T-2921

THE FOURIER THEORETICAL TECHNIQUE AS APPLIED TO FORWARD AND INVERSE MODELING

by

Anthony A. Sirtautas

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science of Geophysics.

Golden, Colorado

Date March 22, 1985

Signed: Anthony A. Sirtautas

Anthony A. Sirtautas

Approved: Will A Schneider

William A. Schneider
Thesis Advisor

Golden, Colorado

Date 22 March 85

P R Romig

Phillip R. Romig
Head
Department of Geophysics

ABSTRACT

The algorithms for the Fourier theoretical solutions of the One-way and Two-way non-reflecting wave equations are presented. The Fourier theoretical method calculates exact spatial derivatives in the spatial frequency (kx,kz) domain, and the time derivatives are calculated by using conventional one-dimensional finite difference schemes. The Fourier theoretical approach requires fewer spatial grid points than full three-dimensional finite difference methods (X,Z,t). Therefore, it is believed that the Fourier theoretical method will be more efficient for both forward (exploding reflector) and inverse (Reverse time migration) modeling.

The Fourier theoretical approach for the One-way wave equation is tested against five earth models: a point diffractor in a homogeneous medium, a point diffractor in a vertically layered medium, a fault block model, a syncline model, and an anticline model. The models are used in both the forward and inverse modeling examples. The results are accurate for all models when compared to the known solutions.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

ACKNOWLEDGMENTS

I wish to thank my advisor, William A. Schneider, for his patience and invaluable help during my three years of graduate studies at the Colorado School of Mines.

This work was developed initially on a Vax - 780 computer, with a Star - 100 array processor, at Golden Geophysical Corporation. While I was there, Robert Shurtleff guided me as my direct supervisor on this project.

During the last year, I made use of a CRAY-XMP 24 at Sohio Petroleum in Dallas. A special thanks goes to Bruce Secrest of Sohio, and Mrinal Sen of the University of Hawaii who helped me understand the theory of finite difference techniques and the Fourier theoretical technique.

Finally, I wish to thank Frank Hadsell and Ron Knoshaug who both served as members of my thesis committee.

And last but not least, I thank Sara Wilson who took the time to type the final copy of my thesis.

INTRODUCTION

Since exact analytical solutions to the elastic/ acoustic wave equation do not exist for most subsurface models of geologic interest, the need for numerical approximations to the wave equation has been of concern to the geophysical community for some time. The numerical approximations are used in applications of forward and inverse modeling. A great deal of time and effort is being dedicated not only to the derivation of algorithms, but also to their implementation on high speed digital.

Until recently, finite difference schemes have been one of the most popular ways of representing the wave equation numerically. Unfortunately, conventional finite difference schemes are limited because of problems with spatial aliasing. Depending on the scheme used, the grid size must be considerably smaller than the shortest wave length component under consideration. The error is in part due to numerical truncation, which tends to dominate the shorter wave lengths. This problem manifests itself in the numerical solution as grid dispersion.

A number of finite difference schemes have been proposed to help alleviate such numerical problems. In recent years, very sophisticated schemes have been developed at Los Alamos to map expanding wave fronts in nuclear blasts. Many of these schemes use dynamic grid sizes. The grid spacing changes as the wave fronts progress through the model.

A second approach is to evaluate the spatial derivatives analytically. By considering the solution of the wave equation as a complete set of orthogonal basis functions, it is possible to compute the exact derivative of the composite function. Numerically, the Fourier Series (the Fast Fourier Transform) is ideally suited for this task, and a method known as the Fourier theoretical approach (Kosloff, 1982) takes advantage of this.

This thesis analyzes the Fourier theoretical approach with applications to both forward and inverse modeling. This method is applied to both One-way and Two-way non-reflecting wave equations. Because of the simplicity of the wave equation involved, the One-way equation is ideally suited to evaluate the capabilities of this algorithm. This approximation has been implemented to demonstrate the advantages and disadvantages of the Fourier theoretical technique. The Fourier theoretical solution to the Two-way non-reflecting wave equation is derived to demonstrate the capability of applying the technique to more sophisticated forms of the wave equations.

FUNDAMENTALS OF THE FOURIER THEORETICAL METHOD

General Theoretical Development

Although finite difference schemes are one of the most elegant and straight forward ways of representing the wave equation numerically, the applications of such algorithms have been limited due to an inherent problem. In most cases, the grid spacing (the spatial sampling rate) in X and Z is severely limited due to stability considerations, and improper choices of grid spacing result in errors which propagate due to numerical truncation of the shortest wave length components. In most cases the minimum spatial sampling rate needs to be at least twelve nodes for the shortest wave length. However, in many applications 25 nodes per shortest wave length are used to avoid grid dispersion:

$$\lambda min = \frac{Vmin}{fmax}; \; \Delta rmax \leq \frac{\lambda min}{12}, \qquad (1)$$

λmin = shortest wave length;

fmax = maximum frequency content of data;

Vmin = minimum velocity;

$\Delta rmax$= maximum allowable grid spacing.

In modeling, a fine grid spacing, while increasing the physical size of the finite difference solution, is not a problem, as it is simple to resample to a larger grid spacing once the solution is computed. In migration, however, since our X-grid spacing (the horizontal spacing) is set during acquisition, it is highly undesirable to resample to a larger grid spacing because recorded events may be spatially aliased. The Fourier theoretical technique is better suited for this purpose, since this method is independent of the spatial sample rate.

In this paper, the acoustic wave equation is used to evaluate the Fourier theoretical technique. Consider the acoustic wave equation of the form

$$\ddot{P}(X,Z,t) = V(X,Z)^2\nabla^2 P(X,Z,t) \qquad (2)$$

(see Appendix A)

where P is the wave field representing the pressure, t is the time of propagation, and X and Z are the horizontal and vertical distances, respectively. If the media represented by P(X,Z,t) is considered to be homogeneous (constant velocity V), then the solution of the above wave equation can be written as a Fourier series (Gazdag, 1981).

$$P(x,z,t) = \Sigma \; \Sigma \; \Sigma \; \hat{P}(k_x,k_z,\omega)e^{i(k_x X+k_z Z-\omega t)} \tag{3}$$

Summing over all k_x's, k_z's and where $P(k_x,k_z,\omega)$ is the three-dimensional Fourier transform of $P(X,Z,t)$. This formulation implies that the pressure field $P(X,Z,t)$ must be band limited to nyquist, and that it becomes periodic in X and Z. A band limited system assumption does not cause any problem, but considering the model periodic, could result in undesirable artifacts.

Since the solution of the wave equation can be written as a Fourier series in X and Z, it is possible to compute the spatial derivatives in the (k_x, k_z) domain.

An Example of the Fourier Theoretical Approximation

Starting with the acoustic wave equation

$$\ddot{P} = V(X,Z)^2 \nabla^2 P \ , \tag{4}$$

the goal is to approximate the solution with a finite difference scheme in time and a Fourier theoretical approximation in space. A classic form of approximating the second derivative explicitly is

$$\frac{d^2P(t)}{dt^2} = \frac{P(t+\Delta t)-2P(t)+P(t-\Delta t)}{\Delta t^2} \ , \tag{5}$$

where

1. $P(t-\Delta t)$, $P(t)$, and $P(t+\Delta t)$ are three consecutive wave fields in time;
2. Δt is the time sampling interval;
3. The error in this approximation is of the order $0(\Delta t^2)$.

Using equation (5), the forward numerical solution is

$$P(t+\Delta t) = 2P(t)-P(t-\Delta t)+\Delta t^2 V^2 \nabla^2 P(t), \qquad (6)$$

which is an approximation of the pressure field at $P(t+\Delta t)$ related directly to $P(t)$ and $P(t-\Delta t)$. By using the relationships in equation (3), the pressure field can be represented as a Fourier series. The second derivative of the pressure field is then represented by

$$\nabla^2 \hat{P} \leftrightarrow - (k_x^{\ 2}+k_z^{\ 2})\hat{P}(k_x,k_z,\omega)e^{i(k_x X+k_z Z-\omega t)} . \qquad (7)$$

Therefore, the solution of the acoustic wave equation involving a one-dimensional finite difference in time and a Fourier approximation of the derivatives in space is a combination of equations (6) and (7).

In theory, this type of approximation to the solution of the wave equation should be identical to that of full two-dimensional (X,Z,t) finite difference approximations. In this case, the spatial finite difference equations are substituted by a hybrid scheme which uses a Fourier domain approximation of the second derivative. This numerical solution to the derivative is exact within the frequency bank of the spatial mesh (Kosloff, 1982), implying that for the given frequencies the derivatives will be numerically accurate.

Theoretical Development of the One-Way Wave Equation

The relationships in equations (6) and (7) can be used in place of conventional finite difference schemes for forward modeling if multiples are desired. In seismic applications, however, it is desirable to be able to separate the upward and downward traveling waves to eliminate multiples which would hinder inverse modeling solutions (Claerbout, 1976). Since the direction of propagation of the wave front is controlled by the wave vector (k_x, k_z) and its temporal frequency (ω), upward and downward traveling waves can be separated by the dispersion relationship in the frequency wave number domain if the velocity V in the medium is assumed to be constant:

$$\omega^2 = (k_x^2 + k_z^2)V^2. \tag{8}$$

(see Appendix B)

By taking the square root of both sides of equation (8)

$$\omega = \pm(k_x{}^2+k_z{}^2)^{\frac{1}{2}}V, \tag{9}$$

the upward and downward traveling waves are separated. When the dispersion relationship in equation (9) is applied to the acoustic wave equation, the sign will control the direction of travel, and since the restriction of waves traveling in one direction only needs to apply in the z direction

$$\omega = Vk_z\left[1+\left(\frac{k_x}{k_z}\right)^2\right]^{\frac{1}{2}} \tag{10}$$

or

$$\omega = V\mathrm{sgn}(k_z)(k_x{}^2+k_z{}^2)^{\frac{1}{2}} \tag{11}$$

where if the wave number vector is given by $\underline{K} = k_x\hat{x}+k_z\hat{z}$ then

1. $k_z<0$: each wave component is displaced in the direction of its wave number vector;
2. $k_z>0$: each wave component is displaced opposite of its wave number vector.

Since it is known that

$$i\omega\hat{P} \quad ; \quad \hat{P}(X,Z,\omega)$$

transforms to

$$\frac{\partial P}{\partial t} \quad ; \quad P(X,Z,t)$$

by using equation (11) in the case of a constant velocity field (homogeneous earth) the One-way wave equation is derived in the Time-wave-number domain (SAL, 1983):

$$\frac{d\bar{P}}{dt} = i\ V\text{sgn}\ (k_z)(k_x{}^2+k_z{}^2)^{\frac{1}{2}}\bar{P}\ . \qquad (12)$$

In the spatial domain equation (12) is

$$\frac{dP}{dt} = VF_{xz}{}^{-1}\left[\ i\ \text{sgn}(k_z)(k_x{}^2+k_z{}^2)^{\frac{1}{2}}F_{xz}{}^{+1}(P)\ \right], \qquad (13)$$

where $F_{xz}{}^{+1}$ and $F_{xz}{}^{-1}$ are the forward and inverse two-dimensional spatial Fourier transform operators respectively. Unfortunately since the multiplication factor, $i\ \text{sgn}(k_z)(k_z{}^2+k_x{}^2)^{\frac{1}{2}}$, cannot be represented in the (X,Z) domain, the forward and inverse two-dimensional transforms will have to be evaluated for each time step.

Implementation of equation (13) for each time step is summarized as follows:

1. Take the spatial two-dimensional Fourier transform of $P(X,Z,t_i)$;
2. Multiply by the one-way derivative operator;
3. Take the inverse two-dimensional Fourier transform of $P(k_x,k_z,t_i)$;
4. Multiply by the velocity field V;
5. Resulting in dP/dt.

Limitation of the One-Way Wave Equation

The One-way wave equation is obtained from the acoustic wave equation. Because of this, it is affected by the set of assumptions made in the derivation of the acoustic wave equation. The initial assumption made is that the acoustic wave equation assumes an isotropic fluid with zero viscosity. Assuming a fluid medium restricts the solution only to compressional waves. While this assumption reduces the validity of the solutions, the primary data of interest in seismic applications is that of compressional waves. In areas of complex geology (high velocity gradients and complex geologic structures), where shear waves and converted waves give important information, the solution from the field assumption may not be accurate enough. The next assumption is done to obtain linear relationships between pressure and particle velocity from the non-linear forms. It can be shown (Berkhout, 1982) that this assumption is valid for practical seismic velocities. The final assumption made in the derivation of the acoustic wave equation is that $\nabla \ln \rho$ is zero. Ignoring inhomogeneity in density is a common assumption made in many seismic applications. However, in the derivation of the acoustic wave equation, two fundamental acoustic parameters are compressibility (K = bulk modulus) and density (ρ). Both parameters determine the acoustic velocity

$$V = \sqrt{\frac{K}{\rho}} .$$

Therefore, if we neglect the density term $\nabla P \cdot \nabla \ln\rho$, it does not mean that the influence of variable density is ignored. In fact, the effect of density on a seismic wave is included in the velocity field (Berkhout, 1982).

The second set of assumptions affects the final solution in a more severe manner. In the derivation of the One-way wave equation, the velocity field is transformed into the wave number domain. To do this, a homogeneous velocity is assumed. If the One-way wave equation could only work for homogeneous velocities, this would be unacceptable. However, in the final form of the wave equation, an inhomogeneous velocity field is allowed. While physically this is incorrect, in practice it is shown that as long as the velocities vary slowly throughout the model, the times of propagation of the wave fronts are correct.

The wave equation in the form of equation (13) is an ordinary differential equation which can be solved by any standard numerical method. Because of the assumptions made during the derivation of the One-way wave equation, it is not necessary to use some of the highly accurate finite difference schemes (Runga Kutta), and hence two simple schemes are considered.

Taylor Series Approximation

Gazdag (1981) proposes to use a Taylor series approximation

$$P(t+\Delta t) = \sum_n \frac{\partial^n P(t)}{\partial t^n} \cdot \frac{\Delta t^n}{n!} \tag{14}$$

to solve equation (13). This term gains accuracy as more terms are considered. By using the first four terms,

$$P(t+\Delta t) = P(t)+\frac{\partial P}{\partial t}\Delta t+\frac{\partial^2 P}{\partial t}\frac{\Delta t^2}{2!} + \frac{\partial^3 P}{\partial t^3}\frac{\Delta t^3}{3!}+O(\Delta t^4) \qquad (15)$$

and a recursive method to solve the higher order derivatives

$$\frac{d^n P}{dt^n} = V(x,z)F_{xz}{}^{-1}\left[i\ \mathrm{sgn}(k_z)(k_x{}^2+k_z{}^2)^{\frac{1}{2}}F_{xz}{}^{+1}\left[\frac{d^{n-1}P}{dt^{n-1}}\right]\right] \qquad (16)$$

the One-way wave equation is solved. If this scheme is implemented it would involve six full two-dimensional transforms per time step. Because of the expense of the Fourier transform this would not be a viable alternative.

Centered Difference Approximation

Instead of evaluating higher order derivatives, it would be more appropriate to approximate the first derivative with a lower order scheme but take smaller time steps. Kosloff (1982) proposes to use a centered difference scheme

$$\frac{dP^n}{dt} = \frac{P^{n+1}-P^{n-1}}{2\Delta t}\ , \qquad (17)$$

where $P^n = P(n\Delta t)$

to approximate the first derivative. In fact, because of the approximations made in deriving the One-way wave equation, the accuracy of the centered difference scheme is sufficient to get a good solution. It is this algorithm which has been implemented for both modeling and reverse time migration.

Implementation of the One-Way Wave Equation

The One-way wave equation

$$\frac{dP}{dt} = V(x,z)F_{xz}^{-1}\left[i\ \mathrm{sgn}(k_z)(k_x^2+k_z^2)^{\frac{1}{2}}F_{xz}^{+1}(P(x,z,t))\right] \quad (18)$$

as previously discussed, is a simplified form in which spatial derivatives involving the velocity are ignored. While it may seem that this a gross approximation of the wave equation, since a Taylor series was not used to derive this solution, there is no restriction on the angles of dip it should be able to handle. It is because of this that it is called the ninety degree wave equation. As long as the velocity gradient is not too large, the wave field will be correctly reconstructed.

Forward Propagation

By using an explicit centered difference approximation to the first derivative,

$$P^{n+1} = 2\Delta t \frac{dP^n}{dt} + P^{n-1} \tag{19}$$

and by combining equations (18) and (19), the first time derivative can be related to the spatial derivatives numerically.

In the implementation of equation (19), there are several factors which must be considered.

Stability

While there are no severe restrictions on the spatial derivatives because of the Fourier theoretical approach, there is a relationship between the spatial sampling rate and temporal frequency spectrum. This restriction is a nyquist-like relationship,

$$\Delta r\mathrm{max} < \frac{V\mathrm{min}}{2f\mathrm{max}} \tag{20}$$

which means that there should be at least two spatial samples for the shortest possible wave length. By restricting the frequency spectrum of the source wavelet, it is possible to increase the grid spacing to an allowable amount. Because of the accuracy of the derivatives computed in the wave number domain, the stability restriction is different from that of a normal finite difference scheme (equation (1)).

The other stability requirement, due to the finite difference in time, is that of the size of the time step. It can be shown that for the centered difference scheme, the time step has to be

$$\Delta t < \frac{\Delta x}{\sqrt{2}\ \pi V_{max}} \tag{21}$$

if $\Delta x = \Delta z$ (see Appendix B).

Therefore, before modeling is done, the source function has to be resampled to a sample rate less than Δt. To do this resampling, I chose a sin(x)/x interpolation scheme. The interpolation is done in the frequency domain by padding the spectrum with zeros and doing an inverse transform back to time. This type of interpolation guarantees that the frequency spectrum of the original data will remain unchanged.

Once the grid spacing is determined and the source function is resampled, the pressure field can be propagated through the earth. By using an exploding reflector approximation (see Figure 1), the source function is inserted along any boundary which is to be mapped into time. As time progresses, the source function is continuously added into the boundaries. Then, as the wave propagates through (X,Z), it can be mapped into time at Z = 0.

Numerical Dispersion

In the Fourier theoretical method, the spatial derivative operator is accurate for any pressure field within the frequency band of the spatial mesh. If the source function $S(\underset{\sim}{x},t)$ has the appropriate frequency spectrum, then errors in the numerical solution will come from the inaccuracy of the time derivative (the finite difference) approximation. The error appears in the solution as numerical dispersion. Unlike conventional finite differences (Alford, 1974), dispersion diminishes rapidly as the size of the time step is decreased. At the stability limit for the one-dimensional case,

$$\frac{V\Delta t}{\Delta x} < \alpha$$

where $\alpha = \dfrac{1}{\sqrt{2}\ \pi}$ (22)

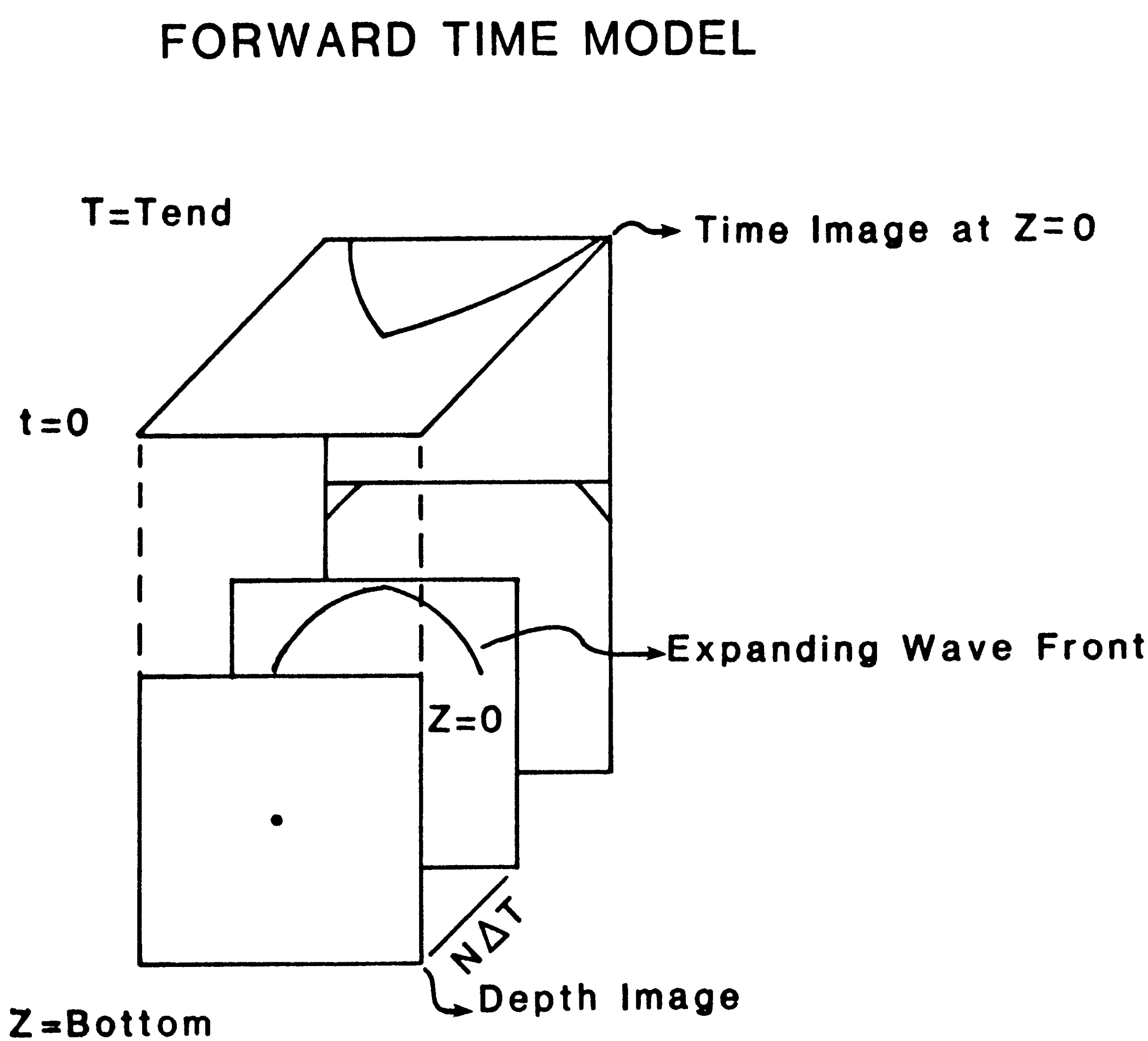


Figure 1. Pictorial representation of the exploding reflector approximation.

the numerical dispersion is considerable. However, as the time step is decreased, the effects of dispersion will almost disappear. When $\alpha = 0.2$, there is almost no numerical dispersion for all frequencies in the band of the mesh.

Comparing the dispersion relationships calculated by Alford (1974) for normal finite difference schemes of the acoustic wave equation (see Figure 2) to those of the Fourier theoretical approach (see Figure 3; Kosloff, 1982), the Fourier approach will have much smaller errors for the same α.

Boundary Conditions

The periodic nature of the Fourier method can cause problems when considering boundary conditions. True absorbing boundaries are very difficult to design in the wave number domain, therefore to simulate absorbing boundary conditions the velocity field around the edges is slowly tapered to zero. This boundary condition, while not a true absorbing boundary, behaves nicely as long as the gradient of the taper on the velocity field is small.

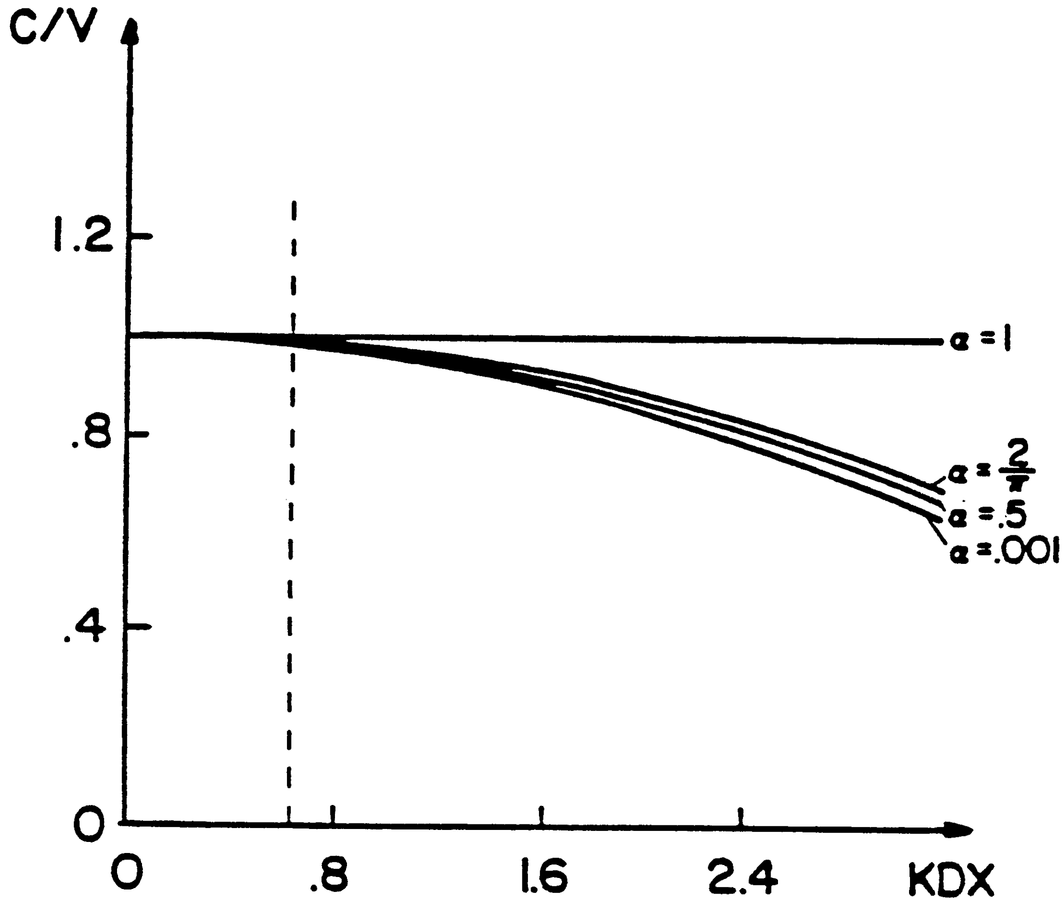


Figure 2. Dispersion relationships of a second order finite difference scheme for the homogeneous wave equation for different ratios of α (Alford, 1974).

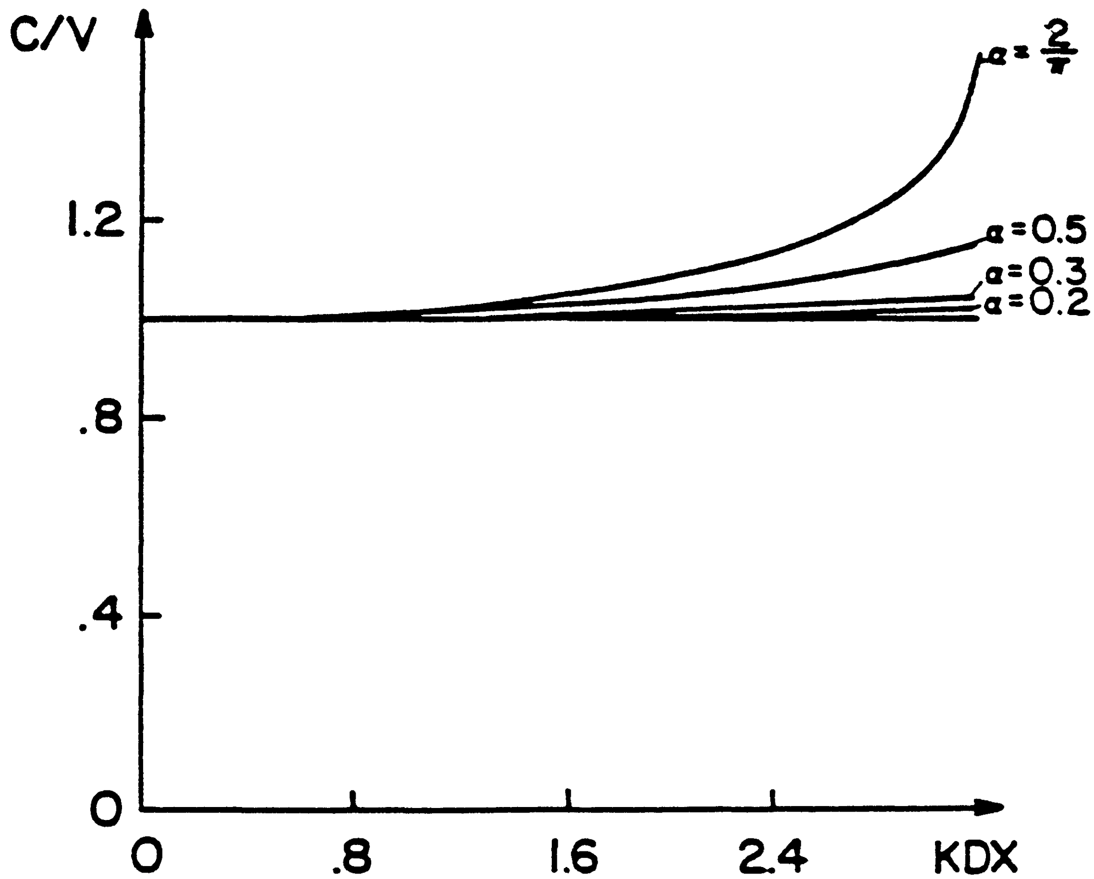


Figure 3. Dispersion relationships of the Fourier theoretical technique approximation of the One-way wave equation for different ratios of α.

Another way of eliminating the "wrap around" problem of the periodic boundary condition is done during the construction of the spatial mesh. The spatial grid for the Fourier theoretical approach has to be large enough to insure that the "critical" events will arrive before the "wrap around", due to the periodic boundary conditions, occurs.

Reverse Time Migration

Migration in Reverse time (Baysal, 1982; see Figure 4) uses the same algorithms and stability requirements as the exploding reflector modeling procedure. There are only two differences. The first is that, instead of picking a grid spacing according to the frequency content of our data, the data is filtered relative to the set grid spacing in X. It is better to filter the data in time than to resample it in X because of problems with events which are spatially aliased. From equation (20),

$$\Delta r = (\Delta x^2+\Delta z^2)^{\frac{1}{2}} \quad \text{and if } \Delta x = \Delta z,$$

$$fmax < \frac{Vmin}{\sqrt{2}\ \pi\Delta r}\ , \tag{23}$$

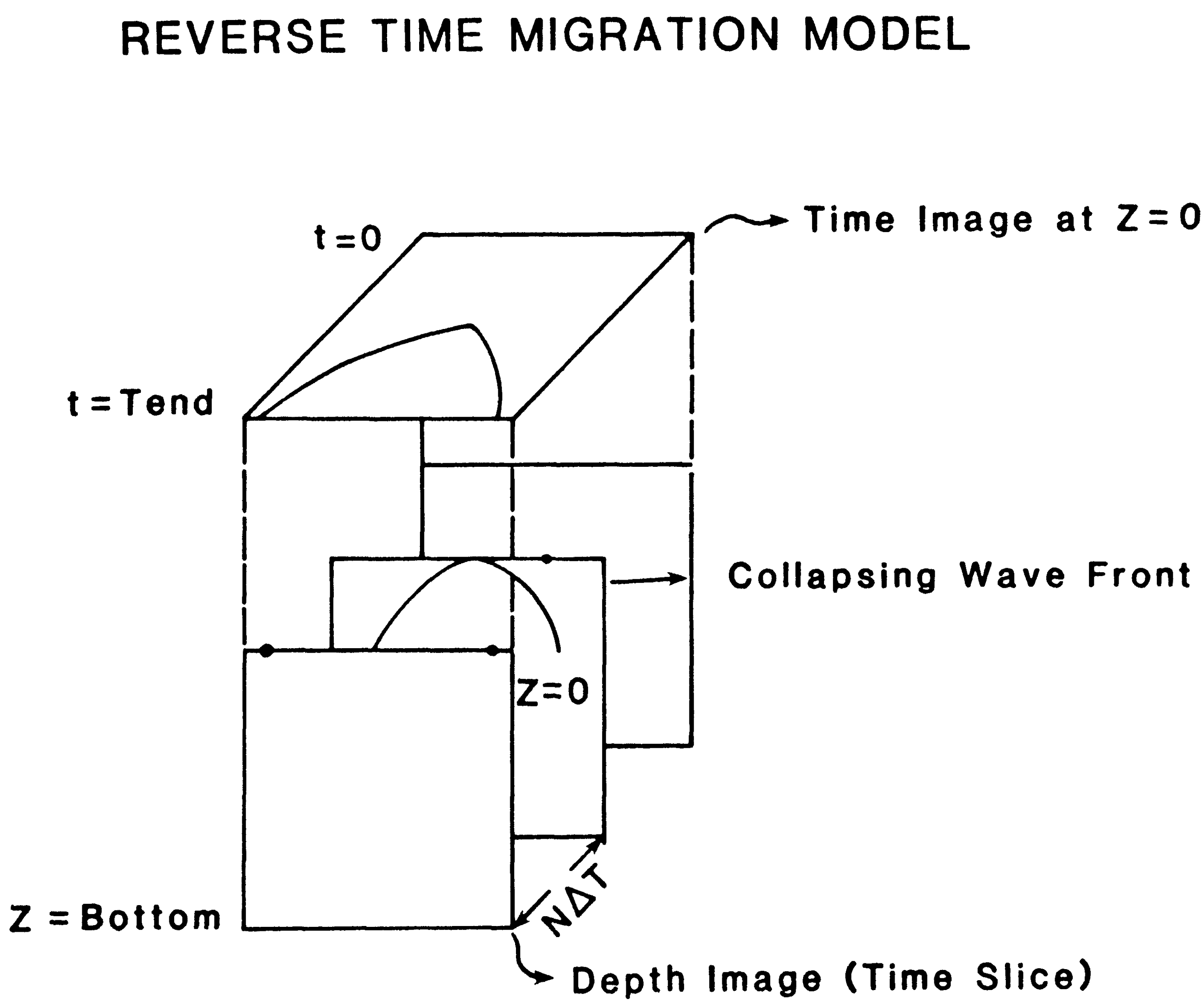


Figure 4. Pictorial representation of Reverse time migration.

the maximum allowable frequency for a given grid spacing is obtained by filtering the input data back to the maximum allowable frequency that the stability criterion will allow. The filtering has to be done before the migration because instabilities due to spatial sampling will appear as periodic components due to aliasing, and will corrupt the solution in a manner which cannot be corrected after the fact.

