.
$n_x \frac{n_y}{p_{io}} + \widetilde{n_x} \frac{\widetilde{n_y}}{p_{io} n_f} + \widehat{n_x} \frac{\widehat{n_y}}{p_{io} n_f}$	float	Memory for stacking the image.

Table 11: Optimized Coefficients for the Square-Root Operator

Maximum Angle		α_l	β_l
5	$m = 0, l = 0$	0.000000000	0.000000000
15	$m = 1, l = 1$	0.500000000	0.000000000
45	$m = 1, l = 1$	0.500000000	0.250000000
60	$m = 1, l = 1$	0.500000000	0.355000000
65	$m = 1, l = 1$	0.478242060	0.376369527
75	$m = 1, l = 1$	0.454814230	0.446184960
80	$m = 2, l = 1$	0.040315157	0.873981642
	$l = 2$	0.457289566	0.222691983
87	$m = 3, l = 1$	0.004210420	0.972926132
	$l = 2$	0.081312882	0.744418059
	$l = 3$	0.414236605	0.150843924
90-	$m = 4, l = 1$	0.000523275	0.994065088
	$l = 2$	0.014853510	0.919432661
	$l = 3$	0.117592008	0.614520676
	$l = 4$	0.367013245	0.105756624
90	$m = 5, l = 1$	0.000153427	0.997370236
	$l = 2$	0.004172967	0.964827992
	$l = 3$	0.033860918	0.824918565
	$l = 4$	0.143798076	0.483340757
	$l = 5$	0.318013812	0.073588213

B Migration: Miscellaneous

B.1 Lee and Suh Expansion Coefficients

For reference, the optimized coefficients for the Pade approximation to the square-root operator, Eq. (3.3), are repeated in Table 11 from Lee and Suh [19]. These coefficients are specified in *src/include/opt_coeff.inc*.

B.2 Finite-Difference Equations of Tridiagonal Solves

Each term in the approximation of the paraxial wave equation has been reduced to a ratio of simple polynomials in S_x or S_y . This allows the efficient Thomas algorithm to be used for the solution. The finite-difference coefficients for the x -direction can be determined (the y -direction can be similarly found)

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \frac{\alpha_1 S_x}{1 + \beta_1 S_x} P. \quad (\text{B.1})$$

Moving the denominator of the right-hand side to the left-hand side, one can find

$$\left[\frac{\partial P}{\partial z} + \beta_1 \frac{v^2}{\omega^2} \frac{\partial^2}{\partial x^2} \left(\frac{\partial P}{\partial z} \right) \right] = \pm \alpha_1 \frac{v^2}{\omega^2} \left(\frac{i\omega}{v} \right) \frac{\partial^2 P}{\partial x^2}$$

The Crank-Nicholson method is used to discretize in the depth direction, which produces

$$\begin{aligned} & \frac{1}{\Delta z} \left\{ P^{k+1} - P^k + \beta_1 \frac{v^2}{\omega^2} \left[\frac{\partial^2 P^{k+1}}{\partial x^2} - \frac{\partial^2 P^k}{\partial x^2} \right] \right\} \\ &= \frac{1}{2} \left[\pm \alpha_1 \frac{v^2}{\omega^2} \left(\frac{i\omega}{v} \right) \frac{\partial^2 P^{k+1}}{\partial x^2} \right] + \frac{1}{2} \left[\pm \alpha_1 \frac{v^2}{\omega^2} \left(\frac{i\omega}{v} \right) \frac{\partial^2 P^k}{\partial x^2} \right]. \end{aligned}$$

The second-order, compact finite-difference operator can be introduced

$$\frac{\partial^2}{\partial x^2} = \frac{\delta_x^2}{1 + \lambda \Delta x^2 \delta_x^2}$$

where

$$\delta_x^2 u = \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}$$

is the central-difference operator, and λ is the compact coefficient to increase the finite-difference accuracy ($\lambda = 1/12$ for Taylor series expansion will produce a fourth-order scheme). Thus the discretization scheme becomes

$$\begin{aligned} & \frac{1}{\Delta z} \left\{ P^{k+1} - P^k + \left(\lambda \Delta x^2 + \beta_1 \frac{v^2}{\omega^2} \right) \left[\delta_x^2 P^{k+1} - \delta_x^2 P^k \right] \right\} \\ &= \frac{1}{2} \left[\pm \alpha_1 \frac{v^2}{\omega^2} \left(\frac{i\omega}{v} \right) \delta_x^2 P^{k+1} \right] + \frac{1}{2} \left[\pm \alpha_1 \frac{v^2}{\omega^2} \left(\frac{i\omega}{v} \right) \delta_x^2 P^k \right], \\ & P_{i,j}^{k+1} + \left(\lambda + \beta_1 \frac{v^2}{\omega^2 \Delta x^2} - \pm \frac{\alpha_1}{2} \frac{\Delta z}{\Delta x^2} \frac{i v}{\omega} \right) \left[P_{i-1,j}^{k+1} - 2P_{i,j}^{k+1} + P_{i+1,j}^{k+1} \right] \\ &= P_{i,j}^k + \left(\lambda + \beta_1 \frac{v^2}{\omega^2 \Delta x^2} + \pm \frac{\alpha_1}{2} \frac{\Delta z}{\Delta x^2} \frac{i v}{\omega} \right) \left[P_{i-1,j}^k - 2P_{i,j}^k + P_{i+1,j}^k \right], \end{aligned}$$

or finally

$$a_i P_{i-1,j}^{k+1} + b_i P_{i,j}^{k+1} + c_i P_{i+1,j}^{k+1} = d_i$$

where

$$\begin{aligned} A &= \lambda + \frac{\beta_1 v^2}{\Delta x^2 \omega^2} - \pm \frac{\alpha_1}{2} \frac{\Delta z}{\Delta x^2} \frac{i v}{\omega} \\ A^* &= \text{conjugate of } A \\ a_i &= A \\ b_i &= 1 - 2A \\ c_i &= A \\ d_i &= A^* P_{i-1,j}^k + (1 - 2A^*) P_{i,j}^k + A^* P_{i+1,j}^k. \end{aligned}$$

