GEOPHYSICS, VOL. 72, NO. 5 (SEPTEMBER-OCTOBER 2007); P. SM213–SM221, 4 FIGS., 2 TABLES.
10.1190/1.2742686

Reverse time migration with optimal checkpointing

William W. Symes[1]

ABSTRACT

Reverse time migration (RTM) requires that fields computed in forward time be accessed in reverse order. Such out-of-order access, to recursively computed fields, requires that some part of the recursion history be stored (checkpointed), with the remainder computed by repeating parts of the forward computation. Optimal checkpointing algorithms choose checkpoints in such a way that the total storage is minimized for a prescribed level of excess computation, or vice versa. Optimal checkpointing dramatically reduces the storage required by RTM, compared to that needed for nonoptimal implementations, at the price of a small increase in computation. This paper describes optimal checkpointing in a form which applies both to RTM and other applications of the adjoint state method, such as construction of velocity updates from prestack wave equation migration.

INTRODUCTION

Reverse time migration (RTM) was introduced in the 1980s (Baysal et al., 1983; Whitmore, 1983), and has recently gained renewed attention from the seismic imaging community (Biondi and Shan, 2002; Yoon et al., 2003; Mulder and Plessix, 2004; Yoon et al., 2004; Bednar et al., 2006). Similar reversed computations have been used to extract velocity updates from wave equation depth migration (Shen et al., 2003; Biondi and Sava, 2004; Albertin et al., 2006; Khoury et al., 2006; Soubaras and Gratacos, 2006), and in many other subjects to link parameter perturbations to simulations (Munk et al., 1995; Wang et al., 1998; Akcelik et al., 2003; Talagrand, 2007).

All of these algorithms are instances of the adjoint state method, a way of organizing the computation of a gradient of a cost or objective function depending on a recursive simulation, with roots in optimal control (Bryson and Ho, 1979; Pontryagin, 1987). Chavent and Lemmonier (1974) introduced this concept into the study of geophysical inverse problems, and later applied it in their study of seismic inversion posed as a nonlinear least-squares problem (Bamberger et al., 1977, 1979, 1982). Very soon after the introduction of RTM, several authors pointed out that it amounted to a single step of a gradient descent algorithm for output least-squares inversion, computed via the adjoint state method (Lailly, 1983, 1984; Tarantola, 1984). Plessix (2006) provides an excellent overview of the method and some of its geophysical applications.

The adjoint state method, in any of its guises, poses an interesting computational complexity problem. The core of the algorithm is the crosscorrelation of two fields at the same (time or depth) level, one computed via forward (in time or depth) recursion, the other via backward recursion. It is natural to carry out the forward recursion first, but then its entire history must be made accessible during the backward recursion. For small problems, one simply stores all states reached during the forward recursion, and reuses them as needed in the rest of the algorithm. For very large applications of the adjoint state method, such as 3D RTM, the required storage is so extensive that implementations must use the slowest levels of memory hierarchy (i.e., disk i/o on modern platforms which amounts to a definition of very large). This need for very large amounts of slow memory complicates the logistics of algorithm implementation and degrades performance.

This paper (re)introduces the seismic imaging community to an algorithm devised by Griewank which largely ameliorates the complexity problem just described (Griewank, 1992; Blanch et al., 1998; Griewank, 2000; Griewank and Walther, 2000; Akcelik et al., 2003). This optimal checkpointing algorithm trades floating point arithmetic for memory. It increases the computational complexity of the adjoint state method by a factor logarithmic in the total number of steps, while reducing the memory complexity to a number of state buffers, also logarithmic in the total number of steps. Griewank's prescription for this tradeoff is provably optimal. For 2D RTM, optimal checkpointing eliminates any need for disk i/o, for a factor of two (or so) increase in floating point operations (flops). For 3D RTM, optimal checkpointing may or may not eliminate the need for disk i/o, but in any case drastically reduces it, with similar floating point penalty. Informal comparisons suggest that with current technology, the reduction in time required for slow memory access more than compensates for the extra flops. If floating point throughput

Manuscript received by the Editor November 18, 2006; revised manuscript received January 20, 2007; published online August 23, 2007.
[1]Rice University, The Rice Inversion Project, Department of Computational and Applied Mathematics, Houston, Texas. E-mail: symes@caam.rice.edu.

advances more quickly than memory bandwidth, as seems likely, optimal checkpointing will become even more attractive.

In the next section, I will detail the adjoint state method in a form which applies to all of the problems mentioned above. The following sections describe the checkpointing concept, optimal checkpoint schedules, a few observations about implementation, and a RTM application. The concluding section assesses the implications of the optimal checkpointing algorithm for 3D RTM in contemporary and near-future computing environments. Appendix A gives a complete derivation of the adjoint state method, in its general form.

ADJOINT STATE METHOD

I will use the notation $\mathbf{u}$ to denote the state vector of a discrete time- (or depth-) dependent system. For leapfrog (three-level) time stepping of a second-order wave equation for the displacement field in elasticity or the pressure in acoustics, $\mathbf{u}$ represents two successive time levels of the discrete displacement or pressure field. For a stress-velocity scheme, $\mathbf{u}$ holds one time level each of pressure and the velocity components. For depth extrapolation, $\mathbf{u}$ is simply a depth slice of the reflected wavefield.

The evolution operator of the system depends on a vector $\mathbf{c}$ of model or control parameters. For most seismic problems, this vector represents the velocity model, and perhaps other parameters that influence the predicted seismic response.

A discrete linear evolution with N time steps takes the form

$$\mathbf{u}^{n+1} = \mathbf{H}^n[\mathbf{c}]\mathbf{u}^n + \mathbf{f}^n, n = 0,1, \ldots, N-1 \qquad (1)$$

in which the evolution operator $\mathbf{H}^n$ may depend on the step n as does the inhomogeneous or source term $\mathbf{f}^n$, and certainly depends on the control $\mathbf{c}$. For example, in a time stepping scheme for seismic modeling, $\mathbf{H}^n$ encodes the finite difference stencil. For depth extrapolation, it is the depth step operator.