The second difference is the way the boundary conditions are handled. In the case of modeling, the surface time response is the unknown. However, in migration, the time response along the Z = 0 axis is known for all time, but the depth response at t = 0 is not. By stepping backwards in time using

$$P^{n-1} = P^{n+1} - 2\Delta t \frac{dP^n}{dt} \tag{24}$$

and equation (18), the wave field in depth is reconstructed back until t = 0.

Problems

The one problem that exists with both the modeling and Reverse time migration algorithm for the One-way wave equation is inherent in the derivative factor which is used to multiply the spatial frequencies. To understand this problem, it is necessary to analyze the derivative operator

$$i\ \mathrm{sgn}(k_z)(k_x^{\,2}+k_z^{\,2})^{\frac{1}{2}} \tag{25}$$

which can be divided into two distinct parts. The first part

$$(k_x^{\,2}+k_z^{\,2})^{\frac{1}{2}} \tag{26}$$

is a two-dimensional form of a first derivative. It is not a smooth operator; it has slight discontinuities in its derivative along the spatial nyquist frequencies and at $k_x = k_z = 0$ (the absolute DC point in a two-dimensional Fourier domain). These discontinuities are very small, and while they may cause slight Gibb's phenomena, it does not affect the solution. The second part

$$i\ \mathrm{sgn}(k_z) \tag{27}$$

is a Hilbert transform operator. During early time in the modeling, when the spatial wave field is almost zero and spikes are being forced in as source/boundary conditions, the side lobes after the inverse spatial transforms are very large. The large side lobes create what appears to be vertical plane waves which give a time domain response very similar to that of a direct wave. Unfortunately, the exploding reflector model does not give a direct wave response. It is interesting to note that the "vertical plane waves" are not apparent in Reverse time migration, because, unlike single point diffractors, the response is defined over all space (all X) on the surface (z = 0). This impedes the vertical plane waves from forming.

The Two-Way Non-Reflecting Wave Equation

The acoustic wave equation obtained for a variable velocity and density field is

$$\rho \nabla \cdot \left(\frac{1}{\rho} \nabla P\right) = \frac{1}{V^2} \ddot{P} \tag{28}$$

(see Appendix C)

To obtain the Two-way non-reflecting wave equation, constant impedance across any boundary is assumed:

$$\rho V = \text{constant} . \tag{29}$$

When equations (28) and (29) are combined, the result is a Two-way non-reflecting wave equation where the reflection coefficient at normal incidence is zero:

$$V\frac{\partial}{\partial x}\left[V\frac{\partial P}{\partial x}\right] + V\frac{\partial}{\partial z}\left[V\frac{\partial P}{\partial z}\right] = \frac{\partial^2 P}{\partial t^2}\ . \qquad (30)$$

(Baysal, 1984)

By using the Fourier theoretical approach in space, and a finite difference in time, this form of the wave equation can be implemented. The second derivative in space is approximated by

$$\frac{d^2P^n}{dt^2} = \frac{P^{n+1}-2P^n+P^{n-1}}{\Delta t^2}\ . \qquad (31)$$

This time, however, the spatial derivatives are implemented in a slightly different fashion. Looking at the one-dimensional case, the wave equation involves two first derivatives in space and a second derivative approximation in time,

$$V\frac{\partial}{\partial x}\left[V\frac{\partial P}{\partial x}\right] = V(x,z)F_x^{-1}\left[ikxFx^{+1}\{V(x,z)F_x^{-1}(ik_xF_x^{+1}(P(x,z,t))\}\right] \qquad (32)$$

where Fx^{+1} and Fx^{-1} are one-dimensional forward and inverse Fourier transforms.

The Fourier theoretical solution of this wave equation involves a series of one-dimensional Fourier transforms, which are easier to implement than a two-dimensional one.

There is great potential in this form of the wave equation, especially for migration, because in theory it would have no dip or velocity gradient limitations.

Implementation of the Two-Way Non-Reflecting Wave Equation

The Two-way non-reflecting wave equation

$$\frac{\partial^2 P}{\partial t^2} = V\frac{\partial}{\partial x}\left[V\frac{\partial P}{\partial x}\right] + V\frac{\partial}{\partial z}\left[V\frac{\partial P}{\partial z}\right] \qquad (33)$$

is a more accurate form than the One-way wave equation to apply to the exploding reflector concept for both the forward and inverse problem. The Two-way non-reflecting wave equation simultaneously allows both up and down going waves. In a homogeneous medium it is identical to the acoustic wave equation. However, when propagating from one medium to another, the wave equation has a zero reflection coefficient for a normal incident wave (see Appendix C). Hence the name Two-way non-reflecting wave equation. While this may seem in contradiction to the exploding reflector concept, unlike the One-way wave equation, it allows the model to approximate the wave field in very high velocity gradients.

In a comparison between the One-way wave equation and the Two-way non-reflecting wave equation, the differences become apparent. The model is an overhanging fault block of 3000 ft/sec, imbedded in a linear velocity gradient medium increasing 4.4 ft/sec per foot from 2000 ft/sec to 24,000 ft/sec (see Figure 5). The One-way wave equation in an exploding reflector approximation, because there is only upgoing waves, cannot properly model the wave front which is propagating through the high velocity gradient layer (see Figure 6). What should really be happening is that the wave front generated by the fault plane initially travels downward and is then turned around by refraction and will propagate to the surface. This is correctly modeled by the Two-way non-reflecting wave equation (see Figures 7 and 8).

The non-reflecting wave equation is conceptually easier to implement than the One-way wave equation and the method of applying this form of the equation by using the Fourier theoretical technique will be demonstrated with the one-dimensional form:

$$\frac{\partial^2 P}{\partial t^2} = V\frac{\partial}{\partial x}\left[V\frac{\partial P}{\partial x}\right] \qquad (34)$$

VELOCITY GRADIENT MODEL

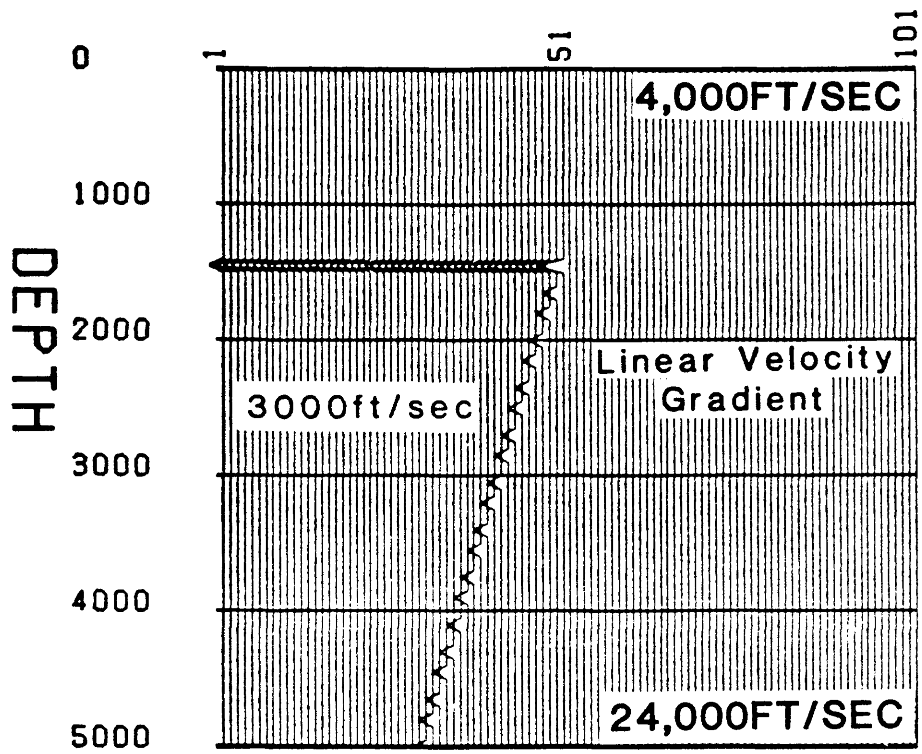


Figure 5. The velocity gradient model used to test the effects of high velocity gradients on the One-way wave equation.

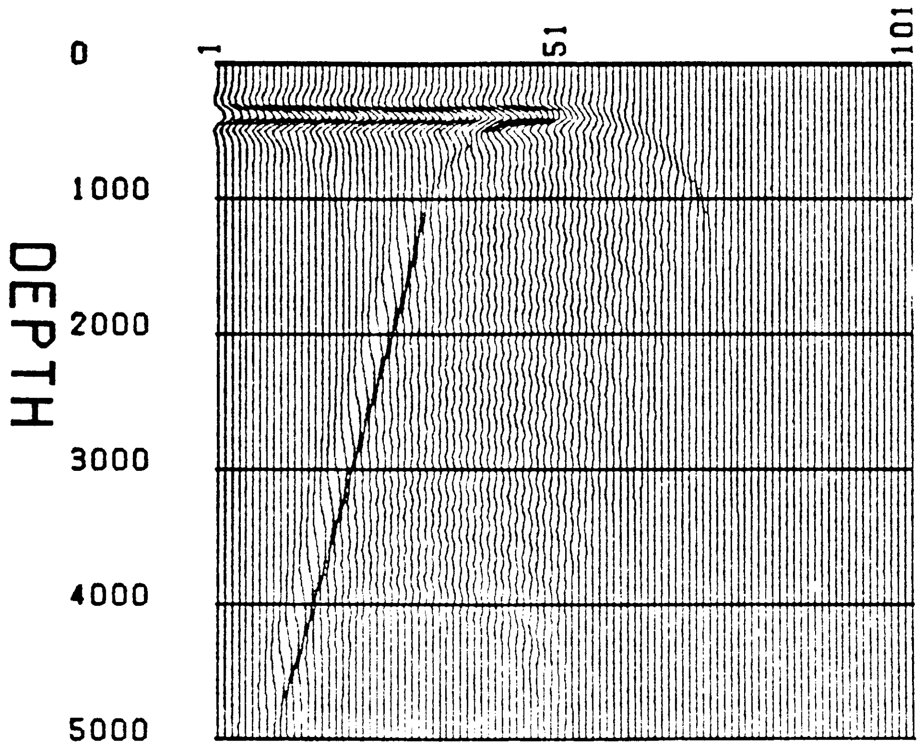


Figure 6. Depth snapshot of the velocity gradient model for the One-way wave equation.

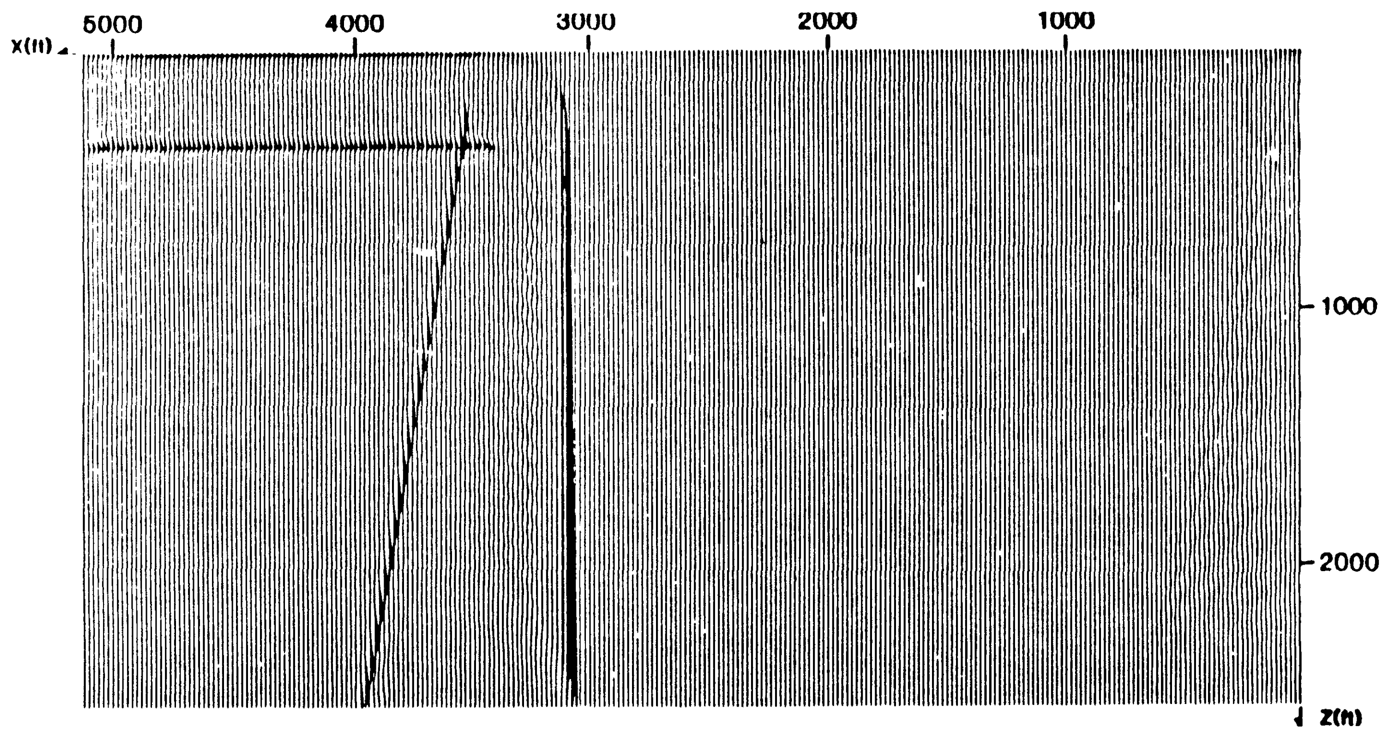


Figure 7. Depth snapshot of the velocity gradient model for the Two-way non-reflecting wave equation.

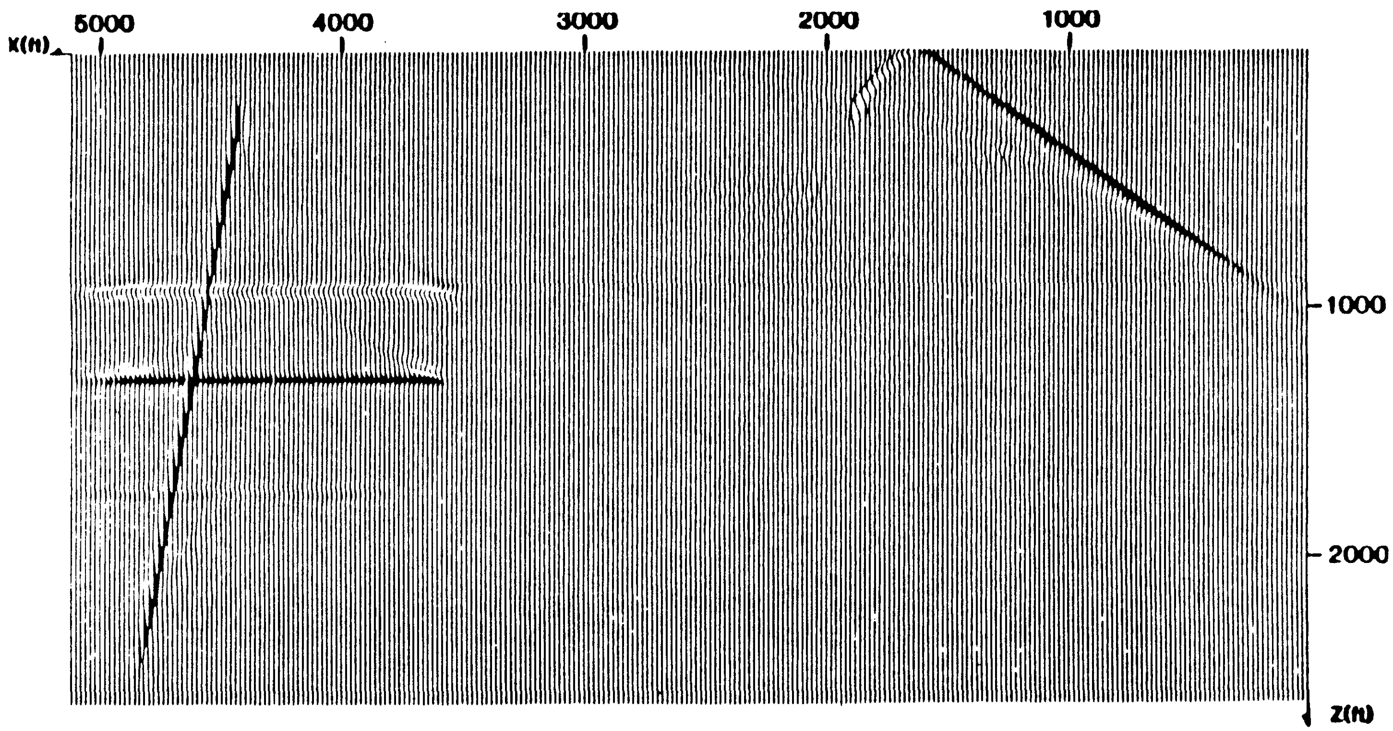


Figure 8. Depth snapshot of the velocity gradient model for the Two-way non-reflecting wave equation. The Two-way non-reflecting wave equation will refract the wave front from the fault plane back to the surface due to the high velocity gradient in the model.

where P is the pressure field and V(X,Z) is a spatially varying velocity field. The second derivative in time is approximated by the explicit finite difference approximation in equation (31), which is a second order ($O(\Delta T^2)$) accurate scheme.

The spatial derivatives are implemented in the Fourier domain. Taking equation (34), it is necessary to

1. Transform the pressure field by a one-dimensional Fourier transform;
2. Multiply by the first derivative operator (ik);
3. Do an inverse transform;
4. Multiply by the velocity field;
5. Again transform the field to the Fourier domain;
6. Multiply by the first derivative operator;
7. Inverse transform;
8. Multiply by the velocity field.

It is apparent that the Two-way non-reflecting wave equation is taking a derivative of the velocity field as well as derivatives of the pressure field. In two dimensions the spatial derivatives would be approximated by

$$V(x,z)F_x^{-1}\left[ik_xF_x^{+1}(V(x,z)F_x^{-1}\{ik_xF_x^{+1}(P)\})\right] +$$
$$V(x,z)F_z^{-1}\left[ik_zF_z^{+1}(V(x,z)F_z^{-1}\{ik_zF_z^{+1}(P)\})\right] =$$
$$\frac{\partial^2 P}{\partial t^2} \tag{36}$$

Stability

As in the One-way wave equation, the only stability restriction in space is the nyquist relationship

$$fmax < \frac{Vmin}{2\Delta r} . \tag{37}$$

The second derivative in time, however, will give a different stability relationship for the finite difference:

$$\frac{Vmax\Delta t}{\Delta x} < \frac{\sqrt{2}}{\pi} \quad \text{if } \Delta x = \Delta z. \tag{38}$$

It will be necessary to resample the data in time. The stability criterion is twice as large as the constraint for the One-way wave equation (see Appendix C).

Boundary Conditions and Numerical Dispersion

It is not necessary to discuss the boundary conditions and dispersion relationships (see Appendix C) as they are also very similar to those of the One-way wave equation.

Problems

The problem which plagues the Two-way non-reflecting wave equation is not a numerical but computational one. Instead of involving two two-dimensional transforms, it entails four forward and four inverse transforms. The amount of computations can cause a number of application problems.

The Numerical Two-Dimensional Fourier Transform

The limiting factor in the Fourier theoretical method is the ability to take two-dimensional transforms. This operation is done thousands of times in the simplest form of the hybrid scheme, therefore it is necessary to develop a scheme which is fast and efficient relative to memory usage and I/O. Because computers have a finite memory, it is necessary for the two-dimensional Fourier transform to be computed with a method which stores most of the data out of core. There are two such schemes which I have implemented.

The first assumes that the data is stored so that one of the directions is stored contiguously (columns) on a mass storage device. First the values are transformed using system FFTs. The non-contiguous element (row) transforms are done by considering each column as a array Vi. During a row Fourier transform operation, every element in array Vi will have the same operation done to it... eg: the complex scale factor is the same for every element in Vi. Because this form of the two-dimensional FFT is I/O bound, it is ideal for a vector processor where the access time to the mass storage device is very fast.

The second algorithm of computing two-dimensional FFTs is proposed by Gazdag (SAL, 1983). The data is blocked into small equal subsets. The idea behind the algorithm is to reconstruct the contiguous elements of the vector on the "fly" and then use the system FFTs in both directions. This form of the two-dimensional FFT is not I/O bound as the transfers can be coded very efficiently. However the coding is very complicated. If memory size is severely restricted (such as in a FPS-100 array processor) this is the way to do the two-dimensional transform.

Computational Problems of the Fourier Theoretical Technique

The Fourier theoretical method for solving partial differential equations is a very powerful method to accurately approximate derivatives. However, until recently, it was not considered as a viable alternative to the more cumbersome finite difference techniques. The reason will become more apparent as I use the One-way wave equation as a case history. The algorithm for the One-way wave equation can be divided into two distinct operations:

1. Preprocessing the data... Filtering and re-sampling in time to compensate for the stability and dispersion due to the finite difference in time;
2. Implementation of the hybrid finite difference scheme.

The preprocessing of the data is necessary but is a standard one time process and does not cause any undue computational problem.

The second step, implementing the One-way wave equation as in equation (18), can be divided into five distinct steps per time increment:

1. A forward two-dimensional Fourier transform;
2. Multiplication by the one-way propagation operator;
3. An inverse two-dimensional Fourier transform;

4. Multiplication by the velocity field;

5. Finite difference addition.

While very simple in nature it is necessary to look at the number of operations involved in the above five steps.

Assuming an N by N grid

1) The most efficient and fast way to approximate the Fourier transform is through the Fast Fourier transform (FFT). This is an N * log(N) operation. Therefore, for one two-dimensional FFT it would involve

N rows * (N*log(N))+
N columns* (N*log(N))

operations.

2) Multiplication by an N*N point grid for the One-way propagation operator:

N squared operations.

3) Inverse two-dimensional FFT:

N rows * (N*log(N))+
N columns* (N*log(N)).

4) Multiplication by an N*N point velocity field:

N squared operations.

5) The finite difference addition:

N squared operations.

The total number of operations is

$$4*N^2*\log(N)+3*N^2$$

Taking a small grid of 512 by 512 grid points, the total number of operations by time step is on the order of 10.2 million operations. If there was five seconds of data and a time step of one millisecond (5000 time steps), the total number of operations involved would be a phenomenal 51 billion.

The second problem inherent in Fourier techniques is of the same order of magnitude. The hybrid scheme involves six distinctly different matrices on which operations are done

$$V(x,z),\ D(k_x,k_z),\ P^n,\ P^{n+1},\ P^{n-1},\ \text{work}\ (k_x,k_z)$$

Each of the matrices are complex valued ($2*N^2$ in size). The total number of words required in the case of the 512 by 512 grid would be 3.1 million words.

Because of these two problems the Fourier theoretical technique has not attracted attention until recently. With the advent of super digital computers and array processors this technique is becoming a viable process.

RESULTS OF FORWARD AND INVERSE MODELING

To investigate the Fourier theoretical method, a series of experiments are done by using the One-way wave equation. In the case of modeling an exploding reflector approximation is assumed, and for migration, Reverse time migration is used. The initial tests involve a single point diffractor with various configurations of velocity fields. The last three experiments are done with three complex earths: a fault block model, a syncline, and an anticline. In the three cases the Reverse time migrations are done with both correct and incorrect velocity models.

Point Diffractor

Forward and Inverse Modeling in a Homogeneous Medium

Initially, a homogeneous earth is used to test propagation of a point source in the earth. The first model is a seventy hertz point diffractor buried at a depth of about 3800 feet in a 10,000 ft/sec velocity medium in a 101 by 101 depth model grid (see Figure 9). The wave fronts as a function of space are shown for progressing time in the forward and inverse problem (see Figures 10-12, 14-17). The final migrated section is in depth (see Figure 18).

Observations of a Point Diffractor in a Homogeneous Medium

In the forward modeling depth snap shots (see Figures 10-12), the wave fronts are very well defined. There is no apparent loss of frequency with angle except for a geometric rotation effect of (f cos (θ)). The FFT buffers in this case are of size 512 by 512. It is apparent only in the time representation that though the wrap around effect has not been completely eliminated, it has been substantially reduced and is not affecting the solution (see Figure 13).

In the Reverse time migration, the snap shots show an interesting aperture problem. To reconstruct the full conical wave front, an infinite hyperbola in time is necessary. Since we do not have such a solution, the full conical wave fronts cannot be formed by the migration (see Figures 13-17). Because of this, the pure spike in the X direction can never be reconstructed. However, it is obvious from the results (see Figure 18) that this should cause little problems, because in most cases migration will always be done over much larger apertures.

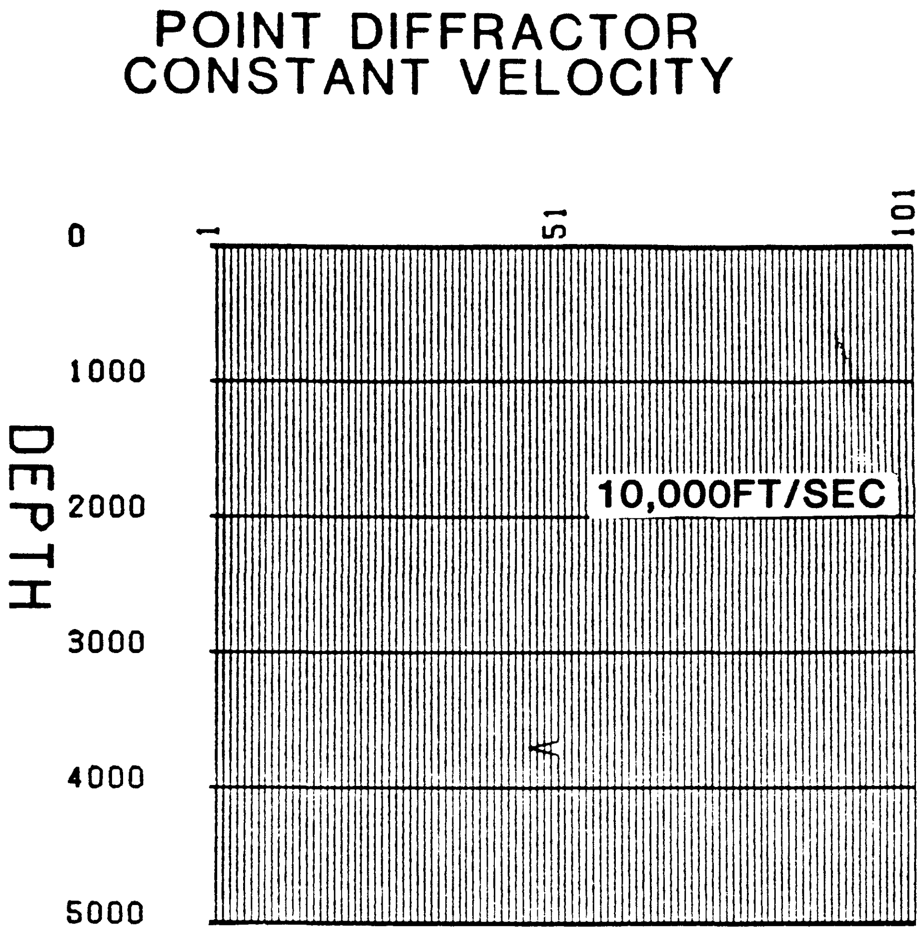


Figure 9. Point diffractor model in a homogeneous earth.

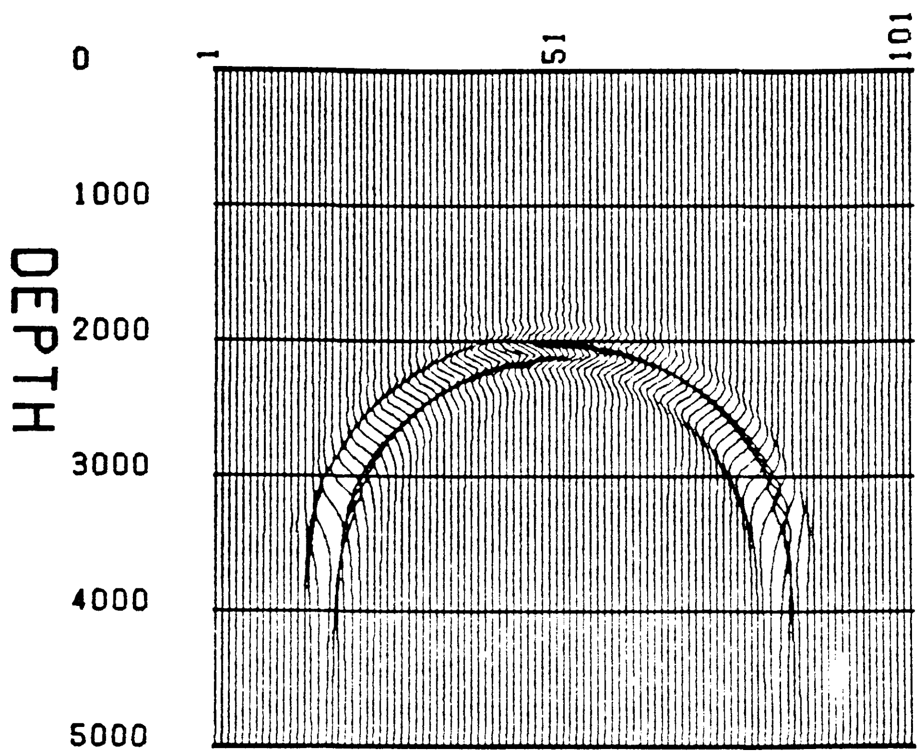


Figure 10. Depth snapshot during early time. The tails at the edges of the conical wave front cause the wrap around problem apparent in the Fourier theoretical technique approximation of the One-way wave equation.

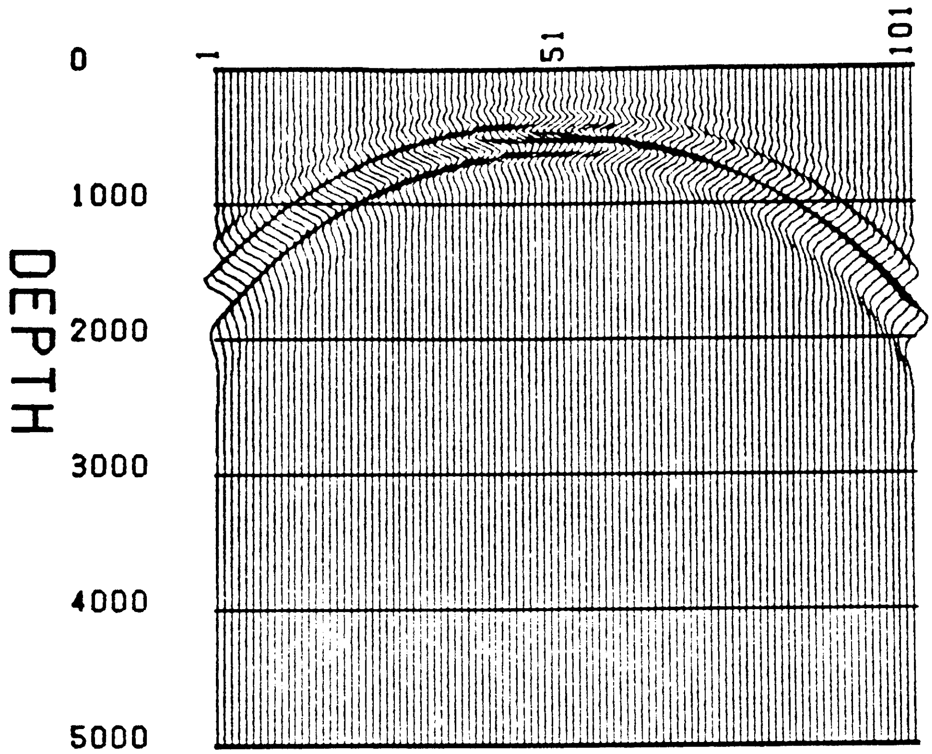


Figure 11. Depth snapshot of the propagating wave front due to a point diffractor just before the wave front arrives at the surface.

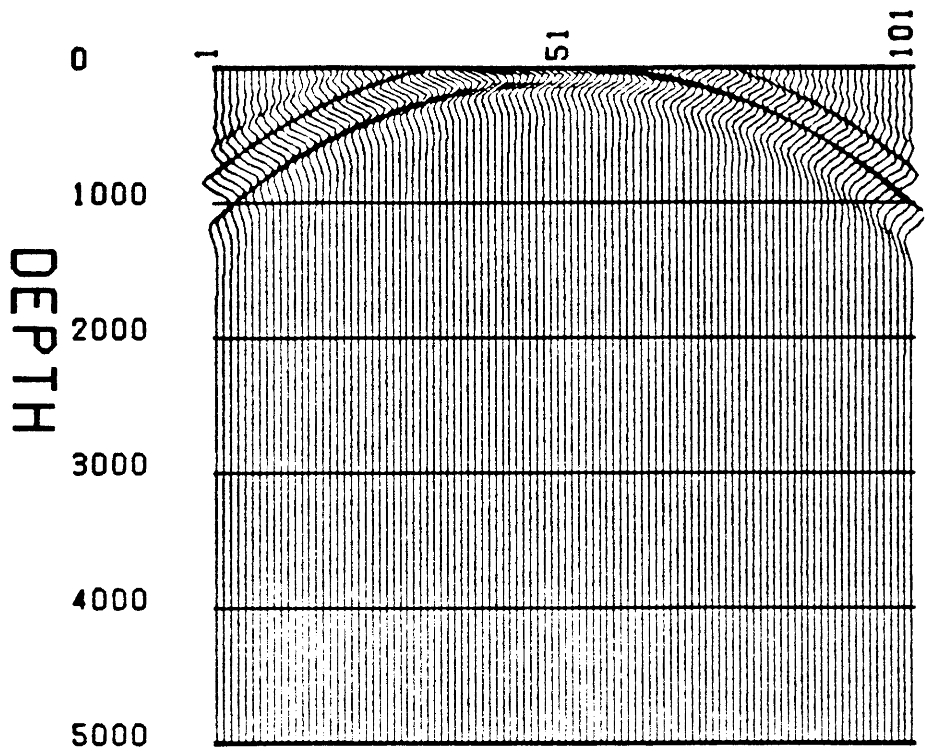


Figure 12. Depth snapshot of the propagating wave front due to a point diffractor just before the wave front arrives to the surface.

