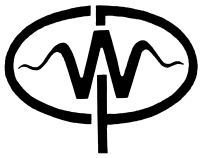
CWP-228
August 1996



3-D Common Offset Inversion in Depth Dependent Media and Its Parallel Implementation

Meng Xu

— Master's Thesis —
Mathematical and Computer Sciences

Center for Wave Phenomena
Colorado School of Mines
Golden, Colorado 80401
303/273-3557

ABSTRACT

For complex geologic structures, in order to preserve whatever structural and amplitude information is in the data, three dimensional inversion and prestack processing should be performed. The objective of the inversion is to obtain correct locations of the interfaces and return a true amplitude (angularly dependent reflection coefficient) consistent with the underlying theory of wave propagation. Given scalar data and the velocity above a reflector, prestack Kirchhoff inversion resolves the location of the interface. When amplitude information has been preserved in the data, the method additionally yields the reflection coefficient at each interface point.

The common-offset-inversion development here is based on high-frequency asymptotic theory. When Kirchhoff modeling data is inserted into a general 3-D inversion operator, asymptotically evaluating the integrals by the method of stationary phase permits an inversion amplitude function to be chosen so that the inversion operator produces a singular function of support on the reflector, weighted by the reflection coefficient.

In a depth-dependent background medium, traveltime and amplitude depend on only offset and depth; tables can be set up containing traveltime and amplitude information as functions of offset and depth. Lateral homogeneity allows us to use this pair of tables for all source/receiver pairs. Thus, the computing times for a 3-D inversion in the $v(z)$ case are only marginally more expensive than for the constant-background velocity algorithms. Moreover, in this implementation, the inversion algorithm is adapted into parallel schemes to achieve a further speedup.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor, Dr. Norman Bleistein, for his advice, guidance and support during my time at the Center for Wave Phenomena. I am indebted to Dr. Ken Larner and Dr. Erik Van Vleck for serving on my committee. I would also like to thank Zhenyue Liu and Mikki Hand for their assistance in this research effort.

I gratefully acknowledge the support by the members of the Consortium Project on Seismic Inversion Methods for Complex Structures at the Center for Wave Phenomena, Colorado School of Mines and the Ocean Acoustics program at the Office of Naval Research. I would also like to acknowledge Los Alamos National Laboratory for providing computer (**CM5**) resources.

Chapter 1

INTRODUCTION

The purpose of this project is to implement the 3-D inversion theory of Bleistein [1986] for a depth-dependent background. Liu [1993] developed a 2-D prestack migration code where he used finite difference method to solve only Eikonal equation. He did not consider amplitude information. Sumner [1990] made a CZ migration program (zero offset, depth dependent background velocity) for 2.5-D [Bleistein et al., 1987] where the amplitude calculation can be largely simplified since parameters along one direction do not count. Currently there is no 3-D inversion code in CWP (Center for Wave Phenomina) that treats amplitude properly. I derive a ray tracing method that solves the ray equations for the parameters that give a relatively true amplitude.

The code developed here is based on the Kirchhoff integral inversion method for common-offset seismic data [Bleistein et al., 1986]. This method treats amplitude in inversion in a WKBJ-consistent manner so that the output is the reflectivity function. Furthermore, Bleistein showed that, based on the stationary-phase principle, one can design weights in the Kirchhoff integral to determine quantities such as the specular reflection angle. The output provides an estimate of the angularly dependent reflection coefficient at specular at the sample points on the scatterer, as well as an estimate of that incident specular angle with respect to the normal to the reflector. By processing the data for a suite of offsets, data for amplitude versus offset (AVO) or amplitude versus angle (AVA) analysis is generated for each point on the reflector.

In Bleistein's integral formulation, the integral requires ray data: traveltime, WKBJ amplitude and other ray parameters for a background model. Two commonly used approaches to calculating traveltime and the other quantities in the Kirchhoff integral are ray tracing and finite differences applied to the Eikonal equation. In a general 3-D case, finite-difference methods are much faster than is ray tracing. For a depth-dependent medium, however, ray equations can be solved analytically, so that ray tracing implementation is quite efficient. Moreover, traveltime and other ray parameters depend on only depth and offset, which means that tables of these quantities need to be calculated only once as a function of lateral offset; tables for all shot or receiver positions are merely shifts and interpolations from the original tables. Compared to a general velocity 3-D inversion, depth-dependent inversion costs much less since the computation cost of traveltime and amplitude for every trajectory from surface grid points to subsurface grid points could dominate the total computation cost of the Kirchhoff integral method in the general background case. Even so, generating ray data for the depth dependent medium is still costly. The amplitude calculation costs at least 20 times (time and memory) more than the traveltime calculation. Further speedup is needed. In Chapter IV, I describe parallel schemes that implement

the inversion process on a Thinking Machines CM-5 and a Silicon Graphics Power Challenge. These parallel algorithms are also suited for general velocity 3-D Kirchholf inversions.

An advantage of a Kirchhoff inversion, such as this one, is that one can image a selected part of the model instead of the whole model when the purpose is to obtain an inversion result for a particular target zone. This is in contrast to finite-difference migration or inversion, which needs to process data for the entire model.

Chapter 2

ALGORITHM FOR 3-D COMMON OFFSET INVERSION

2.1 Introduction

In this chapter, I review an inversion operator for application to prestack common-offset data, producing a subsurface map of the reflector, along with its reflection coefficients. I also develop an efficient ray tracing method for this model to calculate the necessary ray data for the amplitude and traveltime computation.

2.2 Inversion Formula

The inversion formulas of Bleistein [1986] provide a tool for obtaining correct locations of interfaces as well as a model-consistent specular reflection coefficient and incidence angle. The inversion formulas have the form of aperture-limited Fourier-like integrals. The integrand of these integrals will contain a determinant that will characterize the geometry of inverting a particular data set. This determinant is part of a Jacobian that depends on both the background propagation parameters and on the source-receiver configuration. Consequently, the problem of extending the inversion formula to new recording geometries is reduced to a problem of computing the value of this determinant for the specific geometry.

Figure 2.1 depicts a surface containing source and receiver locations, with $\xi = (\xi_1, \xi_2)$ as the surface parameter vector. $\mathbf{x_s}(\xi)$ and $\mathbf{x_g}(\xi)$ are the coordinates of the source and receiver in terms of ξ.

The full 3-D Kirchhoff inversion formula [Cohen et al., 1987] is

$$\beta(\mathbf{y}) = \frac{1}{8\pi^3} \int d^2\xi \frac{h(\mathbf{y},\xi)}{a(\mathbf{y},\xi)|\nabla_y \phi(\mathbf{y},\xi)|} \int i\omega \, d\omega \, e^{-i\omega\phi(\mathbf{y},\xi)} u_S(\mathbf{x}_g, \mathbf{x}_s, \omega). \tag{2.1}$$

In this equation, $\beta(\mathbf{y})$ is the reflectivity function for the imaged section; $\mathbf{y} = (x, y, z)$ is the 3-D position vector of the output. For a common-offset inversion, we denote by A_s and τ_s (A_g and τ_g) the amplitude and phase of the WKBJ Green's functions at $\mathbf{y}$ with initial point at the source or receiver. Thus, in the above equation

$$a(\mathbf{y},\xi) = A_s A_g, \qquad \phi(\mathbf{y},\xi) = \tau_s + \tau_g. \tag{2.2}$$

with the latter being the total traveltime from source to $\mathbf{y}$ to receiver. Furthermore,

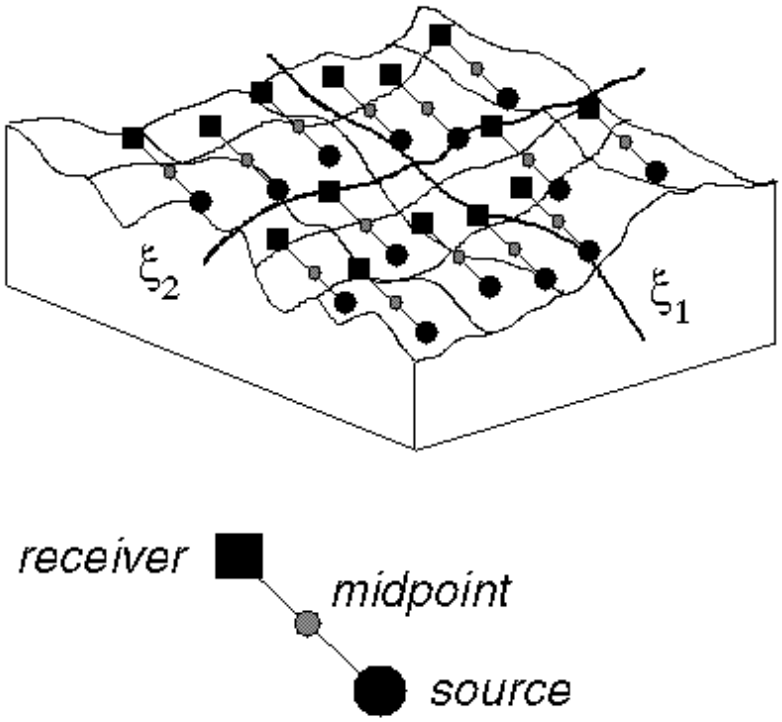


FIG. 2.1. Generalized source and receiver positions. The surface is parameterized by $\boldsymbol{\xi}$ with the generalized common offset geometry.

$u_s(\mathbf{x}_g, \mathbf{x}_s, \omega)$ represents the data in ω (frequency) domain; h is the determinant

$$\det \begin{bmatrix} \nabla_y \phi(\mathbf{y}, \xi) \\ \dfrac{\partial(\nabla_y \phi(\mathbf{y}, \xi)}{\partial \xi_1} \\ \dfrac{\partial(\nabla_y \phi(\mathbf{y}, \xi)}{\partial \xi_2} \end{bmatrix}. \tag{2.3}$$

The expression of this one determinant for any source-receiver configuration and background propagation speed is a major contribution of Beylkin's approach (1985) to high-frequency inversion and is referred as the Beylkin determinant. The details of h will be described later. The $\nabla_y \phi(\mathbf{y}, \xi)$ and the WKBJ amplitude a are the outgrowth of the Cohen/Bleistein inversion theory.

The reflectivity function provides a reflectivity map through a family of bandlimited delta functions that peak on the reflectors in the medium. The peak amplitude is proportional to the angularly-dependent reflection coefficient at a determinable specular angle θ_s, where $2\theta_s$ is the angle between the ray directions from the source and receiver to the output point. Denote by $\mathbf{y}_{peak}$ a point along the reflector normal where β has a local maximum or peak; that is, $\mathbf{y}_{peak}$ is a point on the reflector, . Thus,

$$\beta(\mathbf{y}_{peak}) \sim R(\mathbf{y}_{peak}, \theta_s) \frac{\cos\theta_s}{\pi c(\mathbf{y}_{peak})} \int_{-\infty}^{\infty} F(\omega) d\omega. \tag{2.4}$$

That is, the peak value of β for $\mathbf{y}$ on S is the geometrical-optics reflection coefficient $R(\mathbf{y}_{peak}, \theta_s)$ multiplied by $2\cos\theta_s / c(\mathbf{y}_{peak})$ and multiplied by $1/2\pi$ times the area under the filter $F(\omega)$ in the ω-domain.