With the above coefficients, a tridiagonal solve can be used to determine the solution of Eq. (B.1). Similarly for the y direction, the coefficients can be determined to be

$$a_i P_{i,j-1}^{k+1} + b_i P_{i,j}^{k+1} + c_i P_{i,j+1}^{k+1} = d_i,$$

where a_i , b_i and c_i are unchanged from above except for changing Δx^2 to Δy^2 , and the right-hand side is accordingly

$$d_i = A^* P_{i,j-1}^k + (1 - 2A^*) P_{i,j}^k + A^* P_{i,j+1}^k.$$

B.3 Comparison of Splitting Methods

The method of fractional steps (MOFS) is useful in numerical methods to reduce memory storage (it only requires one depth or time level) and allows the use of very efficient algorithms (Thomas Algorithm) to generate a solution. However fractional step methods are different than approximate factorization (AF) or alternating direction implicit (ADI) schemes and some comparison would be helpful. Looking at the first two terms of Eq. (3.3),

$$\frac{\partial P}{\partial z} = \pm \frac{i\omega}{v} \left[1 + \frac{\alpha_1 S}{1 + \beta_1 S} \right] P = [L_1 + L_2] P \quad (\text{B.2})$$

where $L_1 = \pm(i\omega/v)$ and $L_2 = \pm(i\omega/v)(\alpha_1 S)/(1 + \beta_1 S)$. The two operators would be split as

$$\frac{1}{2} \frac{\partial P}{\partial z} = L_1 P$$

and

$$\frac{1}{2} \frac{\partial P}{\partial z} = L_2 P$$

or after discretization

$$\frac{1}{2} \frac{P^{k+1/2} - P^k}{\Delta z/2} = \frac{1}{2} (L_1 P^{k+1/2} + L_1 P^k)$$

$$\frac{1}{2} \frac{P^{k+1} - P^{k+1/2}}{\Delta z/2} = \frac{1}{2} (L_2 P^{k+1} + L_2 P^{k+1/2}).$$

Rearranging one can find

$$\left[1 - \frac{\Delta z}{2} L_1 \right] P^{k+1/2} = \left[1 + \frac{\Delta z}{2} L_1 \right] P^k \quad (\text{B.3})$$

$$\left[1 - \frac{\Delta z}{2} L_2 \right] P^{k+1} = \left[1 + \frac{\Delta z}{2} L_2 \right] P^{k+1/2} \quad (\text{B.4})$$

This outlines the method of fractional steps. First an intermediate solution, $P^{k+1/2}$, is determined through Eq. (B.3), then the next depth step solution, P^{k+1} , is determined by Eq. (B.4). It should be noted that the first half step is only dependent on L_1 and the second is only dependent on L_2 . $P^{k+1/2}$ needs to be eliminated between these equations so that a comparison with approximate factorization can be made.

$$\left[1 - \frac{\Delta z}{2} L_2 \right] P^{k+1} = \left[1 + \frac{\Delta z}{2} L_2 \right] \frac{\left[1 + \frac{\Delta z}{2} L_1 \right]}{\left[1 - \frac{\Delta z}{2} L_1 \right]} P^k$$

or

$$\left[1 - \frac{\Delta z}{2} L_2 \right] \left[1 - \frac{\Delta z}{2} L_1 \right] P^{k+1} = \left[1 + \frac{\Delta z}{2} L_2 \right] \left[1 + \frac{\Delta z}{2} L_1 \right] P^k \quad (\text{B.5})$$

The L_1 and L_2 operators have been allowed to commute (*i.e.*, $L_1 L_2 \sim L_2 L_1$) in Eq. (B.5). The approximate factorization method can be completed in two stages by replacing $[1 - (\Delta z/2)L_1] P^{k+1}$ with Q , and solving Eq. (B.5) for Q , and then solving $[1 - (\Delta z/2)L_1] P^{k+1} = Q$ for P^{k+1} . The original form of the continued-fraction-expansion equation can be reconstituted by neglecting terms of $(\Delta z^2/4)L_1 L_2(P^{k+1} - P^k)$.

$$\begin{aligned} \left[1 - \frac{\Delta z}{2}(L_1 + L_2)\right] P^{k+1} &= \left[1 + \frac{\Delta z}{2}(L_1 + L_2)\right] P^k \\ \frac{P^{k+1} - P^k}{\Delta z} &= \frac{1}{2}(L_1 + L_2) P^{k+1} + \frac{1}{2}(L_1 + L_2) P^k \\ \frac{\partial P}{\partial z} &= [L_1 + L_2] P = \pm \frac{i\omega}{v} \left[1 + \frac{\alpha_1 S}{1 + \beta_1 S}\right] P \end{aligned}$$

This is exactly Eq. (B.2). This illustrates that the two methods are similar so long as the operators L_1 and L_2 commute, and it also shows that the $(\Delta z^2/4)L_1 L_2 (P^{k+1} - P^k)$ term is the neglected term for both methods.

MOFS, AF and ADI methods are all second-order accurate in space and time with the appropriate boundary conditions, and they are all unconditionally stable for two dimensions (ADI is conditionally stable for three dimensions). However if the operators do not commute, then MOFS is only first-order accurate. Marchuk [22] indicates that this may be circumvented by alternating which direction (x or y ; Eq. (B.3) or Eq. (B.4)) is completed as the first step. ADI methods generally require two evaluations of spatial operators (one for each step), while MOFS and AF methods only require one spatial evaluation. Thus there is a computational savings for MOFS and AF methods. Additionally ADI and AF methods generally require two levels of storage for the solution array, while MOFS only requires one. For the large problems associated with seismic imaging, the additional memory savings is useful.

B.4 Discretization of Boundary Conditions

Five different boundary conditions have been implemented in Salvo: zero-value Dirichlet boundary condition (BC), zero-slope Neumann BC, Clayton and Engquist's [9] absorbing BC, UT at Dallas absorbing BC, and Xu's absorbing BC. The discretization for the first two boundary conditions are simple, and the discretization for the absorbing BCs are similar. Dirichlet BCs require that $P_1 = 0$ along the boundaries on the left-hand side (LHS) and $P_n = 0$ on the right-hand side (RHS). Therefore the tridiagonal coefficients are simply

$$\begin{array}{ll} LHS & (P_1 = 0) \\ b_1 & = 1 \\ c_1 & = 0 \\ d_1 & = 0 \end{array} \quad \begin{array}{ll} RHS & (P_n = 0) \\ a_n & = 0 \\ b_n & = 1 \\ d_n & = 0 \end{array}$$

Neumann BCs need $\partial P / \partial(x, y) = 0$ along the left and right boundaries, and therefore require

<i>LHS</i>	<i>RHS</i>
$b_1 = 1 - 2A$	$a_n = 2A$
$c_1 = 2A$	$b_n = 1 - 2A$
$d_1 = (1 - 2A^*)P_1^k + 2A^*P_2^k$	$d_n = 2A^*P_{n-1}^k + (1 - 2A^*)P_n^k$

where

$$A = \lambda + \frac{\beta_1}{\Delta x^2} \frac{v^2}{\omega^2} - \pm \frac{\alpha_1}{2} \frac{\Delta z}{\Delta x^2} \frac{iv}{\omega}.$$

and A^* is the conjugate of A . The Dirichlet and Neumann boundary conditions are perfect reflectors, which can cause reflected artifacts. The major difference between them is the sign of the reflected wave.