MODELLED POINT DIFFRACTOR
TIME RESPONSE

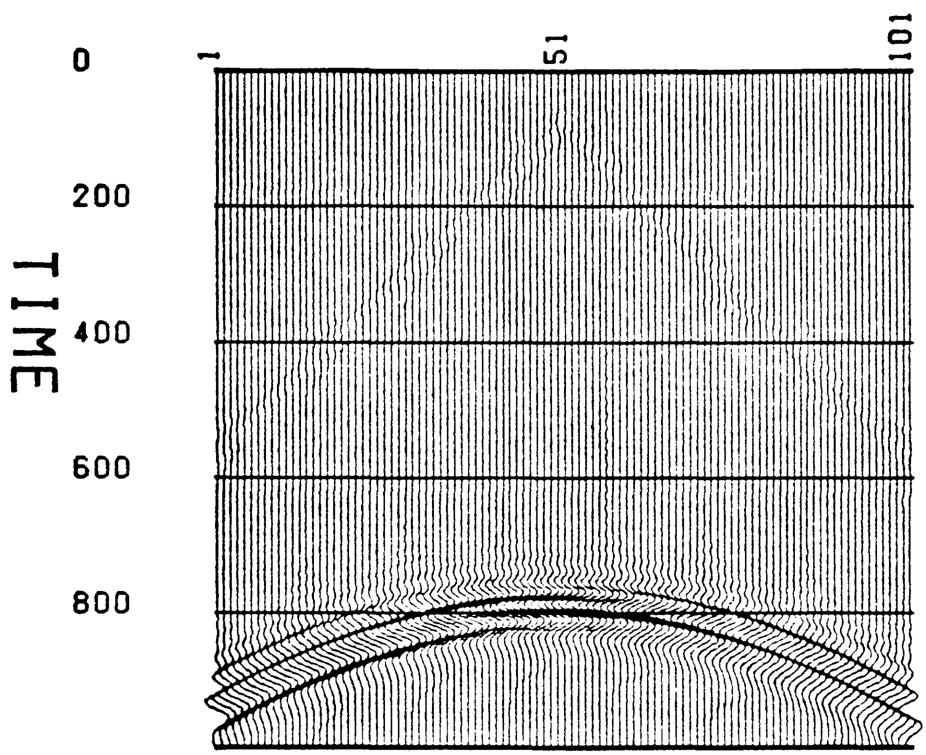


Figure 13. Time section of the point diffractor in a homogeneous medium. The slight straight line artifact is a result of the wrap around effect due to the One-way wave equation.

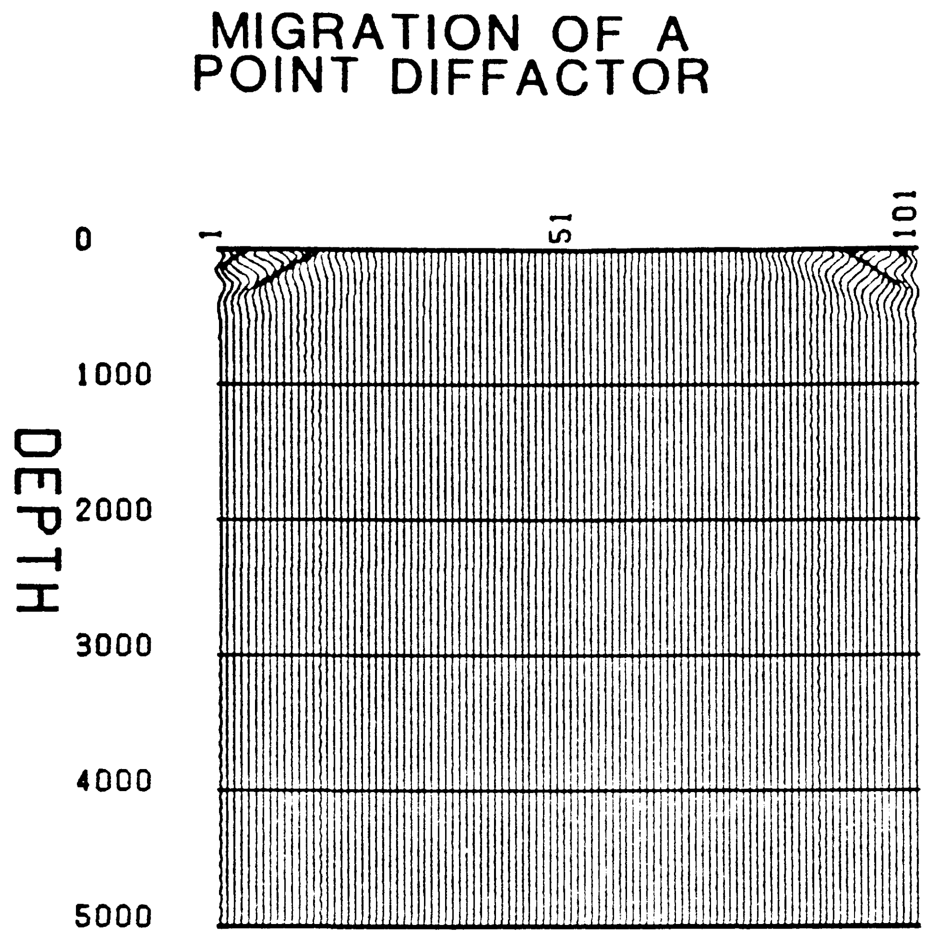


Figure 14. Depth snapshot of the collapsing wave front during Reverse time migration at late time (in Reverse time migration, time regresses).

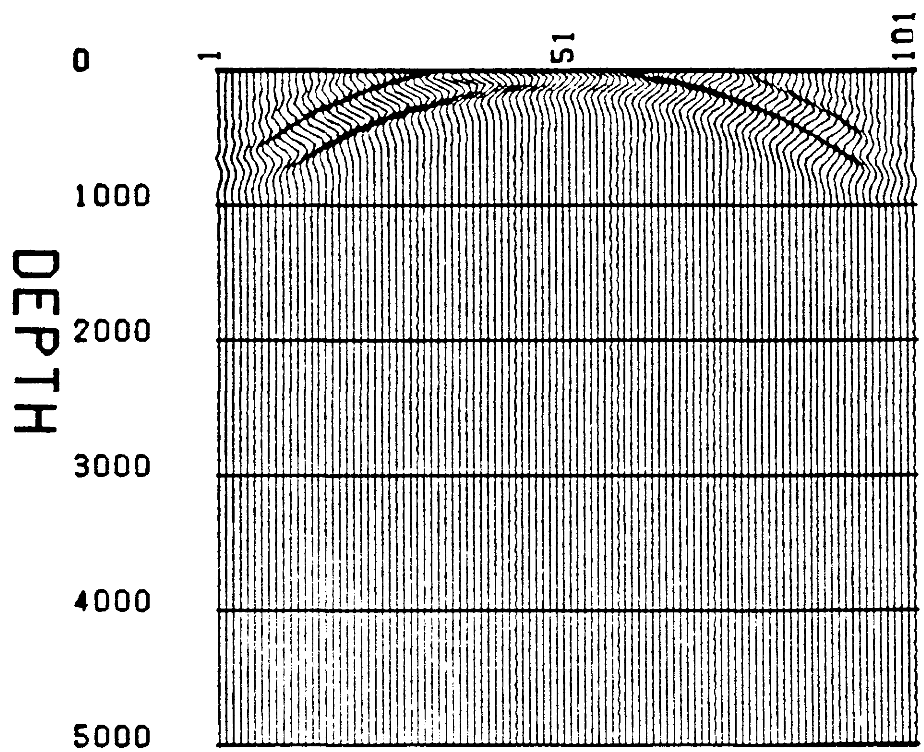


Figure 15. Depth snapshot of the collapsing wave front during Reverse time migration at TO-100 msec of one-way time.

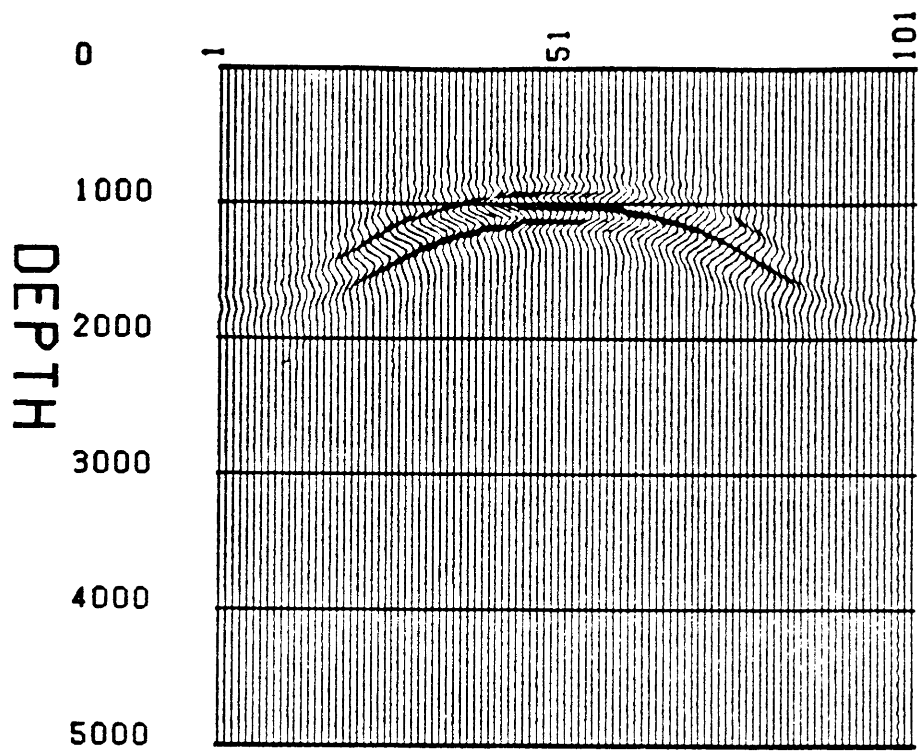


Figure 16. Depth snapshot of the collapsing wave front during Reverse time migration at TO-200 msec of one-way time.

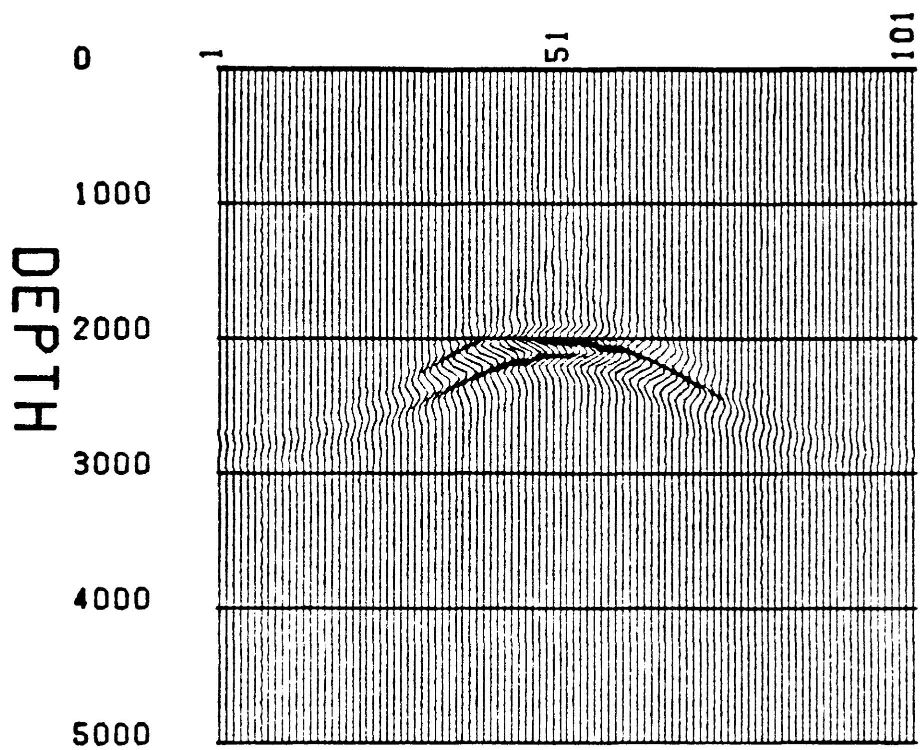


Figure 17. Depth snapshot of the collapsing wave front during Reverse time migration at TO-300 msec of one-way time.

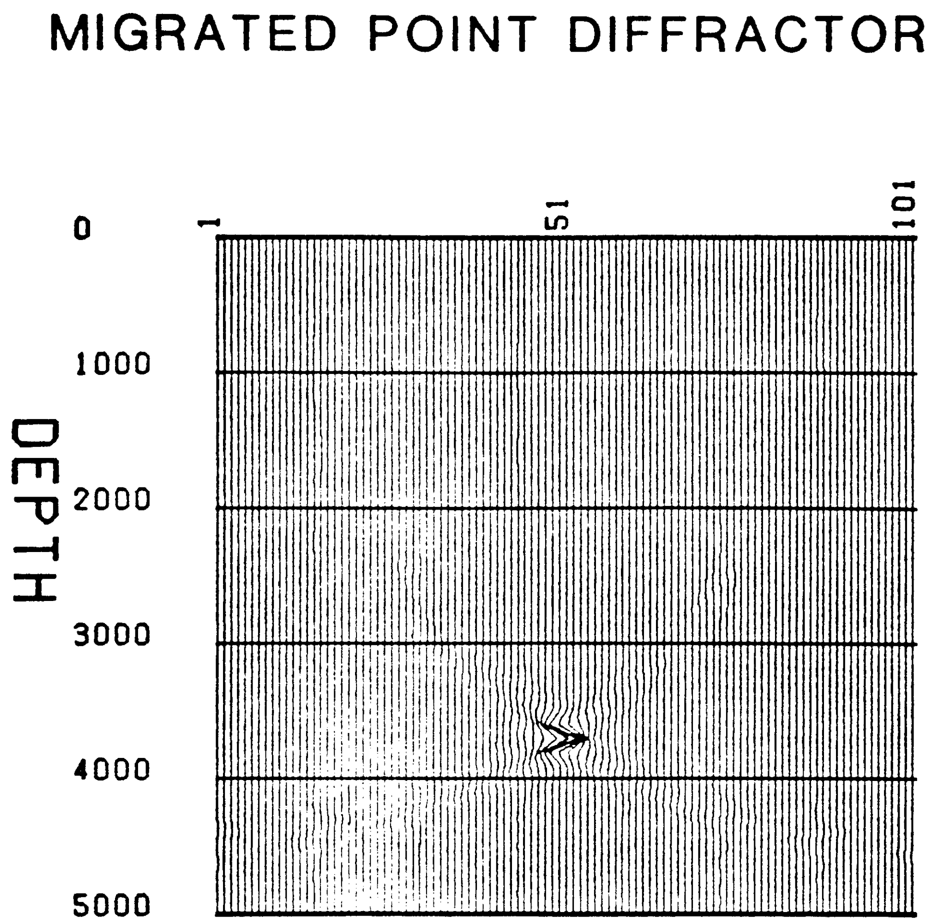


Figure 18. Reverse time migrated point diffractor mapped in depth. Loss of the true point diffractor is due to the "aperture" problem of depth migration.

Forward Problem in a Vertically Layered Medium

To test the effect of lateral velocity variations on the One-way wave equation solution, a forward model is constructed with three equal vertical layers. The velocities from left to right are 8000, 10,000, and 12,000 ft/sec. A point diffractor is buried at a depth of 3800 ft at CDP 51 using a 70 hertz wavelet with a 50 foot spacing in both x and z in a 101 by 101 grid. The forward model with a series of snap shots is shown (see Figures 19 and 20).

Observations of a Point Diffractor in a Vertically Layered Medium

The example clearly shows that horizontal velocity variations do not distort the solution. All the travel times along the wave fronts are correct and again the only loss of frequency is due to the geometrical rotation effect. There is also an interesting effect due to the "two-way" nature of the One-way wave equation in the x direction. The transmitted wave front, the primary wave, travels through the boundaries. At the same time there is also a reflection off the boundaries (see Figures 19 and 20).

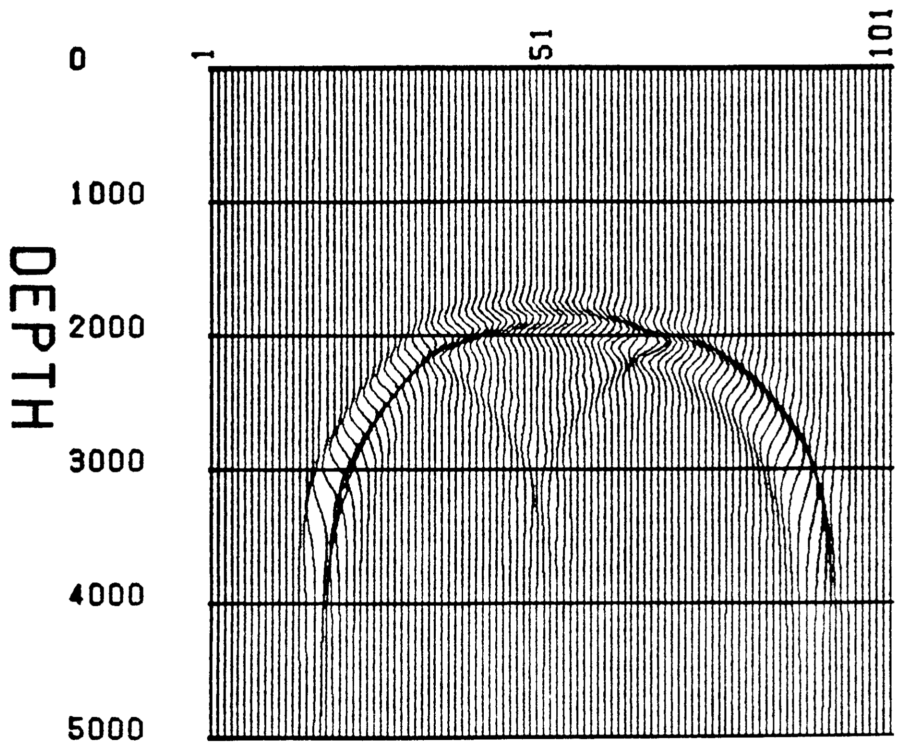


Figure 19. Depth snapshot of the point diffractor in a vertically layered medium.

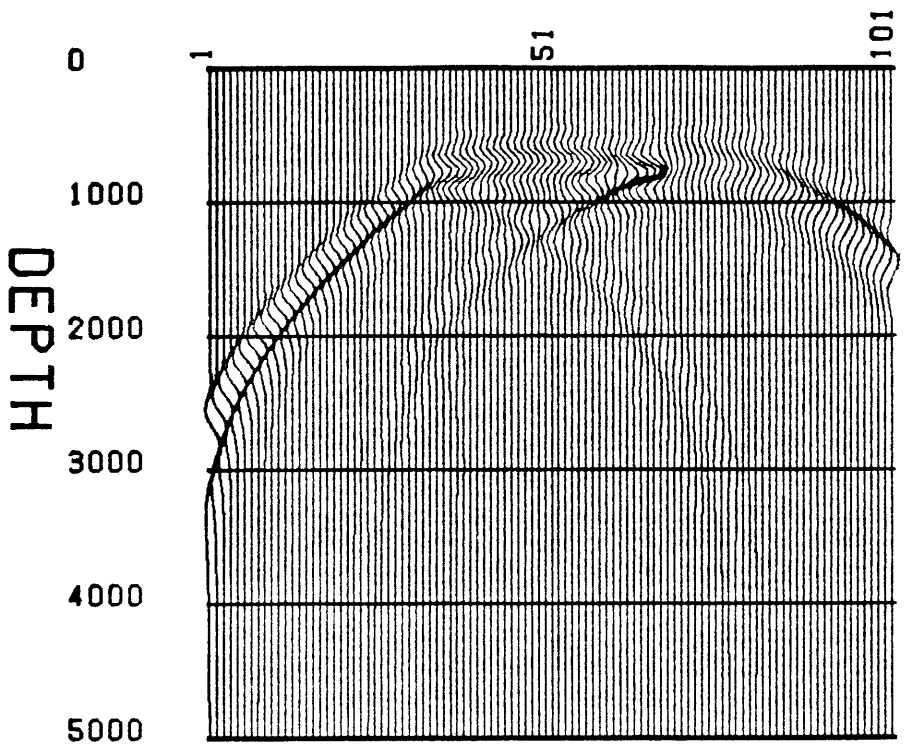


Figure 20. Depth snapshot of the point diffractor in a vertically layered medium. The reflections in the horizontal direction are due to the "two-way" nature of the One-way wave equation in the horizontal direction.

Fault Block Model

Forward and Inverse Problem

The purpose of this model is to determine the ability of the technique to handle dipping events. The fault block model is made up of an overburden of 8000 ft/sec, followed by a 10,000 and a 12,000 ft/sec layer respectively. The flat layers are distorted by a 1000 foot slip at about CDP 51 (see Figure 21). The source wavelet is a Ricker wavelet with a peak in the amplitude spectrum at about 35 Hertz. The boundaries are treated as a series of point diffractors and the source wavelet is added at each diffractor for every time step. The Reverse time migration is done for three separate velocity models. This is done to test Reverse times sensitivity to the depth model. The first model is done with the correct velocity field (see Figure 30). The second and third migrations are done using an anticline (see Figure 33) and a syncline (see Figure 29) models to migrate the fault block.

Observations of the Forward and Inverse Fault Block Model

The wrap around effects which caused problems in the point diffractor models are no longer apparent when full line-like sources are used. This is because much of the wrap around effect is canceled out by the adjacent sources. In the Reverse time migration the fault plane is very well imaged (see Figure 30).

The two examples of migration with incorrect velocity fields show the robustness of the Reverse time migration. The migrated depth model is well defined in both cases. There are slight distortions in the fault plane as well as the underlying flat layer, but from an interpretational viewpoint the fault plane is still very well imaged (see Figures 32 and 34).

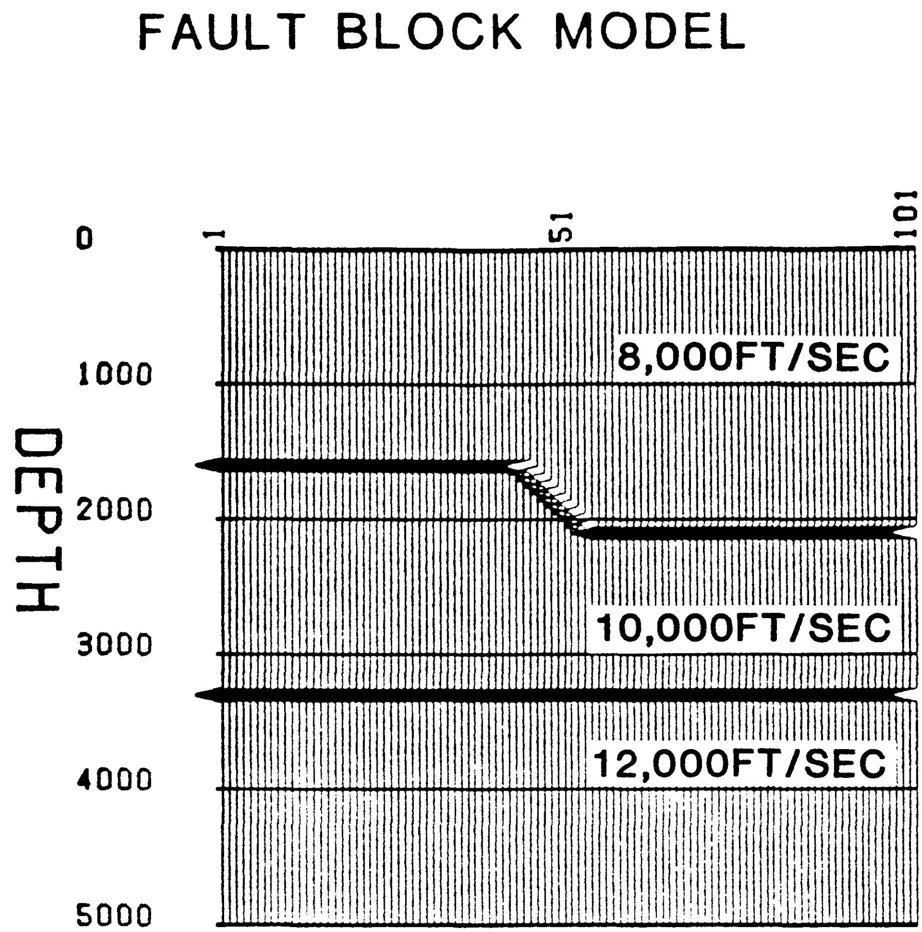


Figure 21. Fault block model used to test the Fourier theoretical technique's ability to handle sharp corners in a model.

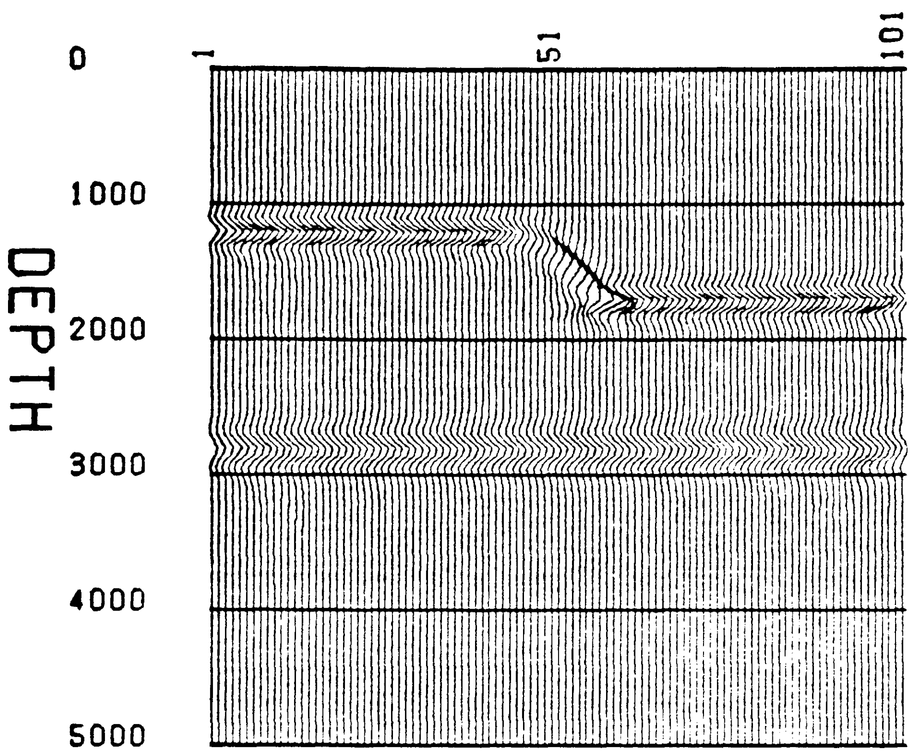


Figure 22. Depth snapshot of the exploding reflector fault block model.

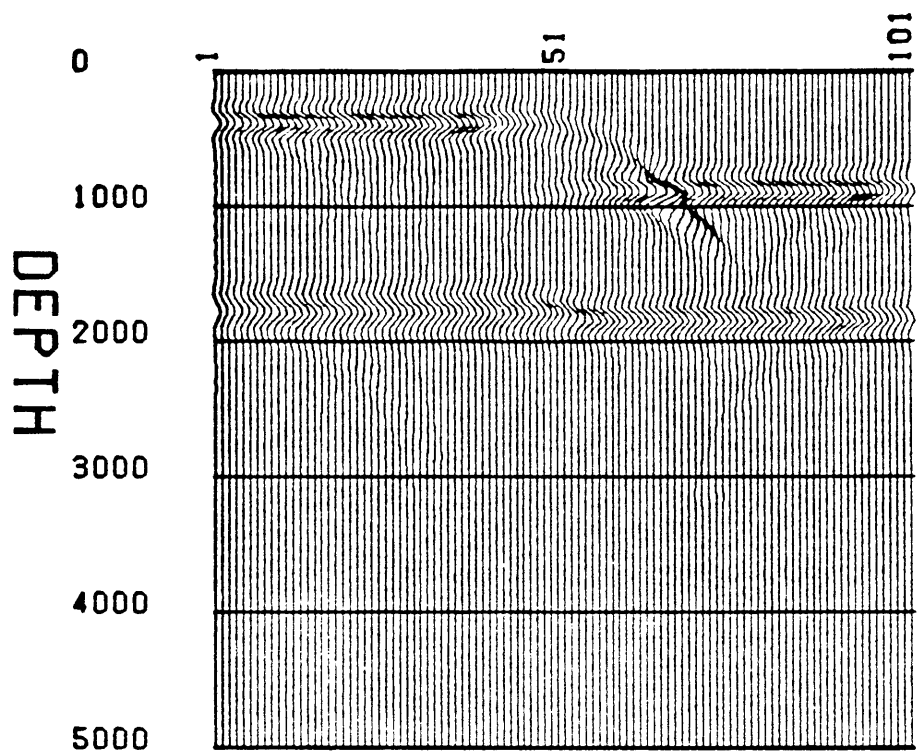


Figure 23. Depth snapshot of the exploding reflector fault block model.

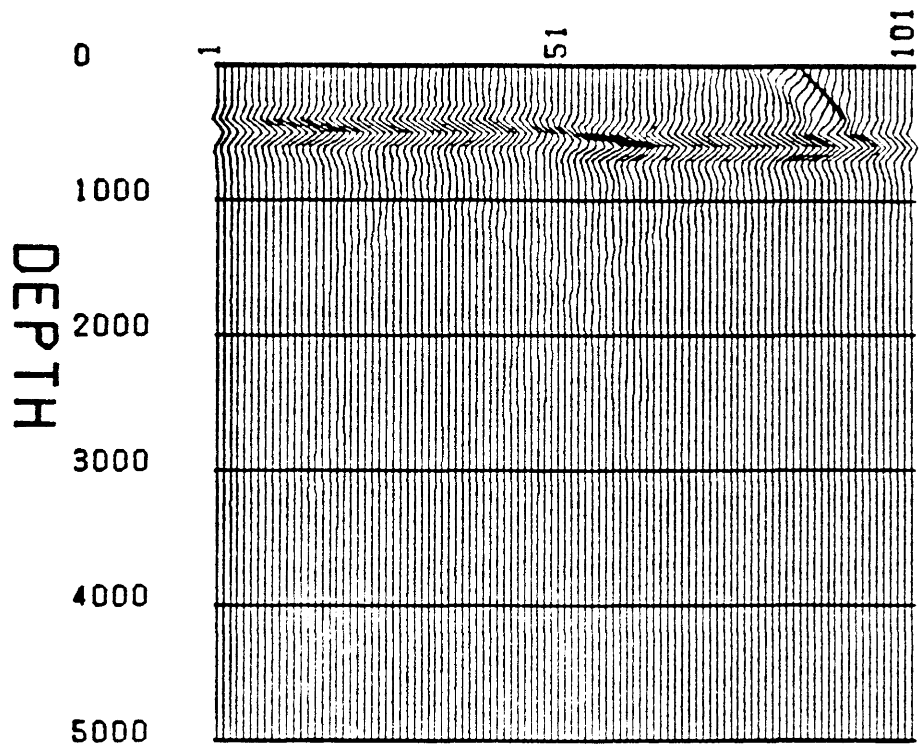


Figure 24. Depth snapshot of the exploding reflector fault block model.

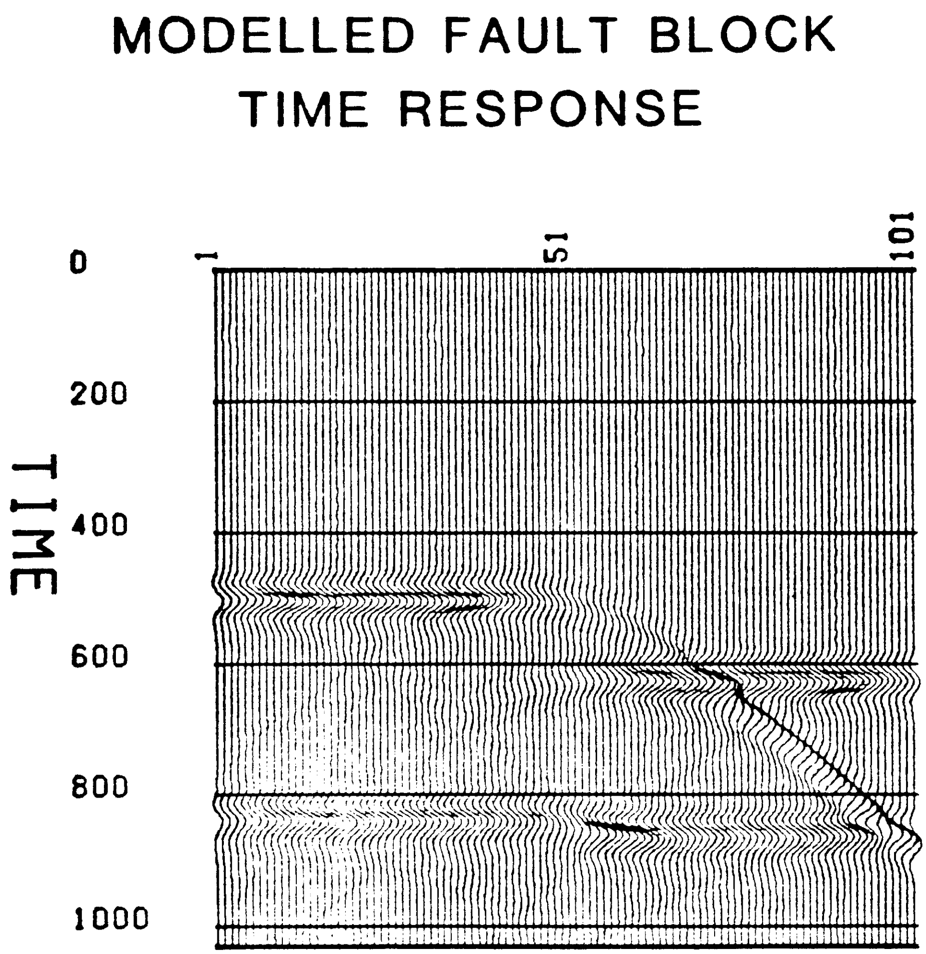


Figure 25. Time section over the fault block model.

