

**A MULTITASKING IMPLEMENTATION
OF A 3D HIGH ORDER FINITE
DIFFERENCE FORWARD MODELING PROGRAM
ON THE CRAY X-MP/416**

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston - University Park

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Omar S. Terki-hassaine

August, 1987

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my advisor, Dr. Ernst Leiss, for his help, patience, and understanding.

I am also grateful to the members of my thesis committee, Dr. Gerald Gardner and Dr. Duane Pyle for their criticisms and suggestions.

I wish to thank everyone in the Allied Geophysical Laboratories without exception, for their assistance in every way.

Finally, I wish to thank my wife, Latifa, for her support and patience constantly displayed in my graduate studies. I dedicate this thesis to her and my parents.

**A MULTITASKING IMPLEMENTATION
OF A 3D HIGH ORDER FINITE
DIFFERENCE FORWARD MODELING PROGRAM
ON THE CRAY X-MP/416**

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston - University Park

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Omar S. Terki-hassaine

August, 1987

ABSTRACT

We present a multitasked implementation of a 3D out-of-core seismic forward modeling on the CRAY XMP/416. The algorithm is based on the forward explicit high order finite difference method.

We give a brief theoretical overview of 3D forward modeling and the resolution of the 3D wave equation. We discuss the coefficient matrix generated by the above mentioned approach. An in-core and an out-of-core version of a reentrant subroutine were designed to perform any fraction of the matrix vector multiplication independently. The rest of the program takes advantage of the microtasking feature which enables the system to treat independent iterations of Do Loops as subtasks to be performed by any available processor.

The comparison of the measured speed-ups obtained of the multitasked programs (two, three, and four processors) versus the unitasked programs shows that the combination of the macrotasking and microtasking features enabled us to reach approximately 80 percent of the ideal speed-up.

The modeling results are significantly improved by using the absorbing boundaries. We tested the program to determine the number of points per wavelength. We found that the number of grid points per wavelength is inversely

proportional to the order of the finite difference; and this led us to conclude that there is a significant reduction in memory requirements and in CPU time for using higher order. We discuss the effect of the elimination of a point from the high order configuration on the modeling result and the execution time. We show why the only acceptable result is when the outer point is eliminated. Finally the time sections of the SALFRH model collected at the plane $Z = 3$ are presented when we have a point source and a plane wave source in the model and for the exploding reflector model.

Due to the availability of the SSD (Solid State Storage Device) and its 1250 Mbps dual channel, the I/O wait time was virtually close to zero.

TABLE OF CONTENTS

	Page
ABSTRACT	v
I. INTRODUCTION	1
II. THEORY	5
2.1 High Order Finite Difference Operators.....	5
2.2 Forward Explicit High Order Finite Difference Method	11
III. CRAY X-MP FEATURES	14
3.1 Cray X-MP Overview	14
3.2 Multitasking Basics	17
3.2.1 Parallelism	18 ✓
3.2.2 Task Granularity	19 ✓
3.2.3 Scope of Variables	20
3.2.4 Factors Affecting Performance	20
3.2.5 Software Support	21
3.2.6 Macrotasking	23
3.2.7 Microtasking.....	25
IV. DESIGN FEATURES OF THE PROGRAM.....	28

4.1	Matrix Vector Multiplication Routine	28 ~
4.2	Use of Microtasking	34
V	RESULTS AND DISCUSSION.....	38
5.1	Performance Results	38
5.2	Forward Modeling Results	40
5.2.1	Choice of Parameters	40
5.2.2	Absorbing Boundaries	43
5.2.3	Point Source Wave Modeling.....	44
5.2.4	Plane Wave Modeling.....	45
5.2.3	Effect of Point Elimination on Modeling Results.....	45
5.2.6	SALFRH Results	47
VI.	USER'S GUIDE TO THE PROGRAMS	51
6.1	Parameters	51
6.2	Input of the Programs	54
6.3	Output of the Programs.....	57
VII.	CONCLUSION.....	64
	REFERENCES	66
	LIST OF FIGURES AND TABLES.....	68

CHAPTER 1

INTRODUCTION

Forward modeling is a numerical method that synthesizes (generates) cross time sections of complex geological subsurface structures. Seismic interpretation attempts to obtain a picture of the subsurface from field data gathered by reflection seismology. If the results of forward modeling agree with the actual field data, chances are very good that the interpreted model is an accurate one.

In the last decade, wave theory was introduced into seismic data processing (see Claerbout, 1970). Wave theory methods are very demanding of computer resources and thus have been applied only after much data reduction has been performed using other, less expensive techniques.

The advent of supercomputers such as the CRAY has now made it possible to use wave equation methods prior to data reduction. Here, the 3D acoustic wave equation is the basis for 3D forward modeling. There are several techniques used for solving the second order partial differential equations, among them the finite element and the Fourier series method. The method we used in our program to perform 3D forward modeling is a high order finite difference method. This method is very sensitive to the number of grid points per wavelength and may cause what is called a grid dispersion which in turn leads to inaccurate results. For this reason, one has to be careful about choosing the order and the grid

size for a given model. The higher the order, the less the number of grid points per wavelength. As is pointed out in Alford et al., 1974, there is a general rule for second order methods that ten grid points are required to resolve a given wavelength.

There have been several successful implementations of 3D forward modeling on uniprocessor vector computers. One attempt has been made by Juang, (1985) to use multitasking to perform 3D forward modeling using FFT's as a basis in the algorithm to solve the acoustic wave equation.

In this study, Chapter 2 introduces the 3D wave equation resolution algorithm based on the high order finite difference method. A brief overview of the CRAY X-MP/416 is presented in Chapter 3 followed by a description of the CRAY multitasking features used to implement the program. The flowtrace analysis of the unitasked program shows that the matrix vector multiplication gets the largest share of CPU time (approximately 96 percent). Chapter 4 concentrates on how we optimize this operation starting from the full vectorization of the unitasked version to the work balancing of the multitasked version.

Two versions of the reentrant subroutine were designed. The first was an out-of-core version where the program segments that performed I/O were considered critical regions. The second was an in-core routine and its implementation was motivated by the latest update of the central memory size from 8 to 16 Mwords. This subroutine is a high granularity task that can be distributed to two, three,

or four processors.

A performance study is carried out in which a theoretical speed-up is derived taking into account the multitasking overhead.

Chapter 5 first discusses the results obtained when the multitasked implementations were run in a dedicated environment. The multitasked implementation provides satisfactory speed-ups. A better performance is achieved when we use the microtasking features for small granularity tasks such as DO loops. A comparison of the measured speed-up values with the theoretical ones is presented.

The program was tested for a triangular block to carry out the parameter study. The number of points per wavelength was tested by the program to be inversely proportional to the order of the finite difference. One is tempted to believe that the higher the order, the more CPU time and memory space are needed to perform 3D forward modeling. We show that this is not so; there is a reduction in CPU time and a significant saving in memory space requirements. The use of absorbing boundaries has enabled us to get better forward modeling results.

The program was designed to perform 3D forward modeling for a point source or a plane wave anywhere in the model as well as for the receivers, and for the exploding reflector scheme.

Simple models such as a horizontal layer and a triangular block were used to test the program for the point source and the plane wave modeling. We finally

present the 3D SALFRH model time sections using the exploding reflector model, the point source and the plane wave.

A user's guide of the different versions of the 3D forward modeling multitasked program is given in Chapter 6.

CHAPTER 2

THEORY

In this chapter, we discuss the resolution of the 3D wave equation using a forward explicit high order finite difference method.

We derive all the formulas related to this approach. We also discuss the structure of the coefficient matrix generated by this technique. Based on this theoretical algorithm, we develop the 3D forward modeling programs and their multitasked versions.

2.1 HIGH ORDER FINITE DIFFERENCE OPERATORS

The basis for 3D forward modeling is the well known 3D acoustic wave equation:

$$\frac{\partial^2 P(x, y, z, t)}{\partial x^2} + \frac{\partial^2 P(x, y, z, t)}{\partial y^2} + \frac{\partial^2 P(x, y, z, t)}{\partial z^2} = \frac{1}{V^2(x, y, z)} \frac{\partial^2 P(x, y, z, t)}{\partial t^2} + f(x, y, z),$$

where $P(x, y, z, t)$ is the pressure at each grid point (x, y, z) and at a given time t . $V(x, y, z)$ is the velocity and $f(x, y, z)$ is the source term at each grid point.

The high order finite difference operators are (see Davis, 1963):

$$\Phi_t^2 P_{k,l,j}^m = P_{k,l,j}^{m-1} - 2P_{k,l,j}^m + P_{k,l,j}^{m+1},$$

$$\Phi_x^2 P_{k,l,j}^m = \sum_{i=1}^n C_i P_{k-i,l,j}^m + B P_{k,l,j}^m + \sum_{i=1}^n C_i P_{k+i,l,j}^m,$$

$$\Phi_y^2 P_{k,l,j}^m = \sum_{i=1}^n C_i P_{k,l-i,j}^m + B P_{k,l,j}^m + \sum_{i=1}^n C_i P_{k,l+i,j}^m,$$

$$\Phi_z^2 P_{k,l,j}^m = \sum_{i=1}^n C_i P_{k,l,j-i}^m + B P_{k,l,j}^m + \sum_{i=1}^n C_i P_{k,l,j+i}^m,$$

$$\Phi_{x,y,z}^2 P_{k,l,j}^m = \frac{1}{\Delta x^2} \Phi_x^2 P_{k,l,j}^m + \frac{1}{\Delta y^2} \Phi_y^2 P_{k,l,j}^m + \frac{1}{\Delta z^2} \Phi_z^2 P_{k,l,j}^m,$$

$$n = \frac{NORD}{2},$$

where NORD is the order of the difference approximation and is always an even positive integer.

The next step is to determine the values of the coefficients C_i ($i = 1, \dots, n$) and B in the previous high order central difference operators. A linear system of equations is needed to solve for the values of these coefficients.

Before we derive the general result of the linear system of equations, let us assume the fourth order scheme and let the pressure P be defined in the one dimensional domain.

From the well known Taylor formula, we obtain the following:

$$\begin{aligned} P(t + \Delta t) = P(t) + P'(t) + \frac{1}{2!}P^{(2)}(t)\Delta t^2 + \frac{1}{3!}P^{(3)}(t)\Delta t^3 \\ + \frac{1}{4!}P^{(4)}(t)\Delta t^4 + \frac{1}{5!}P^{(5)}(t)\Delta t^5 + \frac{1}{6!}P^{(6)}(t)\Delta t^6 + \dots, \end{aligned} \quad (2.1.2)$$

$$\begin{aligned} P(t - \Delta t) = P(t) - P'(t) + \frac{1}{2!}P^{(2)}(t)\Delta t^2 - \frac{1}{3!}P^{(3)}(t)\Delta t^3 \\ + \frac{1}{4!}P^{(4)}(t)\Delta t^4 - \frac{1}{5!}P^{(5)}(t)\Delta t^5 + \frac{1}{6!}P^{(6)}(t)\Delta t^6 + \dots, \end{aligned} \quad (2.1.3)$$

$$\begin{aligned} P(t + 2\Delta t) = P(t) + 2P'(t) + \frac{2^2}{2!}P^{(2)}(t)\Delta t^2 + \frac{2^3}{3!}P^{(3)}(t)\Delta t^3 \\ + \frac{2^4}{4!}P^{(4)}(t)\Delta t^4 + \frac{2^5}{5!}P^{(5)}(t)\Delta t^5 + \frac{2^6}{6!}P^{(6)}(t)\Delta t^6 + \dots, \end{aligned} \quad (2.1.4)$$

$$\begin{aligned} P(t - 2\Delta t) = P(t) - 2P'(t) + \frac{2^2}{2!}P^{(2)}(t)\Delta t^2 - \frac{2^3}{3!}P^{(3)}(t)\Delta t^3 \\ + \frac{2^4}{4!}P^{(4)}(t)\Delta t^4 - \frac{2^5}{5!}P^{(5)}(t)\Delta t^5 + \frac{2^6}{6!}P^{(6)}(t)\Delta t^6 + \dots, \end{aligned} \quad (2.1.5)$$

$$P(t) = P(t) \quad (2.1.6)$$

By multiplying both sides of expressions (2.1.2) and (2.1.3) by C_1 , multiplying both sides of expressions (2.1.4) and (2.1.5) by C_2 , multiplying both sides of expression (2.1.6) by B and adding them, we get the following:

$$\begin{aligned} & C_2P(t - 2\Delta t) + C_1P(t - \Delta t) + BP(t) + C_1P(t + \Delta t) + C_2P(t + 2\Delta t) \\ &= (2C_1 + 2C_2 + B)P(t) + \frac{2}{2!}(C_1 + 2^2C_2)P^{(2)}(t)\Delta t^2 \\ &+ \frac{2}{4!}(C_1 + 2^4C_2)P^{(4)}(t)\Delta t^4 + O(\Delta t^6). \end{aligned}$$

To get the fourth order finite difference formula

$$P^{(2)}(t) = \frac{1}{\Delta t^2} [C_2P(t - 2\Delta t) + C_1P(t - \Delta t) + BP(t) + C_1P(t + \Delta t) + C_2P(t + 2\Delta t)],$$

we must let

$$\begin{cases} 2C_1 + 2C_2B &= 0 \\ \frac{2^2}{2!}(C_1 + 2^2C_2) &= 1 \\ \frac{2}{4!}(C_1 + 2^4C_2) &= 0, \end{cases}$$

and so we get the following linear equation:

$$\begin{cases} 2C_1 + 2C_2B &= 0 \\ C_1 + 2^2C_2 &= 1 \\ C_1 + 2^4C_2 &= 0, \end{cases} \quad (2.1.7)$$

by solving the linear system of equations (2.1.7), we get the values for C_1, C_2 , and B easily, namely $16/12$, $-1/12$, and 16 .