The adjoint state computation associated to equation 1 is a backwards (i.e., backwards in step index) evolution for an adjoint state $\mathbf{w}$, of the same type as the system state $\mathbf{u}$. Input to the adjoint state computation is a datum $\mathbf{r}^n$ of the same type (data structure) as the output data of the simulation. The latter is usually related to the state $\mathbf{u}^n$ by a sampling operator $\mathbf{S}^n$ (for example, extracting time samples at receiver points). The adjoint state computation computes two additional dynamical fields: $\mathbf{w}^n, n = N+1, \ldots, 1$, of the same type as the state $\mathbf{u}^n$, and $\mathbf{g}^n, n = N, \ldots, 0$, of the same type as the control (or model) vector $\mathbf{c}$. The algorithm steps $\mathbf{w}^n$ backwards in index n:

$$\mathbf{w}^{N+1} = 0; \quad \mathbf{w}^n = (\mathbf{H}^n[\mathbf{c}])^T\mathbf{w}^{n+1} + (\mathbf{S}^n)^T\mathbf{r}, \quad n = N, \ldots, 1, \qquad (2)$$

and increments $\mathbf{g}^n$ by

$$\mathbf{g}^n = \mathbf{g}^{n+1} + \mathbf{A}^n[\mathbf{c}, \mathbf{u}^n]^T\mathbf{w}^{n+1}, \quad n = N-1, \ldots, 0. \qquad (3)$$

The output of the adjoint state computation is the final value $\mathbf{g}^0$. This field is the gradient of a function of the output data, interpreted as a function of the control $\mathbf{c}$, as is explained in Appendix A. For RTM, the output of the adjoint state computation is also an image of the subsurface, in the sense explained by Lailly (1983, 1984) and Tarantola (1984). For wave equation migration velocity analysis, it is a velocity update vector (Shen et al., 2003).

Appendix A also explains the relation between the imaging operator $\mathbf{A}$ and the dynamical operator $\mathbf{H}$.

Typical implementations use only one time (or depth) level of working storage for $\mathbf{w}$, $\mathbf{g}$ and implement the expressions in equations 2 and 3 as assignments, i.e., the right-hand side is evaluated then overwritten onto the left-hand side. In particular, the last expression repeatedly updates the output vector $\mathbf{g}$. It is not so easy to reduce the storage required for the state $\mathbf{u}$, however. The imaging condition (equation 3) implies that the fields $\mathbf{u}^n$ and $\mathbf{w}^{n+1}$ must be available at the same time in an implementation of this algorithm. This is not a natural result of the pair of evolutions 1 and 2, as they run the step index in opposite directions. To make $\mathbf{u}^n$ available during the nth step of the backward evolution of $\mathbf{w}^n$, several possibilities suggest themselves, differing in balance between memory and computation:

1) compute $\mathbf{u}^n$ from step $n = 0$, for each $n = N-1, \ldots 0$. This involves $O(N^2)$ evolution steps in total, an unacceptable computational burden for problems of any significant size.
2) store the entire time history of the state, $\{\mathbf{u}^n, n = 0, \ldots, N-1\}$, and access as needed. This option needs only the $O(N)$ steps of the basic evolution, but much larger amounts of storage.
3) store only every kth step, $k > 1$ of the time history, and interpolate the rest in some way. This appears to be the approach taken by a number of commercial RTM implementations.This approach has the computation cost advantage of approach 2 with a k-fold reduction in memory use, but produces only an uncalibrated approximation to $\mathbf{g}^0$.

In the discussion to come, I will refer to these three approaches to implementation of the adjoint state system (equations 2 and 3) as strategies 1, 2, and 3, respectively.

CHECKPOINTING

This section outlines an alternative to the three implementation strategies for the adjoint state method (equations 2 and 3), described in the last section. For details, see Griewank and Walther (2000). I shall give only a brief overview.

Select checkpoints $\{n_i: i = 0, \ldots, N_C\}$ between 0 and N, and provide buffers $\{\mathbf{b}_j: j = 1, \ldots, N_B\}$ to which states can be stored and from which states can be initialized. Typically $N_B \ll N_C \ll N$.

During the forward loop (i.e., solution of equation 1), store an initial selection of states $\mathbf{u}^n$ at checkpoint steps $\{n = n_{i_j}: j = 1, \ldots, N_B\}$ in the buffers. An essential constraint is that the last checkpoint selected must be the last checkpoint $n_{i_{N_B}} = \max\{n_i: i = 0, \ldots, N_C\}$.

During the backwards loop (i.e., solution of equation 2), adopt strategy 1 from the last section, but compute $\mathbf{u}^n$ repeatedly, using the state value stored in the corresponding buffer $\mathbf{b}_{i_{N_B}}$ to initialize the evolution in equation 1.

When the backwards step reaches the last checkpoint: $n = n_{i_{N_B}}$, use the buffer $\mathbf{b}_n$ containing the corresponding state value to store a new state value, at one of the previously unrecorded checkpoints. (Remember there are more checkpoints than buffers!) Compute this new value by initializing the evolution in equation 1 at the last previously recorded checkpoint before the chosen one.

The backwards loop proceeds by repeated application of strategy 1, always from the last recorded checkpoint before the current index (thus over-index intervals are possibly much shorter than $[0,N]$). As each checkpoint n_i is reached, the state stored in its corresponding buffer $\mathbf{b}_{n_i}$ is used in the application of the imaging condition (equation 3), and then another previously unrecorded checkpoint is stored in the buffer. At the end of the algorithm, a subset of the buffers store

consecutive state vectors starting at $i = 0$, corresponding to the remaining summands in equation 3. Then an application of strategy 2 from the last section finishes the computation.

Table 1 illustrates a typical use of this algorithm. It shows how one might step backwards through $N = 15$ forward steps, accumulating the gradient along the way using the imaging condition (equation 3). This computation used $N_B = 3$ buffers and $N_C = 6$ checkpoints (steps 0, 1, 3, 6, 8, and 11). During the forward loop, the state vectors for steps 0, 6, and 11 are stored in the three buffers. The first step of the backwards loop requires combining $\mathbf{w}^{15}$ with $\mathbf{u}^{14}$, per the imaging condition (equation 3). The required state for step 14 has however been overwritten in the last step of the forward loop (equation 1). So it is recomputed via application of strategy 1 starting from the last stored checkpoint, namely 11 in this case. The next backwards step (index 13) requires $\mathbf{u}^{13}$, which has also been overwritten so must be recomputed. So it goes, until step 11. The checkpointed state $\mathbf{u}^{11}$, stored in buffer 3, is exactly what is needed to complete the gradient update for this step. After the update, but before the next backwards step, buffer 3 is reused: strategy 1 is used to compute the state for step 8 which is the checkpointed, i.e., stored in buffer 3. The algorithm proceeds in this way until backwards step 8, in which the checkpointed state is used to perform the update, but no new checkpointing is done. The next backwards step recomputes the state for that index, starting with the checkpointed step 6 stored in buffer 2. Step 5 employs strategy 1 starting with checkpoint 0, storing checkpoints 1 and 3 in buffers 2 and 3. At the end of the algorithm, in the last two steps, all required states are stored in buffers, and the algorithm finishes by employment of strategy 2.