For $\mathbf{y}$ not on the reflector surface, the angle θ is defined by the equation,

$$\nabla_y \tau(\mathbf{y}, \mathbf{x}_s) \cdot \nabla_y \tau(\mathbf{y}, \mathbf{x}_g) = \frac{\cos 2\theta}{c^2(\mathbf{y})}. \tag{2.5}$$

Then, by computing $\nabla\phi(\mathbf{y}, \xi) \cdot \nabla\phi(\mathbf{y}, \xi)$, one is able to show that

$$|\nabla_y \phi(\mathbf{y}, \xi)| = \frac{2\cos\theta}{c(\mathbf{y})}. \tag{2.6}$$

In particular, for $\mathbf{y}$ on the reflector surface, this provides a means for estimating $\cos\theta_s$, through the introduction of another inversion operator, differing from β by one power of $|\nabla_y \phi|$:

$$\beta_1(\mathbf{y}) = \frac{1}{8\pi^3} \int d^2\xi \frac{h(\mathbf{y}, \xi)}{a(\mathbf{y}, \xi) |\nabla_y \phi(\mathbf{y}, \xi)|^2} \int i\omega \; d\omega \; e^{-i\omega\phi(\mathbf{y}, \xi)} u_S(\mathbf{x}_g, \mathbf{x}_s, \omega). \tag{2.7}$$

Inclusion of the extra divisor of $|\nabla_y \phi|$ introduces this divisor into the asymptotic amplitudes of the result. In particular,

$$\beta_1(\mathbf{y}_{peak}) \sim R(\mathbf{y}_{peak}, \theta_s) \frac{1}{2\pi} \int F(\omega)\, d\omega. \tag{2.8}$$

The fact that these two operators differ by a factor of $2\cos\theta_s / c(\mathbf{y}_{peak})$ allows us to estimate $\cos\theta_s$ from the ratio of the outputs. This, in turn allows us to determine $R(\mathbf{y}, \theta_s)$ from either output. With knowledge of θ_s and the background wavespeed, $c(\mathbf{y})$, it is conceivable, within the limits of the accuracy of the data, that we should be able to estimate the jump in the propagation speed across the reflector using the formula for the geometrical-optics reflection coefficient in a constant density medium. For a variable density medium or for elastic reflection coefficients where more parameters need to be determined, outputs from many offsets and curve fitting are required to estimate parameter changes across reflectors (Bleistein, 1986).

2.3 Ray Tracing

Here, I describe the 3-D ray tracing in a depth-dependent medium. Then I derive a method to calculate Beylkin determinant h and WKBJ amplitude A. For a more complete discussion of the underlying ray theory for determining solutions of the wave equation in the form $A\exp[i\omega\tau]$, see Bleistein (1984), where the detailed derivation of Eikonal equation and Transport equation can be found.

2.3.1 The ray equations

Denote $\mathbf{x} = (x_1, x_2, x_3) = (x, y, z)$ as the cartesian coordinates; σ as the running parameter along the ray that comes from solving the wave equation by characteristics; $c(x_1, x_2, x_3)$ as the media velocity and $\mathbf{p} \equiv \bigtriangledown\tau$ as the slowness vector.

The ray equations is obtained by solving the Eikonal equation using the method of characteristics [Bleistein, 1984]. The general form of the ray equations is

$$\begin{aligned} \frac{dx_i}{d\sigma} = p_i, \qquad \frac{dp_i}{d\sigma} = p\frac{dp}{dx_i}, \quad i = 1, 2, 3, \\ \frac{d\tau}{d\sigma} = p^2, \\ {p_1}^2 + {p_2}^2 + {p_3}^2 = \frac{1}{c^2} \equiv p^2. \end{aligned} \tag{2.9}$$

These equations provide the basis for ray-theoretic modeling.

The *slowness* vector $\mathbf{p}$ points in the direction normal to the surfaces of constant τ. Surfaces of constant τ are called wavefronts, and the $\mathbf{p}$ vector points in the direction tangent to the raypaths. These, in turn, are the spatial trajectories of the solution to (2.9).

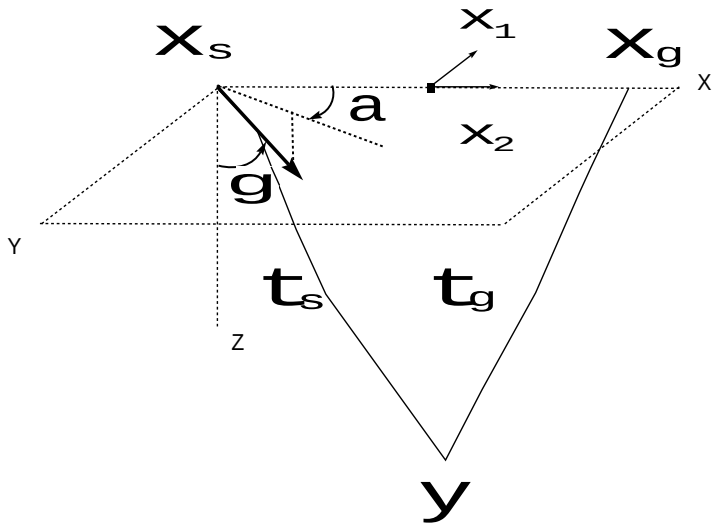


FIG. 2.2. Coordinate system showing an initial ray direction.

We seek solutions of (2.9) for rays emanating from a single point, say $\mathbf{x}_0 = (x_0,\ y_0,\ z_0)$ in an arbitrary downward direction. Those directions are determined by the initial values of α and γ representing the azimuthal and polar angles, respectively, of the spherical coordinate system (Figure 2.2). Then, the initial ray direction $\mathbf{p}_0$ is given by

$$\mathbf{p_0} \equiv p_0(\cos\alpha \sin\gamma, \sin\alpha \sin\gamma, \cos\gamma). \tag{2.10}$$

2.3.2 Specialization to a depth-dependent medium

For a medium that has wavespeed variability in the z-direction only, p_1 and p_2 are constants on each ray path. Denote them by p_{10} and p_{20}. Note that p_{10} and p_{20} are their respective initial values. The ray equations become

$$\begin{aligned} &\frac{dx_1}{d\sigma} = p_{10}, \qquad \frac{dx_2}{d\sigma} = p_{20}, \qquad \frac{dx_3}{d\sigma} = p_3, \\ &\frac{dp_1}{d\sigma} = 0, \qquad \frac{dp_2}{d\sigma} = 0, \qquad \frac{dp_3}{d\sigma} = p\frac{dp}{dx_3}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad \frac{d\tau}{d\sigma} = p^2. \end{aligned} \tag{2.11}$$

From the Eikonal equation, we know the value of p_3

$$p_3 = \sqrt{p^2(x_3) - p_{10}^2 - p_{20}^2} = \sqrt{p^2(x_3) - p^2(x_{30})\sin^2\gamma}. \tag{2.12}$$

The third equation in (2.11) relates σ and x_3, the ray equations may be rewritten in terms of x_3

$$\begin{gathered} \frac{dx_1}{dx_3} = \frac{p_{10}}{p_3}, \quad \frac{dx_2}{dx_3} = \frac{p_{20}}{p_3}, \quad \frac{dp_1}{dx_3} = 0, \quad \frac{dp_2}{dx_3} = 0, \\ \frac{dp_3}{dx_3} = \frac{p}{p_3}\frac{dp}{dx_3}, \quad \frac{d\tau}{dx_3} = \frac{p^2}{p_3}, \quad \frac{d\sigma}{dx_3} = \frac{1}{p_3}. \end{gathered} \tag{2.13}$$

We solve for x_1, x_2, τ, and σ as functions of x_3, α, and γ. Let the initial depth be $x_{30} \equiv z_0$ and the final depth be $x_3 \equiv z$. The solution of (2.13) is Recall that α and γ are the initial angles of the ray. When $p_3 = 0$, the ray is propagating horizontally. It is at a turning point. The equations above all have integrable singularities at the turning point, requiring special care in numerical computation, as discussed below.

2.3.3 The Jacobian J and ray amplitude

Here, we discuss the solution of the transport equation

$$2\nabla\tau \cdot \nabla A + A\nabla^2\tau = 0, \tag{2.14}$$

for the amplitude A. This equation can also be written as an ordinary differential equation in the ray parameter σ

$$\frac{d(A^2 J)}{d\sigma} = 0, \tag{2.15}$$

with solution (Bleistein 1984, 8.3.12)

$$A = \frac{\sqrt{\sin\gamma}}{4\pi\sqrt{c(\xi)J(\sigma,\alpha,\gamma)}}. \tag{2.16}$$

In these equations, J is the Jacobian of the transformation from $\mathbf{x}$ to (σ, α, γ) via the solution of the ray equations,

$$J = \left|\frac{\partial(x,y,z)}{\partial(\sigma,\alpha,\gamma)}\right|. \tag{2.17}$$

We calculate the ray theoretic amplitude A by computing J.

Let us consider the solution of the ray equations in terms of σ, rather than z. The solution is a family of rays, distinguished from one another by the choice of the

parameters α and γ. Along each ray, the values of τ, σ, x, y, z, p_i are known. p_1 and p_2 are constants on each ray; that is, they are independent of σ:

$$p_1 = \frac{1}{c(z_0)} \cos\alpha \sin\gamma,$$
$$p_2 = \frac{1}{c(z_0)} \sin\alpha \sin\gamma. \tag{2.18}$$

With p_1 and p_2 independent of σ, the equations for x and y are

$$x = \xi_1 + \frac{\sigma}{c(z_0)} \cos\alpha \sin\gamma,$$
$$y = \xi_2 + \frac{\sigma}{c(z_0)} \sin\alpha \sin\gamma. \tag{2.19}$$

and z is given in terms of σ by

$$z = \int p_3 d\sigma. \tag{2.20}$$

We do not use equation (2.20) since we do the calculation on a uniform z grid. Instead, we calculate σ for each z by using the third equation in (2.11). Thus, we obtain the ray parameters on a uniform grid in z, but calculate them as if the independent parameter along the ray were σ. This method avoids numerical integration in the neighborhood of a difficult integrable singularity of J as a function of z at the turning point. From (2.19) and (2.11), I derive the nine terms of the determinant J in (2.17) as

$$\frac{\partial x}{\partial \sigma} = \frac{\sin\gamma\cos\alpha}{c(z_0)}, \quad \frac{\partial y}{\partial \sigma} = \frac{\sin\gamma\sin\alpha}{c(z_0)}, \quad \frac{\partial z}{\partial \sigma} = p_3,$$
$$\frac{\partial x}{\partial \alpha} = -\frac{\sigma\sin\gamma\sin\alpha}{c(z_0)}, \quad \frac{\partial y}{\partial \alpha} = \frac{\sigma\sin\gamma\cos\alpha}{c(z_0)}, \quad \frac{\partial z}{\partial \alpha} = 0,$$
$$\frac{\partial x}{\partial \gamma} = \frac{\sigma\cos\gamma\cos\alpha}{c(z_0)}, \quad \frac{\partial y}{\partial \gamma} = \frac{\sigma\cos\gamma\sin\alpha}{c(z_0)}, \quad \frac{\partial z}{\partial \gamma} = \int \frac{-p_{30}\sin\gamma}{c(z_0)p_3} d\sigma.$$

With these values, we can calculate J in (2.16) and then calculate A by using (2.17).