For absorbing BCs, the partial differential equation from Clayton and Engquist has been implemented (shown for the x direction, and the y direction can be similarly written by replacing ∂x with ∂y)

$$s_w i \frac{\partial P}{\partial z} - s_{lr} c_b \frac{v}{\omega} \frac{\partial^2 P}{\partial x \partial z} = -a_b \frac{\omega}{v} P - s_w s_{lr} b_b i \frac{\partial P}{\partial x}.$$

Each of the absorbing boundary conditions in Salvo can be arranged into this general format with differing coefficients. Using the Crank-Nicholson method, the discretized equation can be written

$$\xi P^{k+1} + \eta \Delta x \frac{\partial P^{k+1}}{\partial x} = -\xi^* P^k + \eta^* \Delta x \frac{\partial P^k}{\partial x} \quad (\text{B.6})$$

where

$$\xi = \left(\frac{a_b \omega}{2} \frac{v}{\omega} + i \frac{s_w}{\Delta z} \right)$$

$$\eta = \left(-s_{lr} \frac{c_b}{\Delta x \Delta z} \frac{v}{\omega} + i s_w s_{lr} \frac{b_b}{2 \Delta x} \right)$$

for the Clayton and Engquist absorbing BCs. The coefficients ξ and η will differ for the other absorbing BCs, but Eq. (B.6) will hold. Performing the one-sided differences on the boundaries to maintain tridiagonal system of equations, the finite-difference equations for left side boundaries become

$$(\xi - \eta)P_1^{k+1} + \eta P_2^{k+1} = -(\xi^* + \eta^*)P_1^k + \eta^* P_2^k$$

or

$$b_1 P_1^{k+1} + c_1 P_2^{k+1} = d_1$$

and for right side boundaries

$$-\eta P_{n-1}^{k+1} + (\xi + \eta)P_n^{k+1} = -\eta^* P_{n-1}^k + (-\xi^* + \eta^*)P_n^k$$

or

$$a_n P_{n-1}^{k+1} + b_n P_n^{k+1} = d_n.$$

B.5 Discretization of the Perfectly Matched Layer

Equations (3.14) and (3.15) explicitly define the terms involving $\sigma(x)$ in Eq. (3.13). Equation (3.13) is used to create an alternative central difference operator. Outside the PML the traditional central difference operator

$$\delta_x^2 u = \frac{u_{m+1} - 2u_m + u_{m-1}}{\Delta x^2} \quad (\text{B.7})$$

is used. Equation (3.13) provides the basis for an equivalent operator inside the PML region

$$\delta_x^2 u = \frac{\omega^2}{\Delta x^2 (i\omega + \sigma_m)^3} \left[\frac{1}{2} \frac{p}{m - m_0} \sigma_m (u_{m+1} - u_{m-1}) - (i\omega + \sigma_m) (u_{m+1} - 2u_m + u_{m-1}) \right]. \quad (\text{B.8})$$

The right hand side of either Eq. (B.7) or Eq. (B.8) is substituted for δ_x^2 in the second order compact finite-difference operator

$$\frac{\partial^2}{\partial x^2} = \frac{\delta_x^2}{1 + \lambda \Delta x^2 \delta_x^2}. \quad (\text{B.9})$$

Outside the PML, the definition of δ_x^2 in Eq. (B.7) is used in Eq. (B.9). Inside the PML, the definition of δ_x^2 in Eq. (B.8) is used in Eq. (B.9). Regardless of whether Eq. (B.7) or Eq. (B.8) is used, the Crank-Nicholson discretization in depth of the paraxial wave equation yields a tridiagonal system of equations of the form

$$a_m P_{m-1,n}^{k+1} + b_m P_{m,n}^{k+1} + c_m P_{m+1,n}^{k+1} = d_m.$$

Inside the PML, the coefficients for the tridiagonal system are

$$a_m = AB \quad (\text{B.10})$$

$$b_m = 1 - 2AC \quad (\text{B.11})$$

$$c_m = AB \quad (\text{B.12})$$

$$d_m = A^* B P_{m-1,n}^k + (1 - 2A^* C) P_{m,n}^k + A^* B P_{m+1,n}^k \quad (\text{B.13})$$

where

$$A = \lambda + \frac{\beta_1}{\Delta x^2} \frac{v^2}{\omega^2} - \pm \frac{\alpha_1}{2} \frac{\Delta z}{\Delta x^2} \frac{iv}{\omega} \quad (\text{B.14})$$

$$A^* = \text{conjugate of } A$$

$$B = \frac{\omega^2}{(i\omega + \sigma)^3} \left(\frac{p\sigma}{2(m - m_0)} - (i\omega + \sigma) \right) \quad (\text{B.15})$$

$$C = \frac{\omega^2}{(i\omega + \sigma)^3} (-(i\omega + \sigma)). \quad (\text{B.16})$$

Outside the PML region, the coefficients in the tridiagonal system are calculated using equations are identical to Eq. (B.10) - Eq. (B.14) if B and C were replaced with 1. (This is not a coincidence - a PML region with no absorption should behave the same as the main migration domain. The fact that $B = C = 1$ where $\sigma(x) = 0$ means our expectation is satisfied.) As a result, a single tridiagonal system is used to compute the wavefield both inside the PML region and in the main migration domain.

The above PML formulation was developed for a layer which absorbs energy traveling in the positive x direction. A PML on the left edge of a migration domain will want to absorb energy traveling in the negative x direction. In this case there is a sign change in the basic transformation

$$\frac{\partial}{\partial x} \rightarrow \frac{-i\omega}{i\omega + \sigma(x)} \frac{\partial}{\partial x}.$$

For three-dimensional problems, one will also want to attenuate energy exiting the domain traveling parallel to the y axis. Operator splitting, which simplifies three-dimensional migration, results in separate tridiagonal solves in the x and y directions. Because of this, extending PML to three dimensions requires only modifying tridiagonal equations in the y direction in the exact same manner as those for the x direction were modified in the two-dimensional case.

B.6 Graves and Clayton Filter

The Graves and Clayton phase-correction filter [15] is shown here for reference and to match nomenclature in the previous developments. Graves and Clayton only retain the first cross-product term in Eq. (3.6),

$$\frac{\partial P}{\partial z} = \left[-\pm 2 \frac{i\omega}{v} \alpha_1 \beta_1 S_x S_y \right] P$$

or

$$\frac{\partial P}{\partial z} = (i\hat{F})P,$$

where $\hat{F} = \left[-\pm 2 \frac{\omega}{v} \alpha_1 \beta_1 S_x S_y \right]$, and the solution is $P^{filtered} = \exp \left[i\hat{F}\Delta z \right] P^{unfiltered}$.