In order to get the general linear system for any order we prove the theorem below (see Shen, 1986)

Theorem: If the C_i ($i=1, \dots, n$) and B satisfy:

$$\begin{cases} 2 \sum_{i=1}^n C_i + B &= 0 \\ \sum_{i=1}^n i^{2k} C_i &= 1 \\ \sum_{i=1}^n i^{2k} C_i &= 0 \end{cases} \quad k = 2, \dots, n.$$

then

$$P^{(2)}(t) = \frac{1}{\Delta t^2} \left[\sum_{i=1}^n C_i P(t - i\Delta t) + BP(t) + \sum_{i=1}^n C_i P(t + i\Delta t) \right] + O(\Delta t^{2n+2})$$

Proof: According to Taylor's formula, we have:

$$\begin{aligned} P(t + i\Delta t) &= P(t) + \sum_{i=1}^{n+1} \frac{i^{2k-1}}{(2k-1)!} P^{(2k-1)}(t) \Delta t^{2k-1} \\ &\quad + \sum_{i=1}^n \frac{i^{2k}}{(2k)!} P^{(2k)}(t) \Delta t^{2k} \\ &\quad + \frac{i^{(2n+2)}}{(2n+2)!} P^{(2n+2)}(t) \Delta t^{2n+2} + \dots, \end{aligned} \quad (2.1.8)$$

$$\begin{aligned} P(t - i\Delta t) &= P(t) - \sum_{i=1}^{n+1} \frac{i^{2k-1}}{(2k-1)!} P^{(2k-1)}(t) \Delta t^{2k-1} \\ &\quad + \sum_{i=1}^n \frac{i^{2k}}{(2k)!} P^{(2k)}(t) \Delta t^{2k} \\ &\quad + \frac{i^{(2n+2)}}{(2n+2)!} P^{(2n+2)}(t) \Delta t^{2n+2} + \dots, \end{aligned} \quad (2.1.9)$$

$$i = 1, \dots, n,$$

$$P(t) = P(t), \quad (2.1.10)$$

and if we multiply both sides of expressions (2.1.8) and (2.1.9) by C_i ($i=1, \dots, n$),

multiply both sides of expression (2.1.10) by B , and adding them, we obtain:

$$\begin{aligned}
& \sum_{i=1}^n C_i P(t + i\Delta t) + \sum_{i=1}^n C_i P(t - i\Delta t) + BP(t) = \\
& (2 \sum_{i=1}^n C_i + B)P(t) + 2(\sum_{i=1}^n \frac{i^2}{2!} C_i)P^{(2)}(t)\Delta t^2 \\
& + 2 \sum_{i=1}^n C_i \sum_{k=2}^n \frac{i^{2k}}{(2k)!} P^{(2k)}(t)\Delta t^{2k} + O(\Delta t^{2n+2}) \\
& = (2 \sum_{i=1}^n C_i + B)P(t) + 2(\sum_{i=1}^n \frac{i^2}{2!} C_i)P^{(2)}(t)\Delta t^2 \\
& + 2 \sum_{k=2}^n (\sum_{i=1}^n \frac{i^{2k}}{(2k)!} C_i)P^{(2k)}(t)\Delta t^{2k} + O(\Delta t^{2n+2})
\end{aligned}$$

so if C_i ($i=1, \dots, n$) and B satisfy:

$$\begin{cases} 2 \sum_{i=1}^n C_i + B & = 0 \\ 2 \sum_{i=1}^n \frac{i^2}{2!} C_i & = 1 \\ \sum_{i=1}^n \frac{i^{2k}}{(2k)!} C_i & = 0 \end{cases} \quad k = 2, \dots, n.$$

that is, C_i ($i=1, \dots, n$) and B satisfy

$$\begin{cases} 2 \sum_{i=1}^n C_i + B & = 0 \\ \sum_{i=1}^n i^2 C_i & = 1 \\ \sum_{i=1}^n i^{2k} C_i & = 0 \end{cases} \quad k = 2, \dots, n.$$

it follows that

$$P^{(2)}(t) = \frac{1}{\Delta t^2} \left[\sum_{i=1}^n C_i P(t - i\Delta t) + BP(t) + \sum_{i=1}^n C_i P(t + i\Delta t) \right] + O(\Delta t^{2n+2}).$$

2.2 FORWARD EXPLICIT HIGH ORDER FINITE DIFFERENCE METHOD

The forward explicit high order finite difference method is (see Shen, 1986)

$$\Phi_{x,y,z}^2 P_{k,l,j}^m = \frac{1}{V_{k,l,j}^2 \Delta t^2} \Phi_t^2 P_{k,l,j}^m + f_{k,l,j}^m.$$

The equation is interpreted as follows. At time step m , $P_{k,l,j}^m$ and $P_{k,l,j}^{m-1}$ are known for all values of k , l , and j . We solve for $P_{k,l,j}^{m+1}$.

The complete solution for the forward explicit high order finite difference method for all time steps (see Shen, 1986) can be written as follows:

$$\begin{cases} P^1 = D^{-1} F \\ P^2 = D^{-1} (F - Q * P^1) \\ \vdots \\ P^i = D^{-1} (F - Q * P^{i-1} - D * P^{i-2}) \quad i = 3, \dots, lt \end{cases}$$

where D is a diagonal matrix, whose elements are:

$$d_{k,l,j} = -\frac{1}{V_{k,l,j}^2 \Delta t^2}$$

$$k = 1, \dots, lx \quad ; \quad l = 1, \dots, ly \quad ; \quad j = 1, \dots, lz$$

and F is the source vector. The matrix Q is a large banded matrix which has been

compressed to look like the following:

$$\begin{pmatrix} 0 & 0 & \dots & 0 & M_1 & Z_{11} & \dots & \dots & Z_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & Z_{n,n-1} & \dots & Z_{n,1} & M_n & Z_{n,1} & \dots & \dots & Z_{n,n} \\ Z_{n+1,n} & \dots & \dots & Z_{n+1,1} & M_{n+1} & Z_{n+1,1} & \dots & \dots & Z_{n+1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ Z_{lz-n,n} & \dots & \dots & Z_{lz-n,1} & M_{lz-n} & Z_{lz-n,1} & \dots & \dots & Z_{lz-n,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ Z_{lz,n} & \dots & \dots & Z_{lz,1} & M_{lz} & 0 & \dots & 0 & 0 \end{pmatrix}$$

It is an $(L_x * L_y * L_z, 3 * \text{Nord}+1)$ -matrix. The Z vectors are defined as follows:

$$Z_{i,j} = \frac{C_j}{\Delta z^2} \quad i = 1, \dots, lz \quad ; \quad j = 1, \dots, n.$$

The internal matrices M are $(L_x * L_y, 2 * \text{Nord}+1)$ -matrices that can be written as follows:

$$\begin{pmatrix} 0 & 0 & \dots & 0 & P_1 & Y_{11} & \dots & \dots & Y_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & Y_{n,n-1} & \dots & Y_{n,1} & P_n & Y_{n,1} & \dots & \dots & Y_{n,n} \\ Y_{n+1,n} & \dots & \dots & Y_{n+1,1} & P_{n+1} & Y_{n+1,1} & \dots & \dots & Y_{n+1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ Y_{ly-n,n} & \dots & \dots & Y_{ly-n,1} & P_{ly-n} & Y_{ly-n,1} & \dots & \dots & Y_{ly-n,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ Y_{ly,n} & \dots & \dots & Y_{ly,1} & P_{ly} & 0 & \dots & 0 & 0 \end{pmatrix}$$

The Y vectors are defined as follows:

$$Y_{i,j} = \frac{C_j}{\Delta y^2} \quad i = 1, \dots, ly \quad ; \quad j = 1, \dots, n.$$

The internal matrices P are (Lx, Nord+1)-matrices that can be written as follows:

$$\begin{pmatrix} 0 & 0 & \dots & 0 & a_1 & X_{11} & \dots & \dots & X_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & X_{n,n-1} & \dots & X_{n,1} & a_n & X_{n,1} & \dots & \dots & X_{n,n} \\ X_{n+1,n} & \dots & \dots & X_{n+1,1} & a_{n+1} & X_{n+1,1} & \dots & \dots & X_{n+1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{lx-n,n} & \dots & \dots & X_{lx-n,1} & a_{lx-n} & X_{lx-n,1} & \dots & \dots & X_{lx-n,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{lx,n} & \dots & \dots & X_{lx,1} & a_{lx} & 0 & \dots & 0 & 0 \end{pmatrix}$$

where the central column is defined as follows:

$$a_i = \frac{B}{\Delta x^2} + \frac{B}{\Delta y^2} + \frac{B}{\Delta z^2} + \frac{2}{V^2 \Delta t^2},$$

and the X vectors are defined as follows:

$$X_{i,j} = \frac{C_j}{\Delta x^2} \quad i = 1, \dots, lx \quad ; \quad j = 1, \dots, n.$$

At this point all the computation elements are defined therefore the pressure at each time step can be easily determined.

CHAPTER 3

CRAY X-MP FEATURES USED

The implementation of our algorithm is carried out on the CRAY X-MP system using its multiprocessing capabilities. This section will give a brief overview of this machine's multiprocessing architecture and multitasking features (see Cray mainframe reference manual and multitasking user guide, 1986).

3.1 AN OVERVIEW OF THE CRAY X-MP SYSTEM

The CRAY X-MP is a vector multiprocessor with an option of two to four identical processors, each an enhanced version of the CRAY-1 CPU. A combination of several architectural changes contribute to an improved performance in each processor over the CRAY-1's. The clock period is reduced from 12.5 to 8.5 ns, thereby shortening the time between two consecutive results.

The processors of the CRAY X-MP multiprocessor system as illustrated in Figure 1 share a central memory organized in interleaved memory banks that can be accessed independently and in parallel during each machine clock period. The number of memory ports per processor is increased from one in the CRAY-1 to four. This allows two memory reads, one memory write, and either an input or output call to proceed simultaneously in each processor. Thus each processor

of a CRAY X-MP system has four times the memory bandwidth of a CRAY-1 system. The multiport memory has built-in conflict resolution hardware to minimize delays and maintain the integrity of simultaneous memory references to the same memory bank. Finally, the chaining mechanism which allows results of previous operations to enter the fixed chained slot time has been improved compared to its CRAY predecessors. Chains of operations may now include both memory read and write in addition to the arithmetic pipeline already available in the Cray predecessors.

Each CRAY X-MP processor offers very fast scalar processing with high speed processing of long and short vectors. Additionally, multiprocessor models enable users to exploit the extra dimension of multitasking. The scalar performance of each processor is attributed to its fast clock cycle, short memory access times, and large instruction buffers. Vector performance is supported by the fast clock, parallel memory ports, and flexible hardware chaining. These features allow simultaneous execution of memory fetches, arithmetic operations, and memory stores in a series of linked vector operations.

The mainframe communicates with the front end system and external data storage devices through the I/O subsystem. An optional Solid-State Storage Device or SSD provides an internal second level store that has two channels of 1250 Mbytes/s. The SSD allows the development of algorithms to solve larger and more sophisticated problems in science and engineering. In our implementation, it has been thoroughly used for all intermediate storage.

Special hardware enables the efficient and coordinated application of multiple processors to a single job. All processors assigned to a job share a unique set of binary semaphore and data registers. The semaphore registers allow each processor to signal the other processor that processing should begin, should wait, or has been completed. Semaphore deadlock, a programming error which causes each processor to wait for the other, is automatically detected by the hardware. Variable values and addresses may be passed quickly between processors using the data registers.

The Cray mainframe is designed for use with front-end computers in a computer network. A front end computer system is self contained and executes under the control of its own operating system. Front-end computers provide service to the Cray mainframe in the following ways:

- a. As a master operator station
- b. As a local operator station
- c. As a local batch entry station
- d. As a data concentrator for multiplexing several other stations into a single Cray channel
- e. As a remote batch entry station

- f. As an interactive communication station

This support allows the main frame to concentrate on performing high speed computations and data transfer between main and secondary memory.

3.2 MULTITASKING BASICS

Multitasking is a mode of operation that provides for the execution of two or more parts of a single program in parallel. A job efficiently multitasked requires less execution time, when measured from start to finish (wall clock time) on a dedicated multiprocessor system, than a job that is not multitasked.

Multitasking does not reduce the CPU cycles necessary to execute the program. In fact, multitasking introduces an overhead which increases the cumulative CPU time. The best theoretical gain that can be achieved from multitasking is that a job running on a dedicated system in wall clock time, t , without being multitasked could run on a dedicated system in wall clock time t/n if modified to use n parallel tasks on a machine with n CPUs. On a CRAY X-MP, with n being four, the optimum wall clock speed-up due to multitasking can not exceed a factor of four. Several factors reduce this maximal speed-up for a given program:

- a. Not all parts of a program can be divided into parallel tasks. Many algorithms do not have a parallel structure or have only a portion that can be parallelized.

- b. Those parts that can be multitasked may have dependencies on one another that result, at run time, in one or more tasks having to wait until others complete some operation. During this wait time, the waiting tasks do not contribute to parallelism.
- c. Use of the multitasking features incurs a certain amount of overhead that is lost to the job. The more often these features are used, the greater may be the overhead.

Multitasking is nondeterministic with respect to time; however, tasks must be made deterministic with respect to results. The key to a successful multitasked program is to precisely define and add the necessary communication and synchronization mechanisms between parallel tasks and to provide for the protection of the shared data.

3.2.1 PARALLELISM

Jobs, job steps, programs and subprograms are parallel if they are processed simultaneously rather than sequentially. Levels of parallelism are defined in terms of the types of software processes that are executed in parallel.

Level 1: Independent jobs, each job having a CPU.

Level 2: Job steps: related parts of the same job.

Level 3: Routines and subroutines.

Level 4: Loops.

Level 5: Statements.

The higher the number of the level, the smaller the size or granularity of the tasks. Multitasking is a special case of multiprocessing defining a task to be a job step or a subprogram. Parallelism level 2 and level 3 are macrotasking modes of operation. Parallelism level 4 is also called microtasking.