2.3.4 The Belkyin determinant h

The influence of source-receiver geometry are completely described by the Beylkin determinant

$$h(y,\xi) = \det \begin{bmatrix} p_s + p_g \\ \dfrac{\partial(p_s + p_g)}{\partial \xi_1} \\ \dfrac{\partial(p_s + p_g)}{\partial \xi_2} \end{bmatrix}, \tag{2.21}$$

in terms of the slowness vectors $\mathbf{p}_s$ and $\mathbf{p}_g$ and their derivatives with respect to the surface coordinates, (ξ_1, ξ_2). These vector quantities are evaluated at the output point $\mathbf{y}$.

The determinant must be finite and nonzero for the identification of the cascaded model and inversion integral as an approximate Fourier transform pair [Bleistein, 1986]. So that we could use the value of this matrix to characterize source-receiver configurations. In particular, it is required that this determinant be finite and nonzero for some range of ξ values at any $\mathbf{y}$ where the high-frequency inversion is to be computed.

The general form of the Beylkin determinant can be written for all source receiver geometries as

$$\begin{aligned} h(\mathbf{y},\xi) &= (\mathbf{p}_s + \mathbf{p}_g) \cdot (\mathbf{v}_s + \mathbf{v}_g) \times (\mathbf{w}_s + \mathbf{w}_g) \\ &= (\mathbf{p}_s + \mathbf{p}_g) \cdot \mathbf{v}_g \times \mathbf{w}_g + (\mathbf{p}_s + \mathbf{p}_g) \cdot \mathbf{v}_s \times \mathbf{w}_s + \\ &\quad (\mathbf{p}_s + \mathbf{p}_g) \cdot \mathbf{v}_g \times \mathbf{w}_s + (\mathbf{p}_s + \mathbf{p}_g) \cdot \mathbf{v}_s \times \mathbf{w}_g. \end{aligned} \tag{2.22}$$

where,

$$\mathbf{v}_s \equiv \frac{\partial \mathbf{p}_s}{\partial \xi_1}, \qquad \mathbf{v}_g \equiv \frac{\partial \mathbf{p}_g}{\partial \xi_1}, \qquad \mathbf{w}_s \equiv \frac{\partial \mathbf{p}_s}{\partial \xi_2}, \qquad \mathbf{w}_g \equiv \frac{\partial \mathbf{p}_g}{\partial \xi_2}.$$

The two terms $(\mathbf{p}_s + \mathbf{p}_g) \cdot \mathbf{v}_g \times \mathbf{w}_g$ and $(\mathbf{p}_s + \mathbf{p}_g) \cdot \mathbf{v}_s \times \mathbf{w}_s$ can be recognized as being the Beylkin determinants for the common-shot and common-receiver geometries. We may write the Beylkin determinant as

$$\begin{aligned} h(\mathbf{y},\xi) = 2\cos^2\theta\,[h_s(\mathbf{y},\xi) + h_g(\mathbf{y},\xi)] + \\ (\mathbf{p}_s + \mathbf{p}_g) \cdot [\mathbf{v}_g \times \mathbf{w}_s + \mathbf{v}_s \times \mathbf{w}_g]\,, \end{aligned} \tag{2.23}$$

where $h_g(\mathbf{y},\xi)$=$\mathbf{p}_g \cdot \mathbf{v}_g \times \mathbf{w}_g$ is the determinant for the common receiver case and $h_s(\mathbf{y},\xi)$=$\mathbf{p}_s \cdot \mathbf{v}_s \times \mathbf{w}_s$ is the determinant for the common source case. Unfortunately, the term on the far right in (2.24) contains the cross products $\mathbf{v}_s \times \mathbf{w}_g$ and $\mathbf{v}_g \times \mathbf{w}_s$, which are not easily simplified.

Another way of writing the Beylkin determinant is

$$h(\mathbf{y},\xi) = 2\cos^2\theta\,[h_s(\mathbf{y},\xi) + h_g(\mathbf{y},\xi)] + h_a(\mathbf{y},\xi) + h_b(\mathbf{y},\xi) \tag{2.24}$$

$$+h_c(\mathbf{y},\xi) + h_d(\mathbf{y},\xi), \tag{2.25}$$

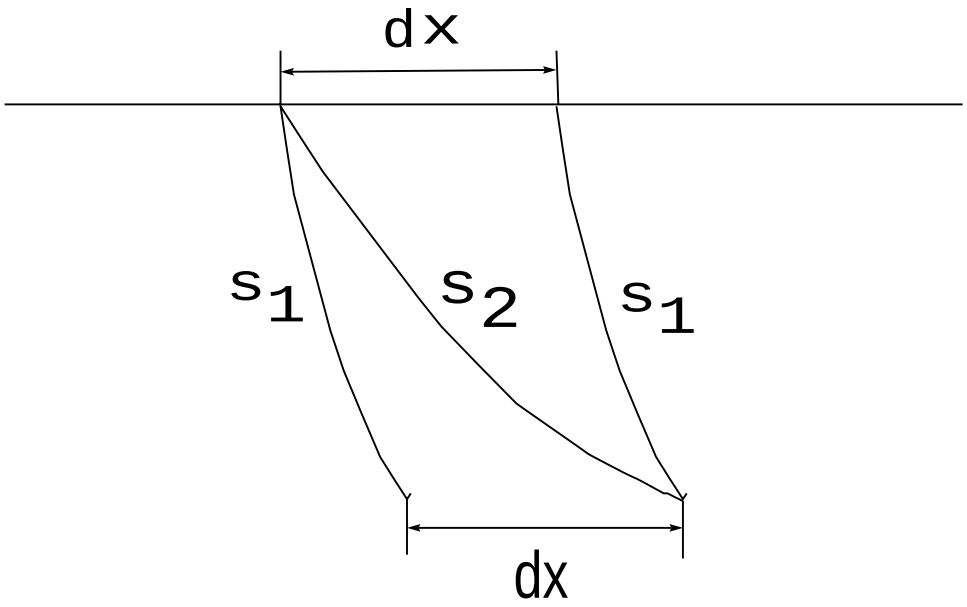


FIG. 2.3. Ray symmetry used in calculating h.

where the h_a, h_b, h_c and h_d are defined by

$$h_a(\mathbf{y},\xi) \equiv \mathbf{p}_s \cdot \mathbf{v}_s \times \mathbf{w}_g, \quad h_b(\mathbf{y},\xi) \equiv \mathbf{p}_s \cdot \mathbf{v}_g \times \mathbf{w}_s,$$
$$h_c(\mathbf{y},\xi) \equiv \mathbf{p}_g \cdot \mathbf{v}_g \times \mathbf{w}_s, \quad h_d(\mathbf{y},\xi) \equiv \mathbf{p}_g \cdot \mathbf{v}_s \times \mathbf{w}_g. \tag{2.26}$$

Unfortunately, we cannot write the general common-offset Beylkin determinant in a form that has a common multiplier of $\cos^2\theta$.

I already know the vectors $\mathbf{p}_s$ and $\mathbf{p}_g$ at the output point from the solution of the ray equations. Now I compute their partial derivatives with respect to ξ_1 and ξ_2 as solutions of another set of ray equations.

By taking derivatives of (2.19) with respect to ξ_1 and ξ_2, respectively, and using (2.18), I derive differential equations for $\partial p_1/\partial\xi_i$ and $\partial p_2/\partial\xi_i$. These equations are

$$\frac{\partial p_1}{\partial \xi_1} = -(1 + p_1\frac{\partial\sigma}{\partial\xi_1})/\sigma, \qquad \frac{\partial p_2}{\partial \xi_1} = -p_2\frac{\partial\sigma}{\partial\xi_1}/\sigma,$$
$$\frac{\partial p_1}{\partial \xi_2} = -p_1\frac{\partial\sigma}{\partial\xi_2}/\sigma, \qquad \frac{\partial p_2}{\partial \xi_2} = -(1 + p_2\frac{\partial\sigma}{\partial\xi_2})/\sigma. \tag{2.27}$$

The only quantities that we do not know in the above equations are $\partial\sigma/\partial\xi_1$ and $\partial\sigma/\partial\xi_2$. It is difficult to implement the derivatives with respect to ξ_1 and ξ_2.

However, using the symmetry of the $v(z)$ medium, we find that $d/d\xi_1 = -d/dx$ and $d/d\xi_2 = -d/dy$ as shown schematically in Figure 2.3. That is, when a ray is shifted horizontally, the quantity σ on that ray does not change. Also, one can see from Figure 2.3, that $d\sigma/d\xi = (\sigma_1 - \sigma_2)/d\xi = -(\sigma_2 - \sigma_1)/dx$.

From the Eikonal equation–the last equation in (2.9) the derivatives of p_3 with respect to ξ_1 and ξ_2 can also be found:

$$\frac{\partial p_3}{\partial \xi_1} = -(p_1 \frac{\partial p_1}{\partial \xi_1} + p_2 \frac{\partial p_2}{\partial \xi_1})/p_3, \qquad \frac{\partial p_3}{\partial \xi_2} = -(p_1 \frac{\partial p_1}{\partial \xi_2} + p_2 \frac{\partial p_2}{\partial \xi_2})/p_3. \tag{2.28}$$

For each output point, we need to calculate the derivatives for both the source and the receiver. They are calculated using (2.27) and (2.28). Combining them into the matrix yields the Beylkin determinant. Once we have h, together with the product of ray theoretic amplitudes, a, the amplitude computation is complete.

In the next chapter, I describe numerical methods to calculate amplitude and traveltime tables.

Chapter 3

NUMERICAL IMPLEMENTATION

3.1 Introduction

In this chapter, I give the computer processing steps and I also describe the process used to transform ray data from a ray coordinate grid to a cartesian grid. A case of constant subsurface speed is presented to verify the algorithm and check its accuracy.

3.2 Processing Sequence

For the common-offset inversion formula in equation (2.1), the computer processing proceeds as follows.

1. *Preprocessing of the data*

The function $u_S(\mathbf{x}_g, \mathbf{x}_s, \omega)$ represents the Fourier transform of a single trace of seismic data, with the source and receiver points given by $\mathbf{x}_s$ and $\mathbf{x}_g$. The ω integration represents a filtering and inverse transform of the trace. Because this is independent of the ξ integration, the computation can be done as a preprocessing step. This is equivalent to establishing an inverse Fourier transform table with time as the variable on a uniform grid.