Table 1. Tabular representation of checkpointing scheme for $N = 15$, $N_B = 3$ buffers. A total of six checkpoints are stored at various times in the three buffers ($N_C = 6$, steps $n = 0$, 1, 3, 6, 8, and 11). In each row of the table, the step index is followed by the indices of the state vectors stored in the three buffers during that step, and by a list of step indices of previously computed state vectors recomputed during that step. The index of the state vector combined with the adjoint state (to increment the gradient) is italicized, and the index of the state vector used as the initial data in strategy 1 (i.e., for recomputation) is bold-faced. For more explanation, see text.

Step	Buffer 1	Buffer 2	Buffer 3	Recomputed
14	0	6	**11**	12,13,*14*
13	0	6	**11**	12, *13*
12	0	6	**11**	*12*
11	0	**6**	*11*	7, 8
10	0	6	**8**	9, *10*
9	0	6	**8**	*9*
8	0	6	*8*	-
7	0	**6**	8	*7*
6	0	*6*	8	-
5	**0**	1	3	1, 2, 3, 4, *5*
4	0	1	**3**	*4*
3	0	1	*3*	-
2	0	**1**	3	*2* -
1	0	*1*	3	-
0	*0*	1	3	-

Note that several steps in the forward loop (equation 1) are repeated in the course of this algorithm. A convenient way to measure the resulting increase in computational cost (over the cost of the simulation, i.e., the forward loop in equation 1) is the recomputation ratio, i.e., the ratio of the total number of steps of the forward loop taken in the algorithm divided by N. For the instance of the checkpointing algorithm described in Table 1, 19 additional forward steps were required, so the recomputation ratio is $34/15 \simeq 2.3$.

The question remains: how should the checkpoints be chosen and in what order should they be used in the backward loop? The strategies 1 and 2 introduced above are obviously end members: strategy 2 uses $N_C = N$ checkpoints, strategy 1 uses none ($N_C = 0$). The reader will easily see that $\sqrt{N}$ evenly spaced checkpoints and $\sqrt{N}$ buffers can be used to produce $\mathbf{g}^0$ in $O(N^{3/2})$ evolution steps, for a recomputation ratio of $\sqrt{N}$, a big improvement over the $N/2$ of strategy 1. As it turns out, it is possible to do much better than this.

OPTIMAL CHOICE OF CHECKPOINTS

In a beautiful piece of combinatorial mathematics, Griewank (1992) completely determined the optimal choice of checkpoints. Griewank's prescription is optimal in the sense that:

- For given numbers of time steps and buffers, the recomputation ratio is minimum amongst all possible checkpointing schedules
- For a given number of time steps and a prescribed maximum recomputation ratio, the number of buffers required is minimum amongst all possible checkpointing schedules.

For a forward loop of length N, the number of buffers required is $O(\log N)$, and the recomputation ratio is also $O(\log N)$. If either of these two numbers is given, the other is determined. In other words, for given recomputation ratio, the maximum length N of the forward loop grows exponentially with the number of buffers used. Contrast this behavior with strategy 3 described above, in which N is linear in the number of buffers, the ratio being determined by the quality of interpolation (hence by bandwidth or some similar signal attribute).

Other accounts of this algorithm, with some improvements, appear in Griewank (2000) and Griewank and Walther (2000). The author and his collaborators used optimal checkpointing in a viscoacoustic least-squares inversion scheme (Blanch et al., 1998). Akcelik et al. (2003) have also used the algorithm in the context of least-squares inversion of basin structure from earthquake data.

Table 2 illustrates the trade-off between computation and memory achieved by optimal checkpointing, for a forward loop with $N = 10{,}000$ steps. This number of steps is more or less a median value for finite difference time-domain simulation of a typical reflection seismic survey. For velocity analysis via wave equation depth migration (i.e., Shen et al., 2003), $N = 1000$ would be more appropriate.

Even for only three buffers, the recomputation ratio is under 28. This seems large, but should be compared to the ratio of 5000 to be expected for a straightforward implementation of strategy 1. The ra-

Table 2. Number of buffers and corresponding recomputation ratio for Griewank's optimal checkpointing scheme. All figures for N = 10,000 steps.

Buffers	3	5	10	15	20	25	30	35	40	60
Ratio	27.9	11.3	5.8	4.5	3.8	3.6	3.4	3.1	2.9	2.8

tio drops rapidly as buffers are added, up to around 15 buffers, after which the incremental gain by adding each buffer becomes small. At about 36 buffers, the recomputation ratio is roughly 3. Because the backwards loop involves the same steps as the forwards loop, plus application of the imaging operator, the cost of the adjoint state computation using optimal checkpointing and 36 buffers is somewhat above four times the cost of a simulation.

A straightforward implementation of strategy 2, i.e., maximal memory and minimal computation, would by the same reasoning cost somewhat more than twice the cost of a simulation. Thus the optimal checkpointing approach with 36 buffers completes the $N = 10{,}000$ adjoint state loop in less than twice the computational cost of strategy 2. This comparison seriously understates the total cost of strategy 2. However, for modern NUMA architectures, it typically requires large amounts of slow (disk) memory access for problems of even modest size. The time devoted to slow memory access can easily shift the comparison in favor of optimal checkpointing.

For a small-scale example of Griewank's optimal schedule of checkpoints, the reader need only return to Table 1, which was constructed using Griewank's algorithm.

IMPLEMENTATION

The algorithm determining optimal checkpoint schedules described by Griewank (1992) is fairly complex. Fortunately, the author has provided full public domain implementations in both C and Fortran 77 (Griewank and Walther, 2000), along with an excellent article covering the topic. The algorithm presented in Griewank and Walther (2000) offers several improvements over that presented in the earlier reference (Griewank, 1992). The code package contains several useful utilities, one of which computes the recomputation ratio from the numbers of steps and buffers. I used this utility to create Table 2.

The attentive reader will note that, up to this point, my discussion of simulation and the adjoint state method has not specified any particular physical system or numerical method. In fact, the framework described up to this point is entirely abstract, posed only in terms of vector calculus concepts. I have implemented the adjoint state method with optimal checkpointing within an object-oriented framework for simulation-driven optimization (Symes et al., 2005; Symes, 2007), which permits the expression of abstract algorithms such as these. Creation of a full-blown simulation package for a specific system, including adjoint state, requires only the implementation of concrete classes defining the specific choices of the step operator **H** and its partial derivatives and their adjoints (one of which is the imaging operator **A**), and the sampling operator **S**.