Thus the phase function, $\hat{F}\Delta z$, has the desired phase to correct for anisotropy along the x - y diagonals. A filter, F , which applies the phase correction can be obtained by

$$\hat{F} = \frac{\Im(F)}{\Re(F)} \Rightarrow F = 1 - \pm 2i \frac{\omega}{v} \alpha_1 \beta_1 \Delta z S_x S_y$$

or in the Fourier domain, one can find

$$F = 1 - \pm 2i \frac{\omega}{v} \alpha_1 \beta_1 \Delta z K_x^2 K_y^2$$

where $K_x = (v/\omega)k_x$, and $K_y = (v/\omega)k_y$. Graves and Clayton found that the filter can be improved through an empirical coefficient, ϵ_0 , and additionally can be made stable through the inclusion of damping functions. They suggested the following form

$$F = D_1 \left[1 - \pm 2i \epsilon_0 \frac{\omega}{v} \alpha_1 \beta_1 D_2 \Delta z K_x^2 K_y^2 \right]$$

where

$$\begin{aligned} D_1 &= (1 + \epsilon_1 K_x^4)^{-1} (1 + \epsilon_1 K_y^4)^{-1} \\ D_2 &= (1 + \epsilon_2 K_x^4)^{-1} (1 + \epsilon_2 K_y^4)^{-1} \\ \epsilon_0 &= 1.5 \\ \epsilon_1 &= 0.01 \\ \epsilon_2 &\stackrel{\text{stable}}{\geq} [\epsilon_0 \alpha_1 \beta_1 \Delta z \omega / (2v)]^2 / \epsilon_1 - (3/8) \epsilon_1 \end{aligned}$$

Later in the solution, the square root of ϵ_1 and ϵ_2 are computed. To remove any difficulties with negative values of ϵ_2 , the second term has been dropped which still satisfies the stability requirement (*i.e.*, $\epsilon_2 = [\epsilon_0 \alpha_1 \beta_1 \Delta z \omega / (2v)]^2 / \epsilon_1$).

The operator-splitting correction can be stably solved in a two stage approach using efficient tridiagonal solves.

$$Q = D_2 P'$$

$$P^{\text{filtered}} = D_1 [P^{\text{unfiltered}} + Q]$$

where

$$P' = - \pm 2i \epsilon_0 \frac{\omega}{v} \alpha_1 \beta_1 \Delta z K_x^2 K_y^2,$$

or in the spatial domain,

$$P' = - \pm 2i \epsilon_0 \frac{\omega}{v} \alpha_1 \beta_1 \Delta z S_x S_y P^{\text{unfiltered}}.$$

The general solution for one stage of the filter, $Q_{\text{out}} = D_\ell Q_{\text{in}}$, is shown by Graves and Clayton, but for this problem

$$\begin{aligned} \left[1 + \eta_0 \frac{v^2}{\omega^2} \frac{\partial^2}{\partial x^2} \right] Q_{\eta_0} &= Q_{\text{in}} \\ \left[1 + \eta_1 \frac{v^2}{\omega^2} \frac{\partial^2}{\partial x^2} \right] Q_{\eta_1} &= Q_{\eta_0} \\ \left[1 + \nu_0 \frac{v^2}{\omega^2} \frac{\partial^2}{\partial y^2} \right] Q_{\nu_0} &= Q_{\eta_1} \\ \left[1 + \nu_1 \frac{v^2}{\omega^2} \frac{\partial^2}{\partial y^2} \right] Q_{\text{out}} &= Q_{\nu_0} \end{aligned}$$

where

$$\begin{aligned} \eta_m &= \sqrt{\epsilon_\ell} \exp \left[-i(2m+1) \frac{\pi}{2} \right] = (-1)^{m+1} i \sqrt{\epsilon_\ell} \\ \nu_n &= \sqrt{\epsilon_\ell} \exp \left[-i(2n+1) \frac{\pi}{2} \right] = (-1)^{n+1} i \sqrt{\epsilon_\ell} \end{aligned}$$

Thus applying this solution to both stages requires a total of 8 tridiagonal solves. The above equations can be generalized to

$$\left[1 + \mu \frac{v^2}{\omega^2} \frac{\partial^2}{\partial x^2} \right] Q_{\text{out}} = Q_{\text{in}}$$

where μ is η_m or ν_n . Using compact finite-differences

$$\frac{\partial^2}{\partial x^2} = \frac{\delta_x^2}{1 + \lambda \Delta x^2 \delta_x^2}$$

where δ_x^2 is the central-difference operator, and λ is a coefficient to increase the finite-difference accuracy to fourth order, one can find

$$\left[1 + \left(\lambda \Delta x^2 + \mu \frac{v^2}{\omega^2} \right) \delta_x^2 \right] Q_{out} = (1 + \lambda \Delta x^2 \delta_x^2) Q_{in}$$

The discretized equation can be written

$$B Q_{out,i-1} + (1 - 2B) Q_{out,i} + B Q_{out,i+1} = \lambda Q_{in,i-1} + (1 - 2\lambda) Q_{in,i} + \lambda Q_{in,i+1}$$

where $B = \lambda + \mu v^2 / (\Delta x^2 \omega^2)$. The boundary conditions for the filter were initially set to zero-value along the edges. However, this produced reflections from the boundaries. To reduce the reflections, the B3 absorbing boundary conditions of Clayton and Engquist were tried, but they produced unstable oscillations along the boundaries. The simple B1 absorbing boundary conditions of Clayton and Engquist were also implemented and found to be stable.

$$k_x = s_w b_b \frac{\omega}{v}$$

or

$$\frac{\partial P}{\partial x} - s_w \frac{i\omega}{v} b_b P = 0$$

For left-side boundaries, one can find

$$\left(1 - i s_w \frac{\omega}{v} b_b \Delta x \right) P_1 - P_2 = 0$$

and for right-side boundaries, one can find

$$-P_{n-1} + \left(1 - i s_w \frac{\omega}{v} b_b \Delta x \right) P_n = 0.$$

Clayton and Engquist suggest $b_b = 1/2$, which will effectively absorb waves with a 30° incidence.