2. *Computation of traveltime and amplitude tables for each source−receiver pair (midpoint)*

Given a source point, we determine the ray paths from the source with uniform polar and azimuthal angles and calculate the ray data along the ray. On each ray grid position, the ray data include traveltime, slowness, ray parameter σ, x and y position, propagation angle and the derivative of depth with respect to propagation angle, $dz/d\gamma$. These ray data are expressed in terms of ray coordinates (α, γ, z). Linear interpolation and extrapolation are used to transform these data onto uniform (x, y, z) grids. Other quantities that are required for the calculation of Jacobian J and Beylkin Determinant h are computed after the transformation. Traveltime and amplitude tables are then established for this source point. Tables for other sources and receivers are shifted from these original tables according to the geometry.

3. *Summation over trace locations*

After calculating traveltime, the program obtains the corresponding value in the trace by calculating a time index ($i = t/dt$) and then interpolating between two adjacent samples on that trace $trace[i]$ and $trace[i+1]$. Multiplying this value with amplitude weights and other factors, we obtain the contribution to the inversion sum from that particular trace location. Finally, summing these contributions over all the

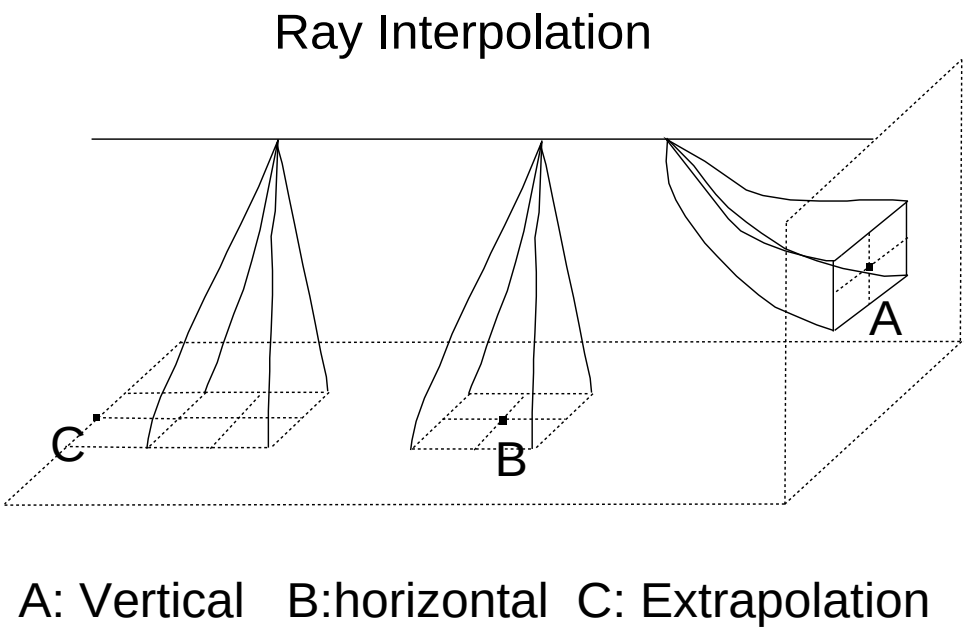


FIG. 3.1. Diagram of interpolation from ray coordinate to uniform grid points.

possible midpoints (traces) that meet the anti-aliasing criteria discussed below, we obtain the angularly dependent reflection coefficient at the particular output point.

Those are the three basic steps in the processing. There are other considerations in the code such as anti-aliasing filter. Spatial aliasing of data is caused by insufficient spatial sampling for a given bandwidth and temporal sampling rate. Receiver spacing and temporal bandwidth determine the maximum dip that the inversion can recover. The energy coming from steeper reflectors becomes aliased and, as in the frequency domain, this energy will appear as a lower than true wavenumber event. Therefore, in the inversion code, we have to remove the aliased energy before we perform inversion. Denote the vector $\nabla\phi = (\nabla\phi_x, \nabla\phi_y, \nabla\phi_z)$, the criterion for common-offset data is

$$\nabla\phi_x \leq \frac{1}{2dxf_{max}}, \qquad \nabla\phi_y \leq \frac{1}{2dyf_{max}}. \tag{3.1}$$

where f_{max} is the maximum frequency that is in the data; dx, dy is the receiver spacing in both x and y direction. For the rays whose emerging angles give a frequency that exceeds f_{max}, we set the amplitude contribution of these rays to zero.

3.3 Transformation and Interpolation

Ray data are generated along each raypath with coordinates (σ, α, γ). Transformation of ray data from this ray system to a uniform grid in (x, y, z) is achieved by linear interpolation as shown in Figure 3.1. Boyi Ou [1994] used two-point linear ray interpolation method for 2.5-D. Here I use Four-point ray interpolation. Rays taking off from the source point intersect each z-plane as they propagate downward. On a z-plane, four nearby ray intersection points that surround a grid point are found as in B in Figure 3.1. Linear interpolation is used to determine the ray data at the grid point from the four intersection points. Linear extrapolation is used near the borders when rays do not surround a grid point as in C in Figure 3.1. This scheme will not be accurate in the vicinity where rays are near turning because the distance between the intersection points of the rays with the turning depth could be too large. To deal with rays propagating nearly horizontally, that is, when p_3 is small (propagation angle over 45 degrees), vertical interpolation is used. Vertical interpolation (C in Figure 3.1) differs from the above scheme in that it uses a vertical plane instead of a horizontal z-plane. This vertical plane could be either an x-plane or a y-plane depending on α (the azimuthal angle). If $\alpha < 45^o$, I use the y-plane, otherwise, I use the x-plane. The nearly horizontally propagating rays will intersect the vertical plane with a closer spacing thus giving a more accurate interpolation than if we used a horizontal plane.

Now we have all the ray data we need on the uniform (x, y, z) grid. We next use these ray data to calculate the Jacobian J, ray amplitude a and Beylkin determinant h. Then combine these values to give the amplitude weight at each grid point for that particular source point. This amplitude table along with the traveltime table will be used for all source and receiver points in the inversion.

3.4 Example: Constant Wavespeed

In this section, I present the specialization of the above results for a constant wavespeed medium, where all calculations can be carried out explicitly. In this case, equation (2.17) reduces to

$$J = \frac{r^2}{c} sin\gamma, \tag{3.2}$$

and equation (2.16) becomes,

$$A = \frac{1}{4\pi r}. \tag{3.3}$$

For the special case of constant wavespeed, with a generally non-flat surface, some simplification of h is possible. Application of Gaussian elimination produces the following simplified forms for h_a, h_b, h_c, and h_d in (2.25)

$$h_a(y,\xi) \equiv \frac{-1}{c^3 r_s} \hat{r}_s \cdot \frac{\partial x_s}{\partial \xi_1} \times \frac{\partial p_g}{\partial \xi_2}, \qquad h_b(y,\xi) \equiv \frac{-1}{c^3 r_s} \hat{r}_s \cdot \frac{\partial p_g}{\partial \xi_1} \times \frac{\partial x_s}{\partial \xi_2}, \tag{3.4}$$

$$h_c(y,\xi) \equiv \frac{-1}{c^3 r_g}\hat{r}_g \cdot \frac{\partial p_s}{\partial \xi_1} \times \frac{\partial x_g}{\partial \xi_2}, \qquad h_d(y,\xi) \equiv \frac{-1}{c^3 r_g}\hat{r}_g \cdot \frac{\partial x_g}{\partial \xi_1} \times \frac{\partial p_s}{\partial \xi_2}.$$

For the special case of constant wavespeed, with a flat recording surface, h_s, h_g, h_a, h_b, h_c, and h_d become

$$\begin{aligned} h_g &\equiv \frac{y_3}{c^3 r_g^3}, \qquad h_s \equiv \frac{y_3}{c^3 r_s^3}, \qquad h_a \equiv \frac{y_3}{c^3 r_s^2 r_g}, \\ h_b &\equiv \frac{y_3 \cos 2\theta}{c^3 r_s r_g^2}, \qquad h_c \equiv \frac{y_3 \cos 2\theta}{c^3 r_s^2 r_g}, \qquad h_d \equiv \frac{y_3}{c^3 r_s r_g^2}. \end{aligned} \tag{3.5}$$

Substituting these results into equation (2.24), yields the full expression for the Beylkin determinant

$$h(y,\xi) = 2\cos^2\theta \; \frac{y_3}{c^3} \left[\frac{(r_s+r_g)(r_s^2+r_g^2)}{r_s^3 r_g^3}\right]. \tag{3.6}$$

The corresponding formulas for $\beta(y)$ and $\beta_1(y)$ for 3D common-offset, constant wavespeed, and with a flat recording surface become

$$\beta(y) = \frac{2y_3}{c^2\pi}\int d^2\xi \left[\frac{(r_s+r_g)(r_s^2+r_g^2)}{r_s^2 r_g^2}\right] \cos\theta \int i\omega \; d\omega \; e^{-i\omega[r_s+r_g]/c} u_S(x_g, x_s, \omega), \tag{3.7}$$

and

$$\beta_1(y) = \frac{y_3}{c\pi}\int d^2\xi \left[\frac{(r_s+r_g)(r_s^2+r_g^2)}{r_s^2 r_g^2}\right] \int i\omega \; d\omega \; e^{-i\omega[r_s+r_g]/c} u_S(x_g, x_s, \omega). \tag{3.8}$$

Equation (3.8) exactly matches equation (30) in Sullivan and Cohen (1987), which was derived using a different approach.

Now we have two amplitude weight expressions for the constant wavespeed case. The first, extracted from quation (2.1) is

$$\frac{1}{8\pi^3}\frac{|h(\mathbf{y},\xi)|}{a(\mathbf{y},\xi)|\nabla_y \phi(\mathbf{y},\xi)|}$$

which I compute numerically by the method I have described in this thesis. The corresponding analytical expression in (3.7) is

$$\frac{2y_3}{c^2\pi}\left[\frac{(r_s+r_g)(r_s^2+r_g^2)}{r_s^2 r_g^2}\right]\cos\theta.$$

which I can evaluate directly. I test my ray tracing program by compare the results from these two approaches.

Table 1: Amplitude weight comparison for a constant background

Depth (Δz)	Analytic	Numerical	Error (%)
4	0.059	0.062	5.19
8	0.287	0.291	1.26
12	0.483	0.490	1.31
16	0.638	0.644	0.95
20	0.754	0.759	0.76
24	0.838	0.843	0.60
28	0.900	0.905	0.51
32	0.946	0.950	0.43
36	0.980	0.983	0.33
40	1.017	1.020	0.28

Table 1 shows the results computed by the numerical and analytic formulas for a constant background. The grids are $40 \times 40 \times 40$ and the ray shooting grids in α and γ are $15^o \times 15^o$, which is rather coarse. The numerical results have two- or three-digit accuracy:

These are the amplitude weights on one trace with a offset of $10\Delta z$. Notice that the accuracy increases with depth. This happens because the polar angle γ associated with the rays increases with depth and thus has a relatively closer ray spacing for interpolation. The computation of travel time was about 10 times more accurate (Table 2). This is because the calculation of traveltime comes directly from ray tracing while the calculation of amplitude has error accumulation from numerical computation of many partial derivatives includes the derivatives of traveltime.