The object-oriented framework described in Symes (2007) uses the code provided by Griewank and Walther (2000) to compute the checkpointing schedule. I believe that the object-oriented approach to expression of high-level algorithms, such as those described here, has much to recommend it. However, it is relatively straightforward to use the codes from Griewank and Walther (2000) to add optimal checkpointing to a procedural implementation of the adjoint state method for a particular system, such as acoustic RTM, see for example Blanch et al. (1998).

EXAMPLE: RTM

This section describes the realization of a particular approach to RTM, namely centered difference time marching for a scalar field (pressure), which will serve as an existence proof and an illustration of the computational efficiency achievable by optimal checkpointing. Both second- and higher-order (in time) schemes can be represented this way. RTM based on the pressure-velocity formulation of acoustics (for example so-called staggered grid schemes) also fit into the framework discussed in this paper, as do elastic RTM schemes.

Centered difference time marching for the constant-density acoustic wave equation can be written as

$$p^{n+1} = 2p^n - p^{n-1} + \Delta t^2 v^2 L p^n + \Delta t^2 f^n. \tag{4}$$

Here p^n approximates the pressure at time n, and L is a spatial operator approximating the Laplacian. For regular grid finite differences, p^n_{ijk}, $i = 0,\ldots,n_x$, $j = 0,\ldots,n_y$, $k = 0,\ldots,n_z$ approximates the pressure at $t = n\Delta t$, $x = i\Delta x$, $y = j\Delta y$, $z = k\Delta z$, and L is a finite difference Laplacian approximation. The inhomogeneous term f^n represents the system excitation via a body force density. The control parameters in this model are the velocity grid values, represented by v.

The scheme described in equation 4 fits into the form given by equation 1 if we define

$$\mathbf{u}^n = \begin{pmatrix} p^n \\ p^{n-1} \end{pmatrix},\ \mathbf{c} = (v), \tag{5}$$

$$\mathbf{H}^n[\mathbf{c}]\mathbf{u}^n = \begin{pmatrix} 2p^n - p^{n-1} + \Delta t^2 v^2 L p^n + \Delta t^2 f^n \\ p^n \end{pmatrix}.$$

In the example presented below, we have used the fourth-order Laplace approximation by centered differences, which results in the so-called (2, 4) displacement scheme, even though it computes pressure (Levander, 1989). The sampling operator S extracts estimates of hydrophone output, typically point samples, at the time-sample rate of the output data traces. Many codes apparently accomplish this with nearest-neighbor interpolation, or assume that the receiver (and source) locations lie on gridpoints. The implementation reported here does not assume any relation between the computation grid and source/receiver locations, or between the simulation time step and the sample rate of the output traces. **S** is implemented using bilinear (space) and cubic spline (time) interpolation to allow for arbitrary source and receiver locations and sample rates. Sources **f** are represented as linear combinations of points sources with (possibly) different wavelets. Source fields are adjoint-interpolated onto the computational grid.

The adjoint state field $\mathbf{w}^n = (q^n, q^{n+1})^T$ has the same structure as does the acoustic field **u**. The imaging operator $(\mathbf{A}^n)^T$ in equation 3 is the partial derivative in **c** of the RHS of the evolution in equation 1, according to the derivation given in Appendix A. For the choices given in equations 4 and 5, imaging operator acts as

$$\mathbf{A}^n[\mathbf{c},\mathbf{u}^n]^T\mathbf{w}^{n+1} = 2\Delta t^2 v L p^n q^{n+1}. \tag{6}$$

That is, equation 3 represents, in this case, accumulation of the expected crosscorrelation between p (source field) and q (receiver field). The various other factors,such as the Laplacian, are necessary to ensure that this crosscorrelation actually computes the output of the adjoint operator to the Born seismic modeling operator. Simpler ad-hoc crosscorrelations may be adequate for imaging purposes, but do not (necessarily) accurately compute the adjoint action.

The code uses perfectly matched layer absorbing boundary conditions to simulate unbounded domain wave propagation (Cohen, 2001), which actually augments the systems of equations 4 and 5 with additional fields and evolution laws which absorb energy near

the boundary. On those parts of the boundary not subject to absorbing boundary conditions, the code employs a method-of-images implementation of the Dirichlet (pressure-release) condition.

I have implemented a 2D version of this algorithm in the object-oriented framework described in the preceding section. All of the finite difference code is written in Fortran 77, partly to use the automatic differentiation package TAMC (Giering and Kaminski, 1998) to generate code for the partial derivatives of **H** and their adjoints. With complicated absorbing boundary condition and numerous subloops, these computations are difficult (though always in principle possible) to carry out by hand. The code has elementary tuning applied (for example, none of the absorbing boundary computations are done away from the boundary), but no explicit source-level loop unrolling, blocking, or tiling. Tests show that at least 99% of the cycles in a simulation of even modest size occur in the Fortran portion of the code, even though in terms of line count Fortran occupies a very small part.

For 2D applications of any reasonable size, each simulation handily fits (with plenty of room to spare) on an individual workstation or cluster node. I built parallelization over subsimulations (shots, in this case) into the structure of the object-oriented framework mentioned earlier (Symes, 2007). A 3D simulation and associated adjoint computations are likely to require loop-level parallelism (domain decomposition) at the level of the finite-difference code, for use with typical contemporary cluster platforms.

The Marmousi 2D synthetic model (Versteeg and Grau, 1991) provides a reasonable test case. Marmousi involves a complex velocity model, lots of reflectivity, geologic plausibility, 240 shots, and a 2.5 km cable. I simulated true Born data by smoothing the original Marmousi velocity model, subtracting a somewhat less stringent smoothing from the original to create a short wavelength perturbation (reflectivity), and computing a synthetic data set with the linearized (Born) simulation code built according to the principles described here. I used the Rice University Cray XD1 cluster, consisting of 336 AMD Opteron 275 dual-core nodes, and the gcc4 compiler suite. On 120 CPUs (cores), the linearized simulation required approximately 39 minutes (wall clock–total CPU time was 78 hours). This is slightly more than twice the cost of a nonlinear simulation which ran in a bit over 18 minutes on the same platform. For completeness, Figures 1 and 2 show the velocity and reflectivity models, and Figure 3 shows a typical shot gather probing the middle (complex) part of the model.

Figure 4 displays the result of RTM applied to the output of this simulation, using the same (reference) velocity. I used 32 buffers in this computation, which for the approximately 8000 time steps in this simulation gave a recomputation ratio of approximately 3. This result required 90 minutes wall clock, also on 120 CPUs, consistent with the analysis given above, which suggests a cost in this case of four to five times the cost of a simulation. No disk i/o was performed during the migration, except for reading in data traces and writing out the final result.