3.2.2 TASK GRANULARITY

A task is a unit of computation that can be scheduled. The instructions in a task are processed in sequential order. A task is a uniquely named process that can have code and data areas in common with other tasks of the same job. Multitasking produces the best speed-up when applied to balanced tasks of significant granularity. The granularity of parallel work must be sufficient to make multitasking worthwhile. For simple multitasking models, the following formula (see Multitasking User's Guide, 1984) gives the size of unpartitioned work of the original task required to obtain a desired speed-up.

$$X = \frac{SP * nCPU * OVERHEAD}{nCPU - SP}$$

To gain a speed-up of SP on $nCPU$ processors, with a multitasking overhead $OVERHEAD$ (measured in clock periods), the original unpartitioned task must

be at least X clock periods in size, assuming equal partitions. This implies that parallelism should be exploited at the highest level possible in order to obtain the greatest speed-up.

3.2.3 SCOPE OF VARIABLES

When converting a program to multitasking one has to be careful about the scope of variables. The ability of processors to access a variable is determined by the variable's scope. The scope of a variable is either global or local with respect to a subroutine. Global variables appear in COMMON blocks, SAVE statements, DATA statements, or in a subroutine's argument list. The result of an operation performed on a global variable by one processor is known to all processors. The local variables can be referenced only within a subroutine. A separate and private storage location for each local variable exists on the stack for each processor entering a multitasked subroutine. Thus the result of an operation on a local variable is known only to the processor that performed it. It is not accessible to other processors.

3.2.4 FACTORS AFFECTING PERFORMANCE

Several factors influence the performance of multitasked code as compared to the original program. Some of these factors come from the computer system such as the library calls overhead and the parallelism ratio (amount of sequential code in the program). However, the user has influence over the following factors, which

frequently have a greater impact on performance:

- a. Level (granularity) of parallelism exploited.
- b. Frequency of calls to the multitasking library.
- c. Partition of work and its distribution among processors.
- d. Programming style in the choice of multitasking mechanisms.

Of all the factors listed, the most important are granularity of task and balanced workload distribution. Figure 2 shows how the performance is affected by the load balancing.

3.2.5 SOFTWARE SUPPORT

The initial implementation of multitasking is at the FORTRAN level, where the user can write CALL statements to ask the system and library software for multitasking functions. Beginning at the FORTRAN level allows flexibility of implementation, ease of use for the programmer, and transportability to other Cray mainframes including single processor configurations.

Enhancements have been made to previously existing Cray system software to support multitasking. The Cray Fortran compiler, CFT, has been modified to produce stack-based, reentrant object code. The local variables of subroutines belonging to a task reside on a memory stack for that task. Reentrancy is a

property of code which allows one copy of the instructions to be executed, without destructive interference, by more than one processor at the same time.

The Cray Operating System, COS, has been modified to handle tasks rather than job steps as the basic unit of work. It permits a user program the use of library routines to create additional tasks belonging to the job. The user interface to the multitasking facilities of the operating system and the hardware is a library of routines contained in the utilities library, UTLIB. The multitasking library takes primary responsibility for managing and scheduling tasks within a program.

The key concept in the COS interface to the library scheduler is that of the Logical CPU. A logical CPU is the entity scheduled by COS for execution on physical CPUs and is identified as an entry in the COS Task Execution Table. Initially, COS assigns a job one logical CPU. But the library scheduler can request additional logical CPUs for a particular job, thereby bringing about multitasking at the COS level. The job of the library scheduler, therefore, is to connect user tasks to logical CPUs in the most efficient manner. If a task must wait for a lock or event, that task is disconnected from its logical CPU. The logical CPU is then freed for use by another task in the job or possibly for return to the system.

The library scheduler manages several queues of tasks. Tasks are moved between queues as their states change through the use of the multitasking facilities. The queues are generally handled in first-in, first-out order. However, when a task calls any of the multitasking subroutines, it is placed at the front of the Waiting

for Logical CPU queue.

3.2.6 MACROTASKING

As previously mentioned, macrotasking is implemented at the subroutine level. There are several library routines provided by the system that enable the programmer to perform macrotasking. Below is a brief description of the most frequently used ones:

TSKSTART

TSKSTART builds a stack for the task. It copies initial information from the task control array into a task Information block in the base of the stack. The task is then placed in the Waiting for Logical CPU queue, and control passes to the library scheduler.

TSKWAIT

TSKWAIT checks the status of the specified task. If the task has completed execution, control returns to the calling task. If the task is active, the calling task is placed in a Suspended queue, the identifier of the task for which it is waiting is saved, and control passes to the library scheduler.

LOCKON

LOCKON checks the status of the lock variable. If the lock variable is unlocked, the subroutine locks it and returns control. If the lock variable is locked,

the calling task is placed in a Suspended queue, the identifier of the lock for which the task is waiting is saved, and control passes to the library scheduler.

LOCKOFF

LOCKOFF changes the status of the lock variable to unlocked. LOCKOFF then removes the first task waiting for that lock from a Suspended queue and puts it in the Waiting for Logical CPU queue.

EVWAIT

EVWAIT checks the status of the event. If the status is posted, control returns to the calling task without further action. If the status is cleared, the task is put in a Suspended queue, the identifier of the event for which it is waiting is saved, and control passes to the library scheduler.

EVPOST

EVPOST changes the status of the event variable to posted. EVPOST then removes all tasks waiting for that event from a Suspended queue and puts them in the Waiting for Logical CPU queue. Control passes to the library scheduler.

EVCLEAR

EVCLEAR changes the status of the event variable to cleared, and control returns to the calling task.

The multitasking routines and library scheduler described above cause user

tasks to change from state to state over the course of a job. Figure 3 shows these transitions.

3.2.7 MICROTASKING

Microtasking has been introduced by Cray Research (see Booth, 1986) to allow programmers to take full advantage of the multiprocessing power when the task size or granularity may be small such as Do-loops.

Microtasking is specified by a small number of user supplied directives that appear as comment lines. A preprocessor, called PREMULT, interprets the directives and inserts the appropriate microtasking code. The processed program is then compiled and executed in the normal way. The addition of the microtasking directives does not reduce the portability of the original program in a CRAY environment.

Because of its low overhead, microtasking is very efficient. In addition microtasking works well when the number of processors available to a job is unknown or varies during the program's execution.

The following briefly describes the main control structures or directives used to convert a conventional program for microtasking on the CRAY X-MP series:

CMIC\$ GETCPUS (N)

This directive specifies the maximum number of processors, N, that can enter a microtasked routine. It must be specified before the call statement to a

microtasked subroutine.

CMIC\$ RELCPUS (N)

This directive appears upon returning from the microtasked section to the calling program. The above pair normally bracket the call statement to the microtasked routine.

CMIC\$ MICRO

This directive specifies a subroutine that is to be multitasked. The subroutine can then be entered by more than one processor. The return statement of the subroutine designates the end of the microtasked section.

CMIC\$ DO GLOBAL

This directive is immediately followed by a Do loop. The iterations of this loop are to be independent and may be executed in parallel. The end of this directive is determined by the statement whose label is referred to by the DO loop.

CMIC\$ PROCESS

This directive defines a process that is a segment of code within a microtasked section that is to be executed only once regardless of the number of processors that enter the microtasked section. A process may be scheduled for execution in parallel with other independent pieces of code.

CMIC\$ END PROCESS

This directive delineates the end of the segment of code that is started by the CMIC\$ PROCESS directive.

CMIC\$ ALSO PROCESS

Several processes may be enclosed within the single pair of directives CMIC\$ PROCESS and CMIC\$ END PROCESS. The directive CMIC\$ ALSO PROCESS separates the individual processes.

CMIC\$ GUARD

This directive protects the code following it from being entered by more than one processor at the same time. If several processors request at the same time to enter a guarded section of code, only one processor will be allowed to enter and the rest are made to wait for later scheduling one at a time.

CMIC\$ END GUARD

This directive marks the end of a guarded section.

Note that all directives begin with the letter C in column 1; thus the regular FORTRAN compiler treats them as comments. In fact, the microtasking preprocessor would also treat a directive as comment if there is any syntax error in the name of the directive; this can result in strange run time errors.

CHAPTER 4

DESIGN FEATURES OF THE PROGRAM

The objective of the program is to compute the pressure of every grid point of the model under study at each time step. Since the pressure at each time step depends on the earlier steps' pressures it makes this program an iterative program. Figure 4 illustrates the general algorithm used to perform 3D forward modeling. Part 1 generates the input data which in our case is the velocity cube. It subsequently computes the source cube, the diagonal cube, and the main column of the Q-matrix; finally it stores them in the Solid State Device (SSD). These operations are performed iteratively for every plane of the model by transferring data to the SSD as shown in Figure 5. The arrays F or source vector, D or diagonal vector, and A2 or Q-matrix main column vector all have the same dimension $LX * LY$. Part 2 starts by computing the matrix vector multiplication $Q * P$. This operation is the most time consuming and is discussed below. After the new pressure vector is computed for each time step the elements needed for the time sections and cross sections are written to output files.

4.1 MATRIX VECTOR MULTIPLICATION ROUTINE

The flowtrace analysis in Figure 6 shows that the matrix vector multiplication routine takes approximately 96 percent of program CPU time. The logical step

would be to try to optimize it and further macrotask it. We initially proceeded with optimizing the entire program by applying thoroughly the vectorizing rules found in the Cray optimization guide. The program mostly used the chaining rule by applying the distributive law in the expressions and the reordering of statements. It also used small outer loops with large inner loops and long vectors. The safest way before one attempts to implement the multitasked version is to predict the performance manually and consequently decide whether macrotasking can be effective. We designed two versions of the multiplication routine:

The first is an out-of-core version and includes I/O calls to transfer the data between memory and the SSD. The speed-up in this case, taking into account that 96 percent of the original program is macrotasked and also assuming a balanced work distribution is written as follows:

$$\begin{aligned}
 \text{Speedup} &= \frac{\text{Time}(1\text{cpu})}{\text{Overhead} + \text{Time}(N\text{cpu})} \\
 &= \frac{\text{Time}(1\text{cpu})}{\text{Overhead} + \frac{0.96 * \text{Time}(1\text{cpu})}{N} + 0.04 * \text{Time}(1\text{cpu})} \\
 &= \frac{N}{0.96 + 0.04 * N + \frac{N * \text{Overhead}}{\text{Time}(1\text{cpu})}}, \tag{1}
 \end{aligned}$$

where

$$\begin{aligned}
\text{Overhead} = & \text{TSKSTART Time} \\
& + \text{TSKWAIT Time} \\
& + (\text{LOCKON} + \text{LOCKOFF}) \text{ Time} \\
& + \text{Memory contention delay and work load imbalance.} \quad (2)
\end{aligned}$$

If we substitute the actual routine times we obtain the following:

$$\begin{aligned}
\text{Overhead} = & (N - 1)((NTS - 2) * 2500 + 1,500,000)cp \\
& + (N - 1)((NTS - 2) * 200 + 1500)cp \\
& + ((NTS - 1) * LZ/N * (Nord + 2) * (1500 + 1500))cp \\
& + 0.01 * Time(1cpu),
\end{aligned}$$

where N is the number of processors used, NTS is the number of time steps, LZ is the number of grid points in the Z direction and cp is a clock period equal to 8.5E-9 s.

The second version was motivated by the update of the CRAY X-MP memory size from 8 to 16 Mwords which has enabled us to store the 3 arrays used for the multiplication subroutine in central memory and eliminate all the I/O calls used in the first case. These arrays are the pressure vector at the previous time step, the

Q-matrix main column array and the multiplication result array. The theoretical performance is derived using the same assumptions as above with the exception that the overhead due to the critical region for I/O is not included. The new overhead is:

$$\begin{aligned} \text{Overhead} = & \text{TSKSTART Time} \\ & + \text{TSKWAIT Time} \\ & + \text{Memory contention delay and work load imbalance.} \end{aligned}$$

The theoretical speed-ups computed from the above formulae for the two versions and different numbers of processors proved satisfactory as shown in the following tables:

Number of Processors	2	3	4
Unitasked Fortran	1.82:1.88	2.61:2.70	3.34:3.45
Unitasked CAL opt	1.80:1.86	2.59:2.69	3.33:3.44

Table 1. Theoretical speed-ups (version1:version2). (500 Time steps).

Number of Processors	2	3	4
Unitasked Fortran	1.83:1.90	2.62:2.71	3.35:3.46
Unitasked CAL opt	1.81:1.88	2.60:2.70	3.34:3.45

Table 2. Theoretical speed-ups (version1:version2) (1800 Time steps).

The multitasking can therefore be carried out efficiently. The task we selected for macrotasking is a large granularity task and this property predicts good speed-ups.

Figure 7 illustrates the three distinct parts of the Q-matrix. The uniform part U is the submatrix that contains only non-zero Z vectors and the two nonuniform parts N1 and N2 are the remaining submatrices that contain both zero and non-zero Z vectors. In order to respect the load balancing rules, we scheduled the computation as shown in Figure 8. We have processor1 take care of both nonuniform parts and a fraction of the uniform one. The rest of the work is equally divided among the remaining processors.

The main characteristic of this subroutine is to be able to compute any vector element resulting from the uniform part independently. This subroutine is designed to be reentrant because it is called by all processors at the same time. The three arrays used in this routine are made global and the rest of the variables is local.

The following illustrates the multitasking mechanism used in the matrix vec-

tor multiplication routine.

MAIN PROGRAM

SUBROUTINE MODEL

CALL TSKSTART(UNIF)—P2

CALL TSKSTART(UNIF)—P3

CALL TSKSTART(UNIF)—P4

CALL UNIF—P1

Compute the result elements from the nonuniform parts N1 and N2

CALL TSKSWAIT

Resume unitasked operations

⋮

UNIF is the name of the reentrant subroutine; MODEL is the name of the subroutine that makes the calls to the multitasking library to perform multitasking on the matrix vector multiplication.

4.2 USE OF MICROTASKING

As mentioned in the previous chapter, microtasking is applied to fine granularity tasks, particularly DO loops.