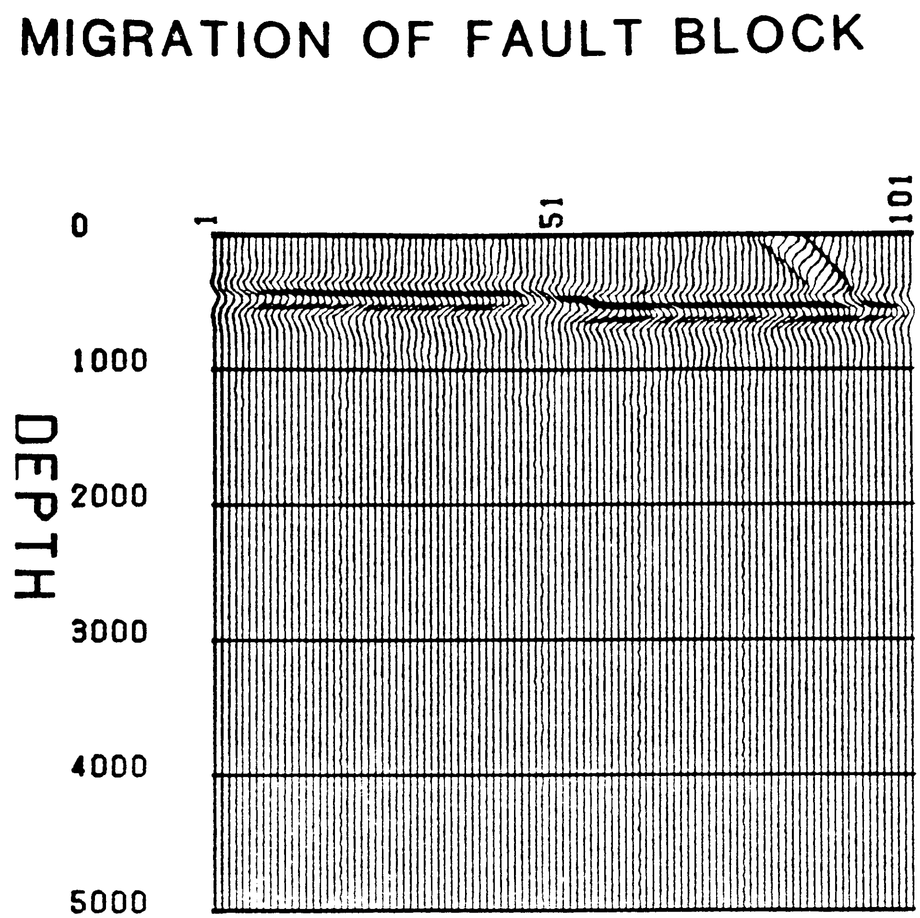


Figure 26. Depth snapshot of collapsing wave fronts during Reverse time migration.

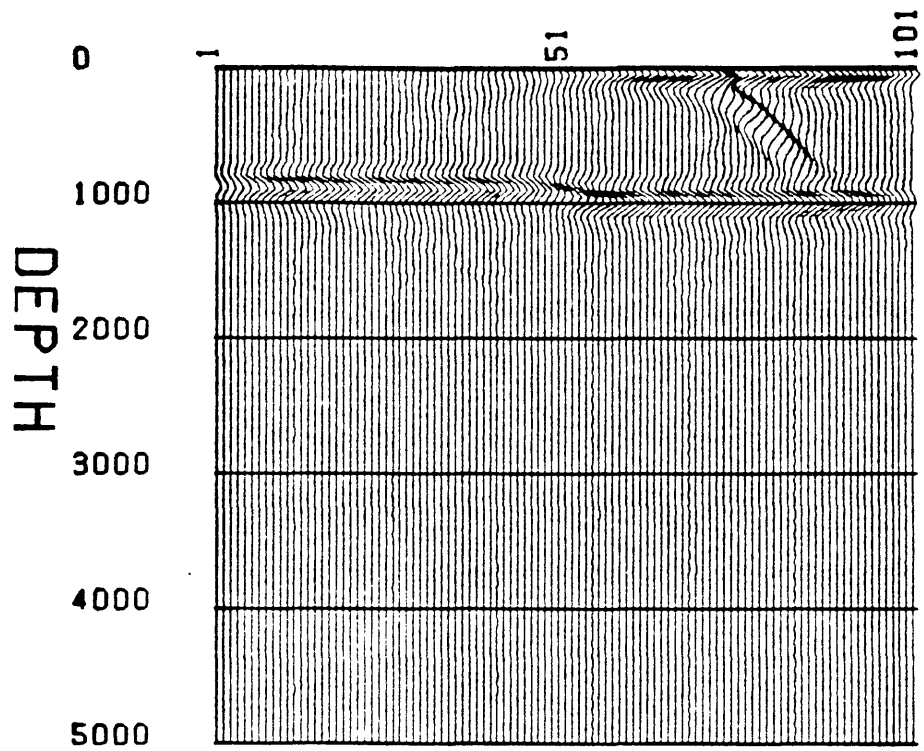


Figure 27. Depth snapshot of collapsing wave fronts during Reverse time migration.

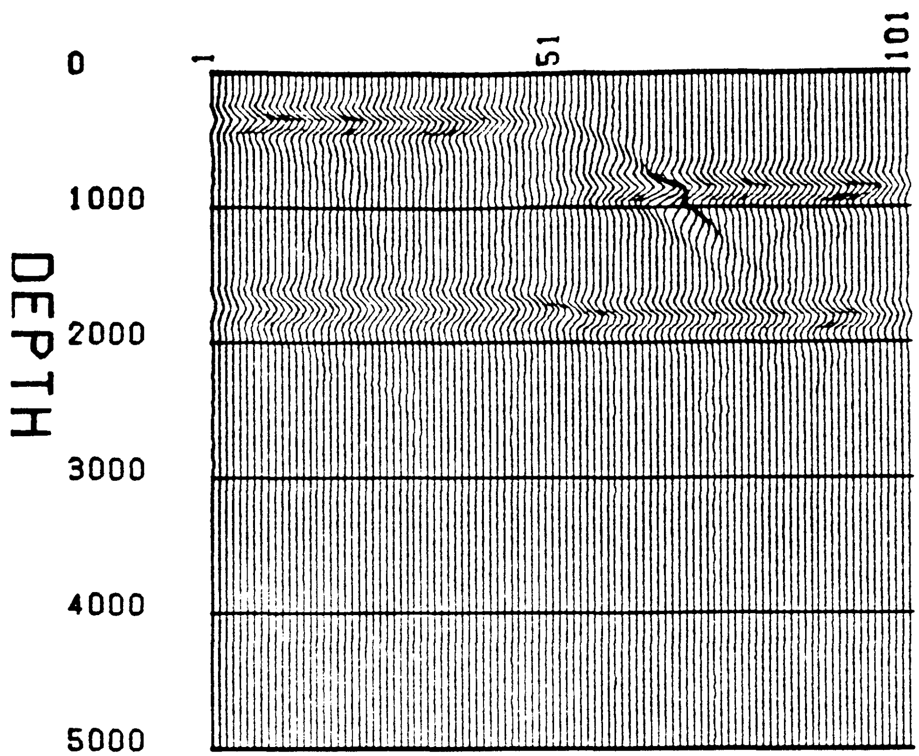


Figure 28. Depth snapshot of collapsing wave fronts during Reverse time migration.

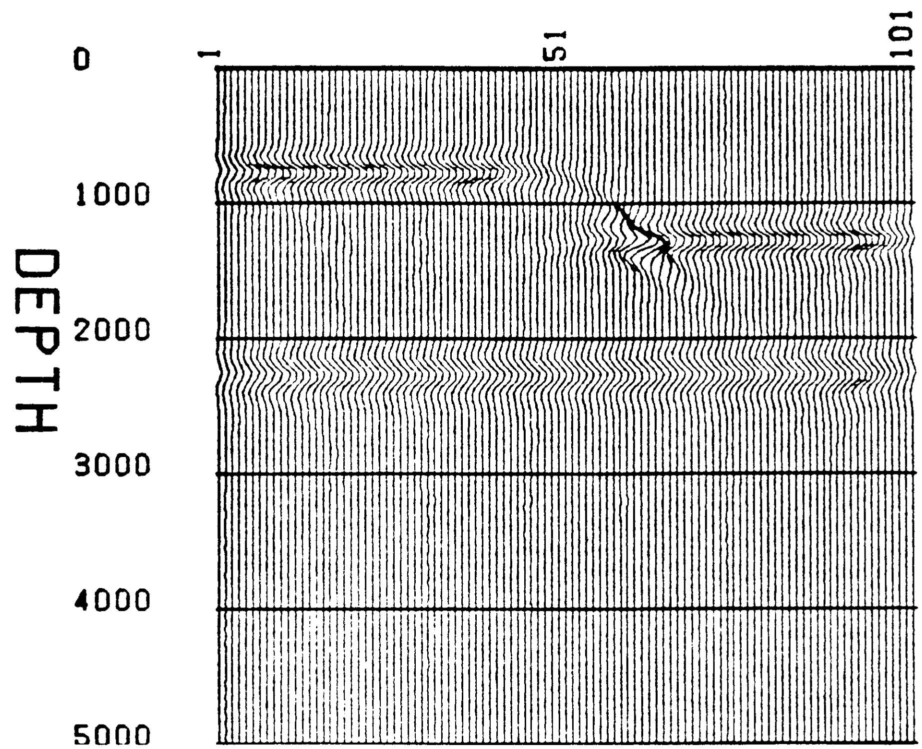


Figure 29. Depth snapshot of collapsing wave fronts during Reverse time migration. These displays can be used to study where migrated events come from in complex geologic structures.

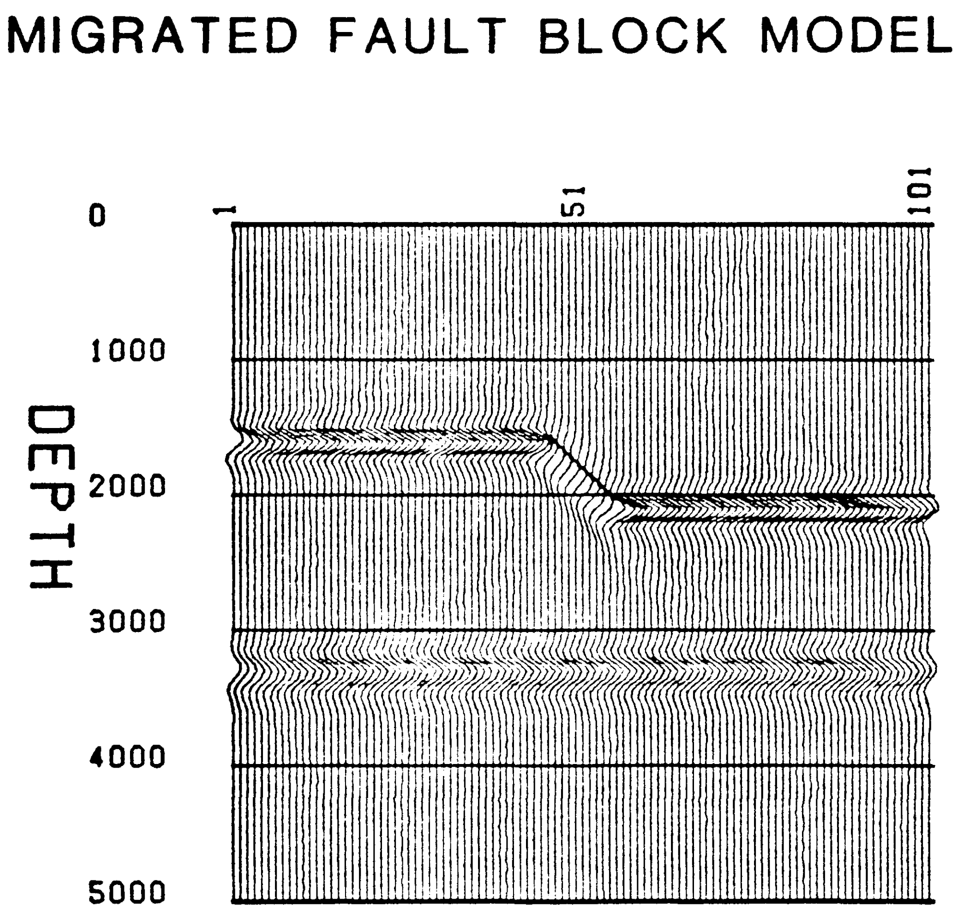


Figure 30. Reverse time migrated fault block done using the exact velocities.

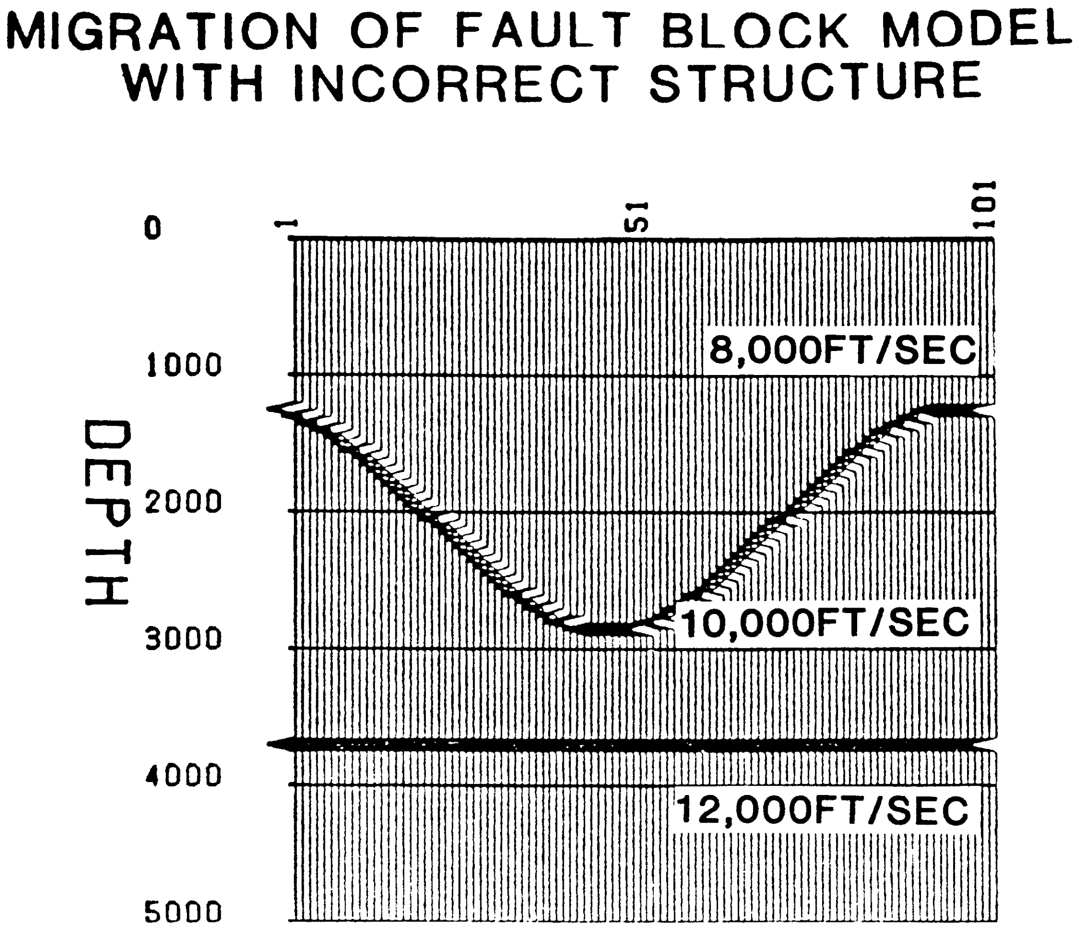


Figure 31. Syncline model used to migrate fault block.

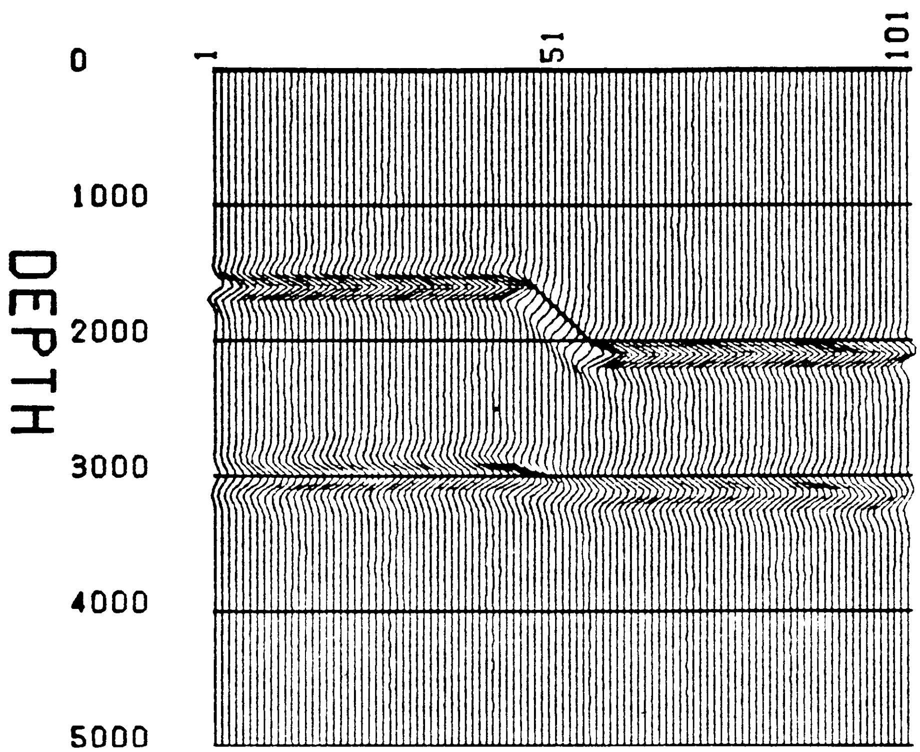


Figure 32. Reverse time migrated fault block done using the syncline model velocities.

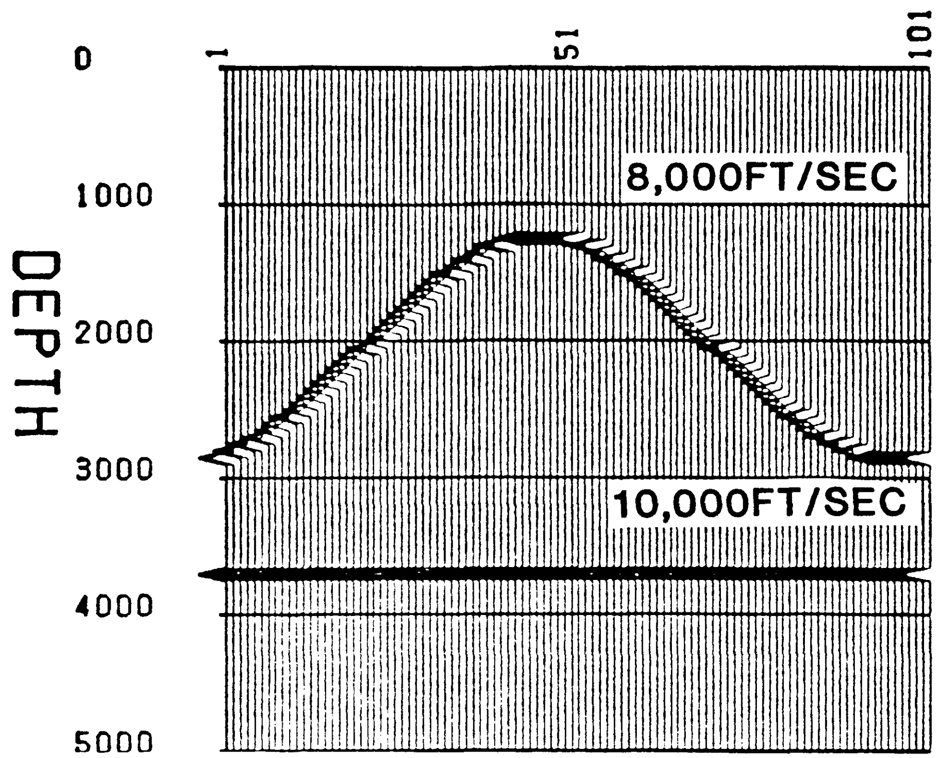


Figure 33. Anticline model used to migrate fault block.

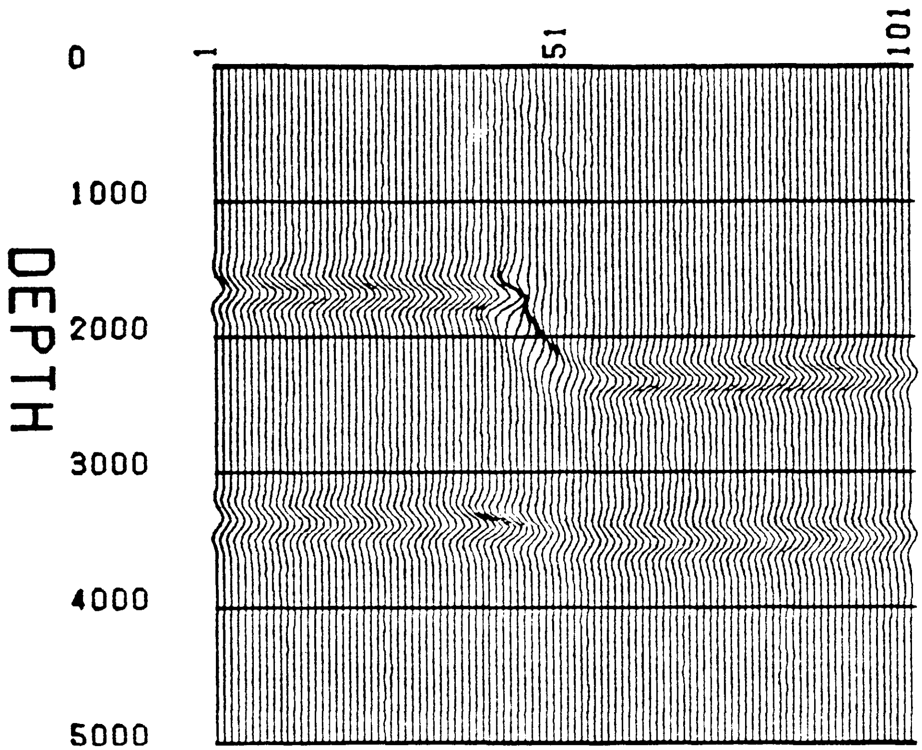


Figure 34. Reverse time migrated fault block done using the anticline model velocities.

Anticlinal Model

Forward and Inverse Problem

The model is an anticline with a flat underlying layer. The purpose of this model is to test the ability of the Fourier theoretical technique to handle a structure which disperses energy. The model consists of an overlying burden of 8000 ft/sec, a 10,000 ft/sec layer, followed by a 12,000 ft/sec "half space". The boundary between the 8 and 10 thousand ft/sec layers makes up the anticline. The anticline is centered about CDP 51 and has a relief of almost 3000 feet (see Figure 35). The migrations done are to show a possible "aperture" problem in depth migration (see Figures 42 and 43). Also there is an example of migration with the incorrect velocity field (see Figure 44).

Observations of the Forward and Inverse Anticlinal Model

In the two migration examples using the correct velocity, the first example has an "aperture" of 101 traces (see Figure 42). The anticline is not properly reconstructed in this case, because of the dispersive nature of an anticline. The energy of the reflections was reflected beyond the 101 trace window, therefore making it impossible to reconstruct the depth model. The second example was done on a 512 trace window (see Figure 43). In this case the anticline is properly reconstructed. The two examples demonstrate that there is an "aperture-like" consideration which needs to be taken into account. The third migration example is the anticline migrated with flat layer velocities (see Figure 44). The anticline structure is correctly reconstructed, but the underlying flat layer is distorted.

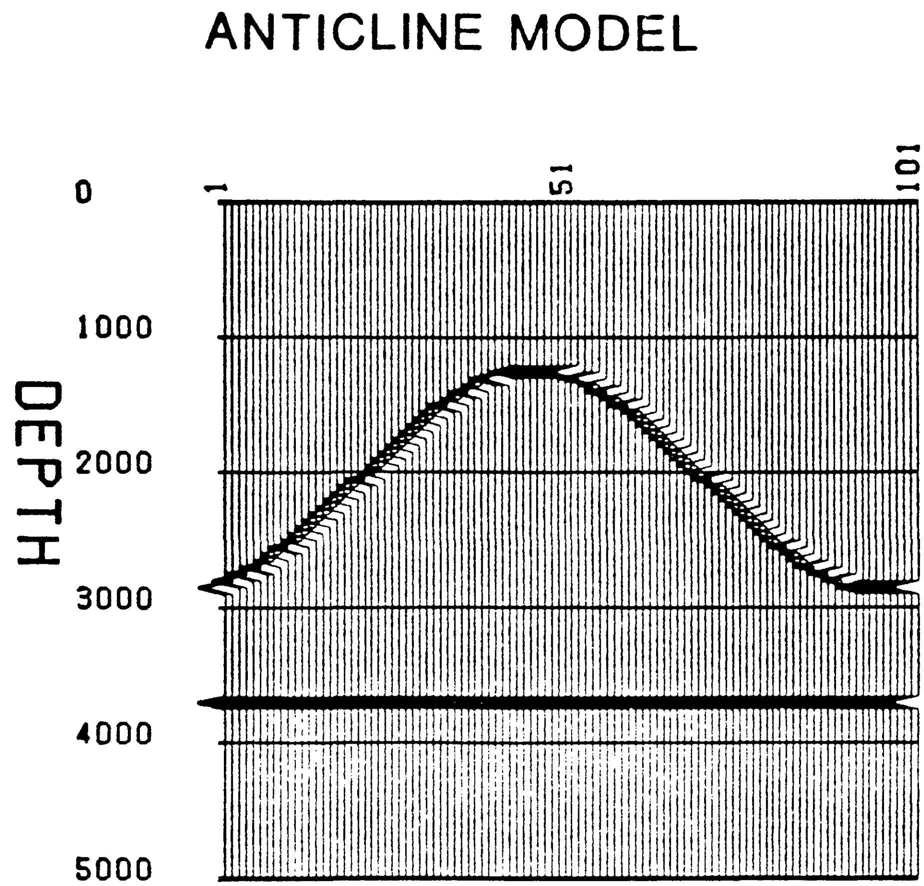


Figure 35. Anticline model used to test the Fourier theoretical technique's ability to handle a model which disperses energy.

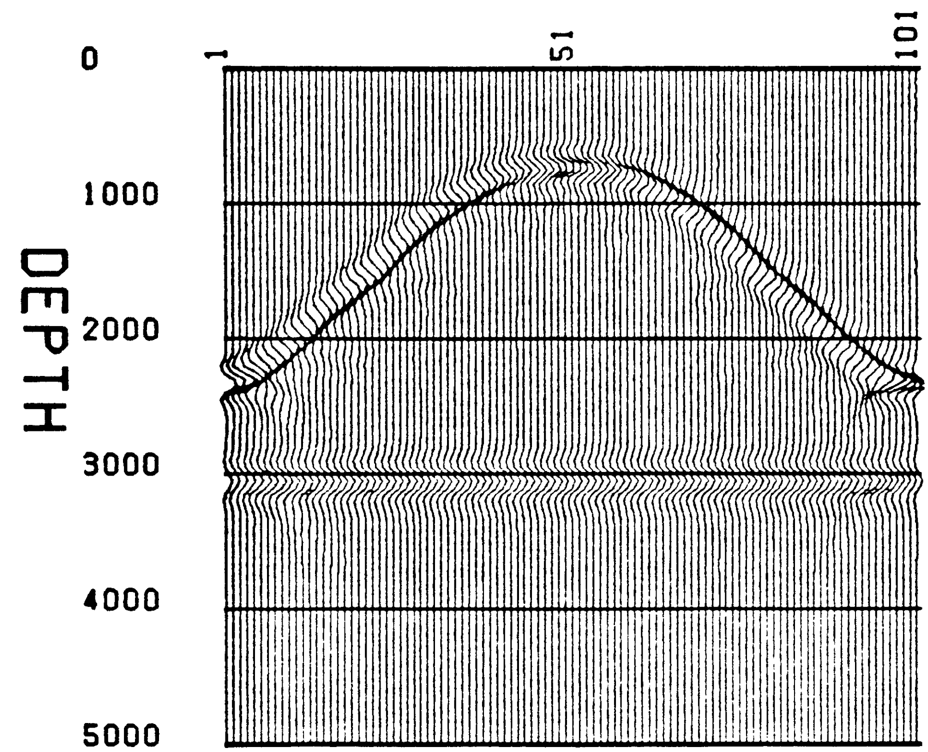


Figure 36. Depth snapshot of the exploding reflector anticline model.

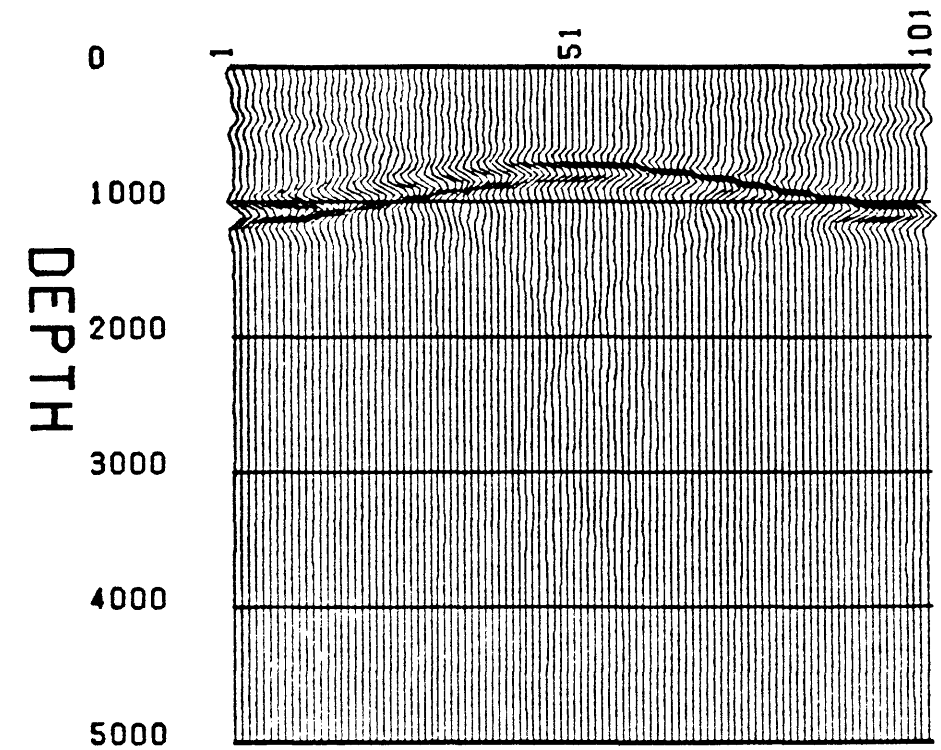


Figure 37. Depth snapshot of the exploding reflector anticline model.

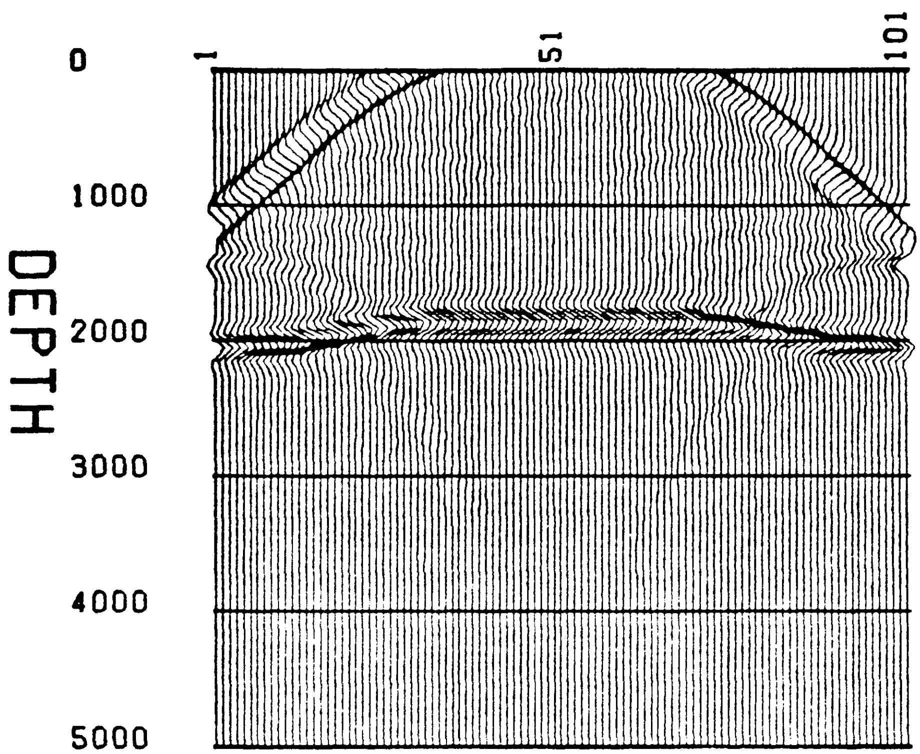


Figure 38. Depth snapshot of the exploding reflector anticline model.