Table 2: Traveltime comparison for a constant background

Depth (Δz)	Analytic	Numerical	Error (%)
4	0.952190	0.952205	0.00
8	1.154701	1.154893	0.02
12	1.526070	1.526313	0.02
16	1.973153	1.973434	0.01
20	2.454927	2.455159	0.00
24	2.954469	2.954733	0.00
28	3.464102	3.464356	0.00
32	3.979950	3.980125	0.00
36	4.499877	4.500115	0.00
40	5.022616	5.022789	0.00

Chapter 4

PARALLEL IMPLEMENTATION

4.1 Introduction

In this chapter, I describe the parallel schemes for the implementation of the inversion algorithm on a Thinking Machines CM-5 and a Silicon Graphics Power Challenge. I first review the parallel programming model that is used on both machines. I then show the algorithm in more detail and discuss performances on each of the two machines.

4.2 Algorithm Design

A parallel programming model is a means of describing parallel algorithms for a certain computer architecture. For example, in the shared-memory programming model it is assumed that the data are in a memory that is shared by all processing elements and can be accessed by any of them. In the message-passing programming model (which is a special case of distributed-memory programming models), a parallel computation consists of several tasks executing concurrently. Each task can read and write its own memory, as well as remote memories (located in other processing nodes) by sending/receiving messages (a message is a set of bytes transferred from one computer to another computer). With the advent of new programming tools such as message-passing libraries and distributed shared memory systems, algorithms designed in a certain programming model are not restricted to execution in a particular architecture. The algorithm designed below is suited for both the CM-5 and PVM (Parallel Virtual Machine).

For one offset, the Kirchhoff inversion algorithm consists of two steps. First, traveltime and amplitude tables have to be computed and interpolated. Then, a summation of data along diffraction curves is performed. The shape of the diffraction curve is described by the impulse response for any image point, the Green's function, which depends on the velocity field and the shot-receiver geometry. Travetime and amplitude are precomputed. The tables needed for the summation are interpolated from these original precomputed tables. This makes the cost for each of the steps comparable.

The step of generating the original traveltime and amplitude table is independent of shot-receiver location. A simple adaptation of a serial algorithm on one processor is all that is required. Table shifting and diffraction summation for different shot and receiver locations are done in parallel. Diffraction summation is not an input/output balanced algorithm in that the input is one data trace and the output is a large block

of data determined by the aperture. Therefore, one master or supervisor processor is needed to distribute and receive data for all the processing elements.

I give the sequential code flow before I introduce the parallel implementation. The sequential code is based on the two-step scheme. The computation for every output point is arranged in five nested loops over the two coordinates of the midpoint traces (the x and y coordinates of the data) and the three dimensions of the output points (x_o, y_o, z_o):

```
/* loop over traces */
for (ixm=0; ixm < nxm; ixm+=1)
for (iym=0; iym < nym; iym+=1)
    ... (get index for tables )

    /* loop over output points */
    for (ixo=0; ixo < nxo; ++ixo)
    for (iyo=0; iyo < nyo; ++iyo)
    for (izo=0; izo < nzo; ++izo)

            distribute amplitudes
            out[iyo][ixo][izo]+=...
```

where:

nxm and nym are of x and y dimensions of traces on the input data

nxo, nyo and nzo are the dimensions of the output block

Based on the above, if the input data section has $N_x \times N_y$ traces and the output section has $N_{xo} \times N_{yo} \times N_{zo}$ points, the cost of the whole process is $O(N_x \times N_y \times N_{xo} \times N_{yo} \times N_{zo})$.

This process can be implemented in a parallel environment by a supervisor/workers paradigm (Figure 4.1). The designed algorithm looks like this:

1. The traces in the input data are laid out among different virtual processors (workers); that is, one trace in the input data, representing one surface location, can be stored in one virtual processor element (PE), and calculations can be done in parallel for all surface locations. Then, the result is sent to the master processor.

2. Use one processor as the master processor to receive the output from each worker processor and add the results together to get the final output.

Basically, this two-step algorithm involves little inter-processor communication. Only

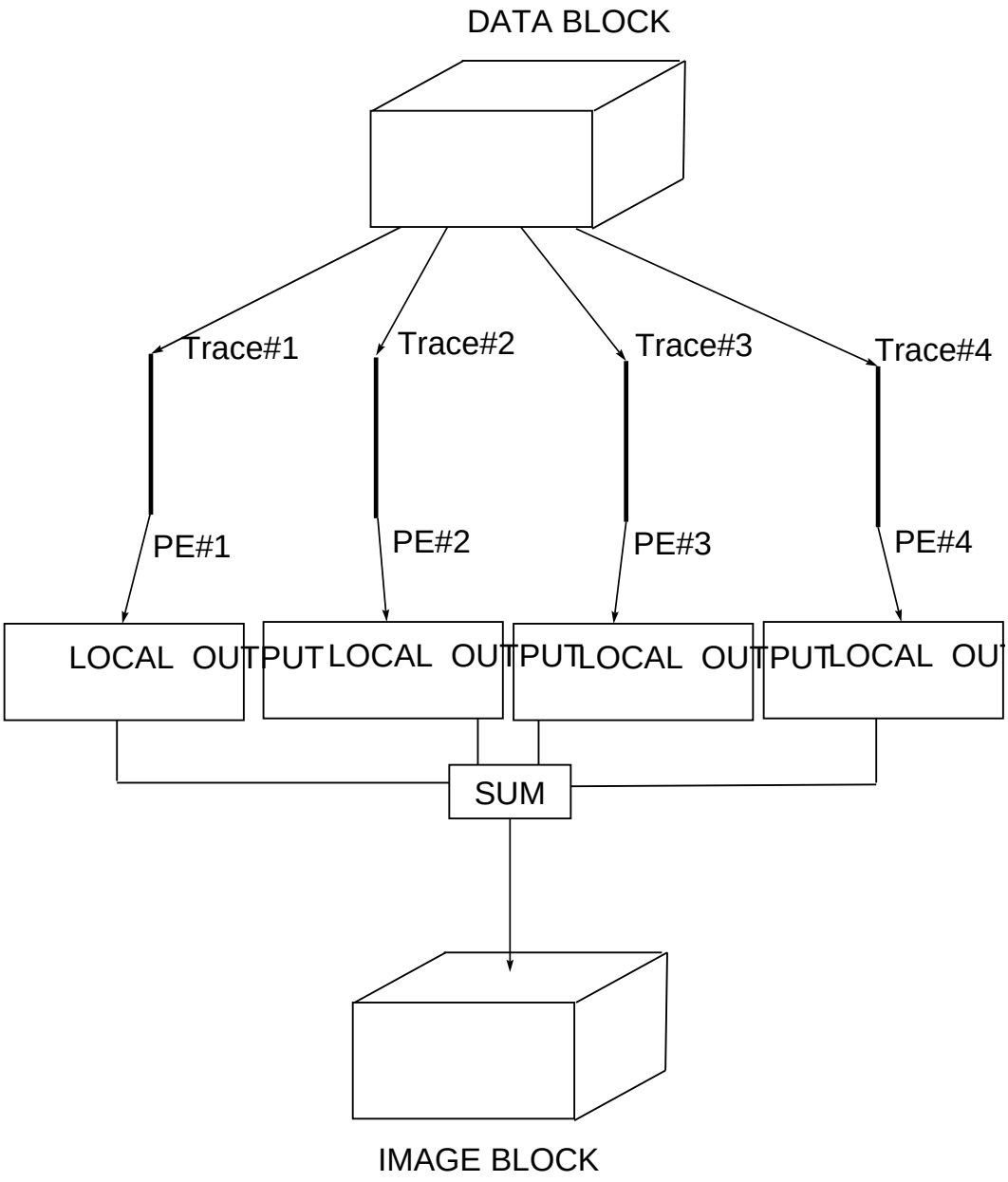


FIG. 4.1. Parallel design

in the beginning and end, each processor element receives data and sends its result to the master processor. Every processor has the same amount of work, so the load balancing is not a problem. Since the major part of work load is divided among N processors, ideally, the time cost would be reduced from $O(N_x \times N_y \times N_x \times N_y \times N_z)$ to $O(N_x \times N_y \times N_x \times N_y \times N_z)/N$. The advantage of this scheme is that it requires minimal communication. The disadvantage is the large demand on memory. Van Trier [1991] used a systolic algorithm for 2-D Kirchhoff migration which took advantage of fast nearest-neighbor communication hardware. That scheme, however, requires more interprocessor communications and is difficult to implement constraints such as aliasing control and other interprocessor operations. I designed my code so that the interprocessor communication among processors is minimum, makes the processing elements more independent from each other. This makes it easier to implement operations such as data filtering and anti-aliasing.

4.3 CM-5

The inversion algorithm requires many numerical operations in the interpolations for traveltime and amplitude, as well as a large number of data transfer operations in the diffraction summation. It is therefore well-suited for implementation on a massively parallel computer like the Connection Machine system from Thinking Machines Corporation.

The CM-5 and the Cray T3D are examples of a MIMD (Multiple Instruction Multiple Data) machine with distributed memory. Distributed memory means that memory is distributed among the processors, rather than placed in a central location, as in multiprocessors. Here, each processing node executes its own program. This program may access local memory and may send and receive messages to or from other nodes. Messages writing to remote memories are used to communicate with other nodes.

The CM-5 that I used had 1024 nodes. Each node contains a SPARC2 CPU, 4 vector units and a local memory. The unit of data on the CM is a parallel variable, and programming languages on the CM incorporate such variables directly into the language. For example, CM C* (c-star), the parallel extension of the C language, treats arrays as first-class objects that can be operated on in a single statement. A wide range of intrinsic functions (such as summation) is available to evaluate or manipulate arrays. The size of arrays is not limited by the number of physical processors: the Connection Machine hardware allows each physical processor to behave as many virtual processors, each with smaller memory. The power of parallelism arises from extending familiar operations to parallel operations.

Below is a summary of characteristics on a CM-5 [CM5: Technical Summary, 1992]. The program views the machine globally as a single machine and locally as a collection of nodes that can communicate. The programmer sees one memory, one thread of control. Data parallel programs can allocate memory, compute, and communicate from the global point of view, each instruction operating on an entire data

set while compilers and system software handle synchronization. A single thread of control simplifies analysis and debugging. Distribution of data and computation is automatic with the same program running on arbitrarily many processors. Programming in a global data parallel language allows one to focus on algorithm instead of hardware.

By following the data parallel strategies, first I find the parallel dimensions of my 3D model arrays for the input data, ray data tables and the output image; then I declare parallel data structures for the data, choosing layouts that minimize communication; finally, I choose an algorithm that allows the program to operate concurrently on as many data items as possible.

The efficiency of the algorithm is computed as follows:

$$efficiency = \left(\frac{parallel\ speedup}{number\ of\ nodes} \right) \times 100\%$$

with

$$parallel\ speedup = \frac{sequential\ time}{parallel\ time}.$$

The C* inversion code was tested on a $100 \times 100 \times 100$ data block (Figure 4.2), yielding a $100 \times 100 \times 100$ output set (Figure 4.3). The data shown are part of a common offset dataset. The actual velocity was not exactly $v(z)$ but we consider it as depth dependent by using the middle velocity trace as the velocity. Although the velocity is not exactly correct, the output still gives a reasonably good inversion image, especially for the deepest feature at the bottom.