B.7 Analytic Solutions to Impulse in a Half Space

This section is taken from notes provided by Louis Romero. To develop a better imaging condition (see Section 3.4), it will be useful to determine analytic solutions for an impulse in a half space for two and three dimensions. The solution to the two-dimensional problem, $p(x, z, t)$, will be determined by solving

$$\frac{\partial^2 p}{\partial t^2} - v^2 \nabla^2 p = 0$$

with

$$\begin{aligned} p(x, 0, t) &= \delta(x)\delta(t) \\ p(x, z, 0) &= 0 \text{ for } z > 0 \end{aligned}$$

where δ is the Dirac delta function. This impulse problem is commonly used to test one-way approximations to the wave equation, and is very close to what is actually carried out in the field. This problem differs from the standard impulse problem for the wave equation because the pressure is required to vanish at the free surface provided $x^2 + y^2 + t^2 \neq 0$. If the earth were homogeneous and only transmitted pressure waves, any real seismic event could be modeled by convolving the response of this signal with functions giving the temporal and spatial distribution of the seismic source. The temporally Fourier transformed impulse response must satisfy

$$\nabla^2 P_2 + \frac{\omega^2}{v^2} P_2 = 0$$

with

$$P_2(x, 0, \omega) = \delta(x)$$

where $P_2(x, z, \omega)$ represents the two-dimensional outgoing waves as $x^2 + z^2 \rightarrow \infty$. The solution to this problem is given in polar coordinates by

$$P_2(r, \theta, \omega) = -\frac{i\omega}{2v} \cos(\theta) H_1^2\left(\frac{\omega r}{v}\right)$$

where $r = \sqrt{x^2 + z^2}$, $\cos(\theta) = z/r$,

$$H_1^2\left(\frac{\omega r}{v}\right) = J_1\left(\frac{\omega r}{v}\right) - iY_1\left(\frac{\omega r}{v}\right)$$

and J_1 and Y_1 are Bessel functions of the first and second kind. This solution satisfies the reduced wave equation, and represents outgoing waves as $r \rightarrow \infty$, and $P_2(x, 0, \omega) = 0$ for $x \neq 0$. It only remains to examine the behavior when $x = z = 0$. Note that as $\omega r/v \rightarrow 0$, J_1 is bounded and $Y_1 \approx -2v/(\pi\omega r)$, and hence

$$P_2(r, \theta, \omega) \approx A(x, z) = \frac{\cos(\theta)}{\pi r} = \frac{1}{\pi} \frac{z}{x^2 + z^2} \text{ as } x^2 + z^2 \rightarrow 0.$$

The function $A(x, z) = \frac{1}{\pi} \frac{z}{x^2 + z^2}$ satisfies $A(x, 0)$ if $x \neq 0$, but

$$\lim_{z \rightarrow 0} \int_{-\infty}^{\infty} A(x, z) dz = 1.$$

It follows that the function $P_2(x, 0, \omega) = \delta(x)$ and hence is the desired solution to the reduced wave equation. The behavior of the solution for $\omega r/v \gg 1$ is of primary interest. The asymptotic expansion in this case is

$$H_1^2\left(\frac{\omega r}{v}\right) = \sqrt{\frac{2v}{\pi\omega r}} \exp\left[-\frac{i\omega r}{v} + i\frac{3\pi}{4}\right]$$

It follows that for $\omega r/v \gg 1$ the wavefield can be represented as

$$\begin{aligned} P_2(r, \theta, \omega) &\approx -i \cos(\theta) \sqrt{\frac{\omega}{2\pi v r}} \exp \left[-\frac{i\omega r}{v} + i\frac{3\pi}{4} \right] \\ P_2(r, \theta, \omega) &\approx \cos(\theta) \sqrt{\frac{\omega}{2\pi v r}} \exp \left[-\frac{i\omega r}{v} + i\frac{\pi}{4} \right] \end{aligned} \quad (\text{B.17})$$

The three-dimensional solution to the following initial boundary value problem can be similarly determined.

$$\frac{\partial^2 p}{\partial t^2} - v^2 \nabla^2 p = 0$$

with

$$p(x, y, 0, t) = \delta(x)\delta(y)\delta(t)$$

$$p(x, y, z, 0) = 0 \text{ for } z > 0.$$

The temporally Fourier transformed impulse response must satisfy

$$\nabla^2 P_3 + \frac{\omega^2}{v^2} P_3 = 0$$

with

$$P_3(x, y, 0, \omega) = \delta(x)\delta(y)$$

where $P_3(x, y, z, \omega)$ represents the three-dimensional outgoing waves as $x^2 + y^2 + z^2 \rightarrow \infty$. The solution to this problem is given in polar coordinates by

$$P_3(r, \theta, \omega) = \frac{\cos(\theta)}{2\pi} \exp \left[-\frac{i\omega r}{v} \right] \left(\frac{1}{r^2} + \frac{i\omega}{vr} \right)$$

where $r = \sqrt{x^2 + y^2 + z^2}$, and $\cos(\theta) = z/r$. This solution satisfies the reduced wave equation, represents outgoing waves as $r \rightarrow \infty$, and $P_3(x, y, 0, \omega) = 0$ for $x^2 + y^2 \neq 0$. It only remains to examine the behavior when $x = y = z = 0$. To do this, note that as $r \rightarrow 0$, $P_3 \approx A(x, y, z) = \frac{z}{2\pi(x^2 + y^2 + z^2)^{3/2}}$. A direct calculation shows that

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} A(x, y, z) dx dy = 1 \quad \forall z$$

It follows that

$$\lim_{z \rightarrow 0} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P_3(x, y, z) dx dy = 1$$

even though $P_3(x, y, 0, \omega) = 0$ for $x^2 + y^2 \neq 0$. One can conclude that $P_3(x, y, 0, \omega) = \delta(x)\delta(y)$. Once again the primary interest is in the solution for $\omega r/v \gg 1$. In this case the Fourier transform of the downwards traveling wave can be approximated by

$$P_3(r, \theta, \omega) \approx \frac{\cos(\theta)}{2\pi} \exp \left[-\frac{i\omega r}{v} \right] \frac{i\omega}{vr}. \quad (\text{B.18})$$

It should be noted that the aliasing criterion requires that $\Delta\omega R/v < 2\pi$, where R is the depth of migration, and $\Delta\omega$ is the spacing of the frequencies. The sampling rate is usually chosen so that this criterion is just barely satisfied. This means that $\Delta\omega R/v \approx 2\pi$, and the smallest frequencies will already satisfy $\omega R/v \gg 1$.

C I/O: Miscellaneous

C.1 Velocity File Information²⁷

The binary velocity file format varies depending on whether a single file or multiple files are used to store the velocity model. In the usual case with a single file, the velocity model is simply stored as a binary volume ($n_{vx} \times n_{vy} \times n_{vz}$). When using multiple files the velocity model is partitioned among the files in a manner similar to that for accumulation files discussed in Section C.2; a one-dimensional partitioning of the model is performed along the y -axis. Each velocity model file includes an initial header indicating the file number (0, 1, ..., number of files - 1) and the size of the velocity x - y plane in words/floats, which is followed by the binary volume. A contiguous x - y plane for the entire velocity model is formed by reading the plane from each of the velocity files in ascending order. With or without an I/O partition, the files are partitioned among the I/O nodes in an even as possible block format that ensures that the data is contiguous across multiple files for each I/O processor (*e.g.*, Figure 41).