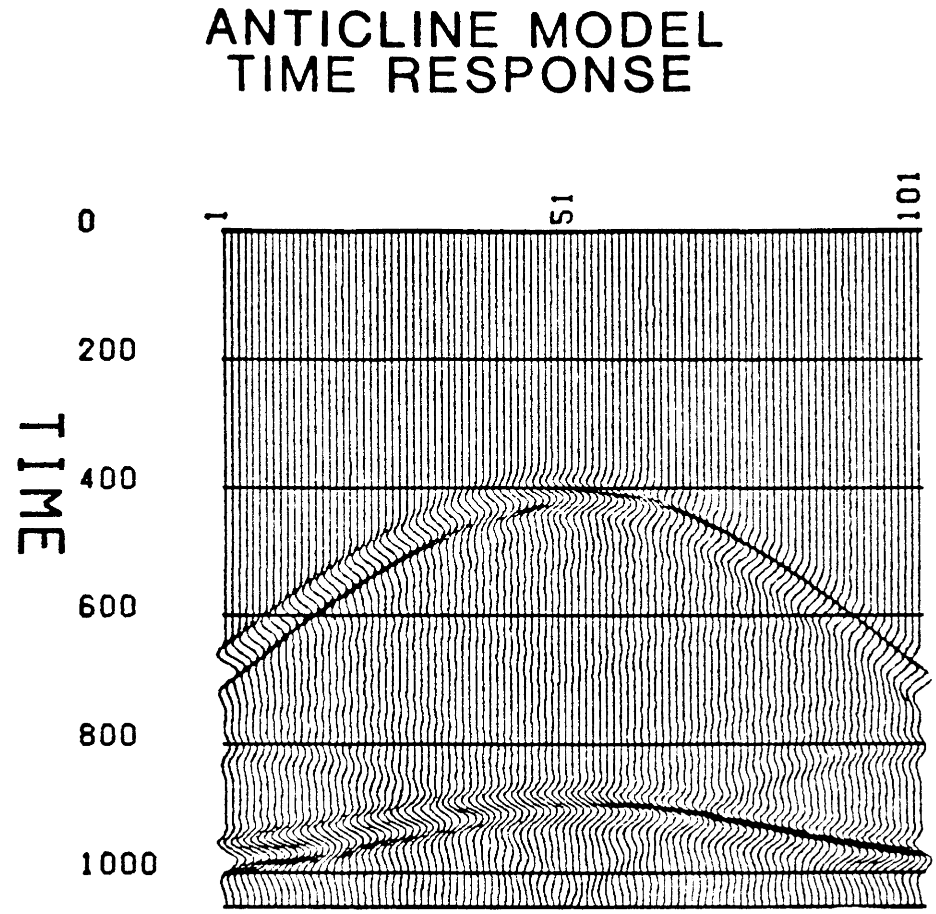


Figure 39. Time section over the anticline.

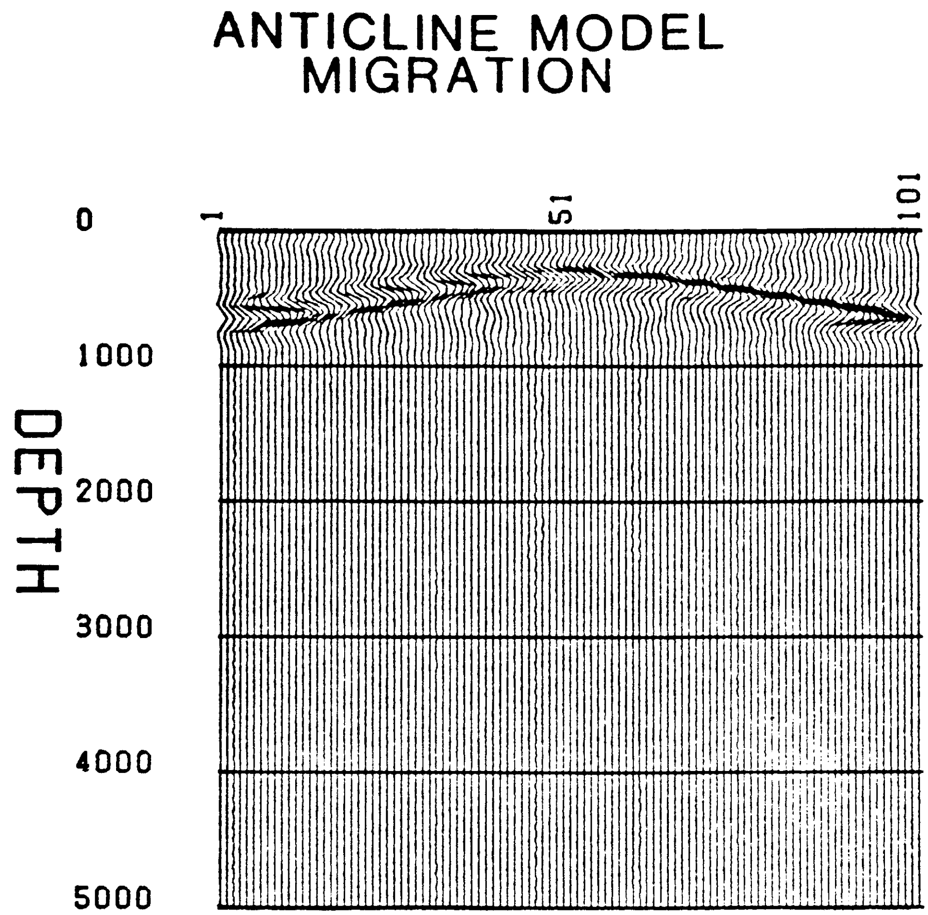


Figure 40. Depth snapshot of collapsing wave fronts during Reverse time migration.

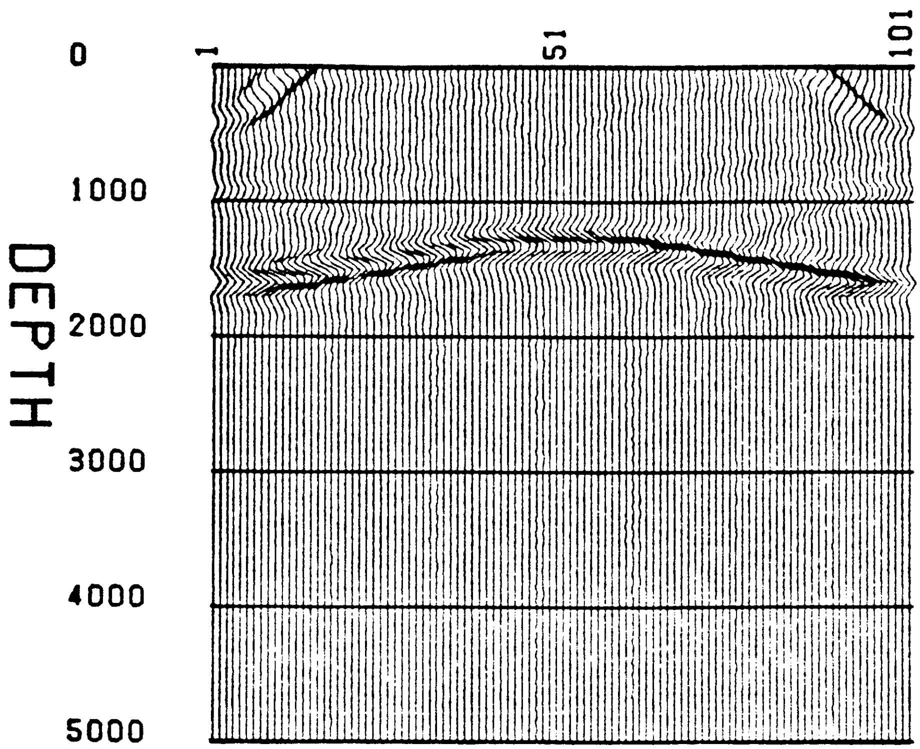


Figure 41. Depth snapshot of collapsing wave fronts during Reverse time migration.

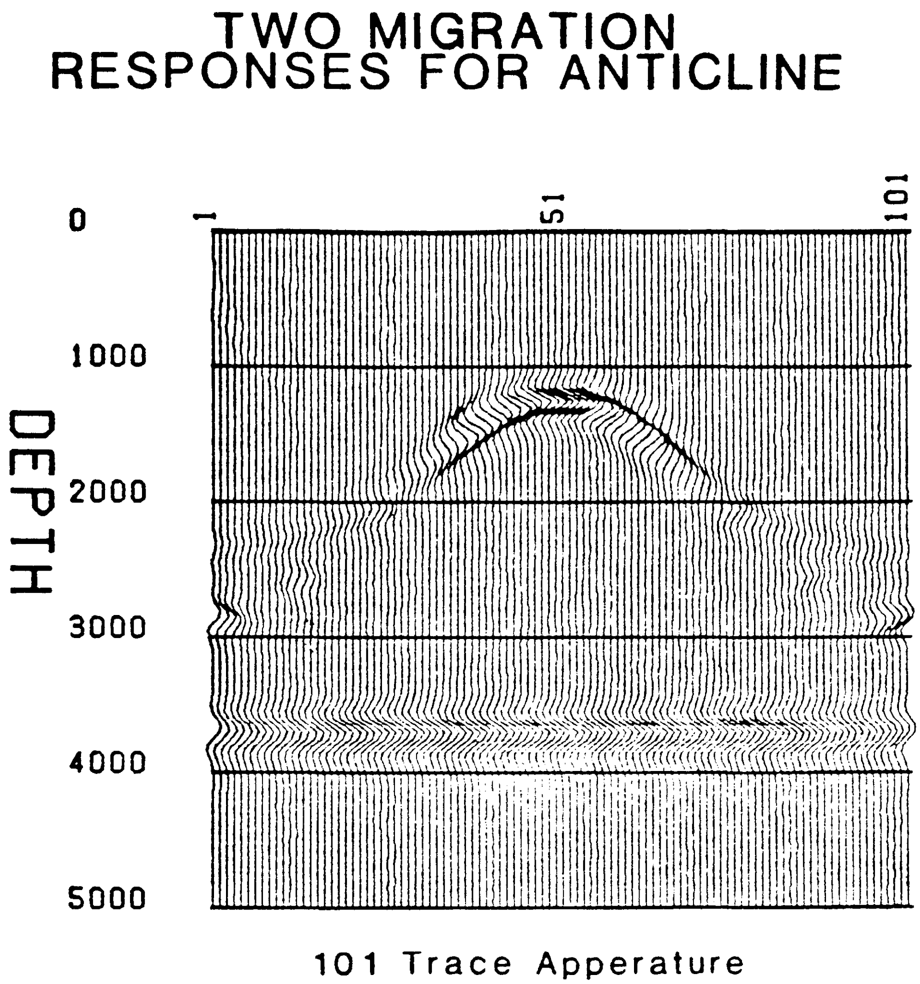


Figure 42. Reverse time migrated response of the anticline model for a 101 trace (5000 ft) aperture.

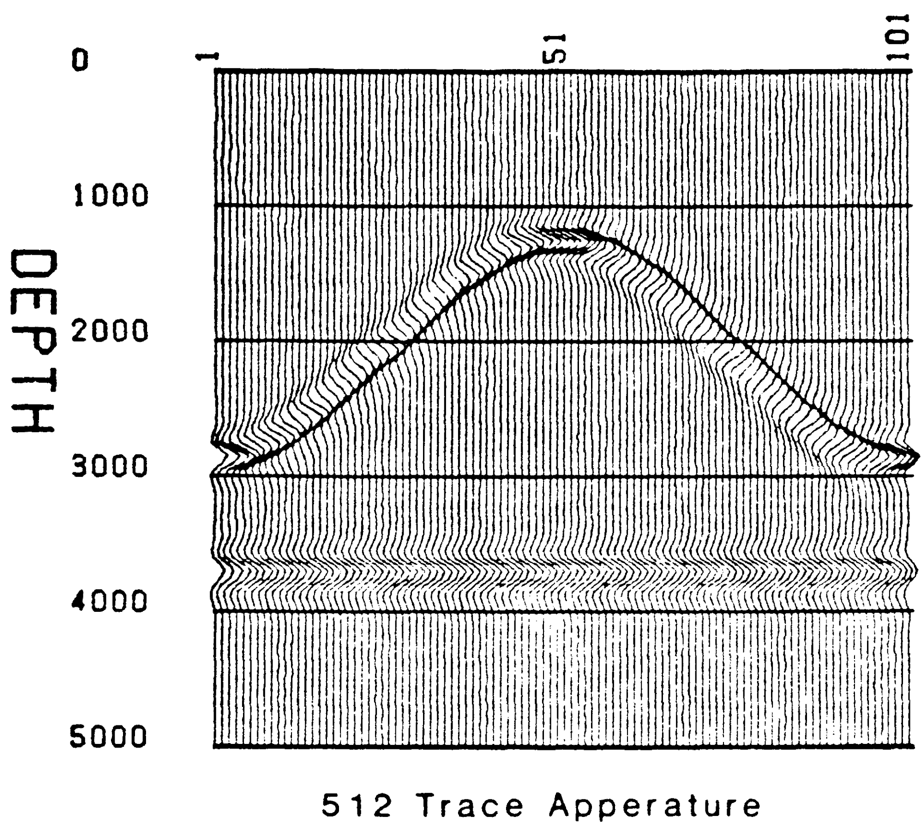
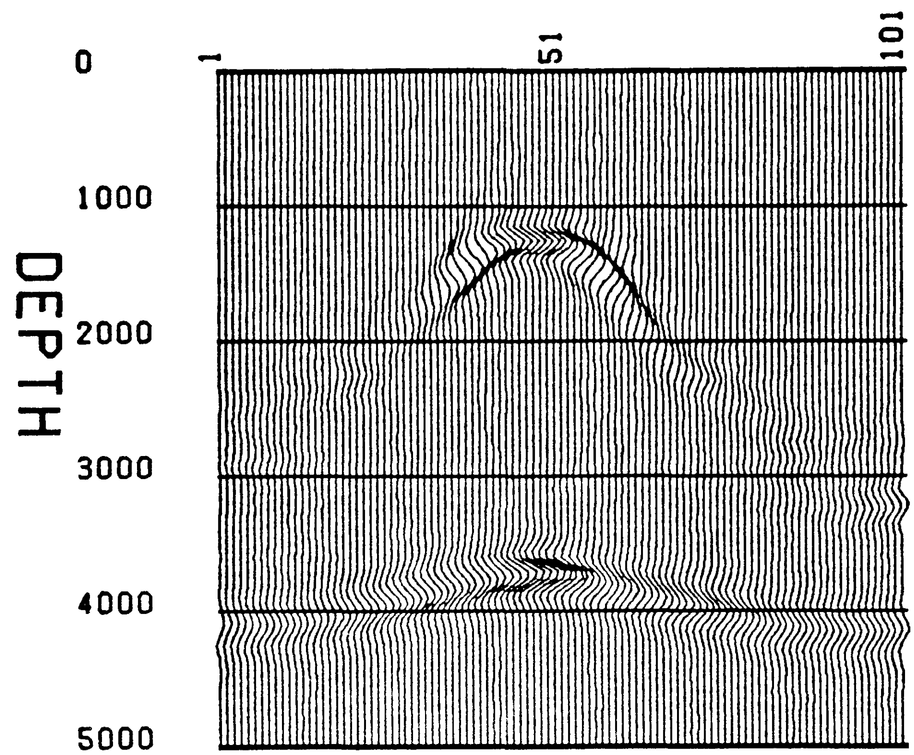


Figure 43. Reverse time migrated response of the anticline model for a 512 trace (25,600 ft) aperture.



ANTICLINE MODEL MIGRATED WITH
FLAT LAYER VELOCITIES

1 0-1300 ft =8000ft/sec

2 1300-3700ft =10000ft/sec

3 12000ft/sec

Figure 44. Reverse time migrated response of the anticline model for flat layer velocities.

Synclinal Model

Forward and Inverse Problem

The model is a syncline with a flat underlying layer. The purpose of this model is to test the ability of the Fourier theoretical technique to handle a structure which focuses energy. The layer consists of an overlying burden of 8000 ft/sec. The next layer is 10,000 ft/sec, followed by a 12,000 ft/sec "half space". The boundary between the 8 and 10 thousand ft/sec layers makes up the syncline. The syncline is centered about CDP 51 and has a relief of 3000 feet (see Figure 45). The migration done is to show the ability of Reverse time migration to properly reconstruct a buried focus (see Figure 52). Also there is again an example of migration with the incorrect velocity field (see Figure 53).

Observations of the Forward and Inverse Synclinal Model

The migration handles the "bow tie diffractions" correctly, moving all events back to their proper location. The second migration example is the syncline migrated with flat layer velocities. The syncline structure is correctly reconstructed, but the underlying flat layer is distorted.

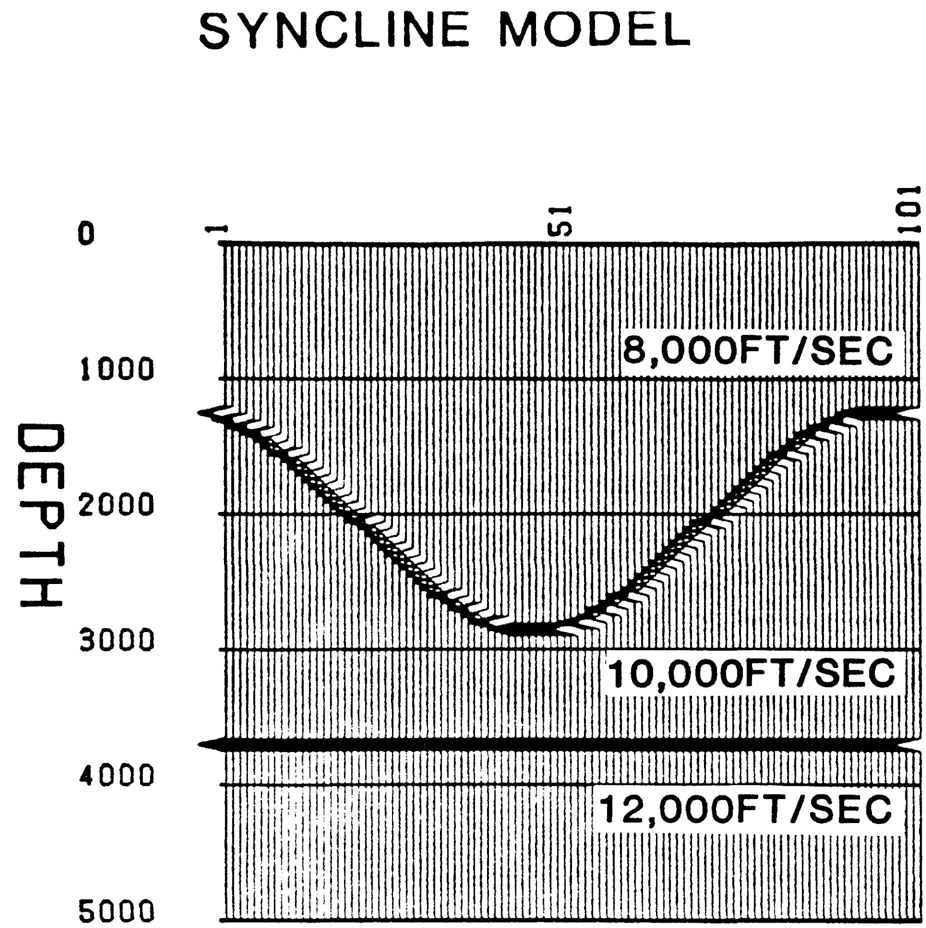


Figure 45. Syncline model used to test the Fourier theoretical technique's ability to handle a model which focuses energy.

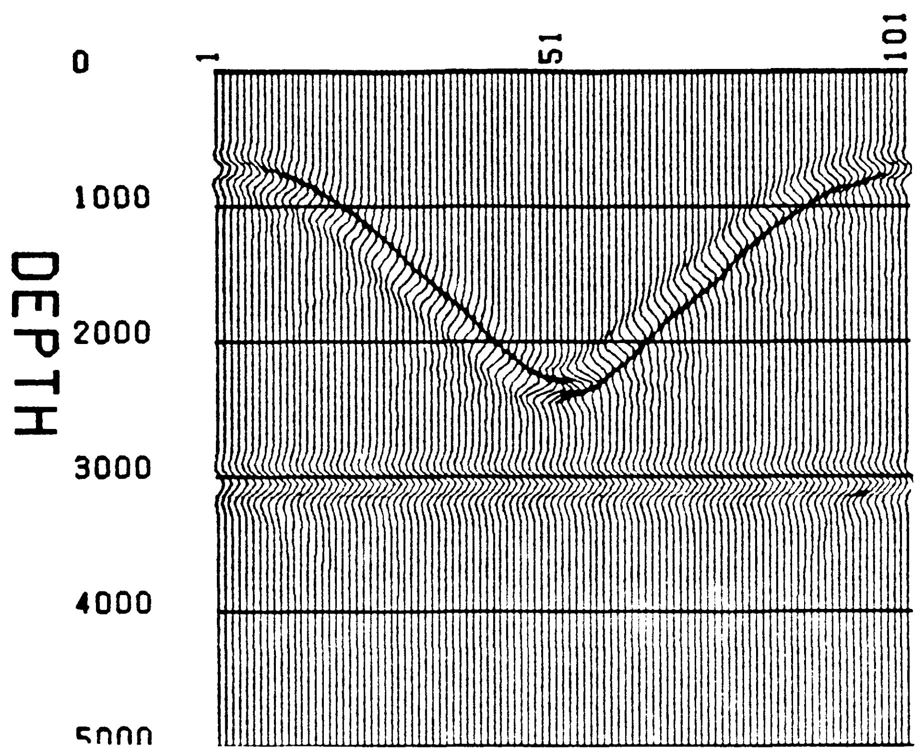


Figure 46. Depth snapshot of the exploding reflector syncline model.

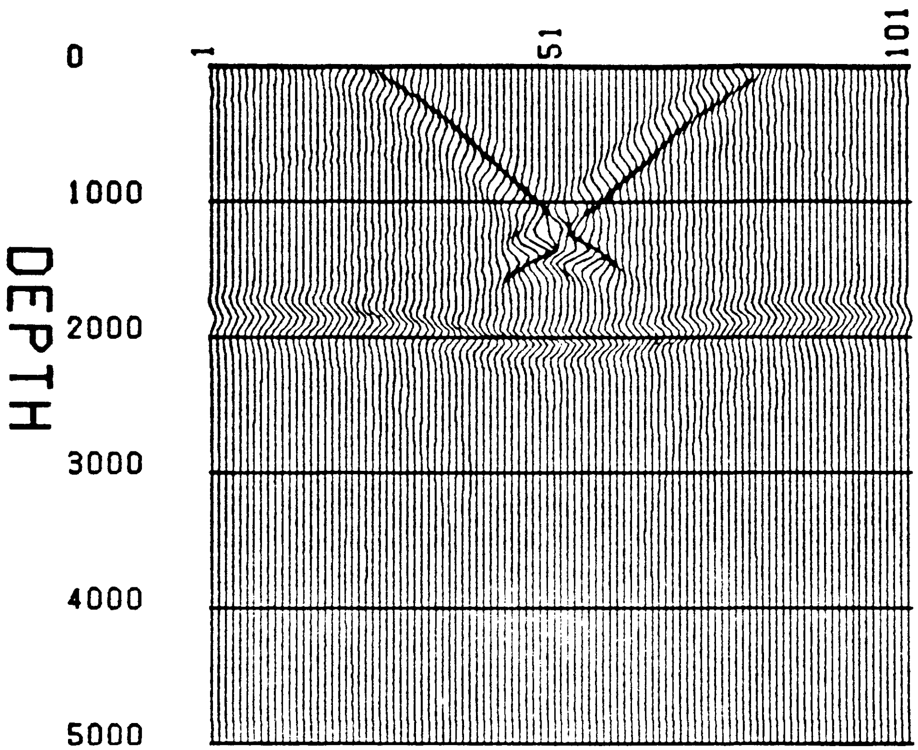


Figure 47. Depth snapshot of the exploding reflector syncline model.

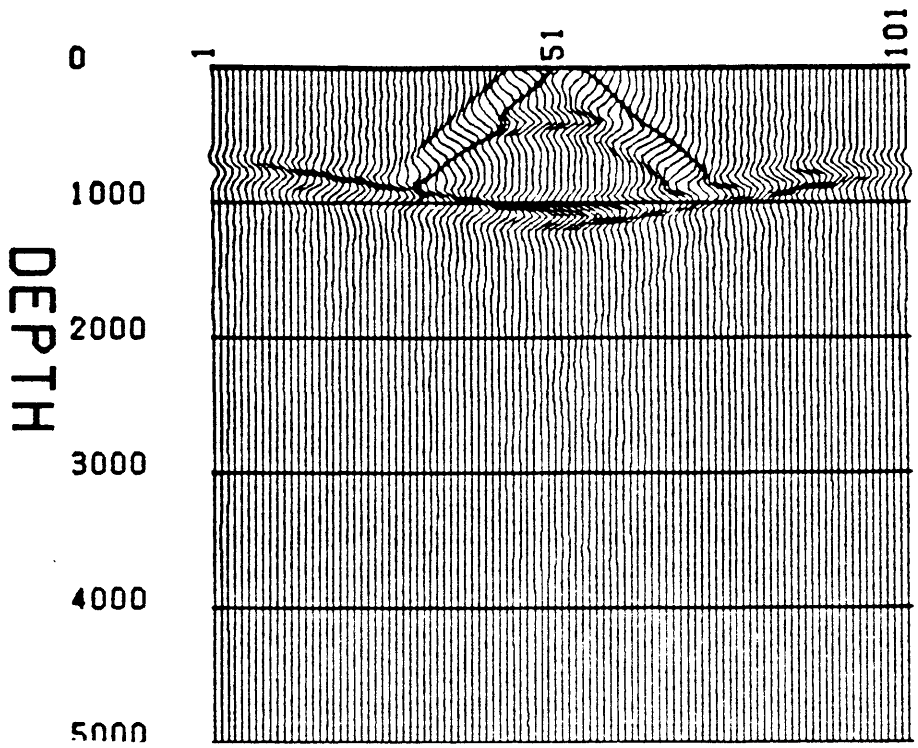


Figure 48. Depth snapshot of the exploding reflector syncline model.

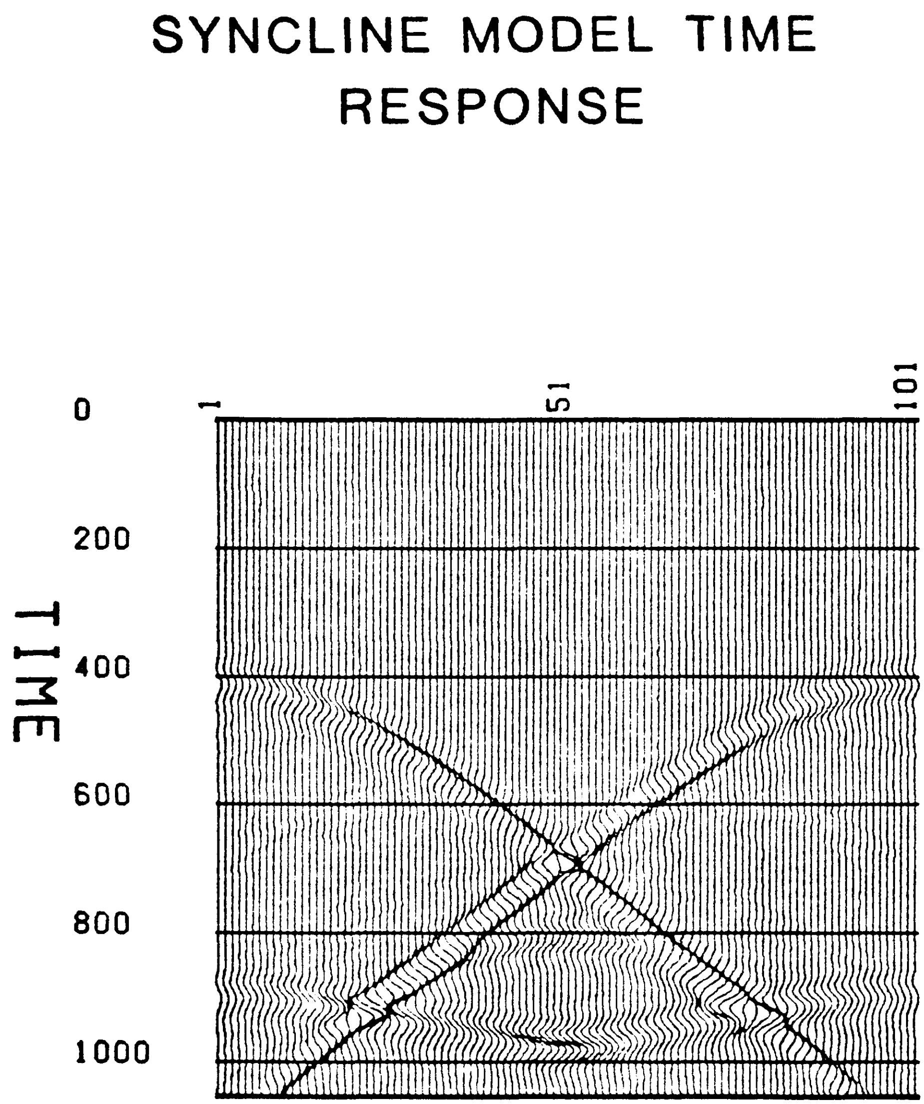


Figure 49. Time section over syncline.

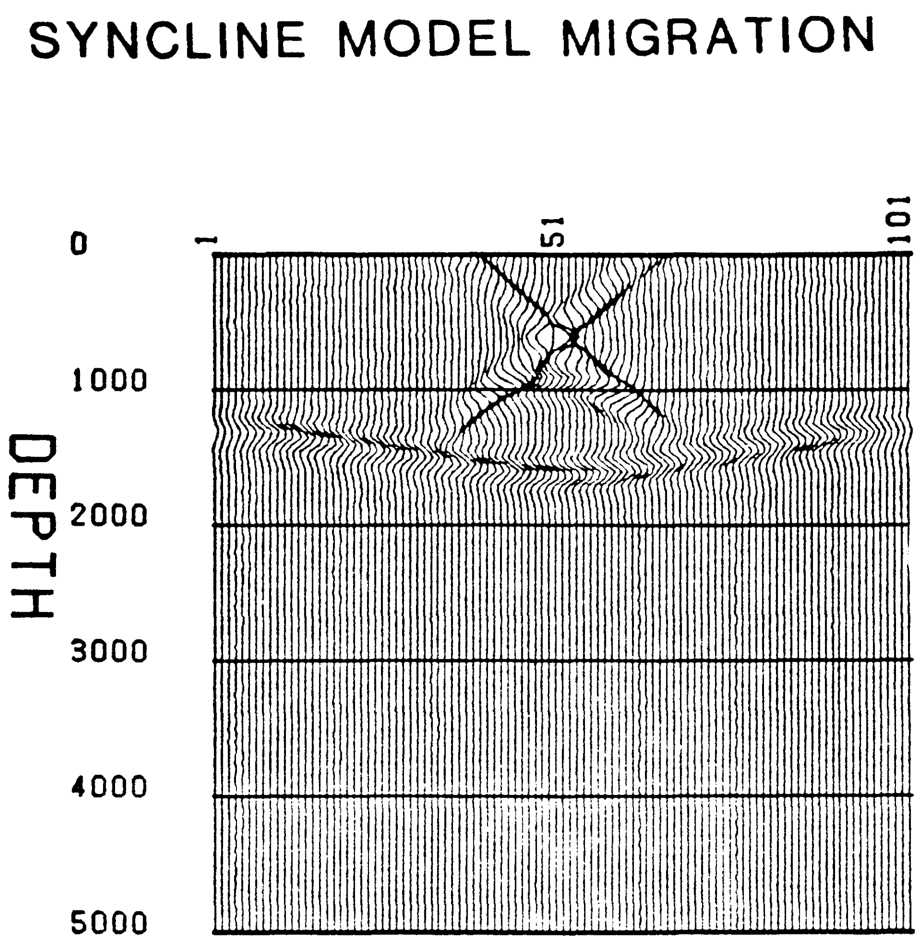


Figure 50. Depth snapshot of collapsing wave fronts during Reverse time migration.

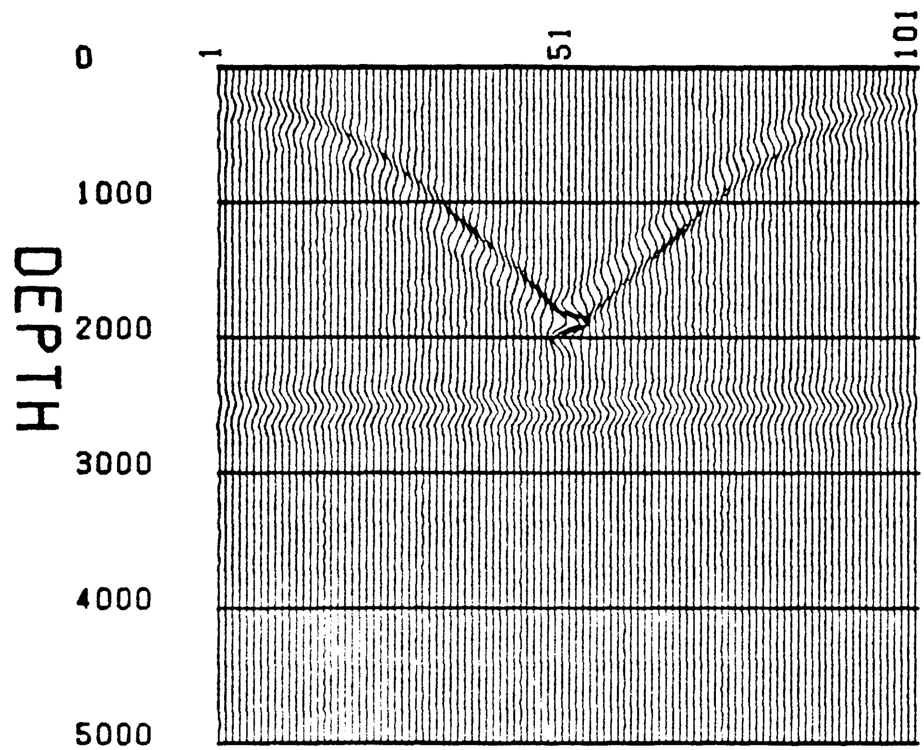


Figure 51. Depth snapshot of collapsing wave fronts during Reverse time migration.