The total time (including I/O and communications) for running this on a CM-5 with 256 processors was 43.42 seconds (the elapsed time), 3 s of the run time was spent on I/O. On a CM-5 with 32 processors, the elapsed time was 305.22 seconds. Figure 4.5 shows the comparison of cpu time for different numbers of processors. The Connection Machine computer is a fully scalable machine, i.e., the scale of the problem that can be solved in a given amount of time scales linearly with the size of the machine. But with the communication overhead, the computation/communication ratio changes, resulting in the non-linear figure in the plot. The parallel speedup from 32 processors to 256 processors is about 7. This is quite a good speedup for such a fairly small-sized problem where the computation versus communication ratio is low.

Normally on a single workstation such as the RS6000, this process requires hours of computation time (112 minutes for a test on a RS6000 workstation). On a CM-5, this reduces to less than a minute. Of course, for larger dimension, the cost will increase enormously. For example, for a $200 \times 200 \times 200$ data block, the cost will increase by approximately 2^5 times. Memory is also a problem, for larger datasets, the local memory on each processor may not be enough. Considerations such as data compression may be needed.

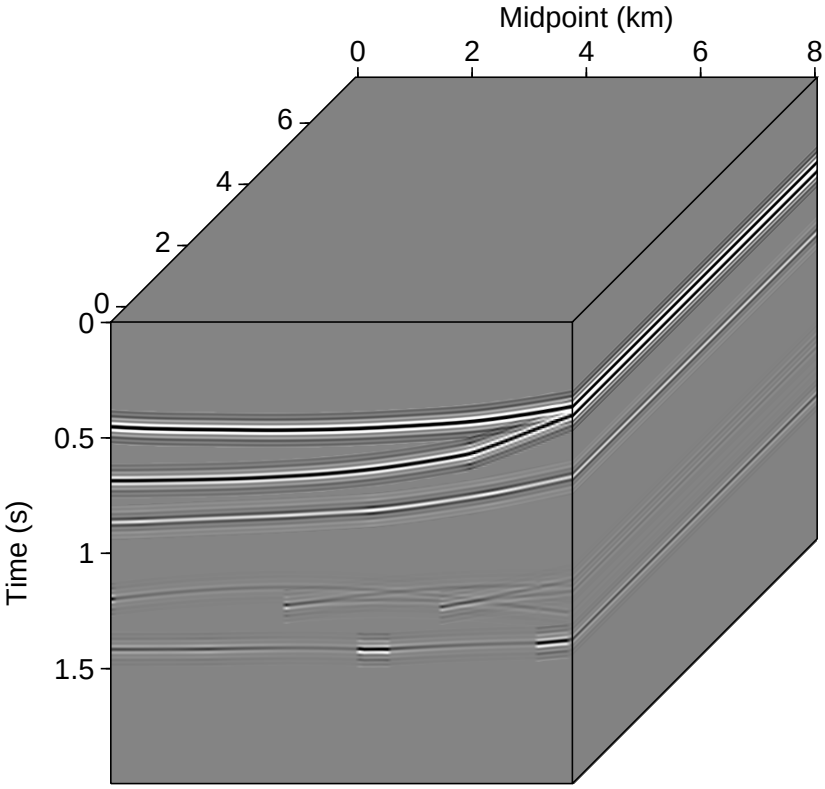


FIG. 4.2. Output of parallel test.

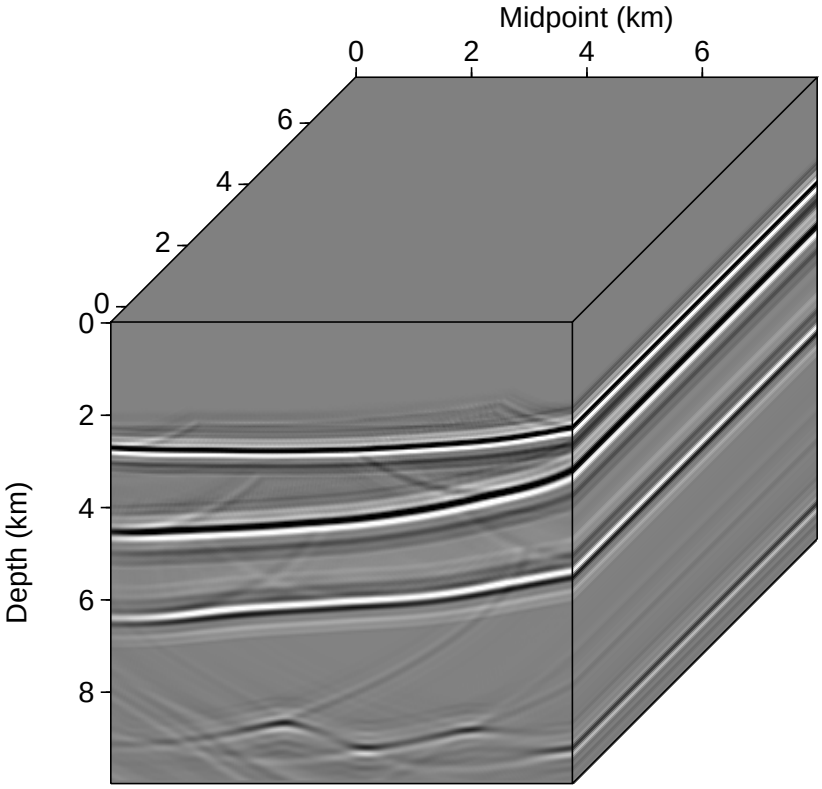


FIG. 4.3. Output of parallel test.

4.4 Implementation Using PVM on a Power Challenge and an IBM Network

The SGI Power Challenge is a shared memory machine. Shared memory implies that all processors share access to a common memory. This machine can also perform distributed computation along with a network of Unix workstations using the PVM [Geist et al., 1994] with a message-passing system. Compared to supercomputers such as the CM-5, Unix workstations have a low cost per node, high availability, high degree of standarization and good software portability.

An algorithm such as in Figure 4.1 can also be implemented using PVM with a slight modification; we change the subdomain of the input. That is, instead of sending one data trace to a single processor, a section with the number of N_x traces with an index in y are to be sent (Figure 4.4). We do this so that the inter-communications between worker-nodes and the master-node will not be too large. This is equivalent to saying that each of the worker-nodes is doing an inversion with only the traces inside that section contributing to the image block. The algorithm is implemented using a message-passing programming model and is implemented using the PVM library as the message passing interface. For the tests, I used a network of IBM RS6000's is used in addition to a 4-node Power Challenge.

The program was coded in C language, making use of the SU (Seismic Unix) library, existing CWP (Center for Wave Phenomena, Colorado School of Mines) software. The interprocessor communication, process creation, and synchronization were implemented using the PVM library. One master node calculates the original traveltime and amplitude tables, and partitioned input, in this case a section of data along with all the necessary parameters, is sent to each node. Each node does the same processing on its part of data. Finally the master node receives the output from each working node and combines them to give the result. The specific strategy for the workload allocation used was the so-called node-only mode, in which multiple instances of the program (which resides in the common file system of the workstation cluster) execute, and each process generated in this way updates its own output grids according to the partitioned input grids, as well as establishes the communication with the master node. In this strategy, one of the processors (the master processor) spawns the rest of the processes, and takes over the initial distribution of data, problem parameters and workload allocation, and the final allocation and output of the results, in addition to contributing to the computation itself.

Taking the same $100 \times 100 \times 100$ data block (Figure 4.2) for testing, using only one node on the Power Challenge, the elapsed time is 9572 s; for four nodes on the Power Challenge, the elapsed time is 2538 s. The speedup is 3.77. And the efficiency is 94 percent.

When a network of IBM RS6000 workstations is networked with the Power Challenge, the speedup decreases (Figure 4.6). This is because the bandwidth of the network is limited, which makes the communication slower. The speedup also decreases as the number of processors increases because the size of the problem is fixed,

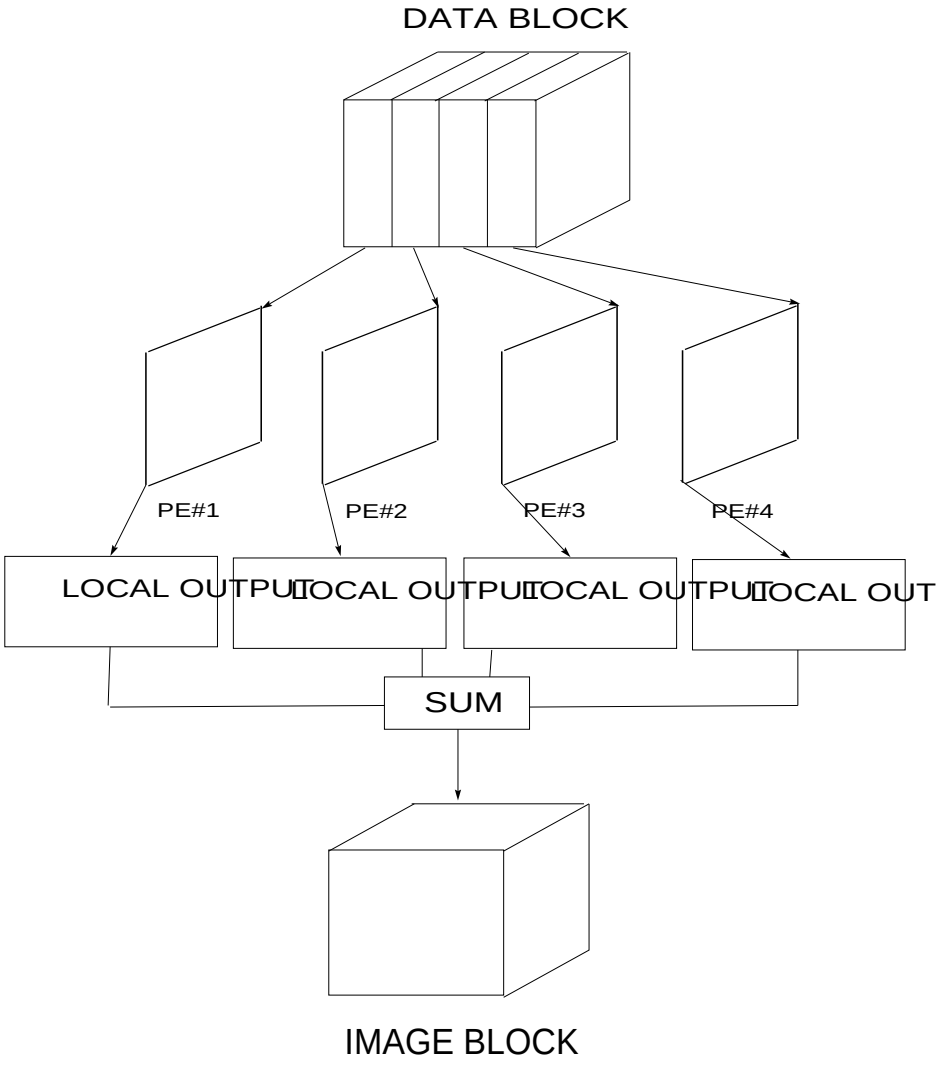


FIG. 4.4. Parallel algorithm using PVM.