DISCUSSION

Direct comparison of optimal checkpointing performance with alternatives is difficult, because of the wide variety of techniques employed to optimize these other strategies. Notably, commercial implementations of strategy 3 described earlier in this paper appear to employ aggressive decimation of the state history, various types of interpolation, reduced word length, and other devices (citable descriptions seem difficult to come by). Of course the seek speed and bandwidth of the disk subsystem also has a dominating influence on the performance of such algorithms. As control of all such factors is impossible, I have restricted myself to pointing out the performance

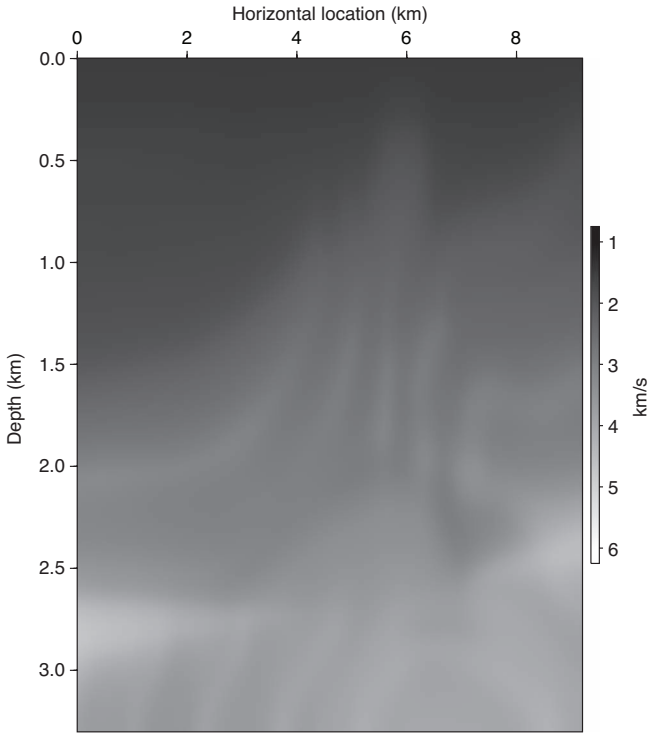

Figure 1. Marmousi velocity model, smoothed by a 160 m tapered moving average.

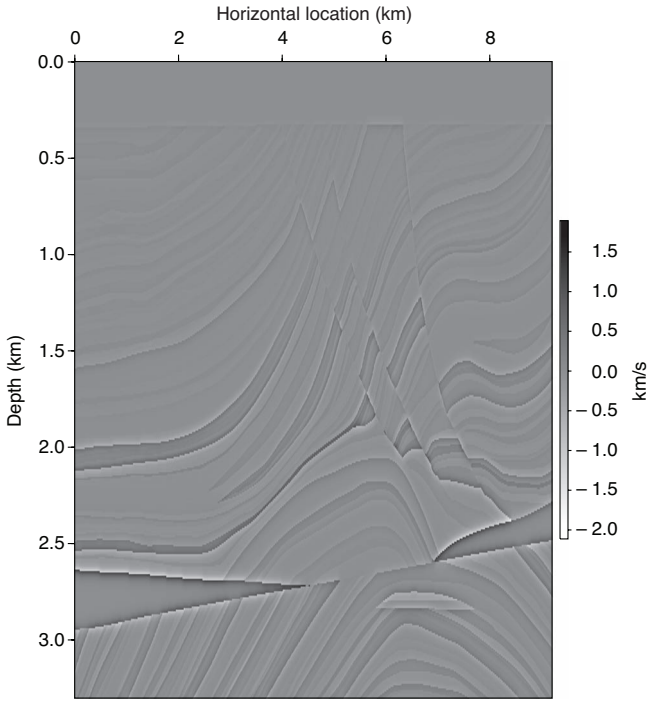

Figure 2. Velocity perturbation model derived from Marmousi velocity model, by subtracting a 40 m tapered moving average.

of checkpointing in units of simulations, with examples, rather than attempt a comparison with a straw man implementation of an alternative strategy. It does seem reasonable to suppose, however, that avoidance of the slowest level of the memory hierarchy might more than compensate for the increase in floating point complexity implicit in checkpointing.

For problems of modest size, the checkpointing approach clearly eliminates altogether any need for disk i/o, assuming typical modern workstation or cluster node resources. This is certainly the case for 2D RTM, and for the single-frequency unit of migration velocity analysis by 3D depth extrapolation (which is effectively a 2D adjoint state computation) (Shen et al., 2003). Even when disk i/o is necessary (as may be the case for 3D RTM, for example), optimal checkpointing requires much less than any alternative approach. Consider a typical 3D RTM involving $n_x = n_y = n_z \simeq 1000$ gridpoints in each direction, so that a state (two time levels) requires storage of 2×10^9 words. For a straightforward implementation of strategy 2 (taking into account that only half of each state need be stored) with a typical 3–4 s simulation time and typical bandwidth and sampling requirements leading to $N \simeq 10^4$, one estimates 10^{13} words of storage, or 20 TB assuming that all floating point numbers are stored as two-byte integers. This is a large, but not unreasonable, amount of fast or near disk for a large contemporary facility. Strategy 3 with a decimation factor of 10 reduces the storage requirement to 2 TB, an amount of disk easily available on a single chassis, and within reach as semiconductor memory on reasonably sized clusters. The effect on performance of reading $O(10^9)$ words from disk at every time step, or even every tenth time step, is likely to be considerable. The optimal checkpointing strategy with $N_B = 36$ buffers, on the other hand, requires that only 72 GW be stored, or 144 GB with compression to two bytes per word. This is a reduction of over an order of magnitude in both storage and reads/writes, and brings the memory required for 3D RTM within the range of core memory available in large shared memory machines or subclusters of larger clusters.

The main point, however, is not so much whether the required states can be stored in core or must be farmed out to disk for one strategy or another. These choices depend on rapidly changing technology. The statements made in the preceding paragraph would have been wildly unrealistic 10 years ago, and will appear quaint 10 years from now. The point is that optimal checkpointing significantly reduces the memory intensity of RTM, and this reduction is likely to become more, rather than less, important in view of foreseeable technology trends. For example, if disk i/o is to be avoided altogether, then any reduction in memory complexity increases the portion of the computation that can be carried out on a single CPU or multicore processor, and therefore reduces both the number of subdomains required in a domain decomposition strategy (a practical necessity for 3D for some time to come) and the attendant number of messages that must be passed. That is, optimal checkpointing permits 3D RTM to be carried out in core in a distributed computing environment with less communication complexity than any other approach. Moreover, various technologies appear poised to dramatically increase floating point throughput available to applications like those discussed here, in the next few years. For example, the IBM Cell Broadband Engine (IBM, 2004) and field programmable gate arrays appear to promise significant speed over contemporary CPUs. If these gains are indeed realized, they will very likely far outstrip the improvements in non-uniform memory access bandwidth, especially at the slow end, i.e., disk i/o.