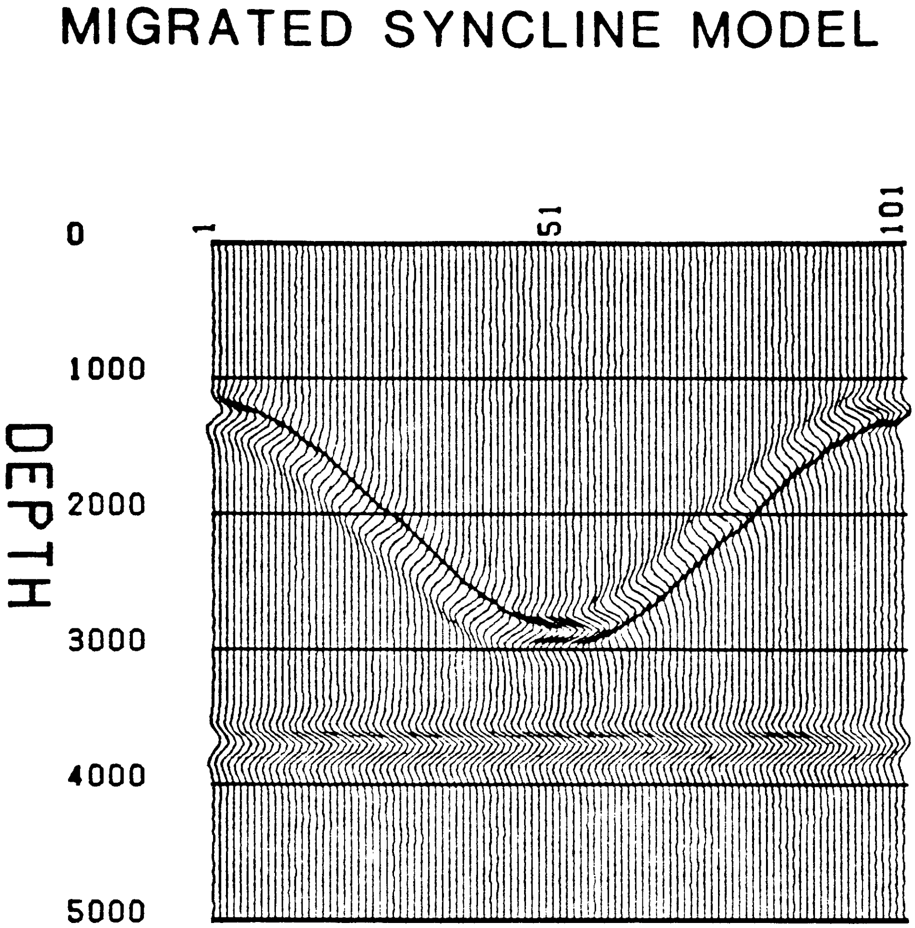


Figure 52. Reverse time migrated response of the syncline model using the exact velocities.

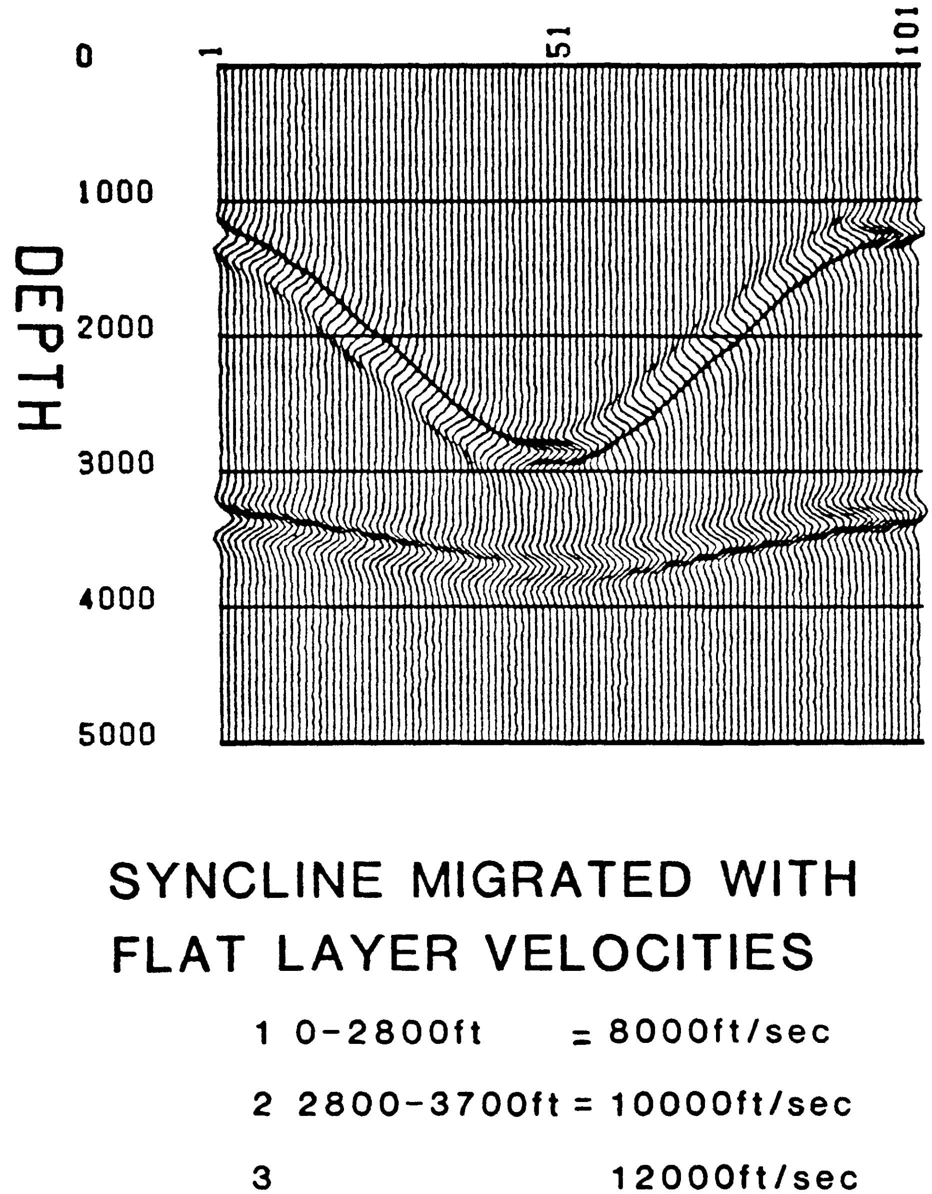


Figure 53. Reverse time migrated response of the syncline model for flat layer velocities.

General Observations

On all the models there is an edge effect which has not been removed. Diffractions propagate off the edges of the exploding reflector. This is indirectly related to the Fourier theoretical method. While the earth is defined over a limited range of data, it has to be placed within a grid which has a power of two-grid spacing to accommodate the two-dimensional FFT. Because of this, the exploding reflectors terminate at the edge of the "known" earth. But, in fact, the earth has to extend beyond that. The edges of the reflectors in turn set up the diffractions. The way to solve this problem would be to taper the reflectors smoothly at the edges.

CONCLUSIONS

The results from the previous section demonstrate the potential of the Fourier theoretical technique. It allows the handling of much steeper events with little dispersion. The only difference between a conventional finite difference scheme and the Fourier theoretical technique is the way in which the spatial derivatives are handled. By going into the Fourier domain and implementing the derivatives, there is little error in the spatial approximation of the wave equation. The primary source of error arises from the finite difference approximation in time. Contrary to normal finite difference in schemes, numerically the Fourier theoretical technique is easy to apply to a wave equation, and it will give better results for larger spatial sample rates because of small numerical dispersion. While the Fourier technique may not seem cost effective for two-dimensional models (X,Z), the results are better and there are no problems of spatial aliasing due to the larger spatial sample rates. In the future the Fourier theoretical technique will be applied to three-dimensional models (X,Y,Z), and because of its independency of spatial sampling, both the time of computation and the memory savings will be evident.

In the forward modeling problem, it seems that while the One-way wave equation gives adequate solutions, there are numerical problems which can arise. The wrap around problem can be minimized by using larger FFT buffers, but this can cause application problems. However, the Two-way non-reflecting wave equation shows some promising results (Baysal, 1984). While not fully implemented at the time of this thesis, enough results and theoretical studies have been analyzed to demonstrate its potential.

In the migration problem through the use of Reverse time migration, the One-way wave equation does give very interesting results. The wrap around problem does not seem to effect the solution. This is because in the case of migration the surface is entirely defined for all time and this form of a boundary condition impedes the wrap around from taking effect. It is important to see the difference between conventional depth and Reverse time migration. Conceptually they are very similar, yet depth migration will have more dispersion at high dips than reverse time for the same equation. Depth migration extrapolates the depth response at $t = 0$ for one z at a time. On the other hand reverse time migration reconstructs and carries along the entire depth wave field for all time.

It is this reconstruction in space and extrapolation in time that make Reverse time migration much more powerful. If the correct form of the wave equation is used there should be no loss of frequency with dip. Like a depth migration, Reverse time migration depends on the velocity model.

The results obtained from the Fourier theoretical technique demonstrate that the method has great potential. With further studies in the Two-way non-reflecting wave equation, it is possible that post stack modeling and migration will be taken the farthest they can go. At this point the solution could approach the solution of some of the prestack migration processes. It is also possible to take the idea of the hybrid Two-way non-reflecting wave equation into prestack. By considering the effects over receivers and shots separately, it could be possible to use a modified form of the hybrid schemes in the prestack domain.

REFERENCES CITED

Aki, K. and Richards, P., 1980, Quantitative seismology: v. 1, W.H. Freeman Co.

Alford, R.M., Kelly, K.R., and Boore, D.M., 1974, Accuracy of finite-difference modeling of the acoustic wave equation: Geophysics, v. 39, p. 834-842.

Baysal, E., Kosloff, D.D., and Sherwood, J.W.C., 1983, Reverse time migration: Geophysics, v. 48, p. 132-141.

Baysal, E., Kosloof, D.D., and Sherwood, J.W.C., 1984, A Two-way non-reflecting wave equation: Geophysics, v. 49, p. 132-141.

Berkhout, A.J., 1982, Seismic migration imaging of acoustic energy by wavefield extrapolation: Elsevier Co.

Bracewell, R.N., 1978, The Fourier transform and its applications: McGraw Hill Co.

Bullen, K.E., 1963, An introduction to the theory of seismology: Cambridge University Press.

Claerbout, J.H., 1976, Fundamentals of geophysical data processing: McGraw Hill Co.

Claerbout, J.H., 1982, Imaging the earth's interior: Stanford Exploration Project, SEP-30.

Emerman, S.H., Schmidt, W., and Stephen, R.A., 1982, An implicit finite-difference formulation of the elastic wave equation: Geophysics, v. 47, p. 1521-1526.

Gazdag, J., 1981, Modeling of the acoustic wave equation with transform methods: Geophysics, v. 46, p.854-859.

Kosloff, D.D., and Baysal, E., 1982, Forward modeling by a Fourier method: Geophysics, v. 47, p. 1502-1412.

Kosloff, D.D., and Baysal, E., 1983, Migration with the full acoustic wave equation: Geophysics, v. 48, p. 677-687.

Papoulis, A., 1977, Signal analysis: McGraw Hill Co.

Seismic Acoustic Laboratory Report, 1983, Fourier theoretical modeling and migration: University of Houston.

Sheriff, R.E., 1973, Encyclopedia Dictionary of Exloration Geophysics: Tulsa, SEG.

Stolt, R., 1978, Migration by Fourier transform: Geophysics, v. 43, p. 23-48.

White, J.., 1983, Applied Seismic Waves; preprint of textbook.

APPENDIX A

DERIVATION OF THE ACOUSTIC WAVE EQUATION

Since all the methods described in this thesis make use of the acoustic wave equation, it would be appropriate to understand how it was derived in order to know its limitations. This derivation is taken from Berkhout (1982).

Considering an isotropic fluid (a fluid is a medium in which static shear forces cannot exist) with zero viscosity, the non-linear basic equations that define transmission of compressional waves in terms of

pressure variations; P

and

particle velocity; $\underline{V}$

are derived.

The total pressure field is

$$P_t = P_o + P, \tag{A1}$$

where P_o is the static pressure and P is the pressure changes caused by the wave field. Also the total density in the fluid is

$$\rho_t = \rho_o + \rho. \tag{A2}$$

The first equation that must be derived describes the relationship between pressure variation in space and particle velocity changes in time. To show this relationship, conservation of momentum (Newton's second law) for a small volume ΔV with constant mass Δm is used,

$$\underline{F}dt = d(\Delta m\underline{V}) \tag{A3}$$

or

$$\underline{F}dt = \Delta m d\underline{V}, \tag{A4}$$

where $\underline{V}$ is the average velocity in ΔV. As time advances from t to t+dt, the average particle velocity inside ΔV changes according to

$$d\underline{V} = \frac{\partial \underline{V}}{\partial t} dt + \frac{\partial \underline{V}}{\partial x}(V_x dt) + \frac{\partial \underline{V}}{\partial y}(V_y dt) + \frac{\partial \underline{V}}{\partial z}(V_z dt) \tag{A5}$$

or

$$\frac{d\underline{V}}{dt} = \frac{\partial \underline{V}}{\partial t} + \frac{\partial \underline{V}}{\partial x} V_x + \frac{\partial \underline{V}}{\partial y} V_y + \frac{\partial \underline{V}}{\partial z} V_z \tag{A6}$$

or

$$\frac{d\underline{V}}{dt} = \frac{\partial \underline{V}}{\partial t} + (\underline{V}\cdot\nabla)\underline{V}, \tag{A7}$$

where $(\underline{V}\cdot\nabla)\underline{V}$ is referred to as the convection term and $\underline{V} = (V_x,V_y,V_z)$.

If the force $\underline{F}$ is written as

$$\underline{F} = (F_x,F_y,F_z), \tag{A8}$$

then

$$F_x = -\Delta P_x \Delta S_x = -\left[\frac{\partial P}{\partial x}\Delta x + \frac{\partial P}{\partial t}dt\right] \Delta S_x \tag{A9}$$

$$= -\frac{\partial P}{\partial x}\Delta V \quad \text{as } dt \rightarrow 0, \tag{A10}$$

where

$$\Delta V = \Delta x \Delta y \Delta z,$$

$$\Delta S_z = \Delta y \Delta x.$$

Similarly it can be shown that

$$F_y = -\frac{\partial P}{\partial y}\Delta V, \tag{A11}$$

$$F_z = -\frac{\partial P}{\partial z}\Delta V, \tag{A12}$$

or by using equations (A8), (A10), (A11), and (A12)

$$\underline{F} = -\Delta V \nabla P. \tag{A13}$$

Then by combining (A4), (A7), and (A13), the relationship for conservation of momentum will become

$$-\nabla P = \rho + \left[\frac{\partial \underline{V}}{\partial t} + (\underline{V}\cdot\nabla)\underline{V}\right]. \tag{A14}$$

The second equation will quantify the relationship between particle velocity variations in space and pressure changes in time. By assuming a fixed amount of mass Δm with some volume ΔV, and exposing the mass to some external force, its position and its volume will change. By the principle of conservation of mass, the mass' change in volume (dV) can be related to its change in total density ($d\rho$):

$$\Delta m(x_1,y_1,z_1,t_1) = \Delta m(x_2,y_2,z_2,t_2) \tag{A15}$$

or

$$\rho_t(x_1,y_1,z_1,t_1)\Delta V(x_1,y_1,z_1,t_1) = \rho_t(x_2,y_2,z_2,t_2)\Delta V(x_2,y_2,z_2,t_2) \tag{A16}$$

or

$$\rho_t \Delta V = (\rho_t + d\rho_t)(\Delta V + dV) \tag{A17}$$

or

$$\frac{d\rho_t}{\rho_t} = -\frac{dV}{\Delta V} - \frac{d\rho_t dV}{\rho_t \Delta V}. \tag{A18}$$

The elemental change in total density can be written as

$$d\rho_t = \frac{\partial\rho}{\partial t}dt + \frac{\partial\rho_t}{\partial x}(V_x dt) + \frac{\partial\rho_t}{\partial y}(V_y dt) + \frac{\partial\rho_t}{\partial z}(V_z dt) \tag{A19}$$

or

$$\frac{d\rho_t}{dt} = \frac{\partial\rho}{\partial t} + (\underline{V}\cdot\nabla)\rho_t. \tag{A20}$$

$\frac{dV}{\Delta V}$ can be written as

$$\frac{dV}{\Delta V} = \frac{dx}{\Delta x} + \frac{dy}{\Delta y} + \frac{dz}{\Delta z} \tag{A21}$$

for small volumes, or

$$\frac{dV}{\Delta V} = \frac{\partial(Vxdt)}{\partial x} + \frac{\partial(Vydt)}{\partial y} + \frac{\partial(Vzdt)}{\partial z}\,, \tag{A22}$$

giving

$$\frac{dV}{\Delta V} = (\nabla\cdot\underline{V})dt. \tag{A23}$$

Now by combining (A20), (A23), and (A18),

$$\frac{\partial\rho}{\partial t} + (\underline{V}\cdot\nabla)\rho_t = -\rho_t(\nabla\cdot\underline{V})+O(dt) \tag{A24}$$

or, for small dt,

$$-\nabla\cdot(\rho_t\cdot\underline{V}) = \frac{\partial\rho}{\partial t} \tag{A25}$$

or in more common form

$$-\nabla\cdot\underline{V} = \frac{1}{\rho_t}\frac{d\rho_t}{dt}\,, \tag{A26}$$

If a linear relationship between density changes and pressure changes,

$$d\rho_t = \frac{1}{v^2}\,dP, \tag{A27}$$

exists within the constant mass Δm, then by substitution into (A26)

$$-\nabla\cdot\underline{V} = \frac{1}{\rho_t V^2}\ \frac{dP}{dt}\ , \tag{A28}$$

or if K the bulk modulus is given by $\rho_t V^2$

$$-\nabla\cdot\underline{V} = \frac{1}{K}\left[\frac{\partial P}{\partial t} + (\underline{V}\cdot\nabla)P\right]\ . \tag{A29}$$

It can be shown that for practical seismic situations

$$|(\underline{V}\cdot\nabla)\underline{V}| << \left|\frac{\partial \underline{V}}{\partial t}\right|$$

and (A30)

$$|(\underline{V}\cdot\nabla)P| << \left|\frac{\partial P}{\partial t}\right|\ ,$$

giving the commonly known relationships

$$-\nabla P = \rho\frac{\partial \underline{V}}{\partial t} \tag{A31}$$

and

$$-\nabla\cdot\underline{V} = \frac{1}{K}\ \frac{\partial P}{\partial t}\ \text{ with } K = \rho V^2. \tag{A32}$$

Note that equations (A31) and (A32) will apply for inhomogeneous fluids if the derivatives of ρ and V exist.

To derive the wave equation for inhomogeneous fluids, the divergence operator is applied to equation (A31),

$$-\nabla\cdot(\nabla P) = \nabla\cdot\ \left(\rho\frac{\partial \underline{V}}{\partial t}\right) \tag{A33}$$

or

$$-\nabla^2 P = \rho \nabla \cdot \left(\frac{\partial \underline{V}}{\partial t} \right) + \frac{\partial \underline{V}}{\partial t} \cdot \nabla \rho \ . \qquad \text{(A34)}$$

Substituting (A32) into (A34) gives

$$\nabla^2 P = \frac{1}{V^2} \frac{\partial^2 P}{\partial t^2} - \frac{\partial \underline{V}}{\partial t} \cdot \nabla \rho \ . \qquad \text{(A35)}$$

Combining (A31) and (A35) then gives

$$\nabla^2 P - \frac{1}{V^2} \frac{\partial^2 P}{\partial t^2} = \nabla P \cdot \nabla \ln \rho . \qquad \text{(A36)}$$

The effect of density inhomogeneity on the wave equation is given by the term $\nabla P \cdot \nabla \ln \rho$. Hence if the inhomogeneity in the $\ln \rho$ can be ignored, then the equation (A36) simplifies to the acoustic wave equation

$$\nabla^2 P = \frac{1}{V^2} \frac{\partial^2 P}{\partial t^2} \ . \qquad \text{(A37)}$$

APPENDIX B

THE ONE-WAY WAVE EQUATION

Derivation of the One-way Wave Equation

Starting with the acoustic wave equation (see Appendix A),

$$V^2\nabla^2 P = \ddot{P} \tag{B1}$$

and assuming that the velocity is constant, take a three-dimensional Fourier transform on both sides of (B1)

$$(k_x{}^2+k_z{}^2)\hat{\bar{P}} = \frac{\omega^2}{V^2}\hat{\bar{P}},$$

where

$$\hat{\bar{P}}(kx,kz,\omega) \leftrightarrow P(x,z,t) \tag{B2}$$

giving the well known dispersion relationship:

$$\frac{\omega^2}{V^2} = k_x{}^2+k_z{}^2. \tag{B3}$$

Then we take the square root of both sides and multiply by i = sqrt(-1) to obtain

$$i\omega = \pm iV(k_x{}^2+k_z{}^2)^{\frac{1}{2}} . \tag{B4}$$

But it is known that

$$i\omega\hat{P} \leftrightarrow \frac{dP}{dt} ,$$

therefore

$$i\omega\hat{\overline{P}} = \pm\ iV(k_x{}^2+k_z{}^2)\hat{\overline{P}}$$

and in the time domain

$$\frac{\partial \overline{P}}{\partial t} = \pm iV(k_x{}^2+k_z{}^2)^{\frac{1}{2}}\overline{P}. \qquad \text{(B5)}$$

Equation (B5) is the One-way wave equation where the sign on the right hand side controls whether it is a forward or backward propagating wave.

Stability

The first derivative with respect to time in the One-way wave equation is approximated by the centered difference scheme. The One-way wave equation in one dimension is given by

$$\frac{\partial P^n}{\partial t^n} = V\frac{\partial P^n}{\partial x^n};\ P^n(x,n\Delta t) = P(x,n\Delta t), \qquad \text{(B6)}$$

where n represents a specific time step. Assuming a sinusoidal solution for $(x,n\Delta t)$ for the centered difference approximation

$$P^n(x,n\Delta t) = e^{i(k_x - \omega n\Delta t)}, \tag{B7}$$

the expression will simplify to

$$Vkx = -\frac{\sin(\omega\Delta t)}{\Delta t}. \tag{B8}$$

If omega, the temporal frequency, is real, then

$$|Vkx\Delta t| \leq 1. \tag{B9}$$

Now by taking the worst possible case at maximum velocity and at maximum spatial frequency

$$V = Vmax$$

$$kx = kx_{nyq} = \frac{\pi}{\Delta x},$$

the stability relationship for the One-way wave equation is derived :

$$\frac{Vmax\Delta t}{\Delta x} < \frac{1}{\pi}. \tag{B10}$$

Extrapolating the solution to two dimensions gives

$$k = (k_x{}^2 + k_z{}^2)^{\frac{1}{2}} \text{ and if } \Delta x = \Delta z$$

$$\frac{Vmax\Delta t}{\Delta x} < \frac{1}{\sqrt{2}\,\pi}. \tag{B11}$$

Numerical Dispersion Due to the Finite Difference in Time

It is known from equation (B8) that

$$Vk\Delta t = \sin(\omega\Delta t).$$

For dispersion we are interested in some kind of measure of phase velocity (temporal frequency/spatial frequency) with respect to spatial frequency. This is done by solving equation (B8) for phase velocity:

$$P_v = \frac{\omega}{K} = \frac{\Delta r}{\Delta t} \cdot \frac{\sin^{-1}(\alpha\Delta rk)}{\Delta rk}$$

where

$$\alpha = \frac{V\Delta t}{\Delta r} < \frac{1}{\pi} \text{ for stability} \tag{B12}$$

In fact we are more interested in the change in phase velocity relative to the true velocity:

$$\frac{Pv}{V} = \frac{\Delta r}{V\Delta t} \frac{\sin^{-1}(\alpha \Delta rk)}{\Delta rk} \tag{B13}$$

APPENDIX C

THE TWO-WAY NON-REFLECTING WAVE EQUATION

Derivation of the Two-way Non-reflecting Wave Equation

In deriving the Two-way non-reflecting wave equation, let us start with the basic equations of particle velocity

$$-\nabla P = \rho \frac{\partial \underline{V}}{\partial t} \tag{C1}$$

$$-\nabla \cdot \underline{V} = \frac{1}{K} \frac{\partial P}{\partial t} \tag{C2}$$

Now taking the divergence operator of equation (C1) and combining with (C2) gives

$$\nabla \cdot \left(\frac{1}{\rho} \nabla P\right) = \frac{1}{K} \frac{\partial^2 P}{\partial t^2} \tag{C3}$$

In the acoustic case where $K = \lambda$, then the acoustic wave equation is

$$\nabla \cdot \left(\frac{1}{\rho} \nabla P\right) = \frac{1}{V^2 \rho} \frac{\partial^2 P}{\partial t^2} \tag{C4}$$

(Berkhout, 1982)

To obtain the Two-way non-reflecting wave equation, constant impedance is assumed in equation (C4)

$$\nabla \cdot (V \nabla P) = \frac{1}{V} \frac{\partial^2 P}{\partial t^2} \tag{C5}$$

giving finally the desired form of the wave equation.

What does the assumption of constant impedance entail? In general

If: Θ_i = angle of incidence

Θ_R = angle of refraction

$$R(\Theta_i, \Theta_R) = \frac{\dfrac{\rho_2 V_2}{\cos\Theta_R} - \dfrac{\rho_1 V_1}{\cos\Theta_i}}{\dfrac{\rho_2 V_2}{\cos\Theta_R} + \dfrac{\rho_1 V_1}{\cos\Theta_i}} \tag{C6}$$

(Aki and Richards, 1980)

or in terms of only the angle of incidence

$$R(\Theta_i) = \frac{\rho_2 V_2 \cos\Theta_i - \rho_1 \sqrt{V_1^2 - V_2^2 \sin^2\Theta_i}}{\rho_2 V_2 \cos\Theta_i - \rho_1 \sqrt{V_1^2 - V_2^2 \sin^2\Theta_i}} \tag{C7}$$

(Berkhout, 1982)

Constant impedance assumes that $\rho_1 V_1 = \rho_2 V_2$, and that the bulk modulus is directly related to the velocity by the impedance.

Stability

Using the same approach as used in Appendix B for the One-way wave equation, a sinusoidal solution is assumed for the finite difference scheme.

$$\frac{d^2 Pn}{dt^2} = \frac{Pn+1 - 2Pn + Pn-1}{\Delta t^2} \tag{C8}$$

Solving the finite difference equations of the assumed solution gives

$$V^2 (k_x{}^2 + k_z{}^2) = \frac{4}{\Delta t^2} \sin^2 \frac{\omega \Delta t}{2} \tag{C9}$$

if the temporal frequency is real. This in turn gives the relationship

$$\left| \frac{\Delta t V}{2} (k_x{}^2 + k_z{}^2)^{\frac{1}{2}} \right| < 1 \tag{C10}$$

and evaluating it at the worst possible case gives the stability relationship for $\Delta x = \Delta z$

$$\frac{\Delta t V max}{\Delta x} < \frac{\sqrt{2}}{\pi} \tag{C11}$$

Numerical Dispersion

Taking the one-dimensional formulation of equation (C9) and solving for the temporal frequency gives

$$\omega = \frac{2}{\Delta t} \sin(\alpha \Delta r K) ; \alpha = \frac{V \Delta t}{\Delta x} \tag{C12}$$

Then by solving for the phase velocity (temporal/spatial frequency) relative to the true velocity gives the dispersion relationship for the Two-way non-reflecting wave equation.

$$\frac{P_v}{V} = \frac{2}{\alpha} \frac{\sin^{-1}(\frac{\alpha}{2} \Delta rk)}{\Delta rk} \tag{C13}$$

where $\alpha < \frac{2}{\pi}$ for stability.

APPENDIX D

SUPER COMPUTERS

Within the last three years technology has advanced rapidly in the geophysical industry. Computing power is reaching a point that many methods of processing originally only considered can now be applied in a realistic time frame. As these methods are the future in the geophysical industry, I believe that a brief description of two separate approaches to new super computing power is appropriate to this thesis.

STAR-100

I have been fortunate to work on two of the world's fastest computers. One is the STAR-100 array processor. An array processor is a peripheral device being fed by a program running on a front-end computer. The program on the front-end treats operations on the AP as simply a call to a subroutine. An array processor is a vector machine, in which all operations are vector type operations. Those operations are done by pipelining vectors through vector-type functional units. This allows for results to be produced every clock period. The STAR-100 is a new generation of super array processors making use of VLSI structure. This AP can be divided into two distinct operational elements. The first controls all the I/O operations and arithmetic processing. The second is the mass storage memory and the high speed cache memory. Data is shipped from the host computer under control of the I/O subsystem to the main memory. Main memory on the STAR is a bulk storage device with a capacity of up to eight million words (32 bits). The storage move processor (SMP), then can move the data from main memory to the data cache, where the data can be accessed by the arithmetic control processor (ACP). The data, after processing, can then be returned to main memory. All these operations are controlled asynchronously by the SMP and the ACP. The effective clock cycle on the STAR is 40 nano seconds, with 100 megabyte port from main memory to cache. At top speed, the STAR can operate at just over 100 million floating point operations per second.

CRAY-XMP 24

The other direction that some geophysical industries are going is the super computers almost exclusively controlled by CRAY Research. The CRAY-XMP 24 is a liquid cooled dual processor CPU with four million words of high speed bi-directional memory which allows access at the same time by both CPUs. The XMP is a main frame machine in which the front-end computers have no program control. The front-ends act only as editing tools and channeling devices allowing multiusers on the CRAY.

The heart of the CRAY is the two CPUs which have a suite of functional units both scalar and floating-point vector. The data and the programs reside in four million words of memory. Data can be transferred from memory to the vector registers at a rate of three words per clock period. Attached to the main memory is a 256 megabyte (32 million words) solid state disk. There is a direct I/O channel into main memory operating at 1250 megabytes per second. On the other side of the CPU is a massive I/O subsystem, which controls up to 48 disk drives and 48 tape drives. Data is transferred across from the I/O subsystem at a rate of 200 megabytes per second. The entire system operates under a 9.5 nano second clock producing up to 250 megaflops for both CPUs.

Both of the machines are very fast and it would be difficult to say which is better. The STAR-100 may produce faster code at high level languages because of its inherent vectorization and minimal scalar operations. The CRAY, however, has a much faster clock period, and can take pure FORTRAN-77 code. What this means is that the speed of the STAR is severely hindered by the I/O being done from its front-end. The CRAY has no such problem. The problem with the CRAY is that vectorization is not a simple process, but requires many hours of work to optimize the code.