In our program, most of the subroutines contain one or more DO loops, each iteration of which is an independent task. This characteristic makes them compatible for microtasking. The following illustrates a typical unitasked DO loop:

```

      N1=LX * LY
      DO 100 J = 1 , N1
100      X(J)= - V(J) * V(J) * T1

```

a) Unitasked do loop

Its microtasked version as follows:

```

      N1 = LX * LY

CMIC$  MICRO

      SUBROUTINE T

      N2 = N1/4

CMIC$  DO GLOBAL

      DO 100 J = 1 , 4

          K=(J - 1) * N2

          DO 100 I= 1 , N2

100      X(K + I)= - V(K + I) * V(K + I) * T1

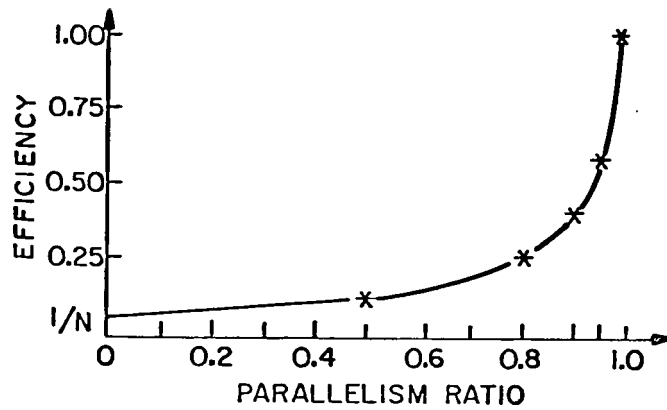
      RETURN

      END

```

b) Microtasked version of above do loop.

According to the previously mentioned flowtrace analysis, 96 percent of the code was used in macrotasking the multiplication routine. Microtasking can be applied to only half of the remaining sequential code which amounts to two percent of the entire code. In this case, the decrease of the amount of sequential code has led to an increase of the parallelism ratio from 96 to 98 percent. The following graph illustrates how a speed-up is improved when the parallelism is slightly increased.



Graph 1. Efficiency vs. parallelism ratio.

If we substitute the new value of the parallelism ratio in the above formulae we will get larger speed-up values than those previously shown in Tables 1 and 2. The following tables display the new obtained values.

Number of Processors	2	3	4
Unitasked Fortran	1.88:1.92	2.72:2.80	3.51:3.63
Unitasked CAL opt	1.84:1.91	2.69:2.78	3.47:3.62

Table 3. Theoretical speed-ups (version1:version2). (500 Time steps).

Number of Processors	2	3	4
Unitasked Fortran	1.89:1.93	2.73:2.81	3.52:3.64
Unitasked CAL opt	1.85:1.92	2.70:2.79	3.48:3.63

Table 4. Theoretical speed-ups (version1:version2) (1800 Time steps).

CHAPTER 5

RESULTS AND DISCUSSION

We discuss two kinds of results. The first one deals with the performance comparison between the unitasked programs and their multitasked versions. The second result relates to the geophysical aspect of the 3D forward modeling program.

5.1 PERFORMANCE RESULTS

An illustrative triangular block model was used to carry out the performance study. The model size used was $128 * 64 * 64$ and the number of time steps considered were kept similar to the ones used in the theoretical performance (500 and 1800 time steps).

The multitasked programs were run in a dedicated environment. Table 1 of Figure 9 shows the running time of the matrix vector multiplication subroutine for the Fortran and CAL unitasked versions versus the two multitasked Fortran versions. Table 2 of Figure 9 shows the speed-ups obtained for the matrix vector multiplication routine. Due to the large granularity of the reentrant subroutine used for the macrotasking part, the values obtained for the speed-up are close to the ideal speed-up, which is equal to the number of processors used. The speed-ups with respect to optimized CAL displayed smaller values because the unitasked

CAL subroutine is faster than its unitasked Fortran counterpart.

Table 1 of Figure 10 shows the experimental speed-up values for the entire program when run with macrotasking. The speed-ups obtained are lower than the ones in the previous table. The main reason in this case is the amount of sequential code with respect to the program. It has been shown in Baer (1980) that the speed-up is very sensitive to the ratio of parallelism. A slight decrease in the parallelism ratio particularly with an increasing number of processors can drastically reduce the speed-up of the multitasked program. For optimized CAL the amount of sequential code is greater than that in Fortran and this led to smaller speed-up values. The results of the previous table are improved in Table 2 of Figure 10 when we added the microtasking feature to the already macrotasked program. The main cause of the improvement is the decrease of the amount of sequential code from 4 percent to approximately 2 percent of the total program. The optimized CAL speed-ups are almost similar to those for Fortran in the last table and this is due to the fact that the amount of sequential code after microtasking is virtually the same.

We noticed that the last speed-ups obtained exceed 80 percent of the ideal speed-up (the number of processors used). The reason for not getting the ideal performance is that the memory contention delay is unavoidable, a perfect workload distribution cannot be attained, and finally some parts of the program are purely sequential and cannot be multitasked.

5.2 FORWARD MODELING RESULTS

5.2.1 CHOICE OF PARAMETERS

The use of high order finite differences may lead to numerical dispersion if the different parameters are not set accordingly. It has been mentioned in the introduction section that 10 points are required to solve a given wavelength for second order methods (see Alford et al., 1974).

The wavelength is obtained by the following formula:

$$\lambda = \frac{V}{F}$$

where V is the minimum model velocity and F is the maximum frequency of the source wave. In the exploding reflector model, the velocity used in the formula is half the minimum velocity of the model.

In the following, we consider the exploding reflector model on the triangular block model already introduced in the previous section to carry out the parameter study. Figure 11 gives the initial snapshot of this model.

We first tested the 4th order method for different number of points and found that for 5 points per wavelength the results obtained were satisfactory. Figures 12 and 13 show the results for 5 points and 4 points per wavelength. The result for 4 points displayed unwanted noise which appeared as early reflections in the modeling result.

The sixth order method displayed good results for a number of points greater or equal to four points per wavelength as shown in Figure 14.

Finally, we tested the eighth order method and the minimum number points needed to solve a given wavelength was at least 3 points. Figure 15 shows the result for the 8th order method with 3 points per wavelength. We conclude here that the higher the order used, the fewer number of points per wavelength are needed for the computation and vice versa. We notice that the results obtained for the different orders tried agree substantially. Consequently, for a particular model under study the number of grid points selected varies with respect to the order of the finite difference used. This introduces a trade off between the decrease of the number of grid points and the increase of the number of columns in the Q-matrix defined in Chapter 2.

To illustrate this trade off we calculated the number of floating point operations performed by the program in every time step; it is given by the following formula:

$$N_{flop} = (LX * LY * LZ) * (2 * DIM * NORD + 1)$$

where DIM is the dimension (three in our case) and the rest of the parameters has already been introduced in Chapter 2.

The following table gives the number of floating point operations with respect to the order of the finite difference method used for our triangular block model case at each time step:

Finite Difference Order	4	6	8
Nflop (Mflop)	13.107	8.013	4.281

We conclude that the CPU time charged to perform 3D forward modeling on a given model is smaller the higher the order. In terms of memory requirements, since the number of grid points per wavelength decreases when the order increases then the overall number of grid points needed decreases. This translates into a saving in memory. The following table illustrates the memory requirements for different orders used on the triangular block model.

Finite Difference Order	4	6	8
Memory (Mwords)	2.7	1.13	0.33

The second parameter we dealt with was the temporal sampling rate. The basis for this study is the stability formula. There are many variations of the formula depending on the dimension used and the order. A formula that is indicative of the practical performance was determined by Loewenthal et al. (1985); it works well for the fourth order program and is given by the following inequality:

$$T \leq \frac{K * DIST}{Vmin}$$

where T is the temporal sampling rate, DIST is the spacing distance between two

grid points (or spacial sampling rate), V_{min} is the minimum velocity, and $K=0.61$ for the fourth order case.

In the case of higher order methods, after several test runs we fixed K to 0.4. The value obtained for the temporal sampling rate in the formula is the maximum value one can use in order to avoid instability.

We know that if the time step is significantly decreased the number of iterations is relatively increased and this leads to the domination of the round-off errors that can distort the results. Figure 16 illustrates the result after 800 time steps with a sampling rate of .0005 ms.

5.2.2 ABSORBING BOUNDARIES

Due to the nature of the 3D wave equation used in our problem that allows exploding waves to travel in both upcoming and downgoing directions we are confronted with unwanted boundary reflections. In order to overcome this problem, it has been shown by Israeli and Orzag (1981) that an addition of a first derivative term to the wave equation can make the wave travel in one direction and dampen it in the opposite. This approach was tested successfully for 2D models by Dablain (see Dablain, 1986). This method is not thoroughly enough developed for 3D models to be applied in the program under discussion here; instead we used the absorbing boundaries method to overcome this reflection noise problem (see Loewenthal et al., 1985).

The Loewenthal method is based on a tapering function which is applied to a certain number of outer grid points of the model. In our case we applied the following tapering function:

$$f(j) = .98^j; \quad j = 1, \dots, N$$

where N is the number of outer grid points where absorption is used. The pressure at the inner most grid point is absorbed by $f(1)$ and the one at the outer point is absorbed by $f(N)$.

We remark that a better result was obtained when we increased the number of significant digits in the function from .98 to .979999. This can be easily explained by the fact that the round-off errors are minimal when the number of significant figures is increased. Figure 17 shows the synthetic time section of the triangular block after we applied the absorbing boundaries for the 13 outer grid points of the cube that holds the triangular block.

5.2.3 POINT SOURCE WAVE MODELING

As opposed to the exploding reflector model, in point source modeling one can have the source anywhere in the model.

We first started by testing the program for a point source in the center of a homogeneous medium. The source wave used is the same as the one previously described. Figure 18 shows the snapshots at $T = 35$ samples for the xy , xz , yz planes. Figure 19 shows the snapshots at $T = 85$ samples and $T = 155$ samples.

We notice reflections when the wave reaches the boundary.

We tested the program for a horizontal layer with a total number of grid points equal to $64 * 32 * 32$. The point source was placed in the center of the first layer whose coordinates are $(32, 17, 8)$. Figure 20 shows the time section recorded at $Z = 3$ after 420 ms of data (350 samples) and Figure 21 shows the snapshot obtained after 100 samples. We then changed the position of the point source to the position $(16, 17, 8)$ for the same model and obtained the time section shown in Figure 22.

5.2.4 PLANE WAVE MODELING

In this case, instead of a point source we have a plane wave source which can be placed anywhere in the model. Figure 23 shows the snapshot in the homogeneous medium at $T = 35$ samples for the xy , xz , yz planes. The plane location is easily determined from the figure. Figures 24 shows the snapshots at $T = 85$ samples and $T = 155$ samples. Figure 25 and 26 respectively show the time section recorded at $Z = 3$ for the horizontal layer after 420 ms of data (350 samples) and the snapshot obtained at 100 samples.

5.2.5 EFFECT OF POINT ELIMINATION ON MODELING RESULTS

The objective of this study is to see if the elimination of one point in every spacial dimension from the computation will still preserve the validity of the forward modeling result.

The study was carried out for the sixth order case in which we have three points on every side of the considered point in every spacial dimension. Figure 27 gives the configuration used in this study and the point numbering adopted for the rest of this section.

The elimination of one point will decrease the size of the Q-matrix by one column and consequently the number of computations involved in the matrix vector multiplication thoroughly discussed in the previous chapter.

First since there is a symmetry around the computed point in every spacial dimension, we found out that whether we eliminate the point from the right or its symmetrical on the left the effect on the result is basically the same. The results discussed in the following are for the points to be eliminated on the right.

We tested this method for the same triangular block considered in the previous section. We first eliminated the inner most point in each spacial dimension namely points 4, 10, 16 in Figure 27; the result obtained is meaningless, the main reason being that the decrease of the number of computations is so significant that valuable data is lost. The elimination of the second inner point in each spacial dimension (points 5, 11, 17) also affects the results and the reason is the same as for the inner most point although the number of computations saved in this case is less than the previous one.

We tried the elimination of one inner most point, one middle point, and one outer point (eg. points 4, 11, 18; points 6, 12, 17; etc...). We also tried the

elimination of two middle points and one outer point (eg. points 5, 11, 18; points 6, 11, 17; and points 5, 12, 17). Finally, we tried the elimination of one middle point and two outer points (eg. points 5, 12, 18; points 6, 12, 17; and points 6, 11, 18). None of the above combinations gave acceptable results.

The only meaningful result was obtained when we eliminated one outer point in every dimension from the computation (points 6, 12, 18). The reason is that the reduction of the amount of computations is less significant than for the previous cases. The number of computations saved in this case for each time step is given below:

$$N_{saved} = 2 * ((LZ - NORD2) * LY * LX + (LY - NORD2) * LX * LZ \\ + (LX - NORD2) * LY * LZ),$$

where NORD2 is half the order of the finite difference used.

As an illustration for the triangular block with $128 * 64 * 64$ grid points and $NORD2 = 2$ the number of computations saved is equal to 3.062 Mflop. This saving is slightly less than 1/4 of the total amount of computations for each time step. Figure 28 and 29 respectively show the results obtained after the elimination of the outer point in every dimension for the triangular block model (exploding reflector) and the horizontal two-layer model (point source).

5.2.6 SALFRH RESULTS

After the program was first tested for the simple triangular block model using

128 * 64 * 64 points, we ran the program for the SALFRH model which is a fairly complicated model that has two domes and one fault. It is given in Figures 30 and 31. The grid size selected was 256 * 256 * 128 grid points. The model has a grid spacing respectively of 5 meters and 6.66 meters in each of the spatial dimensions for the fourth order and sixth order method. Three different velocities and thicknesses are chosen:

Medium of incidence	Scaled Velocity (m/s)	Scaled Thickness (m)
water	3545 / 2	829.06
stycast 3180	6689 / 2	518.16
plexigas	6583 / 2	152.40

Figure 32 is a snapshot of the exploding reflector forward modeling at time $T = 1$ and for the plane defined by $X = 90$ plane which passes through the center of dome 1 and the fault. Figures 33 and 34 are the respective time sections generated by the fourth order program for 600 time steps of .00088 ms each and the sixth order program for 600 time steps of 0.00125 ms each .