C.2 Accumulation File Information²⁸

The binary output image produced by Salvo is in an accumulation file format that facilitates stacking. This format is simply the entire binary volume prepended by a header with fold information. The exact format of this file is as follows

$n_x \ n_y \ n_z$
$dx \ dy \ dz$
$xmin \ ymin \ zmin$
fold = $n_x \times n_y$
image = $n_x \times n_y \times n_z$

The accumulation file includes a nine word header of floats (single precision floating point numbers) that specify the dimensions of the image, the grid spacing, and the image starting or minimum coordinates. This header is followed by the two-dimensional fold ($n_x n_y$ floats) and then the actual migration volume ($n_x n_y n_z$ floats), which consists of n_z (x - y) slices (*i.e.*, the data ordering is x , y , and then z).

When multiple accumulation files are specified for writing (or stacking) the image, the file data domain is partitioned along the y -axis in a one-dimensional distribution. The files are partitioned among the I/O nodes in an even as possible block format that ensures that the data is contiguous across multiple files for each I/O processor. An example can clarify this mapping scheme. Consider the case

²⁷Subroutines: openVel; files: openVel.c

²⁸Subroutines: openAcc, openAccNS, write_acc; files: openAcc.c, write_acc.c

Figure 41: Partitioning the three-dimensional data domain using a one-dimensional partitioning along the y -axis in the case with 5 files and 2 I/O processors.

with 2 I/O processors using 5 files; the x - y image plane (for each depth level z) is partitioned as shown in Figure 41. Each file is an accumulation file that specifies its own subdomain with the header and includes the appropriate image portion.

C.3 Enhanced SEP File Format²⁹

The format chosen for the SEG/EAGE 3D Modeling Committee calculations is an enhanced Stanford Exploration Project (SEP) binary format. This format was used when generating the original data for the SEG/EAGE Salt and Overthrust models, and is how the files are stored at Lawrence Livermore National Laboratory's FAST storage system. Each seismic trace is composed of a trace header and seismic data. There is no file header. Within the file, each seismic trace is preceded by a header which contains twenty-five (25) 4 byte IEEE (floating point) binary words. The 25 header words have the following definitions:

- Word 1 Seismic record length in milliseconds
- Word 2 Number of the shot
- Word 3 Sample interval in microseconds

²⁹The following information was provided by Jeff Long of LLNL.

Comment: Time in the seismic trace starts at $t = 0$. The number of seismic data points per trace is $((1000 * \text{Word 1}) / (\text{Word 3}) + 1)$. The next three words give the source location.

Word 4 X coordinate of the source (meters) in the model

Word 5 Y coordinate of the source (meters) in the model

Word 6 Z coordinate of the source (meters) in the model

The three words give the receiver location

Word 7 X coordinate of the receiver (meters) in the model

Word 8 Y coordinate of the receiver (meters) in the model

Word 9 Z coordinate of the receiver (meters) in the model

Word 10 Receiver or source type. Pressure receiver = 4; Pressure source = 14.

The next three words give the inter-trace spacing (Word 10 = 4) or the inter-source spacing (Word 10 = 14). Inter-source spacing is not very meaningful since we are using point sources in the SSD set. However, this spacing indicates how far away the next shot was in the calculation parameter setup file that contained multiple individual shots. When Word 10 = 14 (meaning the trace is the seismic source signal), Word 14, 18 and 19 are set to -1.0 and Word 17 is set to 0.

Word 11 X spacing (meters)

Word 12 Y spacing (meters)

Word 13 Z spacing (meters)

Word 14 Source-Receiver horizontal offset (meters)

Word 15 Total number of traces for this shot file

Word 16 Number of this trace within this shot file

Word 17 This word describes the type of regular two-dimensional receiver grid or one-dimensional receiver line that this seismic data trace is associated with. A receiver which is part of a regular two-dimensional receiver grid has a value between 1-30000. A two-dimensional grid can also be considered to be a set of one-dimensional lines. Values within the 1-30000 range attempt to define a set of receiver lines within the grid by assigning a unique number to a line within the grid. A value of 0 means that the trace is the seismic source and not a receiver. The numbers have the following meaning:

	0	seismic source signal
	1-10000	receiver part of the line with constant x value
	10001-20000	receiver part of the line with constant y value
	20001-30000	receiver part of the line with constant z value
	1D receiver grid (well) is described by the values:	
	30001-40000	receiver line in the x direction (horz. well)
	40001-50000	receiver line in the y direction (horz. well)
	50001-60000	receiver line in the z direction (vert. well)
Word 18	X Common Depth Point (CDP) location (meters)	
Word 19	Y Common Depth Point (CDP) location (meters)	
Word 20	X offset of sub-model (compute grid) used (meters)	
Word 21	Y offset of sub-model (compute grid) used (meters)	
Word 22	X dimension of sub-model (compute grid) used (meters)	
Word 23	Y dimension of sub-model (compute grid) used (meters)	
Word 24	= 3 meaning 3D wave propagation used in calculations	
Word 25	Number associated with the specific computer at a specific site that carried out the calculation of the shot. Norman Burkhard at LLNL and Pierre Duclos at IFP maintain the master list of numbers for this parameter.	

The seismic data follow the header for each trace and have a 4 byte IEEE (floating point) binary representation. Time in the seismic trace starts at $t=0$. The number of seismic data points per trace is $((1000 \times \text{Word 1}) / (\text{Word 3}) + 1)$. A shot contains the number of traces given by the value stored in Word 15 of the header. The last seismic trace in a shot file is the seismic pressure source used in the calculation.

C.4 GIF Generation of Image Plane Slices and Lines³⁰

Cross-sections of the image planes can be written out to file in GIF format to allow real time visualization of the migration at various depth steps. This capability can be used when either writing or stacking the image (in *process_io_requests.c*, *write_acc.c*, or *stack.c* in the directory *salvo/src/io_partition*). The routines in *action.c* and *write_gif.c* in conjunction with calls to the *gd1.2* GIF-manipulating library (*salvo/src/gd1.2*) can generate user specified *x* or *y* lines or entire image planes at various depth steps. Additional information on using this feature can be found by viewing the man pages for *action.c* and *write_gif.c*.

³⁰Files: *process_io_request.c*, *write_gif.c*, *action.c*

Table 12: Salvo utility listing with summaries.

Utility name	Utility Summary
acc2bin	converts an accumulation (acc) file to a binary file
acc2seggy	converts an acc file to a SEG-Y format file
acc2tec	converts an acc file to a Tecplot format file
accumulate	stacks a binary file with an acc file
apply_fold	applies the acc file fold to the binary image
asc2seggy	converts an ASCII file to a SEG-Y format file
bin2acc	converts a binary image to an acc file
bin2tec	converts a binary image to a Tecplot format file
binExtract	extracts shot gathers from a binary file
byteswap	byteswaps a binary file
change_header	modifies an entry in the acc file header
compare	compares the norms of two acc image files
encode	phase encodes SEP files
extract	extracts plane/volumes from an acc file
factor235	generates numbers with factors of only 2, 3, and 5
freqc	determines frequency content of a single trace
gen_plane_3d	generates a survey of shots for 3D plane reflector
gen_vel	generates a 2 layer velocity model
impulse	generates single source and receiver impulse
makevel	generates a constant velocity model
par_stack	parallel stacking program

D User Manual: Miscellaneous

D.1 Salvo Utilities

Tables 12 and 13 are a listing of the Salvo utilities with brief summaries. Additional information is available through the Salvo man pages.