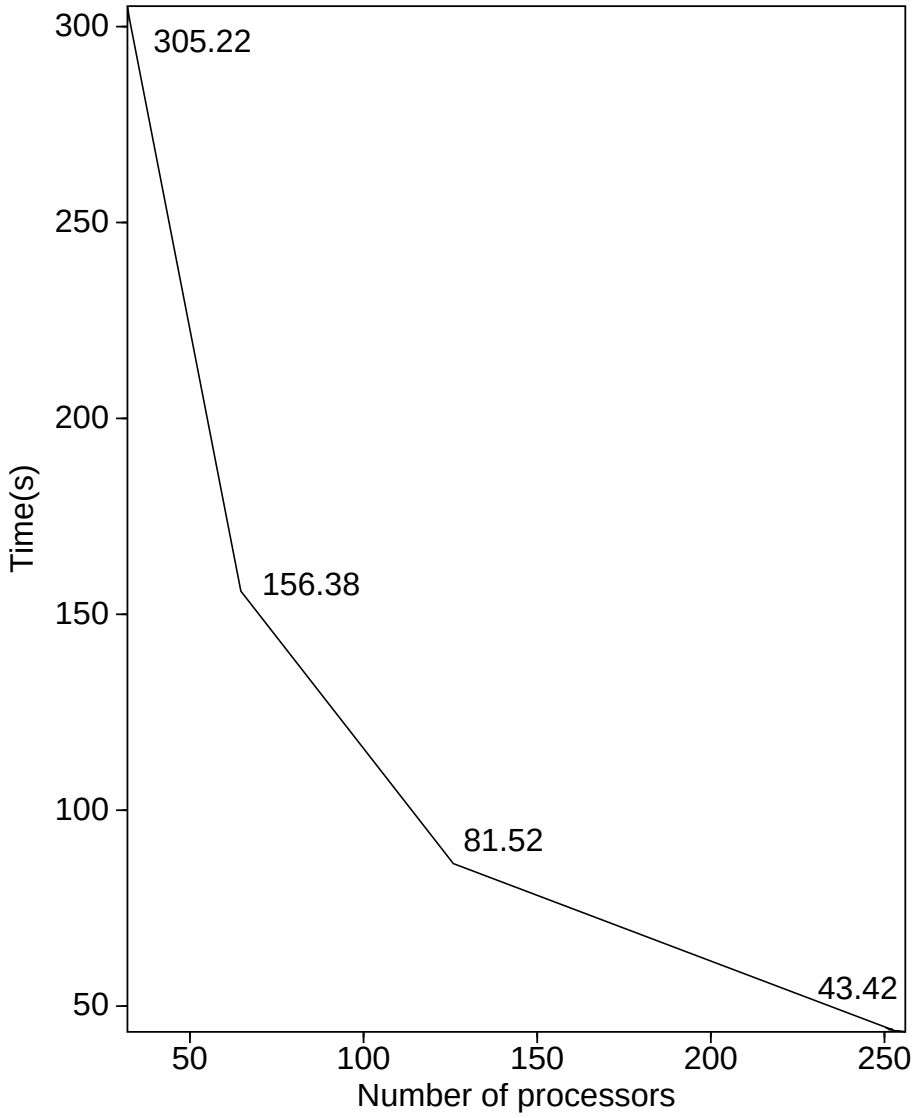


FIG. 4.5. Elapsed time versus number of processors (CM-5).

while, as the number of nodes increases the computation load for each node decreases due to a lower computation/communication ratio. The machines were not run in a dedicated mode for these tests, so the results presented here are representative of a realistic computational environment.

Below is a table of comparison of the CPU time on various numbers of IBM RS6000 workstations.

Table 3: Results on IBM workstations.

Number of workstations	CPU time	Speedup	Efficiency (%)
1	112m,23.32s	1	100
2	61m,47.07s	1.819	91
4	32m,22.12s	3.47	86
8	18m,46.32s	5.99	75

We used a distributed-memory MIMD parallel model for this implementation. The speedup results, as shown in the table, show reduction of efficiency with the decrease of computation/communication ratio. There is a fixed time for I/O and allocation overhead that contributes to the computation/communication ratio while the computation workload decreases.

4.5 Conclusion

I have shown that the inversion algorithm is suitable for parallel computation and that such parallel computation is quite efficient. The efficiency for the parallel implementation will be better for larger problems, where the computation/communication ratio is higher.

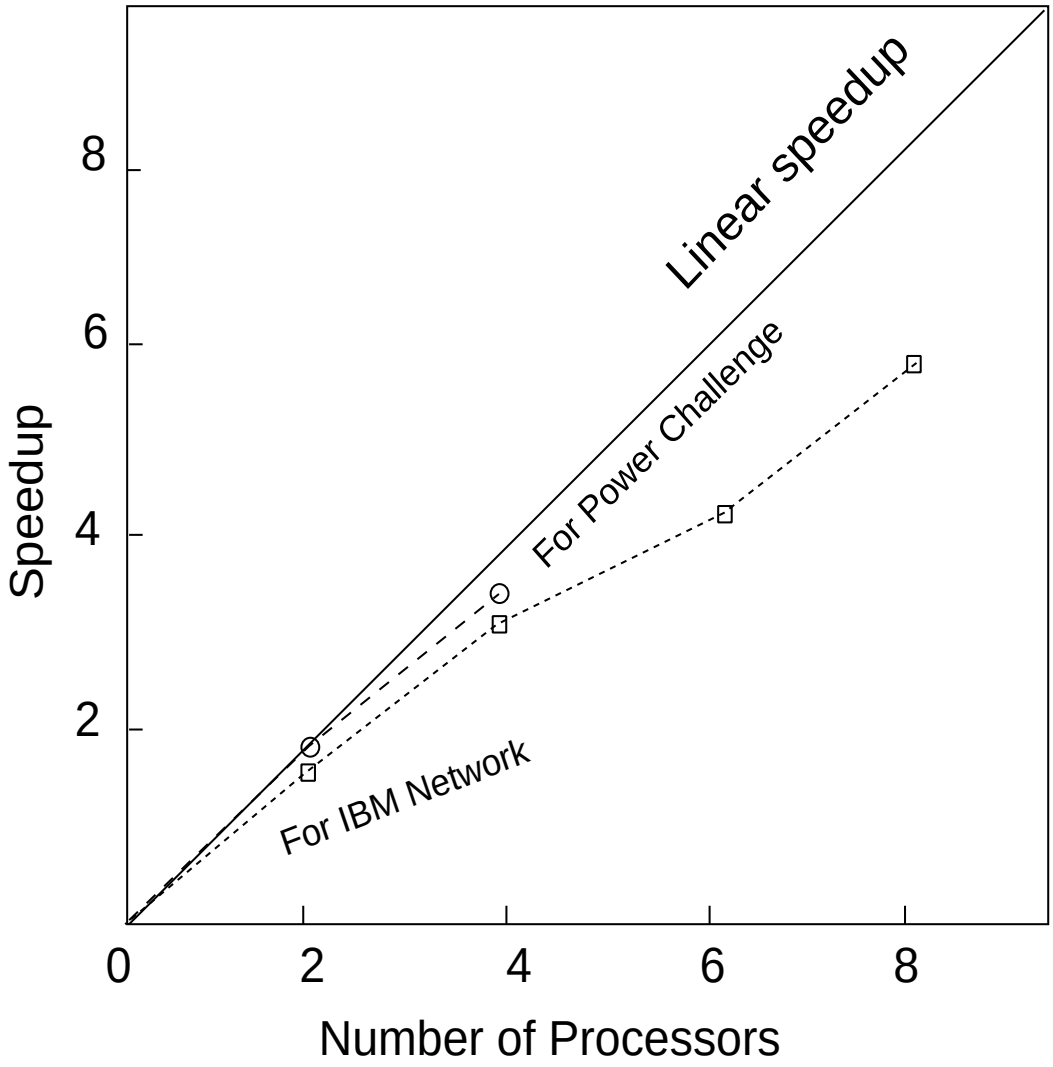


FIG. 4.6. Speedup using PVM.

Chapter 5

DATA EXAMPLES

5.1 Introduction

In this chapter, I present results of tests with the inversion program. The objective of these examples is to test the accuracy of reflector location and reflection coefficient.

5.2 Example 1

The first example provides a simple test of positioning and amplitude accuracy. The model consists of constant-velocity layers, as shown in Figure 5.1. Layer velocities are 2, 3, 6 and 10 km/s from top to bottom. Figure 5.2 shows the model data with offset, 3 km. The inversion produces two outputs: one for β (2.1) and one for β_1 (2.7). Figure 5.3 shows the result for β. The amplitudes in the two outputs should differ by the factor $\cos\theta$, where θ is the specular angle. Tables 4, 5 and 6 show comparisons of exact values and inversion results for R, $R\cos\theta$ and θ; θ is obtained by computing the ratio of R and $R\cos\theta$.

Table 4: True and inversion results for R.

Interface	True	Inversion	Error (%)
1	0.200	0.203	1.5
2	0.333	0.341	2.4
3	0.250	0.256	2.3

Table 5: True and inversion results for R$\cos\theta$.

Interface	True	Inversion	Error (%)
1	0.172	0.174	1.1
2	0.307	0.315	2.6
3	0.235	0.241	2.6

Table 6: True and inversion results for θ (degrees).

Interface	True	Inversion	Error (%)
1	30.90	30.84	0.2
2	22.76	22.71	0.2
3	19.94	19.89	0.3

Note that the error in the calculation of θ is much less than that of R and $R\cos\theta$. Because the processes for computing β and β_1 are very similar, their errors tend to have the same trend and their ratio is more accurate than is either of them, individually.

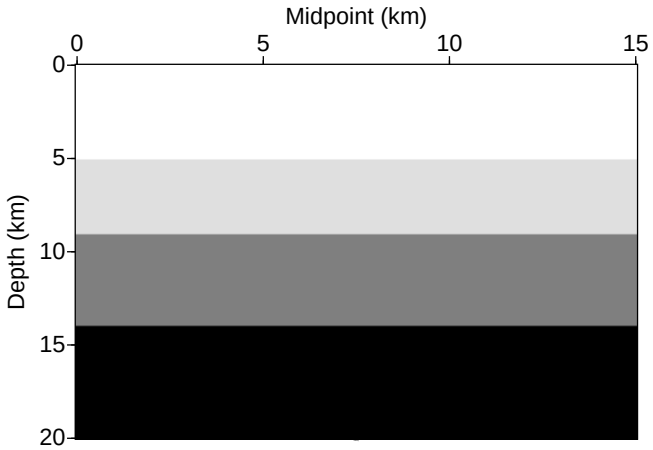


FIG. 5.1. Velocity model for Example 1.