Figure 35 is a snapshot of the exploding reflector forward modeling at time $T = 1$ and for the plane defined by $X = 128$ plane which passes through the center of dome 2. Figures 36 and 37 are the respective time sections generated by the fourth order program for 600 time steps of .00088 ms each and the sixth order program for 600 time steps of 0.00125 ms each .

Figure 38 is a snapshot of the exploding reflector forward modeling at time T

$= 1$ and for the plane defined by $Y = 106$ plane which passes through the center of dome 2 and the fault. Figures 39 and 40 are the respective time sections generated by the fourth order program for 600 time steps of .00088 ms each and the sixth order program for 600 time steps of 0.00125 ms each .

Figure 41 is a snapshot of the exploding reflector forward modeling at time $T = 1$ and for the plane defined by $Y = 168$ plane which passes through the center of dome 2. Figures 42 and 43 are the respective time sections generated by the fourth order program for 600 time steps of .00088 ms each and the sixth order program for 600 time steps of 0.00125 ms each .

Figure 44 is the time section generated along $Z = 3$ and $X = 128$ by the exploding reflector sixth order program for 600 time steps of 0.00125 ms each using absorbing boundaries. This time section is to be compared with the one in Figure 37.

We collected time sections for a point source located at a position with coordinates (128, 128, 20). This source is exactly where the two diagonals of the plane $Z = 20$ intersect. Figure 45 is the time section recorded along the receivers located along the line $Z = 5$ and $X = 90$.

We also collected time sections for a plane wave source located between planes $Z = 20$ and $Z = 26$, planes $Y = 110$ and 146, and planes $X = 125$ and $X = 131$. Figure 46 shows the time section recorded along the receivers located along the line $Z = 5$ and $X = 90$.

Finally, a time section was also collected when the outer point was eliminated from the computation of the sixth order exploding reflector program. Figure 47 shows the time section recorded along the receivers located along the line $Z = 5$ and $X = 128$ in this case and is to be compared with Figure 35.

CHAPTER 6

USER GUIDE TO THE PROGRAMS

The multitasked program developed to perform 3D forward modeling can be run with either of the two versions of the subroutine which carries out the matrix vector multiplication. The name of the program that uses the subroutine version 1 is "MTFORW1". The other program is called "MTFORW2".

6.1 PARAMETERS

The program's parameters are referenced in the `PARAMETER` statements and the `DATA` statements. The following describes the most important ones:

- PLANE: The size of a $Z = \text{constant}$ plane.
- ORD: The order of the finite difference method.
- COF: Array. The coefficients of the high order finite difference operators. Its size is equal to $\text{ORD}/2$.
- B: The remaining coefficient of the high order finite difference operators.
- LX: Number of grid points in X direction.

LY:	Number of grid points in Y direction.
LZ:	Number of grid points in Z direction or depth.
NT:	Total number of time steps used.
PTPOSX:	Position of the point to be eliminated in the X direction from the computation.
PTPOSY:	Position of the point to be eliminated in the Y direction from the computation.
PTPOSZ:	Position of the point to be eliminated in the Z direction from the computation.
DEX:	Distance between two grid points along X direction.
DEY:	Distance between two grid points along Y direction.
DEZ:	Distance between two grid points along Z direction.
T:	Time interval between two time steps.
F:	Frequency of the exploding wave function used.
NS:	Indicates at which time step to collect a particular cross section.

NPS:	Indicates whether point source is used
NPL:	Indicates whether plane wave source is used
NEX:	Indicates whether exploding reflector is used
NSX:	Indicates the X position of the point source.
NSY:	Indicates the Y position of the point source.
NSZ:	Indicates the Z position of the point source.
NSX1:	Indicates the first X position of the plane source.
NSY1:	Indicates the first Y position of the plane source.
NSZ1:	Indicates the first Z position of the plane source.
NSX2:	Indicates the second X position of the plane source.
NSY2:	Indicates the second Y position of the plane source.
NSZ2:	Indicates the second Z position of the plane source.
ABSO:	Indicates the number of grid points used for boundary absorption

NVEL:	The logical number of the first of four files which store the velocity vector
NDIA:	The logical number of the first of four files which store the diagonal of the coefficient matrix
NSOU:	The logical number of the first of four files which store the source vector
NRES:	The logical number of the first of four files which store the residual vector ($R[k-1]$).
NBAS:	The logical number of the first of eight files which store the pressure vector at time step equal to jt and at time step equal to $jt-1$.

6.2 INPUT TO THE PROGRAMS

The input data of the program is the velocity at every grid point. In order for the program to be applied to different models, a subroutine GENVEL is used to generate the velocity cube. This allows the user to rewrite the GENVEL subroutine for his particular model. We present the following example that generates the SALFRH model.

C
 C The subroutine GENVEL is used to calculate the velocity
 C of every grid point. The user can change this subroutine to
 C generate a different model.

C
 C Input parameter:
 C L: depth of Z direction.

C
 C Output parameter:
 C V: velocity at each grid point on L plane

```

SUBROUTINE VEL(V,L)
  DIMENSION V(1)
  COMMON /LEVEL/LX,LY,LZ
  COMMON /LEVEL1/N1,NORD2
  IF(L.LE.70) THEN
    DO 100 J=1,N1
100      V(J)=1772.5
    ENDIF
  IF ((L.GE.44).AND.(L.LE.56)) THEN
    L1=L-44
    L2=(L1*20)/13
    DO 200 K=50,148+L2
    DO 200 I=58+K-L2,206
      J=(K-1)*LX+I
200      V(J)=3344.5
    DO 300 K=70,150
```

```

DO 300 I=50,128
IF (((L-77.83)**2+(K-107.0)**2+(I-90.0)**2)
$      .LE.32.83**2) THEN
      J=(K-1)*LX+I
      V(J)=3344.5
ENDIF
300    CONTINUE
DO 400 K=130,206
DO 400 I=90,165
IF (((L-77.83)**2+(K-169.0)**2+(I-128.0)**2)
$      .LE.32.83**2) THEN
      J=(K-1)*LX+I
      V(J)=3344.5
ENDIF
400    CONTINUE
ENDIF
IF ((L.GE.57).AND.(L.LE.69)) THEN
L1=((L-56)*4)/13
DO 500 K=50,206
DO 500 I=50+L1,206
      J=(K-1)*LX+I
500    V(J)=3344.5
ENDIF
IF (L.GE.70) THEN
DO 600 J=1,N1
600    V(J)=3291.5
ENDIF

```

```

RETURN
END

```

6.3 OUTPUT FROM THE PROGRAMS

The programs output the time sections collected along one or several particular lines on a $Z = \text{constant}$ plane. A subroutine named TIMESC takes care of the output of these time sections. The user is easily able to rewrite this subroutine to obtain the time sections he wishes to explore. The following shows the TIMESC subroutine:

```

C
C   The Subroutine TIMESEC saves the time section at Z = NZP
C   plane. This subroutine can be rewritten by the user
C   according to a different model.
C
C
SUBROUTINE TIMESEC(LTIME1,LTIME2,LTIME3,LTIME4,V,JT)
  DIMENSION V(1)
  COMMON /LEVEL/LX,LY,LZ
  IF (JT .GT. 250) THEN
    WRITE(LTIME1,500) (V(J),J=128,(LY-1)*LX+128,LX)
  ENDIF
  WRITE(LTIME2,500) (V(J),J=90,(LY-1)*LX+90,LX)
  WRITE(LTIME3,500) (V(106*LX+J),J=1,LX)
  WRITE(LTIME4,500) (V(168*LX+J),J=1,LX)
  WRITE(LTIME5,500) (V(I*LX+I-1),I=1,LY)

```



```

500          FORMAT(8E9.2)
          RETURN
          END

```

This particular example collects the data of the time sections along the line $Z = NZP$ and $X = 128$, along the line $Z = NZP$ and $X = 90$, along the line $Z = NZP$ and $Y = 106$, along the line $Z = NZP$ and $Y = 168$, and finally along the diagonal for the SALFRH model.

The second output generated by the programs is any cross section of the model under study at any time step. A subroutine named CROSSC handles the output of these cross sections. The user can easily modify it to generate the desired cross sections. The subroutine CROSSC is shown in the following:

```

C
C   The subroutine CROSSC writes the cross section at time
C   step = NTS into the files. It can be rewritten by the
C   user to get a different cross section.
C
C
      SUBROUTINE CROSSC(X1,NTS,NUMBER)
      COMMON /LEVEL/LX,LY,LZ
      COMMON /FILENUM/NVEL,NDIA,NSOU,NRES,NBAS
      LUNIT= NBAS + MOD(NTS-1,2) * 4
      DO 700 L=1,LZ
      NREC=NUMBER*(L-1)+1
      CALL AREAD4(LUNIT,X1,NREC,NUMBER)

```

```

CALL WAITE4(LUNIT)
WRITE(66,500)(X1(I*LX+I-1),I=1,LY)
WRITE(67,500)(X1(106*LX+J),J=1,LX)
WRITE(68,500)(X1(168*LX+J),J=1,LX)
WRITE(69,500)(X1(J),J=90,(LY-1)*LX+90,LX)
WRITE(70,500)(X1(J),J=128,(LY-1)*LX+128,LX)
500    FORMAT(8E9.2)
700    CONTINUE
ENDIF
RETURN
END

```

The above example generates the cross section diagonal plane, the cross section planes $Y = 106$, $Y = 168$, $X = 90$, and $X = 128$.

In order to use the program, the most critical parameter in the Job Control Statement is the time option. It is the amount of time that sets the priority of the job. Jobs that run longer than 2000 seconds have the lowest priority and those which execute in less than 5 seconds have the highest priority. Since the system is usually loaded during the day, the lowest priority programs usually run at night. A model with $128 * 64 * 64$ grid points tested using a fourth order program for 400 time steps has an execution time of approximately 11 seconds. A model with $256 * 256 * 88$ grid points tested for 600 time steps has an execution time of approximately 320 seconds. It is advisable to always test small models in order to be familiar with the programming environment and the program in particular.

If one wants to run the job in dedicated mode one should add the option `US = BNCHBCH` right after the time option in the Job Control Statement. Also one should make sure that the program runs in less than 5 to 10 minutes in dedicated mode. The Cray system at Mendota Heights runs the dedicated programs between 11 p.m and midnight.

The model parameters are set in the `PARAMETER` and `DATA` statements. First, the model size is to be assigned to the parameter `PLANE` and `LX`, `LY`, `LZ`; second, the order of the finite difference is to be assigned to the parameter `NORD`; finally, the coefficients C'_i 's and `B` are to be stored in array `COF` and `B` respectively. These coefficients have to be determined by the user after solving the linear system of equations previously discussed in Chapter 2. For instance, a sixth order provides the following:

$$C_1 = 1.5, C_2 = -0.15, C_3 = 0.0111, \text{ and } B = -2.7222.$$

The spacial sampling rate along each dimension `DETX`, `DETY`, and `DETZ` have to be set according to the number of points per wavelength of the source used. The temporal sampling rate `T` has to be set to satisfy the stability formula discussed in the previous chapter. The rest of the parameters is set according to the kind of modeling results the user wishes to study.

If the user wants to run the program for a point source, the parameter `NPS` is to be set to 1 otherwise it is kept 0. The parameters `NSX`, `NSY`, `NSZ` have to be set to the values that determine the coordinates of the source in the model.

For a plane wave source, the parameter NPL is to be set to 1 otherwise it is kept 0. The parameters NSX1, NSY1, NSZ1, NSX2, NSY2, NSZ2 have to be set to determine the location of the plane wave in the model.

The same is for the parameter NEX in the exploding reflector model. In this case the sources are the points where there is a velocity change and it is automatically handled by the subroutine MODEL.

If the program is to be tested for the point elimination case, the parameters PTPOSX, PTPOSY, PTPOSZ which are normally set to 0 are to be assigned a positive value. For instance a value of 1 assigned to the three parameters will eliminate the inner most or the closest points in every dimension to the computed point. These parameters cannot exceed a value equal to half the order of the finite difference used.

The output files from the CRAY system contain formatted data. In order to use them for plotting purposes the file has to be unformatted. The following program reads the formatted file and outputs its equivalent unformatted file:

```

PROGRAM HASPLOT
PARAMETER (NXMAX=256, NZMAX=2048)
REAL A(NZMAX,NXMAX)
CHARACTER*80, NAME, REC
TYPE*, 'INPUT FILE NAME:'
ACCEPT 200, NAME
TYPE 200, NAME

```

```

OPEN(UNIT=1, NAME=NAME, IOSTAT = IERR, TYPE = 'OLD',
$      READONLY, FORM='FORMATTED')
TYPE*, 'OUTPUT FILE NAME'
ACCEPT 200, NAME
OPEN(UNIT=2, NAME=NAME, IOSTAT = IERR, TYPE = 'NEW',
$      ERR= 20, FORM='UNFORMATTED')
TYPE*, 'NUMBER OF RECORDS TO SKIP:'
ACCEPT*, NSKIP
DO I=1,NSKIP
READ(1,200) REC
WRITE(6,200) REC
END DO
TYPE*, 'NO OF TRACES'
ACCEPT*, NX
TYPE*, 'NO OF SAMPLES PER TRACE:'
ACCEPT*, NZ
DO IZ=1,NZ
READ(5,100) (A(IZ,IX), IX=1,NX)
END DO
DO II=1,NX
TYPE*, ' TRACE = ',II
WRITE(2) (A(IZ,II), IZ=1,NZ)
ENDDO
STOP
10 STOP 'INPUT ERROR'
20 STOP 'OUTPUT ERROR'
100 FORMAT(8E9.2)

```

```
200  FORMAT(A)
      END
```

After the data file has been made unformatted it is ready to be plotted using the following program:

```
*JOB  OMAR
*CALL GENRATE SHOT SEQNO  2000 1      256
KEYDEF 1 1 0 1 1 METRIC
SINE 10.0 0.0 4000.0 0 0
*CALL READIT F.DAT
*CALL SECTION 64.0      3.0
SCALE PEAK
LABEL  SEQNO  10      1000
TIMING 2      1      0
*END
```

Further information about the use of the above plot program is easily found in the DISCO manual.