APPENDIX E

PROGRAM OF THE FOURIER THEORETICAL TECHNIQUE AS APPLIED TO INVERSE MODELING

```
      SUBROUTINE RTM_EDITP
C
C      Disco edit phase subroutine:
C              Used to set up all disk files, allocate memory
C
C      For description of PTM see RTM_PROCP
C
C      Variable definition
C              Most varibales are described on the same line
C
C
C      common blocks inlcude
C              Monfort                 Disco common block
C              Rtmcb                   Program common block
C
C
      INCLUDE 'MONFORT/NOLIST'
      INCLUDE 'RTMCB.FOR/NOLIST'

      CHARACTER*8         LINE
      CHARACTER*16        IDENT
      CHARACTER*8 SNAM

      DIMENSION MSG(20)         !Message block for communication
                               !with module depth

      EQUIVALENCE         (IDZ,DZ),(IDX,DX),(IV_SCALE,V_SCALE)
      EQUIVALENCE         (IV_MIN,V_MIN)

C
C
C
      RDT=FLOAT(DT)*1E-6                   !Convert sample rate
                                          !from microseconds to seconds

      NTRACES=0                            !Initialize number of traces

      ISEQ=0                                  !sequential number

      ID=0                                  !RCORE memory counter

      OUTFLAG=.FALSE.                         !three logical variables:
      INFLAG=.TRUE.                         !INPUT mode
      IFIRST=.TRUE.                         !FLAG

C
C               Open PROCESS template and get input parameters
C
C

      CALL  SETGBL (NLISTS)                !Disco routine

      DX = FPARM ( 'DX',0,0,0,0)         !Delta x

      FREQ=FPARM('FREQ',0,0,0,0)         !Maximum frequency

      VEL = CPARM ( 'OPER',1, 'MIGR' )!Operation
C
C ***  Get parameters from input list ***
```

```
C
        NAME='CDP'

        DO I=1,NLIST

          CALL NXTLST(LIST_NAME,NNAMES,INDEX,NREP)!GET NEXT LIST CARD

            IF(INDEX.EQ.1)THEN

                SNAM='TIME'
                RL=0
                RU=0
                RDEF=10
                END_TIME=IPARM(SNAM,001,RL,RU,RDEF)
                NTIMES=IPARM('NTIMES',000,0,0,0)

            ELSE IF(INDEX.EQ.2)THEN

                SNAM='PKEY'
                DEFAULT='CDP'
                NAME=CPARM(SNAM,1,DEFAULT)

            ELSE

                WRITE(U$ERR,*),'INDEX PROBLEM IN INPUT LIST'

          END IF

        END DO

C
C        define and/or get trace headers
C

        CALL THDRDEF('TIME',1,HDR$I,IXH_TIME)         !DEFINE time header
                                                      !snap shots
        ORDER=IXH_TIME
C
C                Get index in trace header for
C

        I=THDRGET('LASTTR',LEN,FORMAT,IXH_LASTR,'E') !last trace flag
        I=THDRGET(NAME,LEN,FORMAT,INDEX_CDP,'E') !Primary key
        I=THDRGET('SEQNO',LEN,FORMAT,IXH_ISEQ,'E') !Sequencial number


C
C        Necessary initializations
C                for FPS-100 and Line definition
C

        CALL SETOPT(0)          !not a reentrant module

        CALL  INFOGET ( 'LINE',LINE ) !Get line name

        CALL INFOGET('APMAX',APMAX) !Get maximum size of AP
        CALL APMEM(APMAX) !initialize AP

C
C                Read message from depthvel
```

```
C
      NUM=5
      MSG_LEN=1

      IF(.NOT.MSGGET('RTM',MSG,NUM))THEN

               DZ=DX
               ID_LENGTH=16
               V_MIN=1000
               V_MAX=1000
      ELSE

               IDZ=MSG(1)
               ID_LENGTH=MSG(2)
               IV_MAX=MSG(3)
               IV_MIN=MSG(4)
               IDX=MSG(5)

      ENDIF

      DT=INT(DZ/1E-3)

C
C               Calculate fft lengths for x and z
C
      CALL RTM_LENGTH(2*ID_LENGTH,IPOWER)
      IPOWER=IPOWER+1
      KZ_LENGTH=2.**(FLOAT(IPOWER))

      CALL RTM_LENGTH(2*MAXNTR,IP)
      IP=IP+1
      KX_LENGTH=2.**(FLOAT(IP))

C
C               Calculate number of words need in memory
C

      NWDT4=(MAXNTR+5)*ID_LENGTH
      NWDT4=NWDT4+(KX_LENGTH+4)*(KZ_LENGTH+4)
      NWDT4=(NWDT4/128+1)*128

C
C               Allocate memory and disk space
C

      CALL MEMVAR(NWDT4)

      CALL DSKLCL(THDRLEN,MAXNTR,THRD_FIL)
      CALL DSKLCL(ID_LENGTH,MAXNTR,P_FIL)
      CALL DSKLCL(LENGTH,MAXNTR,I_FIL)
      CALL DSKLCL(KZ_LENGTH+4,KX_LENGTH,K_FIL)

      ITOT=END_TIME/NTIMES+1

      CALL DSKLCL(ID_LENGTH,ITOT*MAXNTR,DEPTH_FIL)

C
C               Scaling factor for stability
C
```

```
      PI=ACOS(-1.)
      OMEGA=2.*PI*FREQ
      SAMP=1./(2*OMEGA)
      AC=SAMP
      DO IC=1,10
              AC=AC*10
              IF(INT(AC).GT.0)THEN
                      SAMP1=FLOAT(INT(AC))*10.**-FLOAT(IC)
                      GOTO 111
              ENDIF
      ENDDO
111      CONTINUE
      SAMP=SAMP1

      V_SCALE_NEW=2.*SAMP

      OLD_LENGTH=LENGTH
      LENGTH=ID_LENGTH

      RETURN
      END
```

```
      SUBROUTINE RTM_PROCP(TRACE,THDR,IFLAG)
C
C      by Tony Sirtautas
C      at Golden Geophysical
C            can be contacted at SOHIO Pet.
C            (214)960-4470

C      Process phase of a Disco Module
C            Note this code is not transportable and there is no
C            guarantees that it is bug free.
C            Transportability problem:
C                  1) This code must run under DISCO
C                        The stand-alone code is available
C                  2) Runs with the STAR-100 array proccessor
C                        the FPS-100 array proccessor

C            The stand-alone code is writtern in Fortran-77
C            and uses some code which needs a Cray XMP to run on.
C            This problem would be easy to fix.
C

C      Variable definition
C            Most varibales are described on the same line
C
C
C      common blocks inlcude
C            Monfort                  Disco common block
C            Rtmcb                    Program common block
C
C
      INCLUDE 'RTMCB/LIST'
      INCLUDE 'MONFORT/LIST'

      REAL TRACE(1),INTERCEPT
      INTEGER THDR(1)

      CHARACTER AISTAT*20

      PARAMETER (ILUN=1)

      ASSIGN 100 TO PROCP
      ASSIGN 200 TO OUTPUT

      XI=0

      IF(OUTFLAG)GOTO OUTPUT          !in output mode

      IF(INFLAG.AND.VEL.NE.'DEPTH')THEN

            CALL MEMCRE(PCORE(FWAVAR),NWDT4)

            LEN_NUM=NWDT4

            DO I=1,KZ_LENGTH
                  RCORE(FWAVAR+(I-1))=0.
            ENDDO

            NUM=NWDT4/KZ_LENGTH

            DO I=1,NUM-1
```

```
                      CALL MOVCOR(RCORE(FWAVAR),RCORE(FWAVAR+I*KZ_LENGTH)
     &                           ,KZ_LENGTH)
                      LEN_NUM=LEN_NUM-KZ_LENGTH

                 ENDDO

                 INFLAG=.FALSE.

            ENDIF

            IFLAG =FLG$MULTI  !set up multi input mode

            NTRACES=NTRACES+1

            ID=ID+OLD_LENGTH

            CALL DSKWRT(I_FIL,NTRACES,TRACE,OLD_LENGTH,1)

            IF(THDR(IXH_LASTR).EQ.1.OR.NTRACES.EQ.MAXNTR)GOTO PROCP

            CALL DSKWRT(THRD_FIL,NTRACES,THDR,THDRLEN,1)

            RETURN

100           CONTINUE                  !Process phase

            THDR(IXH_LASTR)=1

            CALL DSKWRT(THRD_FIL,NTRACES,THDR,THDRLEN,1)

            DEPTH_DATA=FWAVAR

            WORK_SPACE=DEPTH_DATA+ID_LENGTH*NTRACES+1

            CALL RTM_KXKZ(KX_LENGTH,KZ_LENGTH,DX,DZ,FREQ,V_min,K_FIL)

            CALL BLIOOPN('VEL.DSK','OLD',0,0,BL$RDO,0,IVEL_TEMP)
            CALL BLIOOPN('SVEL.DSK','NEW',0,0,0,0,IVS_FIL)
            CALL RTM_VSCALE(IVEL_TEMP,IVS_FIL,ID_LENGTH,NTRACES
     &                      ,V_SCALE_NEW)
            CALL BLIOCLS(IVEL_TEMP,'DELETE')

            IUNIT1=6
            IOUT=0
            ITIME=0

            DO IT=1,NTRACES

                 CALL DSKWRT(P_FIL,IT,RCORE(DEPTH_DATA+(IT-1)*ID_LENGTH)
     &                      ,ID_LENGTH,1)
                 CALL DSKWRT(DEPTH_FIL,IT,RCORE(DEPTH_DATA+(IT-1)*ID_LENGTH)
     &                           ,ID_LENGTH,1)

            ENDDO

                 IOUT=IOUT+NTRACES
*
*
*
```

```
      STAR=.TRUE.

      TIME_S=0.

      AISTAT=' **STOPEN**'

      CALL STOPNW(ILUN,ISTAT,'(AP1)')
               IF(ISTAT.NE.0) CALL STAR_ERROR(AISTAT,ISTAT)

      DO ICOUNT=1,END_TIME/NTIMES           !this form is setup to release
                                           !Star every few minutes.

C
C      go STAR
C

             CALL RTM_FINITE(RCORE(DEPTH_DATA),RCORE(WORK_SPACE)
     &                ,K_FIL,I_FIL,IVS_FIL
     &                ,OLD_LENGTH,ID_LENGTH,NTRACES,P_FIL
     &                ,KX_LENGTH,KZ_LENGTH,NTIMES,RDT,SAMP,TIME_S,STAR)

              DO IT=1,NTRACES
C
C      Roll circular buffer of finite difference
C
                   CALL DSKWRT(DEPTH_FIL,IOUT+IT
     &                          ,RCORE(DEPTH_DATA+(IT-1)*ID_LENGTH)
     &                          ,ID_LENGTH,1)

              ENDDO

              IOUT=IOUT+NTRACES

      ENDDO

      AISTAT= '**FINITE** STCLOS'

      CALL STCLOS(ILUN,ISTAT)
             IF(ISTAT.NE.0)CALL STAR_ERROR(AISTAT,ISTAT)

      CALL BLIOCLS(IVS_FIL,'DELETE')

      OUTFLAG=.TRUE.

      ISEQ=0
      P_FIL=0
      P1_FIL=0
      ID=0

200     CONTINUE

      ISEQ=ISEQ+1
      P1_FIL=P1_FIL+1
      ID=ID+1

      IFLAG=FLG$MULTO          !Multi output mode (DISCO)

      CALL DSKRD(DEPTH_FIL,ID,TRACE(1),ID_LENGTH,1)
      CALL DSKRD(THRD_FIL,1,THDR,THDPLEN,1)
```

```
THDR(IXH_LASTR)=0
THDR(IXH_TIME)=INT(10000*SAMP*(P_FIL))
THDR(IXH_ISEQ)=P1_FIL
THDR(INDEX_CDP)=P1_FIL

IF(P1_FIL.EQ.NTRACES)THEN

        P1_FIL=0
        P_FIL=P_FIL+NTIMES
        THDR(IXH_LASTR)=1

ENDIF

IF(ISEQ.GE.IOUT)THEN

        IFLAG=FLG$NORM          !Hit the last trace
        RETURN

ENDIF

RETURN
END
```

```
      SUBROUTINE RTM_FINITF(A,B,K,BC,FILNAME,ITLEN,LEN,NTRACES
     &                ,SCALAR,FIL2,IKX,IKZ,NTIMES,DT,SAMP,TIME_S,START)
C
C      This subroutine initializes, loads, and starts the microcode
C      the Star-100 Ap.

      IMPLICIT REAL K

      INTEGER FILNAME,FIL2,ILUN
      INTEGER IA(3),IB(3),IXBUF(3),IBC(3),IK(3),IV(3)
      INTEGEP IWORK(3)

      REAL A(LEN*NTRACES),B(2*IKZ+NTIMES*NTRACES),K(IKX*(IKZ+4))
      REAL BC(NTRACES*ITLEN),TIME(500)

      CHARACTER*20 AISTAT

      LOGICAL DEBUG,START

      PARAMETER (ILUN=1,DEBUG=.TRUE.)
C
C      INITIALIZE COUNTERS TO READ THE FILES FROM ASYCHONOUS STORAGE
C              READS IN TERMS OF 512 BYTE BLOCKS
C
      NBYTETR=(LEN)*4          !NUMBER OF BYTES PER TRACES

      IBLKSTR=NBYTETR/512

      IF(NBYTETR.EQ.IBLKSTR*512) THEN  !NUMBER OF BLOCKS PER TRACE

              NBLKSTR=IBLKSTR

      ELSE

              NBLKSTR=IBLKSTR+1

      ENDIF

      IBLK_ST=1          !STARTING BLOCK NUMBER

      TIME_F=TIME_S+(NTIMES-1)*SAMP !FINAL TIME...
                                    ! THE STAR IS RELEASED

      REM=AMOD(TIME_S,DT)              !DETERMINE THE NUMBER OF TIME STEPS
                                    !BETWEEN STARTING TIME AND END TIME

      IST=(TIME_S-REM)/DT+1

      REM=AMOD(TIME_F,DT)

      IFT=(TIME_F+(DT-REM))/DT+3

      IF(IFT.GT.ITLEN)THEN              !IF PAST THE "END OF DATA"
                                    ! ONLY LET CALCLATE TO END

           IFT=ITLEN
           TIME_F=(ITLEN-1)*DT
           NTIMES=(TIME_F-TIME_S)/SAMP+1

      ENDIF
```

```
        NUMB=(IFT-IST)+1

        DO I=1,NUMB

                TIME(I)=DT*FLOAT(I-1)

        ENDDO

        DO I=1,NTRACES          !INTERPOLATE BOUNDARY CONDITINS

                CALL RTM_INTERP(NUMB,TIME,BC((I-1)*ITLEN+IST)
     &                         ,B(2*IKZ+(I-1)*NTIMES+1),NTIMES,SAMP)

        ENDDO

C
C        OPEN THE STAR-100
C

        AISTAT= '**FINITE** STOPNW'

        CALL STOPNW(ILUN,ISTAT,'(AP1)')
                IF(ISTAT.NE.0) GO TO 99999

C
C        CALCULATE THE NUMBER OF WORDS NEEDED ON THE STAR
C

        ISIZE=
     &       3*NTRACES*LEN+2*IKX*(IKZ+4)+4*(IKX)+NTIMES*NTRACES+3*IKX
        ISIZE=(ISIZE/4000+1)*4

C
C        SCHEDULE THE JOB ON THE STAR-100
C

        AISTAT= '**FINITE** STSCHW'

        CALL STSCHW(ILUN,ISTAT,'(DSIZE)',ISIZE,'(PSIZE)',88)
                IF(ISTAT.NE.0) GO TO 99999

        WRITE(*,9991)ISIZE
9991       FORMAT(///,1X,'STAR SCHEDULING COMPLETED WITH ',I5,' WORDS.',//)

C
C        DEFINE MAIN MEMORY ARRAYS
C

        AISTAT= '**FINITE** STARAY'

        CALL STARAY(ILUN,ISTAT,IA,NTRACES*LEN,'(REAL)')
                IF (ISTAT.NE.0) GO TO 99999

        CALL STARAY(ILUN,ISTAT,IB,NTRACES*LEN,'(REAL)')
                IF (ISTAT.NE.0) GO TO 99999

        CALL STARAY(ILUN,ISTAT,IV,NTRACES*LEN,'(REAL)')
                IF (ISTAT.NE.0) GO TO 99999
```

```
      CALL STARAY(ILUN,ISTAT,IWORK,IKX*(IKZ+4),'(REAL)')
              IF (ISTAT.NE.0) GO TO 99999

      CALL STARAY(ILUN,ISTAT,IK,IKX*(IKZ+4),'(REAL)')
              IF (ISTAT.NE.0) GO TO 99999

      CALL STARAY(ILUN,ISTAT,IXBUF,3*IKX,'(REAL)')
              IF (ISTAT.NE.0) GO TO 99999

      CALL STARAY(ILUN,ISTAT,IBC,NTIMES*NTRACES,'(REAL)')
              IF (ISTAT.NE.0) GO TO 99999

C
C      WRITE INPUT DATA TO ST100
C

      IB(1)=1
      IBC(1)=1
      IV(1)=1
      IA(1)=1

      AISTAT= '**FINITE** STWRW'

      DO INT=1,NTRACES

              CALL BLIOGET(FILNAME,IBLK_ST+(INT-1)*NBLKSTR
     &                      ,B(1),NBYTETR)
              CALL DSKRD(FIL2,INT,B(IKZ+1),LEN,1)

              CALL STWRW(ILUN,ISTAT,B(IKZ+1),LEN,IA)
                      IF(ISTAT.NE.0) GO TO 99999

              CALL STWRW(ILUN,ISTAT,A((INT-1)*LEN+1),LEN,IB)
                      IF(ISTAT.NE.0) GO TO 99999

              CALL STWRW(ILUN,ISTAT,B(2*IKZ+(INT-1)*NTIMES+1),NTIMES,IBC)
                      IF(ISTAT.NE.0) GO TO 99999

              CALL STWRW(ILUN,ISTAT,B(1),LEN,IV)
                      IF(ISTAT.NE.0) GO TO 99999

              IV(1)=IV(1)+LEN
              IA(1)=IA(1)+LEN
              IB(1)=IB(1)+LEN
              IBC(1)=IBC(1)+NTIMES

      ENDDO

      IV(1)=1
      IA(1)=1
      IB(1)=1
       IBC(1)=1

      AISTAT= '**FINITE** STWRW'

      IK(1)=1

      DO INT=1,IKX

              CALL STWRW(ILUN,ISTAT,K((INT-1)*(IKZ+4)+1),IKZ+2,IK)
```

```
                    IF(ISTAT.NE.0) GO TO 99999

               IK(1)=IK(1)+IKZ+4

          ENDDO

          IK(1)=1

          LGLEN1=ALOG(FLOAT(IKZ))/ALOG(2.)
          LGLEN2=ALOG(FLOAT(IKX))/ALOG(2.)

C
C          STAR EXECUTION LOOP
C

          CALL HEADER(' STAR PROCESSING')        !TIMING ROUTINES
          CALL TIMRB

          LOGICAL=0
          IF(TIME_S.EQ.0)LOGICAL=1

C
C
C**********************************************************************
C
          DO ICOUNT=1,NTIMES

               AISTAT= '** DERIV **'

               CALL DERIVW(ILUN,ISTAT,JSTAT,
     &                  IB,IV,IK,IWORK,IXBUF,NTRACES,NTRACES*LEN
     &                  IKX*2,LGLEN1,LGLEN2,SCALAR,LOGICAL)
               IF(ISTAT.NE.0.AND.ISTAT.NE.12099)
     &                  WRITE(6,*),AISTAT,' JSTAT=',JSTAT
               LOGICAL=0

               AISTAT= '** FINI **'

               CALL FINIW(ILUN,ISTAT,JSTAT,
     &                  IA,IB,IWORK,IBC,NTRACES,NTRACES*LEN,
     &                  ICOUNT)
               IF(ISTAT.NE.0.AND.ISTAT.NE.12099)
     &                  WRITE(6,*),AISTAT,' JSTAT=',JSTAT
C

          ENDDO
C
C**********************************************************************
C

111        CONTINUE

          CALL TIMRE

          IB(1)=1
          IA(1)=1

          AISTAT= '**FINITE** STRDW'

          DO INT=1,NTRACES
```

```
              CALL STRDW(ILUN,ISTAT,A((INT-1)*LEN+1),LEN,IB)
              IF(ISTAT.NE.0) GO TO 99999

              CALL STRDW(ILUN,ISTAT,B(1),LEN,IA)
              IF(ISTAT.NE.0) GO TO 99999

              CALL DSKWRT(FIL2,INT,B(1),LEN,1)

              IB(1)=IB(1)+LEN
              IA(1)=IA(1)+LEN

      ENDDO

C
C      RELEASE THE ST100
C

      AISTAT= '**FINITE** STREL'

      CALL STREL(ILUN,ISTAT)
              IF(ISTAT.NE.0) GO TO 99999

      AISTAT= '**FINITE** STCLOS'

      CALL STCLOS(ILUN,ISTAT)
              IF(ISTAT.NE.0) GO TO 99999

      TIME_S=TIME_F

      RETURN
C
C      ABNORMAL EXIT
C

99999      WRITE(6,*)AISTAT

      WRITE(6,997) ISTAT
997     FORMAT(3X,'ISTAT RETURN VALUE: ',I8)

      STOP 'ABORTING EXECUTION'

      END
```

```
      SUBROUTINE RTM_INTERP(N,T,Y,F,LENGTH,DELTA)
C
C     CUBIC SPLINE SUBROUTINE USED TO INTERPOLATE THE BOUNDARY
C
      INTEGER N,LENGTH
      REAL T(N),Y(N),D(500)
      REAL C(500),Z(500),F(LENGTH)

      D(1)=1.
      C(1)=0.
      Z(1)=0.

      DO I=2,N-1

              D(I)=2.*(T(I+1)-T(I-1))
              C(I)=T(I+1)-T(I)
              TEMP=(Y(I+1)-Y(I))/(T(I+1)-T(I))
              Z(I)=6.*(TEMP-(Y(I)-Y(I-1))/(T(I)-T(I-1)))

      ENDDO

      D(N)=1.
      C(N-1)=0.
      Z(N)=0.

      CALL TRI(N,C,D,C,Z)      !SOLVE THE TRI-DIAGONAL MATRIX

      XVAL=DELTA

      DO I=1,LENGTH

              F(I)=SPL3(N,T,Y,Z,XVAL) !CALCULATE THE NEEDE PARAMETERS
              XVAL=XVAL+DELTA

      ENDDO

      RETURN
      END
```

```
      SUBROUTINE TRI(N,A,D,C,B)
C
C     TRI-DIAGONAL MATRIX SOLVER
C
      DIMENSION A(N),D(N),C(N),B(N)

      DO I=2,N

              XMULT=A(I-1)/D(I-1)
              D(I)=D(I)-XMULT*C(I-1)
              B(I)=B(I)-XMULT*B(I-1)

      ENDDO

      B(N)=B(N)/D(N)
      DO I=1,N-1

              B(N-I)=(B(N-I)-C(N-I)*B(N-I+1))/D(N-I)

      ENDDO

      RETURN

      END
```

```
      FUNCTION SPL3(N,T,Y,Z,X)
C
C     INTEPPLOTE USING THE VALUES OBTAINED BY TRI
C
      DIMENSION T(N),Y(N),Z(N)

      DO J=1,N-2

            I=N-J
            TEMP=X-T(I)
            IF(TEMP.GE.0)GOTO 3

      ENDDO

      I=1
      TEMP=X-T(1)
3     H=T(I+1)-T(I)
      A=TEMP*(Z(I+1)-Z(I))/(6.*H)+.5*Z(I)
      B=TEMP*A+(Y(I+1)-Y(I))/H-H*(2.*Z(I)+Z(I+1))/6.
      SPL3=TEMP*B+Y(I)

      RETURN

      END

      SUBROUTINE RTM_KXKZ(IKX,IKZ,DX,DZ,FREQ,V_MIN
     &                              ,K_FIL)
C
C      FORM ARRAYS OF KX AND KZ VALUES FOR USE WITH DERIVATIVES IN AP
C             KX(IKX)       --->       ARRAY OF KX VALUES
C             KZ(IKZ)       --->       ARRAY OF KZ VALUES
C
C      WILL THEN GENERATE THE DERIVATIVE MATRIX AND STORE IT ON DISK
C
C
C      INPUT ARGUMENTS
C             KX,IKZ        --->       NUMBER OF SAMPLES
C             DX,DZ               --->       SAMPLE RATE
C
C

      INTEGER IKX,IKZ
      REAL KXN,KZN
      REAL K_MAX,K2
      REAL KX(2000),KZ(2000),D(2000),FX(2000),FZ(2000),OUT(4000)
C
C
C

      KXN=2*ACOS(-1.)/(2.*DX)
      KZN=2*ACOS(-1.)/(2.*DZ)

      DKX=KXN/(FLOAT(IKX)/2.)
      DKZ=KZN/FLOAT(IKZ/2)

      K_MAX=2*ACOS(-1.)*FREQ/V_MIN/SQRT(2.)

      IF(K_MAX.GT.KXN)K_MAX=KXN
```

```
N_K_MAX=K_MAX/DKX+1
LEN=2.**FLOAT(INT(LOG(FLOAT(N_K_MAX))/LOG(2.)))
K_MAX=(LEN-1)*DKX
N_K_MAX=K_MAX/DKX+1
LEN2=2.**FLOAT(INT(LOG((.2*N_K_MAX))/LOG(2.)))*2
DO II=1,LEN2
        D(II)=1.
ENDDO
DO II=1,IKX
        IF((II-1)*DKX.LE.KXN)THEN
                KX(II)=(II-1)*DKX
        ELSE
                KX(II)=(II-1)*DKX-2*KXN
        ENDIF
ENDDO
K_MAX=2*ACOS(-1.)*FREQ/V_MIN/SQRT(2.)
IF(K_MAX.GT.KZN)K_MAX=KZN
N_K_MAX=K_MAX/DKZ+1
LEN=2.**FLOAT(INT(LOG(FLOAT(N_K_MAX))/LOG(2.)))
K_MAX=(LEN-1)*DKZ
N_K_MAX=K_MAX/DKZ+1
LEN2=2.**FLOAT(INT(LOG((.4*N_K_MAX))/LOG(2.)))*2
DO II=1,LEN2
        D(II)=1.
ENDDO
DO JJ=1,IKZ/2+1
        KZ(JJ)=DKZ*(JJ-1)
ENDDO
DO J=1,IKX
IZ=1
        DO I=1,IKZ+1,2
                OUT(I)=0.
```

```
            OUT(I+1)=-SQRT(KZ(IZ)**2.+KX(J)**2.)
            IZ=IZ+1

        ENDDO

        CALL DSKWRT(F_FIL,J,OUT,IKZ+2,1)
    ENDDO

    RETURN

    END
```

```
      SUBROUTINE RTM_LENGTH(LENGTH,IPOWEP)
C
C      SUBROUTINE CALCULATFS THE NEAREST POWER OF TWO
C      RELATIVE TO LENGTH.  TO BE USED WITH THE FFT
C
      POWER=ALOG(FLOAT(LENGTH))/ALOG(2.)
      IP=INT(POWER+1)
      IF((IP-POWER).EQ.1)IP=INT(POWER)
      IPOWER=IP

      RETURN
      END
```

```
      SUBROUTINE RTM_VSCALE(IV_FIL,IVS_FIL,LEN,NTRACES,SCALAR)
      REAL WORK(4000)
C
C      SUBROUTINE SCALES THE VELOCITIES BY SCALAR
C      THE VELOCITYIES ARE STARED OUT OF CORE
C

      NBYTETR=(LEN)*4

      IBLKSTR=NBYTETR/512

      IF(NBYTETR.EQ.IBLKSTR*512) THEN

              NBLKSTR=IBLKSTR

      ELSE

              NBLKSTR=IBLKSTR+1

      ENDIF

      IBLK_ST=1

      DO INT1=1,NTRACES

           CALL BLIOGET(IV_FIL,IBLK_ST+(INT1-1)*NBLKSTR
     &                  ,WORK(1),NBYTETR)

           DO ILEN=1,LEN

                WORK(ILEN)=WORK(ILEN)*(-SCALAR)

           ENDDO

           CALL BLIOPUT(IVS_FIL,IBLK_ST+(INT1-1)*NBLKSTR
     &                  ,WORK(1),NBYTETR)

      ENDDO

      RETURN

      END
     PROCESS FINI(A,B,WORK,BC,NTRACE,NTLN,NTNTM,ID)
     LOCALMEMORY
     INTEGER NTLN,NTNTM,NTRACE
     INTEGER NTIMES,DELEN
     INTEGER BCLEN,BCNTL,BRSLEN,BRSM,BRSM2,CM2,CN,CNL2,CNM2
     INTEGER DF,DLEN,DNTL,I,ICOL,ICOLI,ICOLO,ID,IIN,IOUT,IROW
     INTEGER LEN,LEN2,LGLEN,LGM2,M,M2,MLP2,MN,N,N2,NT,NTL
     INTEGER RLFLG,SCLE
     MAINMEMORY
     REAL A(NTLN),B(NTLN)
     REAL WORK(NTLN)
     REAL BC(NTNTM)
     CACHEMEMORY
     REAL(C1T,AT(8192)),(C1B,AB(8192))
     REAL(C2T,BT(8192)),(C2B,BB(8192))
     REAL(C3T,CT(8192)),(C3B,CB(8192))
C
C
```

```
C
C
      NTIMES=NTNTM/NTRACE
      DELEN=NTLN/NTRACE
C
C         PHYSICAL MERGING OF MANY PROCESSES TO SPEED UP THE CODE
C            ONE OF THE LARGEST PROCESS WHICH CAN RUN ON THE STAR
C
C
CC        PROCESS ADD(NT,LEN,NTL,WORK,A)
CC        LOCALMEMORY
CC        INTEGER NT,LEN,NTL,I,ICOL
CC        MAINMEMORY
CC        REAL WORK(NTL),A(NTL)
CC        CACHEMEMORY
CC        REAL(C1T,AT(8192)),(C1B,AB(8192))
CC        REAL(C2T,BT(8192)),(C2B,BB(8192))
CC        REAL(C3T,CT(8192)),(C3B,CB(8192))
C
      NT=NTRACE
      LEN=DELEN
      NTL=NTRACE*DELEN
C
C
      CALL STSYNC(00 00 00)
C
C  LOAD FIRST TWO VECTORS
C
      CALL SMM2C(WORK(1),1,4,0,AT(1),1,LEN)
      CALL SMM2C(A(1),1,4,0,BT(1),1,LEN)
C
      CALL STSYNC(10 10 10)
C
C  AVADD FIRST COLUMN
C
      CALL AVADD(AT(1),1,BT(1),1,CT(1),1,LEN)
C
C  GET 2ND COL
C
      CALL SMM2C(WORK(1+LEN),1,4,0,AB(1),1,LEN)
      CALL SMM2C(A(1+LEN),1,4,0,BB(1),1,LEN)
C
C  MAIN PROCESS LOOPS
C
      DO 80 I = 3,NT,2
C
C  ACP -- ODD COLS; SMP -- EVEN COLS
C
      CALL STSYNC(01 01 01)
C
C  AVADD - 2'ND, 4'TH, 6'TH, ... COLS
C
      CALL AVADD(AB(1),1,BB(1),1,CB(1),1,LEN)
C
C  WRITE 1'RST, 3'RD, 5'TH. ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-3) * LEN  + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CT(1),1,LEN)
C
C  READ 3'RD, 5'TH, 7'TH, ... COLS FROM MAIN TO CACHE
```

```
C
      ICOL = (I-1) * LEN + 1
      CALL SMM2C(WORK(ICOL),1,4,0,AT(1),1,LEN)
      CALL SMM2C(A(ICOL),1,4,0,BT(1),1,LEN)
C
C  ACP -- EVEN COLS; SMP -- ODD COLS
C
      CALL STSYNC(10 10 10)
C
C  AVADD 3'RD, 5'TH, 7'TH, ... COLS
C
      CALL AVADD(AT(1),1,BT(1),1,CT(1),1,LEN)
C
C  WRITE 2'ND, 4'TH, 6'TH, ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-2) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CB(1),1,LEN)
C
C  READ 4'TH, 6'TH, 8'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = I * LEN + 1
      CALL SMM2C(WORK(ICOL),1,4,0,AB(1),1,LEN)
      CALL SMM2C(A(ICOL),1,4,0,BB(1),1,LEN)
80    CONTINUE
C
C  FLUSH DO LOOP 80
C
      CALL STSYNC(01 01 01)
      CALL AVADD(AB(1),1,BB(1),1,CB(1),1,LEN)
C
C  MOVE NEXT TO LAST COL TO MAIN
C
      ICOL = (NT-2) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CT(1),1,LEN)
C
C  MOVE LAST COL TO MAIN
C
      CALL STSYNC(00 00 00)
      ICOL = (NT-1) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CB(1),1,LEN)
CC      CALL STWAP
CC      RETURN
CC      END
C
CC      PROCESS STBC(NT,DLEN,DNTL,WORK,BCLEN,BCNTL,BC,ID)
CC      LOCALMEMORY
CC      INTEGER NT,DLEN,BCLEN,DNTL,BCNTL,I,ID
CC      INTEGER IIN,IOUT
CC      MAINMEMORY
CC      REAL WORK(DNTL),BC(BCNTL)
CC      CACHEMEMORY
CC      REAL(C1,AT(8192))
CC      REAL(C2,BT(8192))
CC      REAL(C3,CT(8192))
C
      NT=NTRACE
      DLEN=DELEN
      BCLEN=NTIMES
      DNTL=NTRACE*DELEN
      BCNTL=NTRACE*NTIMES
```

```
C
      CALL STSYNC(00 00 00)
C
      DO 90 I = 1,NT
      IIN=(I-1)*DLEN+1
      IOUT=(I-1)*BCLEN+ID
      CALL SMM2C(WORK(IIN),1,4,0,AT(1),1,1)
      CALL SMM2C(BC(IOUT),1,4,0,BT(1),1,1)
C
      CALL STSYNC(11 11 11)
C
C  AVADD BC INTO COLUMNS
C
      CALL AVADD(AT(1),1,BT(1),1,CT(1),1,1)
C