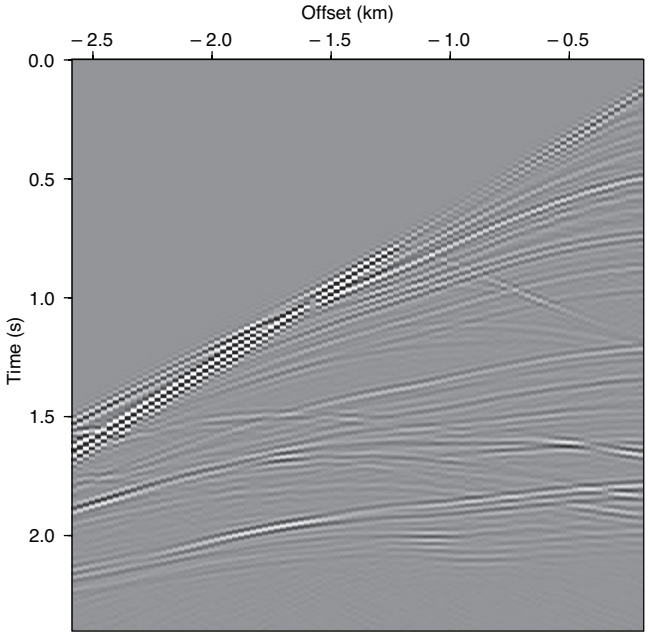

Figure 3. Linearized (Born) simulation of shot gather at 7.5 km from left edge of model. Source depth is 8 m, receiver depth 12 m, group interval 25 m, near offset −200 m, 96 traces per shot, 240 shots were simulated at 25-m intervals, the left-most shot sited 3 km from the left edge of the model. Source is point radiator, zero phase band-pass filter 5-13-40-55 Hz.

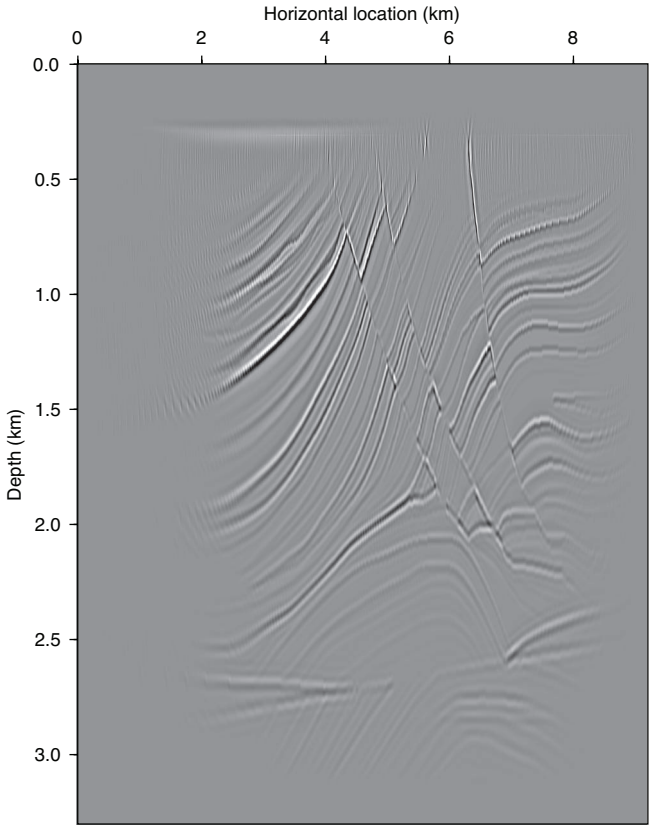

Figure 4. RTM of data described in Figure 3.

CONCLUSION

I have described a checkpointing algorithm due to Griewank for practical implementation of the adjoint state method, and its application to RTM. A straightforward implementation of the adjoint state method stores the entire history of the reference state (source wavefield for RTM). The checkpointing method trades floating point operations for some, or even most, of this storage. With optimal choice of checkpoints, the increase in computation time is logarithmic in the number of steps, and the memory complexity is also logarithmic in the number of steps.

Optimal checkpointing's judicious trade-off of modest increase in floating point complexity for dramatically reduced memory complexity would therefore appear to offer an attractive alternative to contemporary practice for RTM and similar adjoint computations in applied seismology.

ACKNOWLEDGMENT

This work was supported in part by a Major Research Infrastructure grant from the National Science Foundation (CNS-0421109) and partnerships with AMD and Cray, and by the sponsors of The Rice Inversion Project. I am grateful to associate editor J. B. Bednar and to several anonymous reviewers for helpful remarks and suggestions which enabled me to materially improve the exposition. My understanding of optimal checkpointing owes a great deal to discussions with Andreas Griewank and Joakim Blanch.

APPENDIX A

DERIVATION OF GENERAL ADJOINT STATE METHOD

In this appendix, I derive the discrete-in-time form of the adjoint state method. For an analogous account of the method for continuous time, see Plessix (2006).

In addition to the evolution in equation 1, modeling involves a sampling step, which extracts a data prediction from the system state. I assume that the sampling operator $\mathbf{S}^n$ is linear and independent of control parameters, while possibly depending on the step index. For example, seismic time traces are recorded at specific physical locations, depth imaging extracts the zero-offset section at each depth, and so on. If you write the entire time history of the state as a column vector of length $N + 1 =$ number of time samples

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}^0 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{u}^N \end{pmatrix}$$

and define $\mathbf{S}$ to be the row vector of sampling operators

$$\mathbf{S} = (\mathbf{S}^0 \mathbf{S}^1 \ \dots \ \mathbf{S}^N),$$

then the predicted data for control $\mathbf{c}$, denoted $\mathbf{F}[\mathbf{c}]$, is give by $\mathbf{F}[\mathbf{c}] = \mathbf{S}\mathbf{u}$. $\mathbf{F}$ is the prediction operator and is generally nonlinear in $\mathbf{c}$, even for linear evolutions.

The adjoint state method, described in equations 2 and 3, computes the adjoint action of the linearized prediction operator, via a recursion similar to the evolution defining the state. The need for this computation arises in computing the gradient of an objective function of the control $J[\mathbf{c}]$, which is actually a function E of the predicted data

$$J[\mathbf{c}] = E[\mathbf{F}[\mathbf{c}]].$$

For output least-squares (waveform) inversion, E computes the mean square difference between predicted and observed data. For migration velocity analysis, E computes the semblance or differential semblance or some other function of the depth extrapolated reflected field.