CHAPTER 7

CONCLUSION

A 3D forward modeling program has been developed and run on a complicated structure. The amount of computations involved is significant. The high order finite difference method was used as the algorithmic basis for the model.

In this program, approximately 96 percent of the CPU time is devoted to computing the matrix vector multiplication. The macrotasking part of our multitasked program concentrated on this large granularity routine to make it a reentrant subroutine that can be called by all processors in the system at the same time. The program took advantage of the microtasking feature to run fine granularity tasks such as DO loops in parallel. The final speed-up obtained from the multitasked program is in excess of 80 percent of the ideal speed-up. An even better performance is expected if the CAL optimized multiplication routine is multitasked.

The finite difference techniques have a major problem which is related to numerical dispersion. One has to be careful about choosing the number of grid points per wavelength. We found that the number of grid points per wavelength decreases when the order of the finite difference increases and this leads to a significant saving in memory requirements and CPU time. The temporal sampling rate selected for a particular computation has to satisfy a stability formula which

depends on the order used.

We applied the method of absorbing boundaries to smooth out the forward modeling result. We tested the effect of eliminating one point in every spacial dimension from the computation and found that the elimination of the outer points is the only one that provided acceptable results. This program can be tested for point sources and receivers located anywhere in the model as well as for plane wave sources and the exploding reflector model.

REFERENCES

- Alford, R.M., Kelly, K.R., and Boore, D.M., 1974, Accuracy of finite difference modeling of acoustic waves equations: *Geophysics*, 39, 834.
- Baer, J., 1980, *Computer systems architecture*: Computer Science Press, Maryland.
- Booth, M., and Misegades, K., 1986, Microtasking: a new way to harness multiprocessors: *Cray Channels*, Summer 1986, 24.
- Claerbout, J.F., 1970, Coarse grid calculations of waves in inhomogeneous media: *Geophysics*, 35, 407.
- CRAY Research Inc., 1984, *Fortran reference manual*.
- CRAY Research Inc., 1986, *Multitasking users guide*.
- CRAY Research Inc., 1986, *Optimization guide*.
- Dablain, M.A., 1986, The application of high order differencing to the scalar wave equation: *Geophysics*, 1, 54
- Davis, H.T., 1963, *Tables of the mathematical functions*: The Principal Press of Trinity University, Ireland.
- Israeli, M., and Orzag, S.A., 1981, Approximation of radiation boundary conditions: *Journal of Computational Physics*, 41, 115.

- Juang, S.A., 1985, 3D forward modeling by four way I/O concurrent management and absorbing boundaries: Research Computation Laboratory Annual Progress Review, 1, 279.
- Loewenthal, D., Wang, G.J., and Johnson, O., 1985, High order finite difference modeling and reverse time migration: Research Computation Laboratory Annual Progress Review, 1, 189.
- Shen, L., and Johnson, O., 1986, Three well known 3D high order finite difference techniques: Research Computation Laboratory Annual Progress Review, 2, 347.

LIST OF FIGURES AND TABLES

Figure 1. CRAY X-MP system organization.

Figure 2. Workload distribution cases.

Figure 3. Transitions of user tasks.

Figure 4. Program general algorithm.

Figure 5. 3D datacube storage.

Figure 6. Flowtrace analysis.

Figure 7. Q matrix parts.

Figure 8. Processor scheduling.

Figure 9. Running times and speed-ups.

Figure 10. Measured speed-ups.

Figure 11. Snapshot of the triangular block at Time = 1 and $Y = 32$.

Figure 12. Synthetic time section of the triangular block using fourth order method with 5 points per wavelength.

Figure 13. Synthetic time section of the triangular block using fourth order method with 4 points per wavelength.

Figure 14. Synthetic time section of the triangular block using sixth order method with 4 points per wavelength.

Figure 15. Synthetic time section of the triangular block using eighth order method with 3 points per wavelength.

Figure 16. Effect of roundoff errors due to a small time step on the synthetic time section of the triangular block.

Figure 17. Synthetic time section of the triangular block using absorbing boundaries.

Figure 18. Snapshot of point source at 35 samples in planes XY, YZ, XZ.

Figure 19. Snapshot of point source at 85 and 155 samples in plane yz.

Figure 20. Synthetic time section recorded using a point source wave located at (32, 17, 8) in the horizontal two-layer model for 400 ms of data (350 samples).

Figure 21. Snapshot of the horizontal two-layer at 100 samples for the point source case. (Relate to Figure 20)

Figure 22. Synthetic time section recorded using a point source wave located at $(16, 17, 8)$ in the horizontal two-layer model for 400 ms of data (350 samples).

Figure 23. Snapshot of plane wave located between planes ($Z = 5$ and $Z = 8$, $Y = 13$ and $Y = 19$, and $X = 32$ and $X = 35$) at 35 samples in planes XY, YZ, XZ.

Figure 24. Snapshot of plane wave at 85 and 155 samples in plane YZ.

Figure 25. Synthetic time section recorded using a plane wave located between planes ($Z = 5$ and $Z = 8$, $Y = 13$ and $Y = 19$, and $X = 32$ and $X = 35$) in the horizontal two-layer model for 420 ms of data (350 samples).

Figure 26. Snapshot of the horizontal two-layer at 100 samples for the plane wave case. (Relate to Figure 25)

Figure 27. Point configuration and numbering used in the point elimination case

Figure 28. Synthetic time section of the triangular block model for the point elimination case. (Compare with Figure 14)

Figure 29. Synthetic time section of the horizontal two-layer model for the point elimination case. (Compare with Figure 20)

Figure 30. FRH physical model. All sizes scaled.

Figure 31. Cross-section of SALFRH physical model.

Figure 32. Snapshot of SALFRH at Time = 1 and X = 90.

Figure 33. Synthetic time section using fourth order method of the exploding reflector model. (Compare with Figure 32)

Figure 34. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 32)

Figure 35. Snapshot of SALFRH at Time = 1 and X = 128.

Figure 36. Synthetic time section using fourth order method of the exploding reflector model. (Compare with Figure 35)

Figure 37. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 35)

Figure 38. Snapshot of SALFRH at Time = 1 and Y = 106.

Figure 39. Synthetic time section using fourth order method of the ex-

ploding reflector model. (Compare with Figure 38)

Figure 40. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 38)

Figure 41. Snapshot of SALFRH at Time = 1 and Y = 168.

Figure 42. Synthetic time section using fourth order method of the exploding reflector model. (Compare with Figure 41)

Figure 43. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 41)

Figure 44. Synthetic time section generated along $X = 128$ using the exploding reflector sixth order program for the absorbing boundaries case. (Compare with Figure 35)

Figure 45. Synthetic time section recorded at $Z = 5$ for the point source wave case. Point source location: $(128, 128, 20)$.

Figure 46. Synthetic time section recorded at $Z = 5$ for the plane wave case. Plane wave located between the planes: $X = 125$ and $X = 131$; $Y = 122$ and $Y = 134$; and $Z = 17$ and $Z = 23$.

Figure 47. Synthetic time section generated along $X = 128$ using the exploding reflector sixth order program for the point elimination case. (Compare with Figure 34)

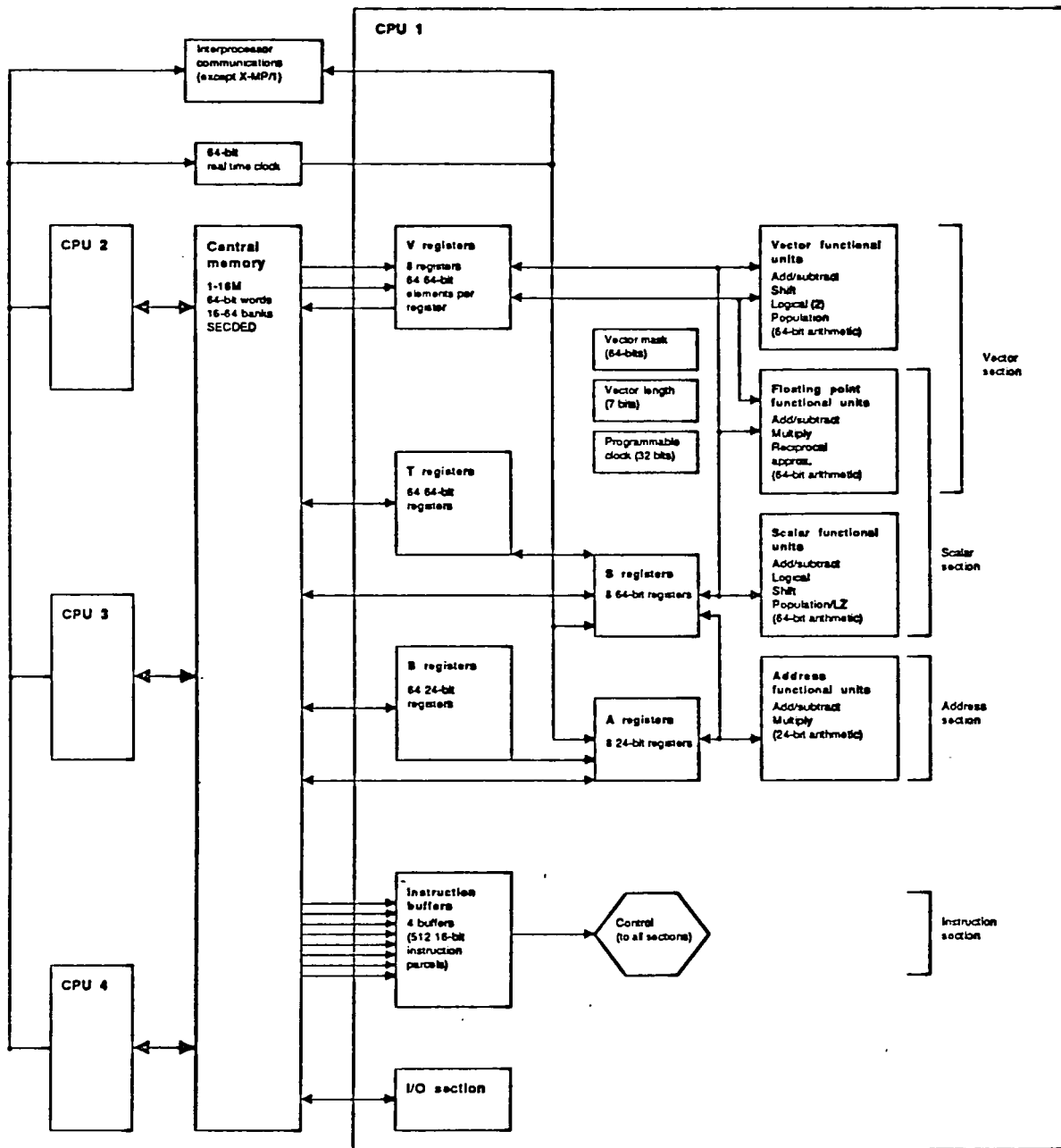


Figure 1. CRAY X-MP system organization.

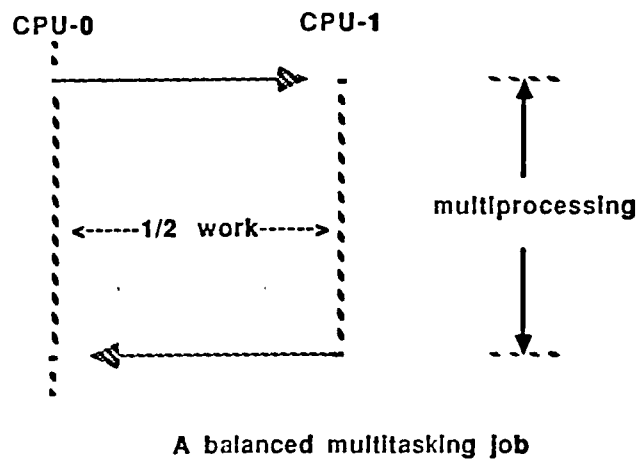
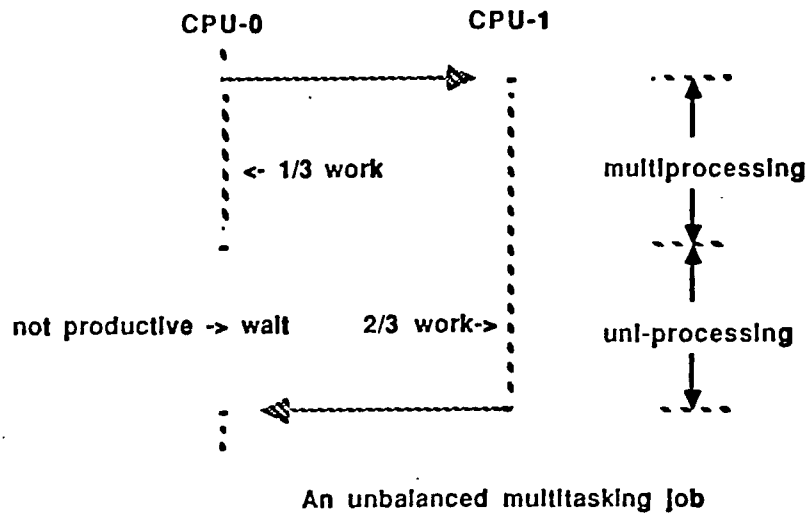


Figure 2. Workload distribution cases.