C  WRITE  COLS FROM CACHE TO MAIN
C
      CALL STSYNC(00 00 00)
      CALL SMC2M(WORK(IIN),1,4,0,CT(1),1,1)
90    CONTINUE
CC      CALL STWAP
CC      RETURN
CC      END
C
CC      PROCESS STTMOV(NT,NTL,A,B,WORK)
CC      LOCALMEMORY
CC      INTEGER NT,NTL,I,ICOL,LEN
CC      MAINMEMORY
CC      REAL A(NTL),B(NTL),C(NTL)
CC      CACHEMEMORY
CC      REAL(C1,AT(16384))
CC      REAL(C2,BT(16384))
CC      REAL(C3,CT(16384))
C
      NT=NTRACE
      NTL=NTRACE*DELEN
      LEN=NTL/NT
C
      CALL STSYNC(00 00 00)
C
C        ROTATE CIRCULAR BUFFER
C
      DO 100 I = 1,NT
      ICOL=(I-1)*LEN+1
      CALL SMM2C(B(ICOL),1,4,0,AT(1),1,LEN)
      CALL SMM2C(WORK(ICOL),1,4,0,BT(1),1,LEN)
      CALL SMC2M(A(ICOL),1,4,0,AT(1),1,LEN)
      CALL SMC2M(B(ICOL),1,4,0,BT(1),1,LEN)
100   CONTINUE
C
C
      CALL STWAP
      RETURN
      END
```

```
      PROCESS DERIV(B,V,K,WORK,XBUF,NTRACE,NTLN,KXKZP4,IKX2,
     *LGLEN1,LGLEN2,S1,LOGICL)
      LOCALMEMORY
      INTEGER NTLN,KXKZP4,IKX2,NTRACE,IKX
      INTEGER LOGICL,LGLEN1,LGLEN2
      INTEGER IKZ,DELEN,IKZP4
      INTEGER BCLEN,BCNTL,BRSLEN,BRSM,BRSM2,CM2,CN,CNL2,CNM2
      INTEGER DF,DLEN,DNTL,I,ICOL,ICOLI,ICOLO,ID,IIN,IOUT,IROW
      INTEGER LEN,LEN2,LGLEN,LGM2,M,M2,MLP2,MN,N,N2,NT,NTL
      INTEGER RLFLG,SCLE
      REAL S1,SCALAR
      MAINMEMORY
      REAL B(NTLN),V(NTLN)
      REAL K(KXKZP4),WORK(KXKZP4)
      REAL XBUF(IKX2)
      CACHEMEMORY
      REAL(C1T,AT(8192)),(C1B,AB(8192))
      REAL(C2T,BT(8192)),(C2B,BB(8192))
      REAL(C3T,CT(8192)),(C3B,CB(8192))
C
C         TAKE THE DERIVATIVE IN THE SPATIAL FOURIER DOMAIN
C            PROCESS IS A PHYSICAL MERGE OF MANY PROCESSES TO SPEED UP
C            EXECUTION
C
      IKX=IKX2/2
      IKZP4=KXKZP4/IKX
      IKZ=IKZP4-4
      DELEN=NTLN/NTRACE
C
C
C
      SCALAR=S1
      IF(LOGICL.EQ.0)GOTO 99
      SCALAR=S1/2
      LOGICL=0
99    CONTINUE
C
C
C
CC       PROCESS CLRMM( WORK, CNM2, N, M2 )
CC       LOCALMEMORY
CC       INTEGER CNM2, N, M2, I, ICOL
CC       MAINMEMORY
CC       REAL WORK(CNM2)
C
       CNM2=IKX*(IKZ+4)
       N=IKZ+4
       M2=IKX
C
C
C                 CLEAR ROWS BEFORE LOADING AND FOURIER TRANSFORMING
C
       CALL STSYNC( 000000 )
       DO 2 ICOL = 1, M2
         I = ( ICOL-1 ) * N + 1
         CALL SCLRMM( WORK(I), N )
2      CONTINUE
C
C                 WAIT FOR COMPLETION AND RETURN
C
```

```
CC        CALL STWAP
CC        RETURN
CC        END
C
CC       PROCESS RFTCOL(M,MN,B,LEN,WORK,MLP2,LGLEN)
C
C  DO COLUMN RFFTS IN ST100
C  WITH INTERNAL ZERO-PADDING.
C
C  INPUT:
C         B(N,M)= INPUT ARRAY; N & M MUST BE EVEN (BECAUSE OF DOUBLE
C  M MUST BE >= 4 TO ALLOW PIPELINE TO BE SET UP.
C  THE TRANSFORM IS DONE OVER THE 'M' REAL COLUMNS OF 'XIN'.
C  I.E. - IN THE 'N' DIRECTION. THE PROCESS WILL ZERO-PAD
C  THE INPUT LENGTH, N, TO LENGTH, LEN, BEFORE TRANSFORMING
C  EACH COLUMN.
C         MLEN          = M * LEN
C         LGLEN           = LOG2 ( LEN )
C  WITH         LEN          = DESIRED OUTPUT VECTOR TRANSFORM LENGTH
C                     (MUST BE POWER OF 2)
C
C  OUTPUT:
C         WORK(LEN+4,M) = OUTPUT COLUMN FFT ARRAY IN PACKED FORMAT
C
CC       LOCALMEMORY
CC       INTEGER N,M,LEN,LGLEN,DF,MLP2,MN
CC       INTEGER SCLE,BRSLEN,N2,LEN2,RLFLG,ICOL,I
CC       MAINMEMORY
CC       REAL B(MN),WORK(MLP2)
CC       CACHEMEMORY
CC       REAL(C1T,AT(8192)),(C1B,AB(8192))
CC       REAL(C2T,BT(8192)),(C2B,BB(8192))
CC       REAL(C3T,CT(8192)),(C3B,CB(8192))
C
      M=NTRACE
      LEN=IKZ
      LGLEN=LGLEN1
      MLP2=IKX*(IKZ+4)
      MN=NTRACE*DELEN
C
      DF = 1
      RLFLG = 1
      SCLE = 1
      N = MN/M
      BRSLEN = 30 + LGLEN
      N2 = N/2
      LEN2 = LEN/2
C
      CALL STSYNC(00 00 00)
C
C  LOAD SN/CS TABLE IN CACHE
C
      CALL SMSTMC(0,CB,CT,LEN2,BRSLEN)
C
C  GET M'TH COLUMN;
C  NEED TO DO TRIPLE XFER (MAIN - CACHE - MAIN - CACHE)
C                TO EFFECT ZERO-PADDING
C
      ICOL = (M-1) * (LEN+4) + 1
      CALL SMXMC2(B((M-1)*N+1),AT(1),BT(1),N2)
```

```
      CALL SCLRMM(WORK(ICOL),LEN)
      CALL SMXCM2(WORK(ICOL),AT(1),BT(1),N2)
      CALL SXMC2B(WORK(ICOL),AT(1),BT(1),LEN2,BRSLEN)
C
      CALL STSYNC(10 10 11)
C
C  RFFT M'TH COLUMN
C
      CALL FFTB(AT(1),BT(1),CB,CT,LGLEN,DF,SCLE,RLFLG)
      CALL RFFTPK(AT(1),BT(1),LGLEN,1)
C
C  GET (M-1)TH COL
C
      ICOL = (M-2) * (LEN+4) +1
      CALL SMXMC2(B((M-2)*N+1),AB(1),BB(1),N2)
      CALL SCLRMM(WORK(ICOL),LEN)
      CALL SMXCM2(WORK(ICOL),AB(1),BB(1),N2)
      CALL SXMC2B(WORK(ICOL),AB(1),BB(1),LEN2,BRSLEN)
C
C  MAIN PROCESS LOOPS
C
      DO 10 I = 1,M-2,2
C
C  ACP -- ODD COLS; SMP -- EVEN COLS
C
      CALL STSYNC(01 01 11)
C
C  RFFT - (M-1)'TH, (M-3)'RD, (M-5)TH, ... COLS
C
      CALL FFTB(AB(1),BB(1),CB,CT,LGLEN,DF,SCLE,RLFLG)
      CALL RFFTPK(AB(1),BB(1),LGLEN,1)
C
C  WRITE M'TH, (M-2)'ND, (M-4)'TH. ... COLS FROM CACHE TO MAIN
C
      ICOL = (M-I) * (LEN+4)  + 1
      CALL SMXCM2(WORK(ICOL),AT(1),BT(1),LEN2+1)
C
C  READ (M-2)ND, (M-4)'TH, (M-6)'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = (M-I-2) * (LEN+4) + 1
      CALL SMXMC2(B((M-I-2)*N+1),AT(1),BT(1),N2)
      CALL SCLRMM(WORK(ICOL),LEN)
      CALL SMXCM2(WORK(ICOL),AT(1),BT(1),N2)
      CALL SXMC2B(WORK(ICOL),AT(1),BT(1),LEN2,BRSLEN)
C
C  ACP -- EVEN COLS; SMP -- ODD COLS
C
      CALL STSYNC(10 10 11)
C
C  RFFT (M-2)ND, (M-4)'TH, (M-6)'TH, ... COLS
C
      CALL FFTB(AT(1),BT(1),CB,CT,LGLEN,DF,SCLE,RLFLG)
      CALL RFFTPK(AT(1),BT(1),LGLEN,1)
C
C  WRITE (M-1)TH, (M-3)RD, (M-5)TH, ... COLS FROM CACHE TO MAIN
C
      ICOL = (M-I-1) * (LEN+4) + 1
      CALL SMXCM2(WORK(ICOL), AB(1), BB(1), LEN2+1)
C
C  READ (M-3)RD,(M-5)TH,(M-7)TH, ... COLS FROM MAIN TO CACHE
```

```
C
      ICOL = (M-I-3) * (LEN+4) + 1
      CALL SMXMC2(B((M-I-3)*N+1),AB(1),BB(1),N2)
      CALL SCLRMM(WORK(ICOL),LEN)
      CALL SMXCM2(WORK(ICOL),AB(1),BB(1),N2)
      CALL SXMC2B(WORK(ICOL),AB(1),BB(1),LEN2,BRSLEN)
10    CONTINUE
C
C  FLUSH DO LOOP 10
C
      CALL STSYNC(01 01 11)
      CALL FFTB(AB(1), BB(1), CB, CT, LGLEN, DF, SCLE, RLFLG)
      CALL RFFTPK(AB(1),BB(1),LGLEN,1)
C
C  MOVE 2'ND COL TO MAIN
C
      ICOL =  LEN+4 + 1
      CALL SMXCM2(WORK(ICOL), AT(1), BT(1), LEN2+1)
C
C  MOVE LAST COL TO MAIN
C
      CALL STSYNC(00 00 00)
      CALL SMXCM2(WORK(1), AB(1), BB(1), LEN2+1)
CC      CALL STWAP
CC      RETURN
CC      END
CC
CC      PROCESS ROWFFT(N,CNM2,XIN,M2,CM2,XBUF,LGM2,DF)
C
C  DO ROW FFTS IN ST100
C
C  INPUT:
C        WORK(N,M)= INPUT ARRAY; N MUST BE EVEN (BECAUSE OF DOUBLE
C  THE TRANSFORM IS DONE OVER THE 'N' ROWS OF 'XIN'.
C  I.E. - IN THE 'M' DIRECTION.
C        CM2          = 2 * M2
C        CNM2          = 2 * N * M2
C        M2          = DESIRED OUTPUT TRANSFORM LENGTH
C                     (MUST BE POWER OF 2)
C        LGM2          = LOG2 ( M2 )
C        DF          = DIRECTION FLAG FOR TRANSFORM
C                  = 1 FOR FORWARD TRANSFORM
C                  =-1 FOR INVERSE TRANSFORM
C  OUTPUT:
C        XOUT( N,M2 )  REPLACES  WORK( N,M2 )
C
C
CC      LOCALMEMORY
CC      INTEGER N,CNM2,M2,M,LGM2,DF,BRSM
CC      INTEGER SCLE,BRSM2,RLFLG,IROW,I,CN,CM2
CC      MAINMEMORY
CC      REAL WORK(CNM2),XBUF(CM2)
CC      CACHEMEMORY
CC      REAL(C1T,AT(8192)),(C1B,AB(8192))
CC      REAL(C2T,BT(8192)),(C2B,BB(8192))
CC      REAL(C3,CB(8192),CT(8192))
C
      N=IKZ/2+2
      CNM2=2*(IKZ/2+2)*IKX
      M2=IKX
```

```
      LGM2=LGLEN2
      DF=1
      CM2=2*IKX
C
C
C
      RLFLG = 0
      SCLE = 0
      IF(DF.EQ.1)SCLE=1
      BRSM2 = 31 + LGM2
      CN = 2 * N
      M = M2 / 2
      BRSM = 30 + LGM2
C
C               LOAD SN/CS TABLE IN CACHE AND
C               CLEAR MAIN MEM VECTOR BUFFER
C
      CALL STSYNC(00 00 00)
      CALL SMSTMC(0,CB,CT,M,BRSM)
C
C               MUST DO TRIPLE XFER ( MAIN - CACHE - MAIN - CACHE )
C                       TO OBTAIN ROW VECTORS
C
      CALL SMM2C(WORK(1),CN,4,0,AT,1,M2)
      CALL SMM2C(WORK(2),CN,4,0,BT,1,M2)
C
C               FFT FIRST ROW
C
      CALL STSYNC(10 10 11)
      CALL FFTN(AT,BT,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C               GET 2ND ROW
C
      CALL SMM2C(WORK(3),CN,4,0,AB,1,M2)
      CALL SMM2C(WORK(4),CN,4,0,BB,1,M2)
C
C               MAIN PROCESS LOOPS
C
      DO 20 I = 3,N,2
C
C               ACP -- EVEN ROWS; SMP -- ODD ROWS
C
      CALL STSYNC(01 01 11)
C
C               FFT - 2'ND, 4'TH, 6'TH, ... ROWS
C
      CALL FFTN(AB,BB,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C               WRITE 1'RST, 3'RD, 5'TH. ... ROWS FROM CACHE TO MAIN
C
      IROW = (I-3) * 2  + 1
      CALL SXCM2B(XBUF(1),AT(1),BT(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AT,BT,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AT,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BT,1,M2)
C
C               READ 3'RD, 5'TH, 7'TH, ... ROWS FROM MAIN TO CACHE
C
      IROW = (I-1) * 2 + 1
      CALL SMM2C(WORK(IROW),CN,4,0,AT,1,M2)
```

```
      CALL SMM2C(WORK(IROW+1),CN,4,0,BT,1,M2)
C
C                 ACP -- ODD ROWS; SMP -- EVEN ROWS
C
      CALL STSYNC(10 10 11)
C
C                 FFT 3'RD, 5'TH, 7'TH, ... ROWS
C
      CALL FFTN(AT,BT,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C                 WRITE 2'ND, 4'TH, 6'TH, ... ROWS FROM CACHE TO MAIN
C
      IROW = (I-2) * 2 + 1
      CALL SXCM2B(XBUF(1),AB(1),BB(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AB,BB,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AB,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BB,1,M2)
C
C                 READ 4'TH, 6'TH, 8'TH, ... ROWS FROM MAIN TO CACHE
C
      IROW = I * 2 + 1
      CALL SMM2C(WORK(IROW),CN,4,0,AB,1,M2)
      CALL SMM2C(WORK(IROW+1),CN,4,0,BB,1,M2)
20    CONTINUE
C
C                 FLUSH DO LOOP 20
C
      CALL STSYNC(01 01 11)
      CALL FFTN(AB,BB,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C                 MOVE NEXT TO LAST ROW TO MAIN
C
      IROW = (N-2) * 2 + 1
      CALL SXCM2B(XBUF(1),AT(1),BT(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AT,BT,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AT,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BT,1,M2)
C
C                 MOVE LAST ROW TO MAIN
C
      CALL STSYNC(00 00 00)
      IROW = (N-1) * 2 + 1
      CALL SXCM2B(XBUF(1),AB(1),BB(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AB,BB,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AB,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BB,1,M2)
CC      CALL STWAP
CC      RETURN
CC      END
C
CC      PROCESS CVMUL(NT,LEN,CNL2,WORK,K)
CC      LOCALMEMORY
CC      INTEGER NT,LEN,CNL2,I,ICOL,LEN2
CC      MAINMEMORY
CC      REAL WORK(CNL2),K(CNL2)
CC      CACHEMEMORY
CC      REAL(C1T,AT(8192)),(C1B,AB(8192))
CC      REAL(C2T,BT(8192)),(C2B,BB(8192))
CC      REAL(C3T,CT(8192)),(C3B,CB(8192))
C
```

```
C          COMPLEX VECTOR MULTIPLY
C            MULTIPLY WORK(I)=WORK(I)*K(I)
C
      NT=IKX
      LEN=IKZ/2+2
      CNL2=IKX*(IKZ+4)
C
C
      LEN2=2*LEN
      CALL STSYNC(00 00 00)
C
      CALL SMXMC2(WORK(1),AT(1),BT(1),LEN)
      CALL SMXMC1(K(1),CT(1),LEN)
C
      CALL STSYNC(10 10 10)
C
C  ACVM FIRST COLUMN
C
      CALL ACVM(AT(1),BT(1),1,CT(1),1,AT(1),BT(1),1,LEN,0)
C
C  GET 2ND COL
C
      CALL SMXMC2(WORK(1+LEN2),AB(1),BB(1),LEN)
      CALL SMXMC1(K(1+LEN2),CB(1),LEN)
C
C  MAIN PROCESS LOOPS
C
      DO 30 I = 3,NT,2
C
C  ACP -- ODD COLS; SMP -- EVEN COLS
C
      CALL STSYNC(01 01 01)
C
C  ACVM - 2'ND, 4'TH, 6'TH, ... COLS
C
      CALL ACVM(AB(1),BB(1),1,CB(1),1,AB(1),BB(1),1,LEN,0)
C
C  WRITE 1'RST, 3'RD, 5'TH. ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-3) * LEN2  + 1
      CALL SMXCM2(WORK(ICOL),AT(1),BT(1),LEN)
C
C  READ 3'RD, 5'TH, 7'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = (I-1) * LEN2 + 1
      CALL SMXMC2(WORK(ICOL),AT(1),BT(1),LEN)
      CALL SMXMC1(K(ICOL),CT(1),LEN)
C
C  ACP -- EVEN COLS; SMP -- ODD COLS
C
      CALL STSYNC(10 10 10)
C
C  ACVM 3'RD, 5'TH, 7'TH, ... COLS
C
      CALL ACVM(AT(1),BT(1),1,CT(1),1,AT(1),BT(1),1,LEN,0)
C
C  WRITE 2'ND, 4'TH, 6'TH, ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-2) * LEN2 + 1
      CALL SMXCM2(WORK(ICOL),AB(1),BB(1),LEN)
```

```
C
C  READ 4'TH, 6'TH, 8'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = I * LEN2 + 1
      CALL SMXMC2(WORK(ICOL),AB(1),BB(1),LEN)
      CALL SMXMC1(K(ICOL),CB(1),LEN)
30    CONTINUE
C
C  FLUSH DO LOOP 30
C
      CALL STSYNC(01 01 01)
      CALL ACVM(AB(1),BB(1),1,CB(1),1,AB(1),BB(1),1,LEN,0)
C
C  MOVE NEXT TO LAST COL TO MAIN
C
      ICOL = (NT-2) * LEN2 + 1
      CALL SMXCM2(WORK(ICOL),AT(1),BT(1),LEN)
C
C  MOVE LAST COL TO MAIN
C
      CALL STSYNC(00 00 00)
      ICOL = (NT-1) * LEN2 + 1
      CALL SMXCM2(WORK(ICOL),AB(1),BB(1),LEN)
CC      CALL STWAP
CC      RETURN
CC      END
C
CC       PROCESS ROWFFT(N,CNM2,WORK,M2,CM2,XBUF,LGM2,DF)
C
C  DO ROW FFTS IN ST100
C
C  INPUT:
C        XIN(N,M)= INPUT ARRAY; N MUST BE EVEN (BECAUSE OF DOUBLE
C  THE TRANSFORM IS DONE OVER THE 'N' ROWS OF 'XIN'.
C  I.E. - IN THE 'M' DIRECTION.
C        CM2         = 2 * M2
C        CNM2         = 2 * N * M2
C        M2         = DESIRED OUTPUT TRANSFORM LENGTH
C                     (MUST BE POWER OF 2)
C        LGM2          = LOG2 ( M2 )
C        DF         = DIRECTION FLAG FOR TRANSFORM
C                 = 1 FOR FORWARD TRANSFORM
C                 =-1 FOR INVERSE TRANSFORM
C  OUTPUT:
C        XOUT( N,M2 )  REPLACES  WORK( N,M2 )
C
C
CC       LOCALMEMORY
CC       INTEGER N,CNM2,M2,M,LGM2,DF,BRSM
CC       INTEGER SCLE,BRSM2,RLFLG,IROW,I,CN,CM2
CC       MAINMEMORY
CC       REAL WORK(CNM2),XBUF(CM2)
CC       CACHEMEMORY
CC       REAL(C1T,AT(8192)),(C1B,AB(8192))
CC       REAL(C2T,BT(8192)),(C2B,BB(8192))
CC       REAL(C3,CB(8192),CT(8192))
C
C
      N=IKZ/2+2
      CNM2=2*(IKZ/2+2)*IKX
```

```
      M2=IKX
      LGM2=LGLEN2
      DF=-1
      CM2=2*IKX
C
      RLFLG = 0
      SCLE = 0
      IF(DF.EQ.1)SCLE=1
      BRSM2 = 31 + LGM2
      CN = 2 * N
      M = M2 / 2
      BRSM = 30 + LGM2
C
C              LOAD SN/CS TABLE IN CACHE AND
C              CLEAR MAIN MEM VECTOR BUFFER
C
      CALL STSYNC(00 00 00)
      CALL SMSTMC(0,CB,CT,M,BRSM)
C
C              MUST DO TRIPLE XFER ( MAIN - CACHE - MAIN - CACHE )
C                     TO OBTAIN ROW VECTORS
C
      CALL SMM2C(WORK(1),CN,4,0,AT,1,M2)
      CALL SMM2C(WORK(2),CN,4,0,BT,1,M2)
C
C              FFT FIRST ROW
C
      CALL STSYNC(10 10 11)
      CALL FFTN(AT,BT,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C              GET 2ND ROW
C
      CALL SMM2C(WORK(3),CN,4,0,AB,1,M2)
      CALL SMM2C(WORK(4),CN,4,0,BB,1,M2)
C
C              MAIN PROCESS LOOPS
C
      DO 40 I = 3,N,2
C
C              ACP -- EVEN ROWS; SMP -- ODD ROWS
C
      CALL STSYNC(01 01 11)
C
C              FFT - 2'ND, 4'TH, 6'TH, ... ROWS
C
      CALL FFTN(AB,BB,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C              WRITE 1'RST, 3'RD, 5'TH. ... ROWS FROM CACHE TO MAIN
C
      IROW = (I-3) * 2  + 1
      CALL SXCM2B(XBUF(1),AT(1),BT(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AT,BT,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AT,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BT,1,M2)
C
C              READ 3'RD, 5'TH, 7'TH, ... ROWS FROM MAIN TO CACHE
C
      IROW = (I-1) * 2 + 1
      CALL SMM2C(WORK(IROW),CN,4,0,AT,1,M2)
      CALL SMM2C(WORK(IROW+1),CN,4,0,BT,1,M2)
```

```
C
C               ACP -- ODD POWS; SMP -- EVEN ROWS
C
      CALL STSYNC(10 10 11)
C
C               FFT 3'RD, 5'TH, 7'TH, ... ROWS
C
      CALL FFTN(AT,BT,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C               WRITE 2'ND, 4'TH, 6'TH, ... ROWS FROM CACHE TO MAIN
C
      IROW = (I-2) * 2 + 1
      CALL SXCM2B(XBUF(1),AB(1),BB(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AB,BB,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AB,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BB,1,M2)
C
C               READ 4'TH, 6'TH, 8'TH, ... ROWS FROM MAIN TO CACHE
C
      IROW = I * 2 + 1
      CALL SMM2C(WORK(IROW),CN,4,0,AB,1,M2)
      CALL SMM2C(WORK(IROW+1),CN,4,0,BB,1,M2)
40    CONTINUE
C
C               FLUSH DO LOOP 40
C
      CALL STSYNC(01 01 11)
      CALL FFTN(AB,BB,CB,CT,LGM2,DF,SCLE,RLFLG)
C
C               MOVE NEXT TO LAST ROW TO MAIN
C
      IROW = (N-2) * 2 + 1
      CALL SXCM2B(XBUF(1),AT(1),BT(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AT,BT,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AT,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BT,1,M2)
C
C               MOVE LAST ROW TO MAIN
C
      CALL STSYNC(00 00 00)
      IROW = (N-1) * 2 + 1
      CALL SXCM2B(XBUF(1),AB(1),BB(1),M2,BRSM2)
      CALL SMXMC2(XBUF,AB,BB,M2)
      CALL SMC2M(WORK(IROW),CN,4,0,AB,1,M2)
      CALL SMC2M(WORK(IROW+1),CN,4,0,BB,1,M2)
CC      CALL STWAP
CC      RETURN
CC      END
C
CC      PROCESS IFTCOL(M,N,LEN,WORK,MLP2,LGLEN)
C
C        INVERSE RFT COLUMN TRANSFORM
C
CC      LOCALMEMORY
CC      INTEGER N,M,LEN,LGLEN,DF,MLP2,ICOLI
CC      INTEGER SCLE,BRSLEN,N2,LEN2,RLFLG,ICOLO,I
CC      MAINMEMORY
CC      REAL WORK(MLP2)
CC      CACHEMEMORY
CC      REAL(C1T,AT(8192)),(C1B,AB(8192))
```

```
CC       REAL(C2T,BT(8192)),(C2B,BB(8192))
CC       REAL(C3T,CT(8192)),(C3B,CB(8192))
C
      N=DELEN
      M=NTRACE
      LEN=IKZ
      LGLEN=LGLEN1
      MLP2=NTRACE*(IKZ+4)
C
      DF = -1
      RLFLG = 1
      SCLE = 0
      BRSLEN = 30 + LGLEN
      N2 = N/2
      LEN2 = LEN/2
C
      CALL STSYNC(00 00 00)
C
C  LOAD SN/CS TABLE IN CACHE
C
      CALL SMSTMC(0,CB,CT,LEN2,BRSLEN)
C
C  GET FIRST COLUMN;
C
      CALL SMXMC2(WORK(1),AT(1),BT(1),LEN2+1)
C
      CALL STSYNC(10 10 11)
C
C  RFFT FIRST COLUMN
C
      CALL RFFTPK(AT(1),BT(1),LGLEN,0)
      CALL FFTN(AT(1),BT(1),CB,CT,LGLEN,DF,SCLE,RLFLG)
C
C  GET 2ND COL
C
      CALL SMXMC2(WORK(LEN+4+1),AB(1),BB(1),LEN2+1)
C
C  MAIN PROCESS LOOPS
C
      DO 50 I = 3,M,2
C
C  ACP -- ODD COLS; SMP -- EVEN COLS
C
      CALL STSYNC(01 01 11)
C
C  RFFT - 2'ND, 4'TH, 6'TH, ... COLS
C
      CALL RFFTPK(AB(1),BB(1),LGLEN,0)
      CALL FFTN(AB(1),BB(1),CB,CT,LGLEN,DF,SCLE,RLFLG)
C
C  WRITE 1'RST, 3'RD, 5'TH. ... COLS FROM CACHE TO MAIN
C
      ICOLO = (I-3) * (N)  + 1
      CALL SXCM2B(WORK(ICOLO),AT(1),BT(1),LEN2,BRSLEN)
      CALL SMXMC2(WORK(ICOLO),AT(1),BT(1),N2)
      CALL SMXCM2(WORK(ICOLO),AT(1),BT(1),N2)
C
C  READ 3'RD, 5'TH, 7'TH, ... COLS FROM MAIN TO CACHE
C
      ICOLI = (I-1) * (LEN+4) + 1
```

```
      CALL SMXMC2(WORK(ICOLI),AT(1),BT(1),LEN2+1)
C
C  ACP -- EVEN COLS; SMP -- ODD COLS
C
      CALL STSYNC(10 10 11)
C
C  RFFT 3'RD, 5'TH, 7'TH, ... COLS
C
      CALL RFFTPK(AT(1),BT(1),LGLEN,0)
      CALL FFTN(AT(1),BT(1),CB,CT,LGLEN,DF,SCLE,RLFLG)
C
C  WRITE 2'ND, 4'TH, 6'TH, ... COLS FROM CACHE TO MAIN
C
      ICOLO = (I-2) * (N) + 1
      CALL SXCM2B(WORK(ICOLO),AB(1),BB(1),LEN2,BRSLEN)
      CALL SMXMC2(WORK(ICOLO),AB(1),BB(1),N2)
      CALL SMXCM2(WORK(ICOLO),AB(1),BB(1),N2)
C
C  READ 4'TH, 6'TH, 8'TH, ... COLS FROM MAIN TO CACHE
C
      ICOLI = I * (LEN+4) + 1
      CALL SMXMC2(WORK(ICOLI),AB(1),BB(1),LEN2+1)
50    CONTINUE
C
C  FLUSH DO LOOP 50
C
      CALL STSYNC(01 01 11)
      CALL RFFTPK(AB(1),BB(1),LGLEN,0)
      CALL FFTN(AB(1), BB(1), CB, CT, LGLEN, DF, SCLE, RLFLG)
C
C  MOVE NEXT TO LAST COL TO MAIN
C
      ICOLO = (M-2)*N+1
      CALL SXCM2B(WORK(ICOLO), AT(1), BT(1), LEN2, BRSLEN)
      CALL SMXMC2(WORK(ICOLO), AT(1), BT(1), N2)
      CALL SMXCM2(WORK(ICOLO), AT(1), BT(1), N2)
C
C  MOVE LAST COL TO MAIN
C
      CALL STSYNC(00 00 00)
      ICOLO = (M-1)*N+1
      CALL SXCM2B(WORK(ICOLO), AB(1), BB(1), LEN2, BRSLEN)
      CALL SMXMC2(WORK(ICOLO), AB(1), BB(1), LEN2)
      CALL SMXCM2(WORK(ICOLO), AB(1), BB(1), N2)
CC        CALL STWAP
CC        RETURN
CC        END
C
CC        PROCESS SMUL(NT,LEN,NTL,SCALAR,WORK)
CC        LOCALMEMORY
CC        INTEGER NT,LEN,NTL,I,ICOL
CC        REAL SCALAR
CC        MAINMEMORY
CC        REAL WORK(NTL)
CC        CACHEMEMORY
CC        REAL(C1T,AT(8192)),(C1B,AB(8192))
CC        REAL(C2T,BT(8192)),(C2B,BB(8192))
CC        REAL(C3T,CT(8192)),(C3B,CB(8192))
C
C             MULTIPLY WORK(I)=SCALAR*WORK(I)
```

```
C          SCALAR IS TRANSFERED FROM LOCAL TO CACHE MEMORY
C
      NT=NTRACE
      LEN=DELEN
      NTL=NTRACE*DELEN
C
C
      CALL STSYNC(0C 00 00)
C
      CALL SMM2C(WORK(1),1,4,0,AT(1),1,LEN)
      CALL STWRCM(SCALAR,BT(1))
C
      CALL STSYNC(10 10 10)
C
C  AVSMUL FIRST COLUMN
C
      CALL AVSMUL(AT(1),1,BT(1),CT(1),1,LEN)
C
C  GET 2ND COL
C
      CALL SMM2C(WORK(1+LEN),1,4,0,AB(1),1,LEN)
      CALL STWRCM(SCALAR,BB(1))

C
C  MAIN PROCESS LOOPS
C
      DO 60 I = 3,NT,2
C
C  ACP -- ODD COLS; SMP -- EVEN COLS
C
      CALL STSYNC(01 01 01)
C
C  AVSMUL - 2'ND, 4'TH, 6'TH, ... COLS
C
      CALL AVSMUL(AB(1),1,BB(1),CB(1),1,LEN)
C
C  WRITE 1'RST, 3'RD, 5'TH. ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-3) * LEN  + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CT(1),1,LEN)
C
C  READ 3'RD, 5'TH, 7'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = (I-1) * LEN + 1
      CALL SMM2C(WORK(ICOL),1,4,0,AT(1),1,LEN)
      CALL STWRCM(SCALAR,BT(1))
C
C  ACP -- EVEN COLS; SMP -- ODD COLS
C
      CALL STSYNC(10 10 10)
C
C  AVSMUL 3'RD, 5'TH, 7'TH, ... COLS
C
      CALL AVSMUL(AT(1),1,BT(1),CT(1),1,LEN)
C
C  WRITE 2'ND, 4'TH, 6'TH, ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-2) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CB(1),1,LEN)
C
```

```
C  READ 4'TH, 6'TH, 8'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = I * LEN + 1
      CALL SMM2C(WORK(ICOL),1,4,0,AB(1),1,LEN)
C      CALL SMM2C(S(1),1,4,0,BB(1),1,1)
      CALL STWRCM(SCALAR,BB(1))
60    CONTINUE
C
C  FLUSH DO LOOP 60
C
      CALL STSYNC(01 01 01)
      CALL AVSMUL(AB(1),1,BB(1),CB(1),1,LEN)
C
C  MOVE NEXT TO LAST COL TO MAIN
C
      ICOL = (NT-2) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CT(1),1,LEN)
C
C  MOVE LAST COL TO MAIN
C
      CALL STSYNC(00 00 00)
      ICOL = (NT-1) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CB(1),1,LEN)
CC      CALL STWAP
CC      RETURN
CC      END
C
CC      PROCESS MUL(NT,LEN,NTL,WORK,V)
CC      LOCALMEMORY
CC      INTEGER NT,LEN,NTL,I,ICOL
CC      MAINMEMORY
CC      REAL WORK(NTL),V(NTL)
CC      CACHEMEMORY
CC      REAL(C1T,AT(8192)),(C1B,AB(8192))
CC      REAL(C2T,BT(8192)),(C2B,BB(8192))
CC      REAL(C3T,CT(8192)),(C3B,CB(8192))
C
C         MATRIX MULTIPLY
C              MULTIPLY WORK(I)=WORK(I)*V(I)
C
      NT=NTRACE
      LEN=DELEN
      NTL=NTRACE*DELEN
C
C
      CALL STSYNC(00 00 00)
C
      CALL SMM2C(WORK(1),1,4,0,AT(1),1,LEN)
      CALL SMM2C(V(1),1,4,0,BT(1),1,LEN)
C
      CALL STSYNC(10 10 10)
C
C  AVMUL FIRST COLUMN
C
      CALL AVMUL(AT(1),1,BT(1),1,CT(1),1,LEN)
C
C  GET 2ND COL
C
      CALL SMM2C(WORK(1+LEN),1,4,0,AB(1),1,LEN)
      CALL SMM2C(V(1+LEN),1,4,0,BB(1),1,LEN)
```

```
C
C  MAIN PROCESS LOOPS
C
      DO 70 I = 3,NT,2
C
C  ACP -- ODD COLS; SMP -- EVEN COLS
C
      CALL STSYNC(01 01 01)
C
C  AVMUL - 2'ND, 4'TH, 6'TH, ... COLS
C
      CALL AVMUL(AB(1),1,BB(1),1,CB(1),1,LEN)
C
C  WRITE 1'RST, 3'RD, 5'TH. ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-3) * LEN  + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CT(1),1,LEN)
C
C  READ 3'RD, 5'TH, 7'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = (I -1)  * LEN + 1
      CALL SMM2C(WORK(ICOL),1,4,0,AT(1),1,LEN)
      CALL SMM2C(V(ICOL),1,4,0,BT(1),1,LEN)
C
C  ACP -- EVEN COLS; SMP -- ODD COLS
C
      CALL STSYNC(10 10 10)
C
C  AVMUL 3'RD, 5'TH, 7'TH, ... COLS
C
      CALL AVMUL(AT(1),1,BT(1),1,CT(1),1,LEN)
C
C  WRITE 2'ND, 4'TH, 6'TH, ... COLS FROM CACHE TO MAIN
C
      ICOL = (I-2)  * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CB(1),1,LEN)
C
C  READ 4'TH, 6'TH, 8'TH, ... COLS FROM MAIN TO CACHE
C
      ICOL = I * LEN + 1
      CALL SMM2C(WORK(ICOL),1,4,0,AB(1),1,LEN)
      CALL SMM2C(V(ICOL),1,4,0,BB(1),1,LEN)
70    CONTINUE
C
C  FLUSH DO LOOP 70
C
      CALL STSYNC(01 01 01)
      CALL AVMUL(AB(1),1,BB(1),1,CB(1),1,LEN)
C
C  MOVE NEXT TO LAST COL TO MAIN
C
      ICOL = (NT-2) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CT(1),1,LEN)
C
C  MOVE LAST COL TO MAIN
C
      CALL STSYNC(00 00 00)
      ICOL = (NT-1) * LEN + 1
      CALL SMC2M(WORK(ICOL),1,4,0,CB(1),1,LEN)
      CALL STWAP
```

```
RETURN
END
```