The chain rule and the definition of gradient give

$$\begin{aligned} \nabla J[\mathbf{c}]^T \delta\mathbf{c} = DJ[\mathbf{c}]\delta\mathbf{c} &= (\nabla E[\mathbf{F}[\mathbf{c}]])^T D\mathbf{F}[\mathbf{c}]\delta\mathbf{c} \\ &= (D\mathbf{F}[\mathbf{c}]^T \nabla E[F[\mathbf{c}]])^T \delta\mathbf{c}, \end{aligned}$$

in which D denotes the derivative, i.e. $DF[\mathbf{c}]$ is the Jacobian matrix of F at $\mathbf{c}$. Because this relation must hold for every perturbation $\delta\mathbf{c}$, it follows that

$$\nabla J[\mathbf{c}] = D\mathbf{F}[\mathbf{c}]^T \nabla E[\mathbf{F}[\mathbf{c}]].$$

The gradient of the error function E tends to be very easy to compute, so the issue posed by the preceding formula is the application of the transpose Jacobian $D\mathbf{F}[\mathbf{c}]^T$.

From its definition,

$$D\mathbf{F}[\mathbf{c}]\delta\mathbf{c} = \mathbf{S}\delta\mathbf{u} \tag{A-1}$$

where the perturbation (Born) field $\delta\mathbf{u}$ has its own evolution law, derived from that for $\mathbf{u}$:

$$\begin{pmatrix} \delta\mathbf{u}^0 \\ \delta\mathbf{u}^1 \\ \vdots \\ \delta\mathbf{u}^N \end{pmatrix} = \begin{pmatrix} 0 & 0 & & 0 \\ \mathbf{H}^0[\mathbf{c}] & 0 & & 0 \\ & \ddots & & \\ 0 & & \mathbf{H}^{N-1}[\mathbf{c}] & 0 \end{pmatrix} \begin{pmatrix} \delta\mathbf{u}^0 \\ \delta\mathbf{u}^1 \\ \vdots \\ \delta\mathbf{u}^N \end{pmatrix} + \begin{pmatrix} 0 \\ D\mathbf{H}[\mathbf{c}]\mathbf{u}^0 \\ \ddots \\ D\mathbf{H}[\mathbf{c}]\mathbf{u}^{N-1} \end{pmatrix} \delta\mathbf{c}. \tag{A-2}$$

Denote by $\mathcal{H}$ the first $(N \times N)$ matrix of operators on the right-hand side of the preceding equation, and by $\mathcal{K}$ the second $(N \times 1)$. Note that $\mathcal{H}$ is lower triangular, and so defines a forward evolution in step index (representing time or depth or…).

The preceding equation may be abbreviated as

$$\delta\mathbf{u} = \mathcal{H}\delta\mathbf{u} + \mathcal{K}\delta\mathbf{c}.$$

Solving this equation for $\delta\mathbf{u}$ and inserting in equation A-1 yields an expression for the Jacobian operator $D\mathbf{F}[\mathbf{c}]$:

$$D\mathbf{F}[\mathbf{c}] = \mathbf{S}(I - \mathcal{H})^{-1}\mathcal{K},$$

when a similar expression follows for its transpose:

$$D\mathbf{F}[\mathbf{c}]^T = \mathcal{K}^T(I - \mathcal{H}^T)^{-1}\mathbf{S}^T. \tag{A-3}$$

The adjoint state method consists in unwinding the equation A-3 to reveal a recursion for the action of the transposed Jacobian on a (data-like) vector $\mathbf{r}$ (which, for the gradient calculation, will be $\nabla E[\mathbf{F}[\mathbf{c}]]$). The algorithm proceeds as follows:

Step 1: Apply the adjoint sampling operator:

$$\mathbf{r} \mapsto \mathbf{S}^T\mathbf{r} = \begin{pmatrix} (\mathbf{S}^0)^T \\ (\mathbf{S}^1)^T \\ \vdots \\ (\mathbf{S}^N)^T \end{pmatrix} \mathbf{r}.$$

Because $\mathbf{S}^n$ extracts the data from a state at time step n, its adjoint inserts the data into the state at time step n.

Step 2 (backpropagation): Solve the adjoint state system

$$(I - \mathcal{H}^T)\mathbf{w} = \mathbf{S}^T\mathbf{r}$$

for the adjoint state vector $\mathbf{w} = (\mathbf{w}^0, \ldots, \mathbf{w}^N)^T$. Because $\mathcal{H}^T$ is upper triangular, solution of this system unfolds into a recursion backwards in the step index. A simple way to take into account the beginning of this backsubstitution process is to extend the adjoint state by one step, and initialize the additional adjoint state step to zero:

$$\mathbf{w}^{N+1} = 0; \quad \mathbf{w}^n = (\mathbf{H}^n[\mathbf{c}])^T\mathbf{w}^{n+1} + (\mathbf{S}^n)^T\mathbf{r}, \quad n = N, \ldots, 1 \tag{A-4}$$

Step 3 (imaging): Apply the operator $\mathcal{K}^T$ to $\mathbf{w}$, which amounts to computing

$$\sum_{n=0}^{N-1} (D\mathbf{H}^n[\mathbf{c}]\mathbf{u}^n)^T\mathbf{w}^{n+1}. \tag{A-5}$$

It is natural to accumulate the sum in equation A-5 term-by-term as the factors $\mathbf{w}^n$ are produced in the backpropagation loop (equation A-4). With the notation

$$\mathbf{A}^n[\mathbf{c}, \mathbf{u}^n] \equiv D\mathbf{H}^n[\mathbf{c}]\mathbf{u}^n$$

defining the imaging operator $\mathbf{A}^n$, the representation given in equations 2 and 3 of the adjoint computation is established.

The definition of an operator adjoint depends on the inner products used in domain and range spaces. The computation outlined here assumes the unscaled Euclidean norm in both domain and range. If scaled norms are used in domain or range of the modeling operator, these scale factors must also be taken into account in the definition of the adjoint. For example, the Euclidean inner product might be scaled to create a quadrature rule for the integral or L^2 inner product of functions. This sort of scaling makes the inner products of sampled grid functions essentially independent of grid size. However, the cell volumes used to scale the Euclidean inner product must then be included in the definition of the adjoint.