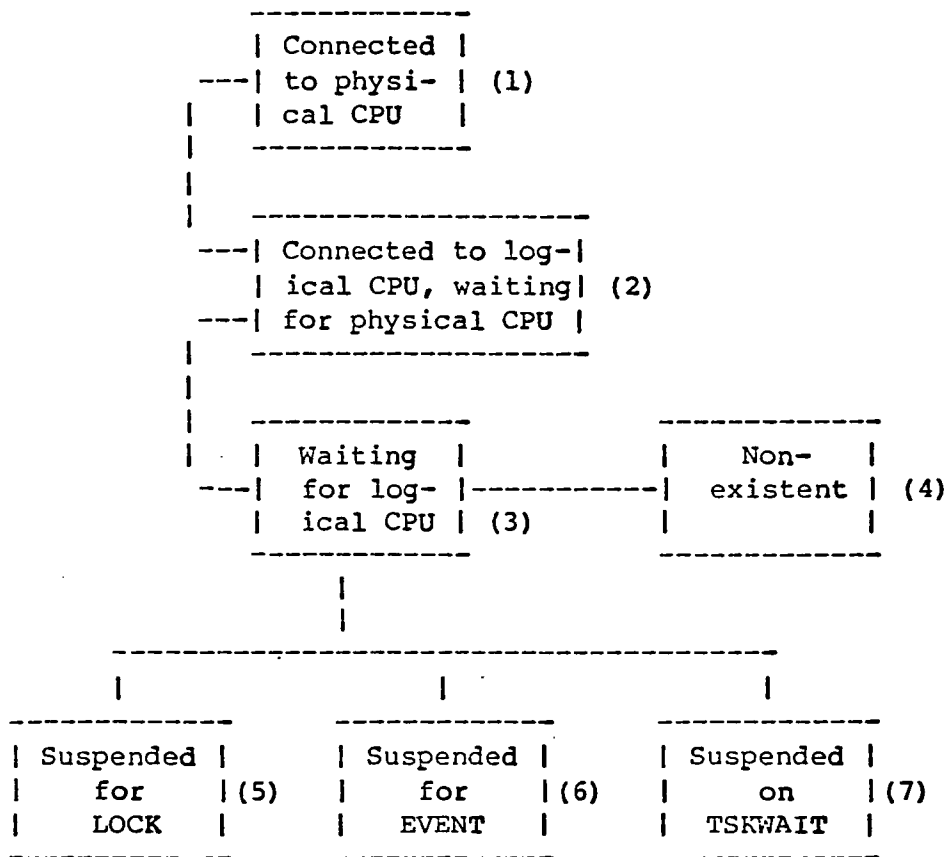


Figure 3. Transitions of user tasks.

PART1: INPUT GENERATION AND STORAGE

```

For each plane LZ
begin
  Generate the velocity vector : V
  Generate the source vector:F
  Compute the inverse diagonal vector D:  $-V*V^T$ 
  Compute the main diagonal of Q matrix A2:
    Constant-2/D
  Store the 3 vectors F,D,and A2 in SSD.
end

```

PART2: MODELING

```

For each time step NTS
begin
  For each plane LZ
    begin
      Fetch vectors F and D from SSD
      If NTS=1 Then  $P1 = D * F$ 
      else Compute matrix-vector
        multiplication:  $M=Q*Pnts-1$ 
      If NTS=2 Then  $P2 = D * (F - M)$ 
      If NTS>2 Then  $Pnts= D*(F - M)-Pnts-2$ 
    end
  Write time section at LZ=0
  Optional:write cross section at a given time step.
end

```

Figure 4. Program general algorithm

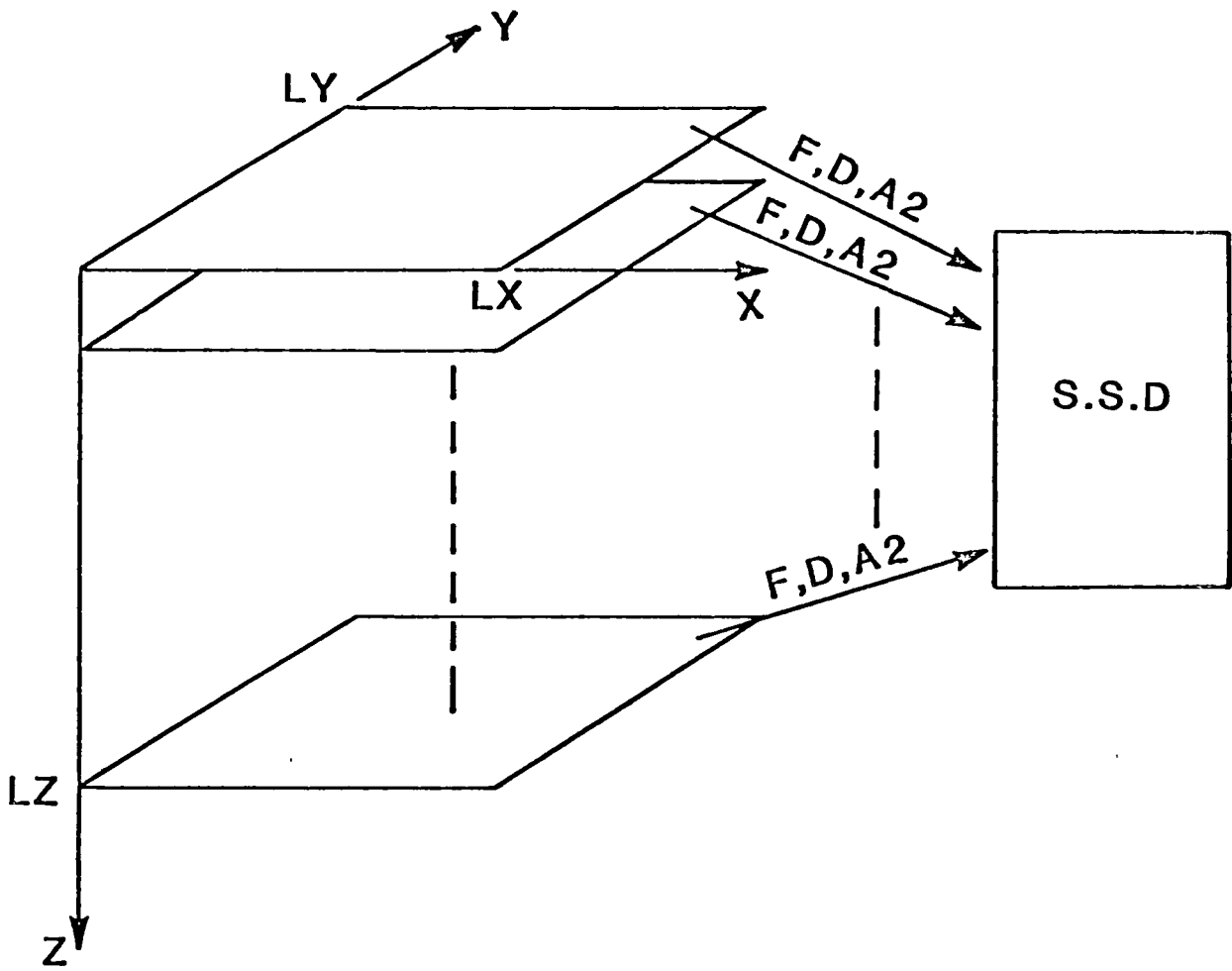


Figure 5. 3D datacube storage.

F L O W T R A C E -- ALPHABETIZED SUMMARY

	ROUTINE	TIME EXECUTING	CALLED
14	AREAD4	C.752 (0.18%)	170800
	00C226242A		
2	AVF	C.017 (0.00%)	1
3	AWRITE4	C.159 (0.04%)	32064
	00C226406A		
8	DIAG	C.011 (0.00%)	32
15	DREAD	4.767 (1.12%)	683200
6	DWAIT	3.962 (0.93%)	811456
4	DWRITE	0.910 (0.21%)	128256
1	MAIN	C.073 (0.02%)	1
11	MODEL	4.333 (1.02%)	500
16	SMMO	409.032 (95.98%)	499
21	ST	1.336 (0.31%)	500
7	VEL	C.006 (0.00%)	33
5	WAITE4	C.810 (0.19%)	202864
	000226644A		
10	WAVE	C.003 (0.00%)	500
* * * TOTAL		426.169	2030706 TOTAL CALLS

F L O W T R A C E -- CALLING TREE

1	MAIN	CCCCC226A	
2	AVF	C0222416A	
3	AWRITE4	C0226406A	
4	DWRITE	00224430A	
5	WAITE4	C0226644A	
6	DWAIT	00224510A	
7	VEL	C0227110A	
8	DIAG	C0224266A	
9	WAITE4	00226644A(TREE AT	5)
10	WAVE	C0222306A	
11	MODEL	C0221044A	
12	AWRITE4	C0226406A(TREE AT	3)
13	WAITE4	C0226644A(TREE AT	5)
14	AREAD4	00226242A	
15	DREAD	00224350A	
16	SMMO	00227334A	
17	AWRITE4	00226406A(TREE AT	3)
18	WAITE4	00226644A(TREE AT	5)
19	AREAD4	00226242A(TREE AT	14)
20	AREAD4	00226242A(TREE AT	14)
21	ST	C0226724A	

Figure 6. Flowtrace analysis.

0	0	...	0	M_1	$Z_{1,1}$	$Z_{1,n}$	Nonuniform Part N1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	
0	$Z_{n,n-1}$...	$Z_{n,1}$	M_n	$Z_{n,1}$	$Z_{n,n}$	
$Z_{n+1,n}$	$Z_{n+1,1}$	M_{n+1}	$Z_{n+1,1}$	$Z_{n+1,n}$	Uniform Part U
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	
$Z_{l_2-n,n}$	$Z_{l_2-n,1}$	M_{l_2-n}	$Z_{l_2-n,1}$	$Z_{l_2-n,n}$	
$Z_{l_2-n+1,n}$	$Z_{l_2-n+1,1}$	M_{l_2-n+1}	$Z_{l_2-n+1,1}$...	$Z_{l_2-n+1,n-1}$	0	Nonuniform Part N1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	
$Z_{l_2,n}$	$Z_{l_2,n}$	M_{l_2}	0	...	0	0	

Figure 7. Q matrix parts.

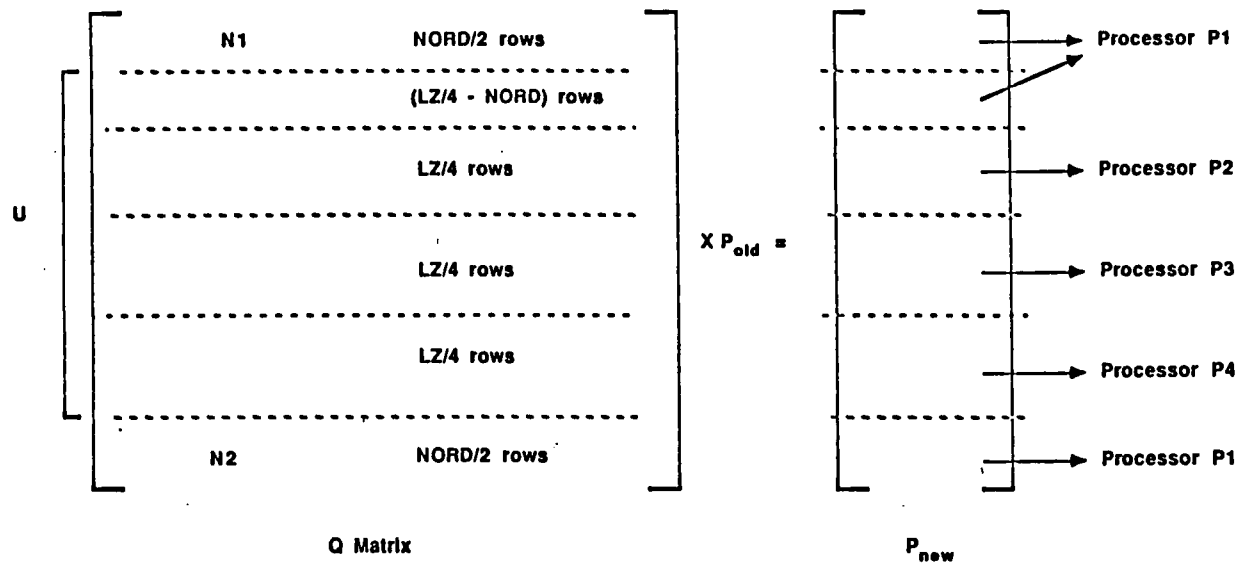


Figure 8. Processor scheduling.

Table 1: Matrix Vector Multiplication Running Times (s).

Number of Processors	1	2	3	4
Unitasked Fortran	.300517	_____	_____	_____
Unitasked Cal opt.	.226225	_____	_____	_____
Multit. version 1	_____	.153668	.101583	.07578
Multit. version 2	_____	.152546	.100844	.075317

Table 2: Speed-ups Obtained for Matrix Multiplication.

Number of Processors	2	3	4
Unitasked Fortran	1.84:1.85	2.88:2.91	3.93:3.96
Unitasked Cal opt.	1.38:1.39	2.17:2.19	2.95:2.98
version1: version2			

Figure 9. Running times and speed-ups.

Table 1. Program Speed-ups Obtained from Macrotasking.

Number of Processors	2	3	4
Unitasked Fortran	1.83:1.84	2.61:2.62	3.30:3.32
Unitasked Cal opt.	1.80:1.81	2.52:2.53	3.14:3.16
version1: version2			

Table 2. Program Speed-ups Obtained from Macro and Microtasking.

Number of Processors	2	3	4
Unitasked Fortran	1.87:1.89	2.70:2.72	3.48:3.50
Unitasked Cal opt.	1.84:1.86	2.65:2.67	3.38:3.41
version1: version2			

Figure 10. Measured speed-ups.