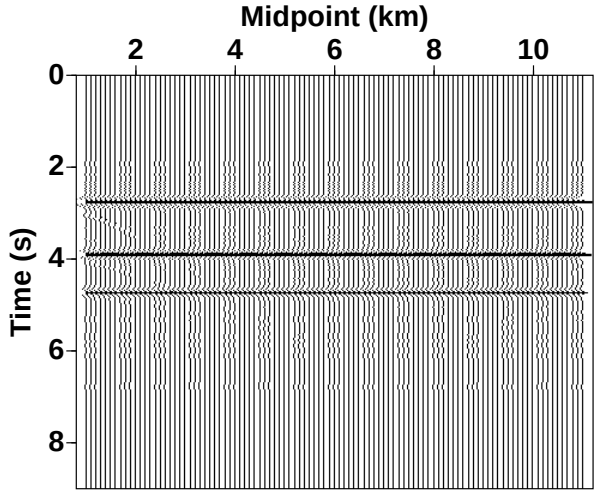


FIG. 5.2. Synthetic data for Example 1.

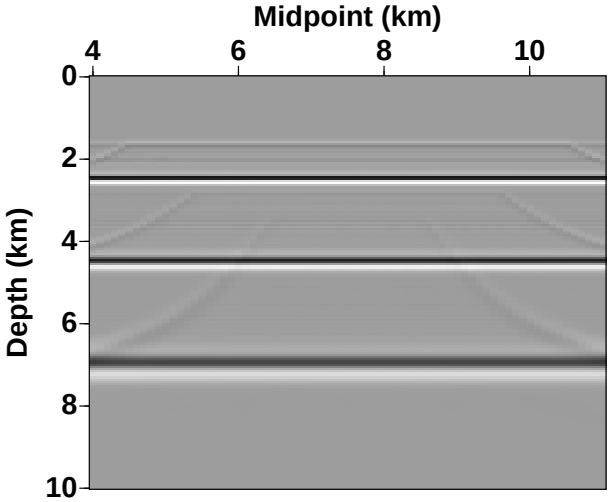


FIG. 5.3. Output β for Example 1.

5.3 Example 2

Another example, in Figure 5.4 and Figure 5.5, shows a synthetic dataset and its inversion results. The model is a spherical object located in a constant velocity medium. The synthetic data are generated in 3D with an offset of approximately five times the diameter of the object. The displays were generated by GOCAD, which is a software system for constructing 3D geological models and 3D grids [GOCAD, 1995]. In the inversion image, an isosurface is created from the peak amplitude so that the spherical object can be shown in 3D without the diffraction events. The reflection coefficient at the normal-incidence position (the top of the sphere) has an error of only 3.1 percent. For positions on the top portion of the sphere, the reflection coefficient is comparably accurate except in the neighborhood of the equator, where we have no specular returns. There, the sphere is filled in by diffraction returns rather than by imaged true specular reflections. The lower half of the sphere is inaccurately positioned because the change in propagation speed across the upper sphere surface does not satisfy the criterion of being a depth-dependent background velocity model. Thus, I used a constant velocity (1.5 km/s) down to the water/seabed interface and another constant below (2 km/s). Both are lower than the true velocity of 6 km/s in the sphere. Since the data were generated with the correct velocity, the specular returns from the lower part of the sphere came sooner than they would have at 2 km/s. Thus, in the imaging, the lower part of the sphere is *pulled up*, making a somewhat elliptical surface.

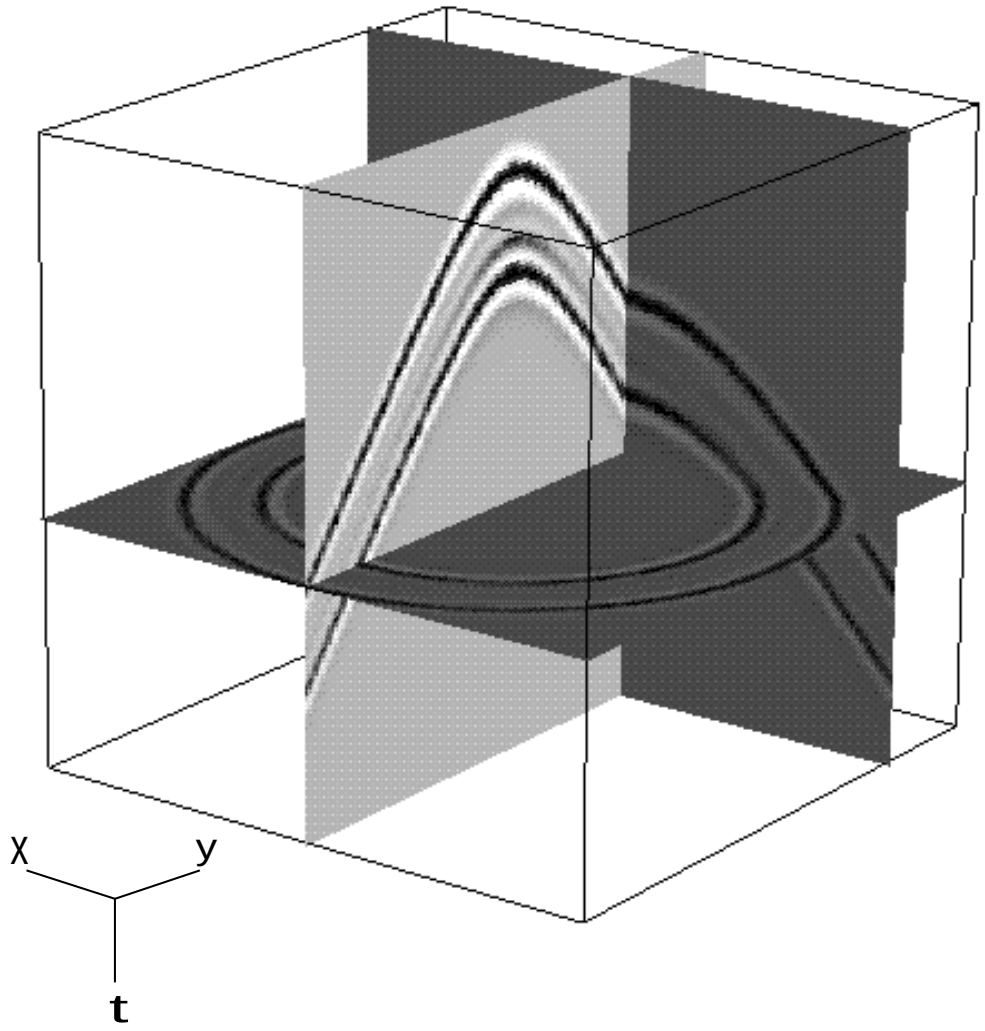


FIG. 5.4. Synthetic data for Example 2.

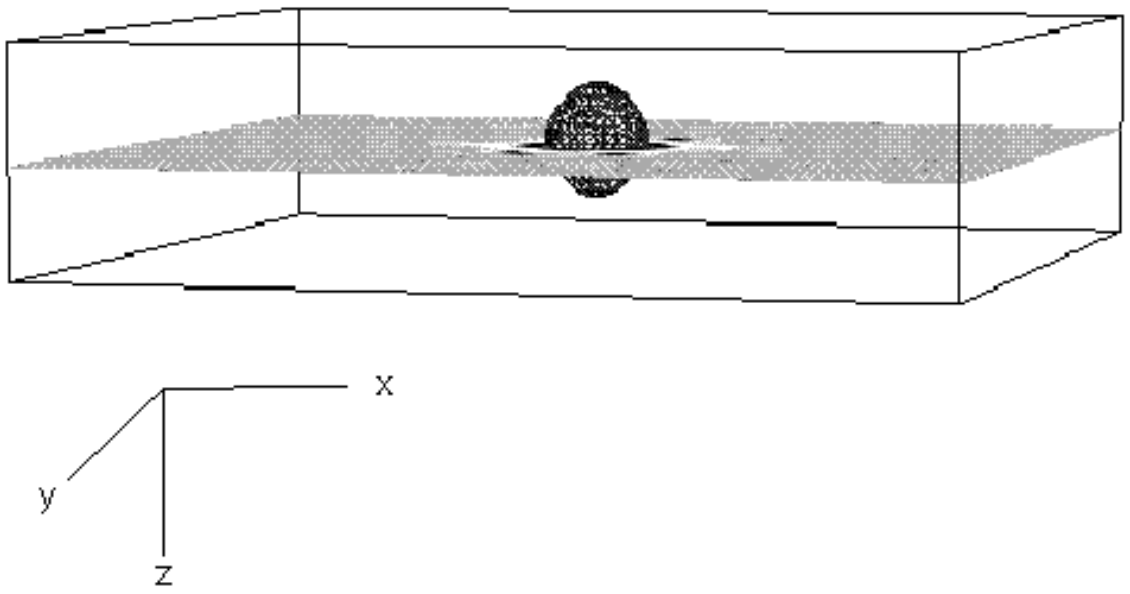


FIG. 5.5. Inversion from the synthetic data for Example 2.

Chapter 6

CONCLUSION

I have developed a 3-D common-offset inversion code that works for depth-dependent medium. The algorithm produces relatively true amplitude weights. I derived a ray tracing algorithm to compute the necessary travetime and amplitude information. The ultimate output of the inversion operator is the reflector map delineated by the singular functions of the interface with angularly dependent reflection coefficients. The results show that the amplitude weights and thus the reflection coefficients obtained by inversion has a small percentage error, compared with the true reflection coefficients. The method is stable and efficient.

I also derived a parallel scheme to further speedup the process. The results show that this kind of 3-D problem is applicable to middle-sized computers such as a network of workstations.

REFERENCES

Beylkin, G., Miller, D., and Oristaglio N., 1985, Spatial resolution of migration algoritms, "Acoustic Imaging 14," Plenum, New York, 155-167.

Bleistein, N, 1984, Mathematical Methods for Wave Phenomena, Academic Press, San Diego.

Bleistein, N, 1986, Kirchhoff inversion for reflector imaging and sound speed and density variations, Proceedings of EAEG/SEG workshop on Deconvolution and Inversion.

Bleistein, N, Cohen, J. K., and Hagin F. G., 1987, Two and one-half dimensional Born inversion with an arbitrary reference: Geophys, 52, 26-36.

CM5: Technical Summary, 1992, Thinking Machines Corporation

Cohen, J., Hagin, F., and Bleistein, N., 1986, Three-dimensional Born inversion with an arbitrary reference: Geophys., **51**, 1552-1558.

Geist, A. and Beguelin, A. and Dongarra, J. and Jiang, W. and Manchek, R. and Sunderman, V., 1994, PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing, MIT-Press, Boston.

GOCAD, 1995, GOCAD++ User's Manual.

Liu, Z, 1993, A Kirchhoff approach to seismic modeling and prestack depth migration: CWP-137, Colorado School of Mines, 217–240.

Ou, B, 1995, 2.5-D Common Offset Inversion in Triangulated Background Models of the Earth. Master's thesis, Colorado School of Mines.

SGI documentation, MIPSpro 64-Bit Porting and Transition Guide.

Sullivan, M. and Cohen, J. K. 1987. Pre-stack Kirchhoff inversion of common offset data. Geophysics, 52, 745-754.

Sumner, B., 1990, CZ1: Fortran program for 2.5-D stratified velocity inversion, descriptions, and instructions: Computer program documentation CWP-U06R, Colorado School of Mines.

Trier, V., 1991, A systolic approach for prestack migration on Connection machine. Geophysics, 53, 708-714.