To verify the proper inclusion of scale factors and realization of other facets of the calculation, the author strongly recommends that any adjoint state implementation be subjected to the obvious null test: for random choices of control and data perturbations $\delta\mathbf{c}$ and $\delta\mathbf{d}$, ensure that

$$\langle D\mathbf{F}[\mathbf{c}]\delta\mathbf{c}, \delta\mathbf{d}\rangle_R \simeq \langle \delta\mathbf{c}, D\mathbf{F}[\mathbf{c}]^T\delta\mathbf{d}\rangle_D$$

in which $\langle\cdot,\cdot\rangle_D$ and $\langle\cdot,\cdot\rangle_R$ are domain and range (of $\mathbf{F}$) inner products respectively. The difference between the two sides should be a modest multiple of machine precision.

REFERENCES

Akcelik, V., J. Bielak, G. Biros, I. Epanomeritakis, A. Fernandez, O. Ghattas, E. Kim, J. Lopez, D. O'Hallaron, T. Tu, and J. Urbanic, 2003, High resolution forward and inverse earthquake modeling on terascale computers: Conference on Supercomputing, Association for Computing Machinery/ Institute of Electrical and Electronics Engineers, Inc., 52.

Albertin, U., P. Sava, J. Etgen, and M. Maharramov, 2006, Adjoint wave equation velocity analysis: 76th Annual International Meeting, SEG, Expanded Abstracts, 3345–3349.

Bamberger, A., G. Chavent, C. Hemon, and P. Lailly, 1982, Inversion of normal incidence seismograms: Geophysics, **47**, 757–770.

Bamberger, A., G. Chavent, and P. Lailly, 1977, Etude mathématique et numérique d'un problem inverse pour l'Équation des ondes à une dimension: Rapport Interne 14, Centre de Mathématiques Appliqués, École Polytechnique, Paris.

———, 1979, About the stability of the inverse problem in 1-D wave equation — Application to the interpretation of seismic profiles: Applied Mathematics and Optimization, **5**, 1–47.

Baysal, E., D. D. Kosloff, and J. W. C. Sherwood, 1983, Reverse time migration: Geophysics, **48**, 1514–1524.

Bednar, J. B., C. J. Bednar, and C. S. Shin, 2006, Two-way vs. one-way: A case study style comparision: 76th Annual International Meeting, SEG Expanded Abstracts, 2343–2347.

Biondi, B., and P. Sava, 2004, Wave-equation migration velocity analysis - I: Theory, and II: Subsalt imaging examples: Geophysics, **52**, 593–623.

Biondi, B., and G. Shan, 2002, Prestack imaging of overturned reflections by reverse time migration: 72nd Annual International Meeting, SEG, Expanded Abstracts, 1284–1287.

Blanch, J., W. Symes, and R. Versteeg, 1998, A numerical study of linear inversion in layered viscoacoustic media, in R. Keys and D. Foster, eds., Comparision of seismic inversion methods on a single real dataset: SEG, 13–44.

Bryson, A., and Y.-C. Ho, 1979, Applied optimal control: John Wiley & Sons, Inc.

Chavent, G., and P. Lemmonier, 1974, Identification de la non-linéarité d'une équation parabolique quasilinéaire: Applied Mathematics and Optimization, **1**, 121–162.

Cohen, G. C., 2001, Higher order numerical methods for transient wave equations: Springer.

Giering, R., and T. Kaminski, 1998, Recipes for adjoint code construction: ACM Transactions on Mathematical Software, **24**, 437–474.

Griewank, A., 1992, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation: Optimization Methods and Software, **1**, 35–54.

———, 2000, Evaluating derivatives: Principles and techniques of algorithmic differentiation: SIAM.

Griewank, A., and A. Walther, 2000, Algorithm 799: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation: ACM Transactions on Mathematical Software, **26**, 19–45.

IBM, 2004, The Cell project at IBM Research: http://www.research.ibm-.com/cell/, accessed October 10, 2006.

Khoury, A., W. W. Symes, P. Williamson, and P. Shen, 2006, DSR migration velocity analysis by differential semblance optimization: 76th Annual International Meeting, SEG, Expanded Abstracts, 2450–2454.

Lailly, P., 1983, The seismic inverse problem as a sequence of before-stack migrations, *in* J. Bednar, ed., Conference on inverse scattering: Theory and Applications: SIAM, 206–220.

———, 1984, Migration methods: partial but efficient solutions to the seismic inverse problem, *in* F. Santosa, ed., Inverse problems of acoustic and elastic waves: SIAM.

Levander, A., 1989, Finite difference forward modeling in seismology, *in* D. E. James, ed., The encyclopedia of solid Earth geophysics: Springer, 410–431.

Mulder, W., and R.-E. Plessix, 2004, A comparison between one-way and two-way wave equation migration: Geophysics, **69**, 1491–1504.

Munk, W., P. Worcester, and C. Wunsch, 1995, Ocean acoustic tomography: Cambridge University Press.

Plessix, R.-E., 2006, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications: Geophysical Journal International, **167**, 495–503.

Pontryagin, L., 1987, Mathematical theory of optimal process: CRC Press.

Shen, P., W. Symes, and C. Stolk, 2003, Differential semblance velocity analysis by wave-equation migration: 73rd Annual International Meeting, SEG, Expanded Abstracts, 2135–2139.

Soubaras, R., and B. Gratacos, 2006, Velocity model building by semblance maximization of modulated-shot gathers: 76th Annual International Meeting, SEG, Expanded Abstracts, 3046–3050.

Symes, W. W., 2007, A time-stepping library for simulation-driven optimization: Technical Report: 07-04: Department of Computational and Applied Mathematics, Rice University.

Symes, W. W., A. D. Padula, and S. D. Scott, 2005, A software framework for the abstract expression of coordinate-free linear algebra and optimization algorithms: Technical Report 05-12: Department of Computational and Applied Mathematics, Rice University.
Talagrand, O., 2007, Data assimilation in meterology and oceanography: Academic Press Inc.
Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, **49**, 1259–1266.
Versteeg, R., and G. Grau, eds. 1991, The Marmousi experience: Proceedings of the European Association Exploration Geophysics workshop on practical aspects of seismic data inversion.
Wang, D. Z. K. K. Droegemeier, and L. White, 1998, The adjoint Newton algorithm for large-scale unconstrained optimization in meterology applications: Computational Optimization and Applications, **10**, 281–318.
Whitmore, N. D., 1983, Iterative depth migration by backwards time propagation: 53rd Annual International Meeting, SEG, Expanded Abstracts, 382–385.
Yoon, K., K. Marfurt, and E. W. Starr, 2004, Challenges in reverse time migration: 74th Annual International Meeting, SEG, Expanded Abstracts, 1057–1060.
Yoon, K., C. Shin, S. Suh, L. Lines, and S. Hong, 2003, 3D reverse-time migration using the acoustic wave equation: An experience with the SEG/EAGE data set: The Leading Edge, **22**, 38–41.