D.2 Code Limitations and Assumptions

The following is a list of known limitations and assumptions within Salvo.

- Paraxial approximation limits the wave propagation to one-way travel, thus multiples and turning waves are not correctly placed.
- Square-root or Pade approximation limits the propagation angle through the number of terms retained for the square-root operator.
- Operator splitting the x and y directions introduces anisotropic phase error

Table 13: Salvo utility listing with summaries continued.

Utility name	Utility Summary
plane_3d	generates the model 3D plane problem
salvo2sep	converts a SALVO file to an SEP file
scale	scales a binary file
segy2bin	converts a SEG Y file to a binary file
segy2pfs	breaks up a SEG Y file into multiple SEG Y files
segy2salvo	converts a SEG Y file to a SALVO file
segy2sep	converts a SEG Y file to an SEP file
segyChange	change values in SEG Y headers
segyCombine	concatenates two SEG Y files
segyDims	generates ASCII output file with trace coords.
segyExtract	extract SEG Y traces for source/receiver gather
segyHeader	prints out header information of SEG Y traces
sep2bin	converts an SEP file to binary format
sep2loc	writes out trace locations from an SEP trace file
sep2pfs	breaks up an SEP file into multiple SEP files
sep2salvo	converts an SEP file to a SALVO format file
sep2segy	converts an SEP file to a SEG Y format file
sepChange	change values in SEP headers
sepExtract	extracts and separates shot gathers from an SEP file
stack_acc	stacks multiple acc files
tl2tec	converts timeline files into a single Tecplot file
transpose	transposes 2D and 3D binary files
vel_interp	interpolates a binary velocity model

in the azimuthal direction. However this can be improved by the phase correction filters.

- Memory scalability issues exist when reading the velocity model and writing the image plane because the entire image plane is stored on each I/O node. Large poststack runs may reveal this problem with memory; generally this has minimal effect with prestack runs given that they are usually performed with frequency parallelism on smaller domains and the entire image plane and velocity model is stored on each compute node anyway.
- Currently the stacking operation is only allowed with frequency parallelism; user selection of spatial parallelism and stacking will default to simply writing out the image.
- Partitioning accumulation files along y -axis of the data volume implies that with two-dimensional problems Salvo can only use one file and one I/O node for writing and stacking the image (Section C.2). The same restriction applies to reading the velocity model. This decision is based on the fact that two-dimensional problems are fairly small and have limited I/O requirements.
- The stacking operation is designed to favor lower memory usage over higher parallelism on the assumption that the cumulative stack file will become extremely large (and main memory is the limiting factor). This should not interfere with the migration process given that I/O is performed in the background and the migration dominates the total runtime.
- Migration is performed on a rectilinear uniform grid, however, dx does not have to equal dy .
- Two-dimensional runs are performed along the x direction only, *i.e.*, parallel to the x axis.
- Traces are placed at the nearest grid point. If a second trace is placed at the same grid point, then it will replace the first trace, unless phase encoding is invoked in which case the traces are summed together.
- The spatial Gaussian source distribution option currently only works with two-dimensional problems.
- The reduced-source frequency migration is implemented for two-dimensional problems only.
- Currently the reduced-source frequency migration is incompatible with spatial parallelism.
- The imaging advantages of Salvo may not be realized with poststack data, since important features can be stacked out of the trace data.

```

MIGRATE
INITPART
INITPROB
INITVARS
  UPDATE_N
  UPDATE_S
  UPDATE_E
  UPDATE_W
INIT_VEL
IMAGE
WRITE_ACC
STACK_IMG
DO 10 FOR EACH DEPTH STEP
  READ_VEL
  SOLVE
  THIN_LENS
  IF ONLY FREQUENCY PARALLELISM
    PIPE_SINGLE
    IF APPLY FILTER THIS DEPTH STEP
      FILTER_LJ
      PHASE_CORRECTION
      CFFT1DI
      CFFT1D
    ENDIF
  ELSE IF SPATIAL PARALLELISM
    DIFFRACT
    TRIDAG
    PIPE_START
    START_BACK
    PIPE_SINGLE
    PIPE_RETURN
    PIPE_MIDDLE
    MIDDLE_FORE
    MIDDLE_BACK
    UPDATE_N
    UPDATE_S
    UPDATE_E
    UPDATE_W
    IF APPLY FILTER THIS DEPTH STEP
      FILTER_GRAVES
      UPDATE_E
      UPDATE_W
      UPDATE_N
      UPDATE_S
      F_TRIDAG
      F_PIPE_START
      F_START_BACK
      F_PIPE_SINGLE
      F_PIPE_RETURN
      F_PIPE_MIDDLE
      F_MIDDLE_FORE
      F_MIDDLE_BACK
    ENDIF
  ENDIF
  IF APPLY FREQUENCY REDUCTION
    FREQ_REDUCTION_CALC
  IF APPLY NO TEMPORAL WRAP AROUND
    NO_WRAP
10 CONTINUE
STATS
VEL_CLEAN
IO_FINALIZE

```

Figure 42: Pseudo code for migration portion of Salvo.

- Currently the PML is not available with spatial parallelism.

D.3 Pseudo Coding for Migration

In Figure 42, pseudo coding for the migration algorithm is shown. Each indentation implies a routine call.

Index

- 1/6 trick, 33, *see* compact finite difference
- accumulation file, 93, 122, 126, 128
- Advanced Computational Technology Initiative, 10
- alternating direction implicit (ADI) scheme, 15, 113, 114, *see* approximate factorization (AF), method of fractional steps (MOFS)
- approximate factorization (AF), 15, 113, 114, *see* alternating direction implicit (ADI) scheme, method of fractional steps (MOFS)
- boundary conditions, 15, 28, 95, 114
 - absorbing
 - Clayton and Engquist, 16, 115
 - Xu, 16, 34
 - absorbing layer, *see* PML
 - specification, 76
- coefficient separation, 77
- column communicator, 62, 64
- compact finite difference, 33, 78, 112, 119
- complex exponential, 76, 77, *see* thin-lens term
- compute node, *see* I/O node
 - during reading of traces, 52, 57, 59
 - during reading of velocity, 68
 - frequency distribution, 53, 58
 - limitations, 128
 - memory allocation, 52
 - optimal number, 59
 - root, *see* FFT communicator, root node
- compute partition, 51, 53, 56, 58, 59, 67
 - definition, 53
 - during reading of traces, 52
 - during writing of image, 69
 - memory requirements, 109
 - number of nodes, 79
 - performance, 94
 - processor cube, 60, 82
- Crank-Nicholson, 112
- DEBUG, 77, 94
- diffraction terms, 15, 24
- dispersion relation, 14, 16, 29
- environment file, 7, 73, 75
- evanescent energy, 14, 19, 34, 62
- FFT, 52, 75, 77
- FFT communicator, 51, 53, 56–58
 - root node, 56, 57
- frequency
 - decay, 89
 - distribution, 52, 58, 62, 77
 - doubling, 23
 - parallelism, 60, 62, 69
 - and phase correction, 90
 - and stacking the image, 69
 - limitations, 128
 - performance, 80
 - phase correction, 19
 - range, 84
- I/O node, 56, 59, 67, 79, 122, *see* compute node
 - compute partition, 56, 68
 - during reading of velocity model, 67
 - during writing of image, 69
 - I/O partition, 56, 69
 - limitations, 128
 - optimal number, 60
 - root, 69
- I/O partition, 51, 53, 122

- definition, 53
- during migration, 60, 66, 69, 79
- during reading of traces, 52, 56
- during writing of image, 69
- memory requirements, 110
- number of nodes, 79, 82
- omitted, 56, 68–70, 79
- performance, 94
- image
 - aliasing, 96, 121
 - GIF, 72, 79, 125
 - phase encoding, 38
 - stacking, 68, 69, 91, 128
 - writing, 68
- imaging condition, 20, 90, 119
 - deconvolution, 23
 - phase encoding, 37
 - poststack, 21
 - prestack, 22
- impulse response, 95, 120
 - analytic solution, 20, 119
- initial
 - process, 51, 82
- machine dependent, 87
- machine parameters, 66
- Marmousi model, 8, 38, 96
- method of fractional steps (MOFS),
 - 15, 24, 113, 114, *see* alternating direction implicit (ADI) scheme,
 - approximate factorization (AF)
- migration
 - process, 88
 - input parameters, 93
 - memory requirements, 109
- MPI, 7, 72
- numerical dispersion, 28, 95
- numerical dissipation, 28
- Nyquist frequency, 85
- operator splitting, 15, 96, 118, 126
- Ormsby wavelet, 23
- OUTPUT, 78
- perfectly matched layer (PML), 17, 91,
 - 92, 96, 116, 129, *see* boundary condition, absorbing
- phase correction filter, 90
 - Graves and Clayton, 19, 96, 117
 - boundary conditions, 119
 - Li, 19, 34, 35, 62, 63, 78, 95, 96
- phase encoding, 35, 53, 85
- poststack
 - migration, 85
- propagation angle, 77, 126
- reading traces
 - process, 52, 53, 81
 - input parameters, 85
 - memory requirements, 109
- reduced-source migration, 38–50, 63,
 - 87, 92, 128, *see* spatial Gaussian source
- row communicator, 62, 64
- SALVO, 85, 92
- Salvo
 - command line, 8
 - compiling, 8, 73
 - definition, 12
 - description, 10
 - installation, 7, 71
 - man pages, 72
 - processes, 81
 - units of length, 84
 - utilities, 8, 72
- SEG/EAGE Salt model, 11, 100
- SEGY, 85, 92
 - byte ordering, 78
 - utilities, 126
- SEP, 85, 92
- spatial Gaussian source, 40, 44, 57,
 - 78, 87, 128, *see* reduced-source migration
- scaling requirements for reduced source, 41
- weighted, 47, 49, 57

- spatial parallelism, 60, 64
 - and boundary conditions, 76
 - and I/O partition, 69
 - and phase correction, 19, 90
 - and writing the image, 69
 - communicators, 64
 - limitations, 128
 - performance, 80
- square-root approximation, 13, 95, 96, 126
 - expansion coefficients, 14, 16, 20, 90, 111
- temporal wrap around, 24, 78
- thin-lens term, 14, 24, 31
- thins-lens term, *see* complex exponential
- time shift, 85, 86
- tower
 - base node, 59, 67–70
 - communicator, 58, 60, 69
 - processors, 52
- trace file format, 85, 92, *see* SALVO, SEG Y, SEP
- tridiagonal blocks, 66, 90, 108
- validation problems, 95
- velocity model, 10
 - description, 93, 122
 - file, 122
 - generation, 126
 - interpolation, 67, 79, 127
 - limitations, 128
 - memory requirements, 110
 - planes, 67
 - reading, 66
 - used, 79, 93
- wave equation
 - paraxial, 13, 28, 111
 - reduced, 120
 - scalar, 13
 - split, 15, 18, 28
 - finite-differenced, 29
 - unsplit, 29

Distribution

External Distribution

Matthew Brzostowski
GX Technology
5847 San Felipe, Suite 3500
Houston, TX 77057

Conoco Inc.
Attn Charles C. Burch, RW 4488
P.O. Box 1267
Ponca City, OK 74602-1267

Rob Gjertsen
IBM
11400 Burnet Rd.
MS 9571
Bldg 905, 7c16
Austin, TX 78758

Leon Hu
PGS
10550 Richmond Ave. Suite 300
Houston, TX 77042

Jim Manatt
Providence Technologies, Inc.
400 East College Boulevard, Suite A
Roswell, NM 88201

George McMechan
Center for Lithospheric Studies
FA 31
University of Texas at Dallas
2601 N. Floyd Road
P.O. Box 830688
Richardson, TX 75083-0688

Scott Morton
Amerada Hess Corporation
One Allen Center
500 Dallas Street
Houston, TX 77002

Charles Mosher
ARCO Expl. and Prod. Technology
2300 West Plano Parkway
Plano, Texas 75075

Ron A. Oldfield
6211 Sudikoff Laboratory
Dartmouth College
Hanover, NH 03755

Billy R. Robinson
IBM
1507 LBJ Freeway
16-07-7118
Dallas, TX 75234

Marcelo Solano
Kerr-McGee
13155 Noel Road
Dallas, TX 75240-5067

Mike Sullivan
Fairfield Industries
1746 Cole Blvd. Suite #100
Golden, Colorado 80401

Internal Distribution

1	MS 0321	William Camp, 9200
1	MS 0750	Marianne Walck, 6116
1	MS 0750	Henry Westrich, 6118
1	MS 1005	Michael Holzrichter, 2617
1	MS 1110	Susan E. Minkoff, 9222
5	MS 1110	David Womble, 9222
1	MS 1111	Sudip Dosanjh, 9221
5	MS 1111	Curtis Ober, 9221
1	MS 1380	Michael Harper, 4331
1	MS 9018	Central Technical files, 8940-2
2	MS 0899	Technical Library, 4916
1	MS 0612	Review and Approval Desk, 4912 For DOE/OSTI
1	MS 0161	Patent and Licensing Office, 11500