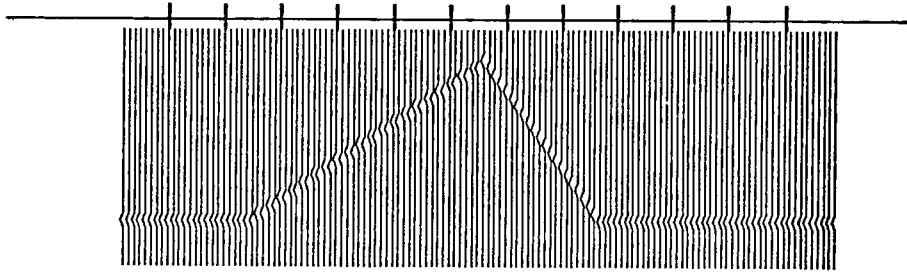


Figure 11. Snapshot of the triangular block at Time = 1 and Y = 32.

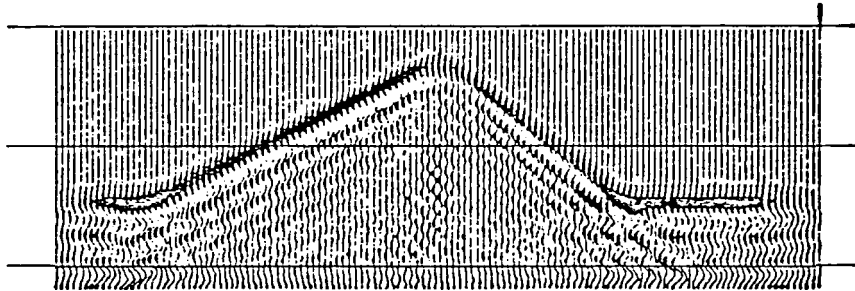


Figure 12. Synthetic time section of the triangular block using fourth order method with 5 points per wavelength.

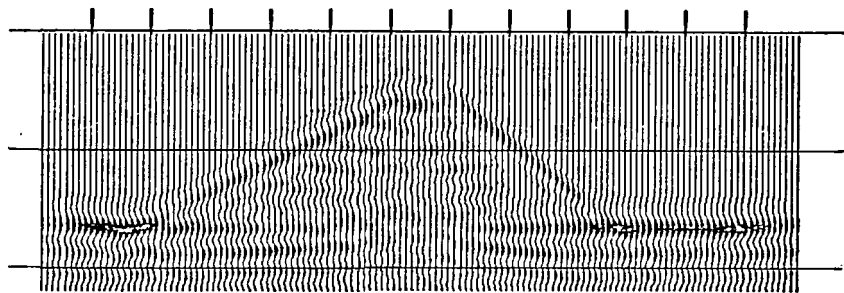


Figure 13. Synthetic time section of the triangular block using fourth order method with 4 points per wavelength.

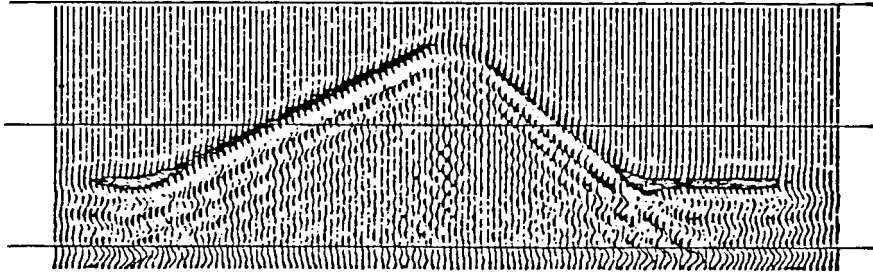


Figure 14. Synthetic time section of the triangular block using sixth order method with 4 points per wavelength.

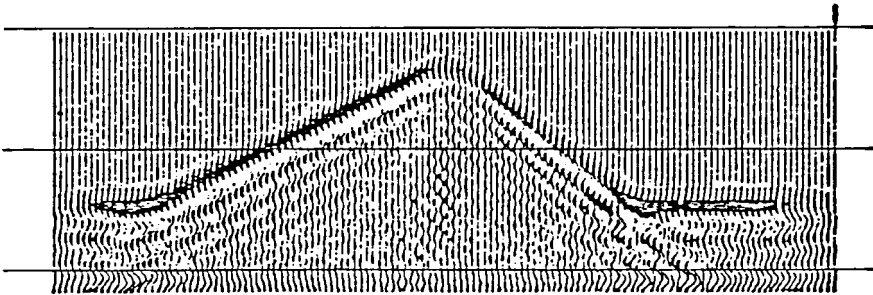


Figure 15. Synthetic time section of the triangular block using eighth order method with 3 points per wavelength.

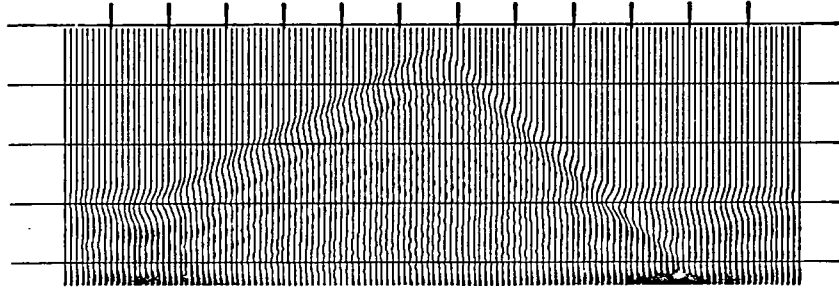


Figure 16. Effect of roundoff errors due to a small time step on the synthetic time section of the triangular block.

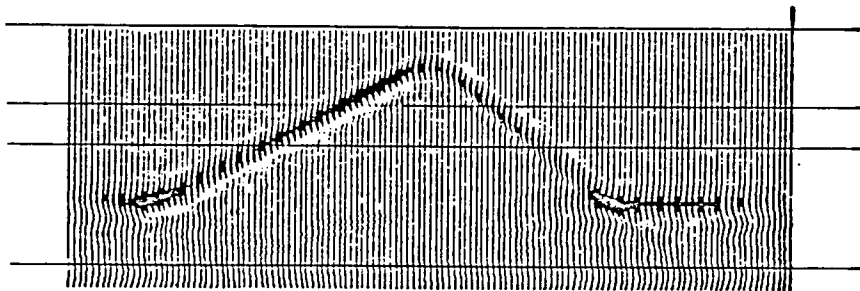


Figure 17. Synthetic time section of the triangular block using absorbing boundaries.

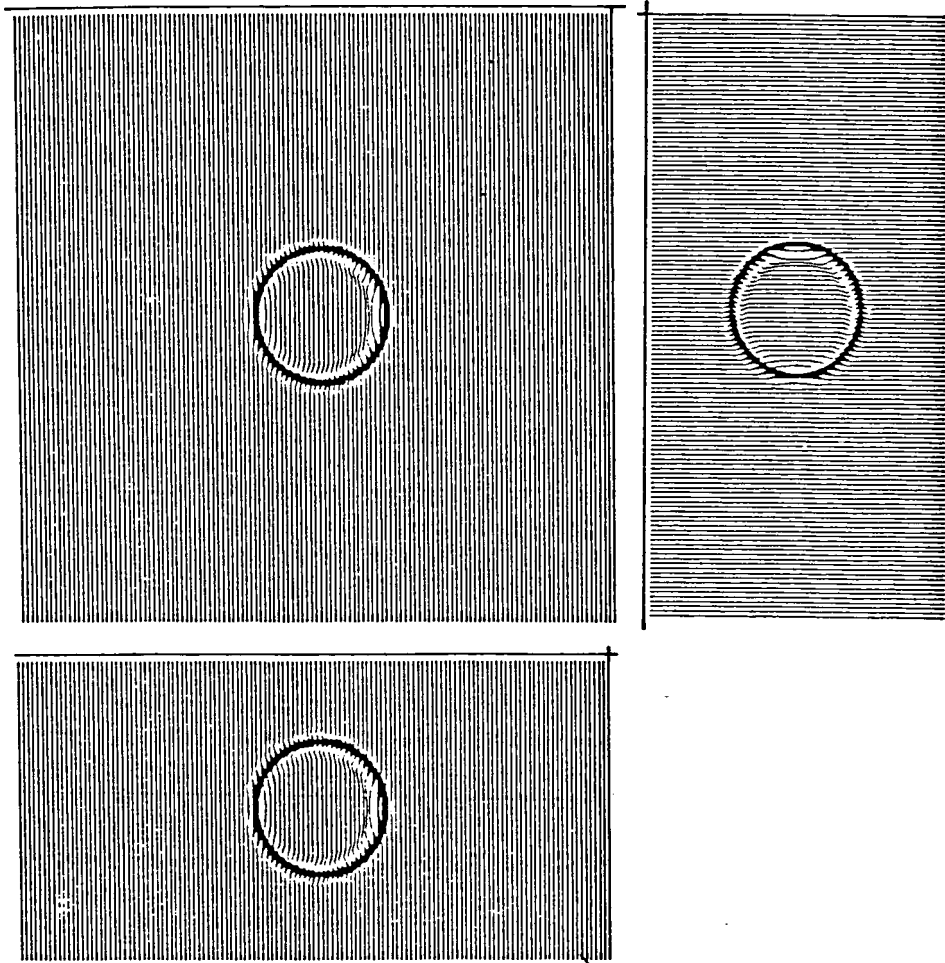
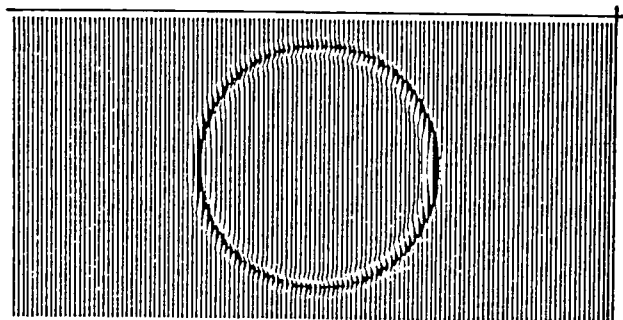
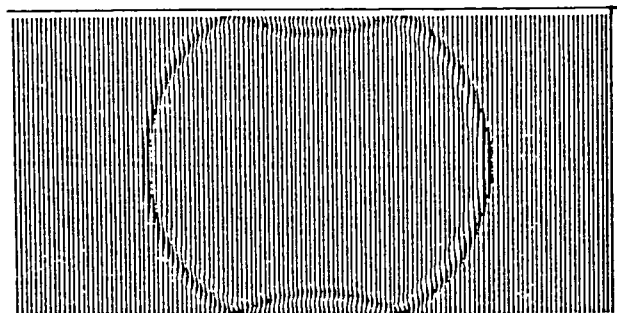


Figure 18. Snapshot of point source at 35 samples in planes XY, YZ, XZ.



a) 85 samples



b) 155 samples

Figure 19. Snapshot of point source at 85 and 155 samples in plane yz .

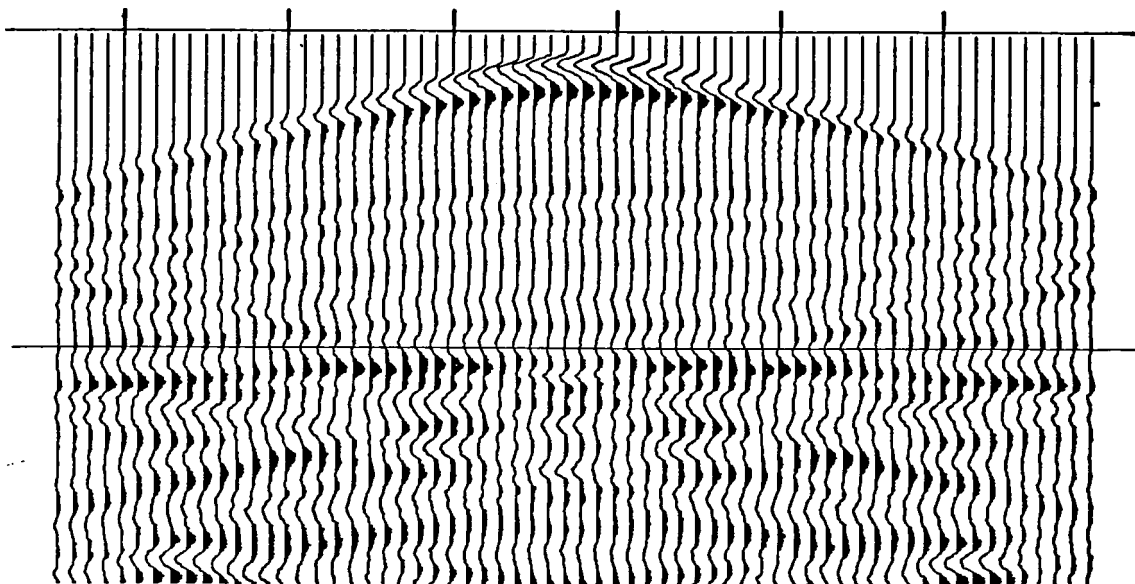


Figure 20. Synthetic time section recorded using a point source wave located at (32, 17, 8) in the horizontal two-layer model for 400 ms of data (350 samples).

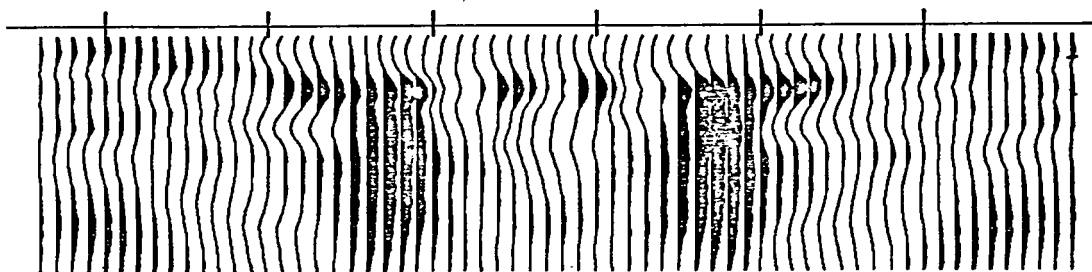


Figure 21. Snapshot of the horizontal two-layer at 100 samples for the point source case. (Relate to Figure 20)

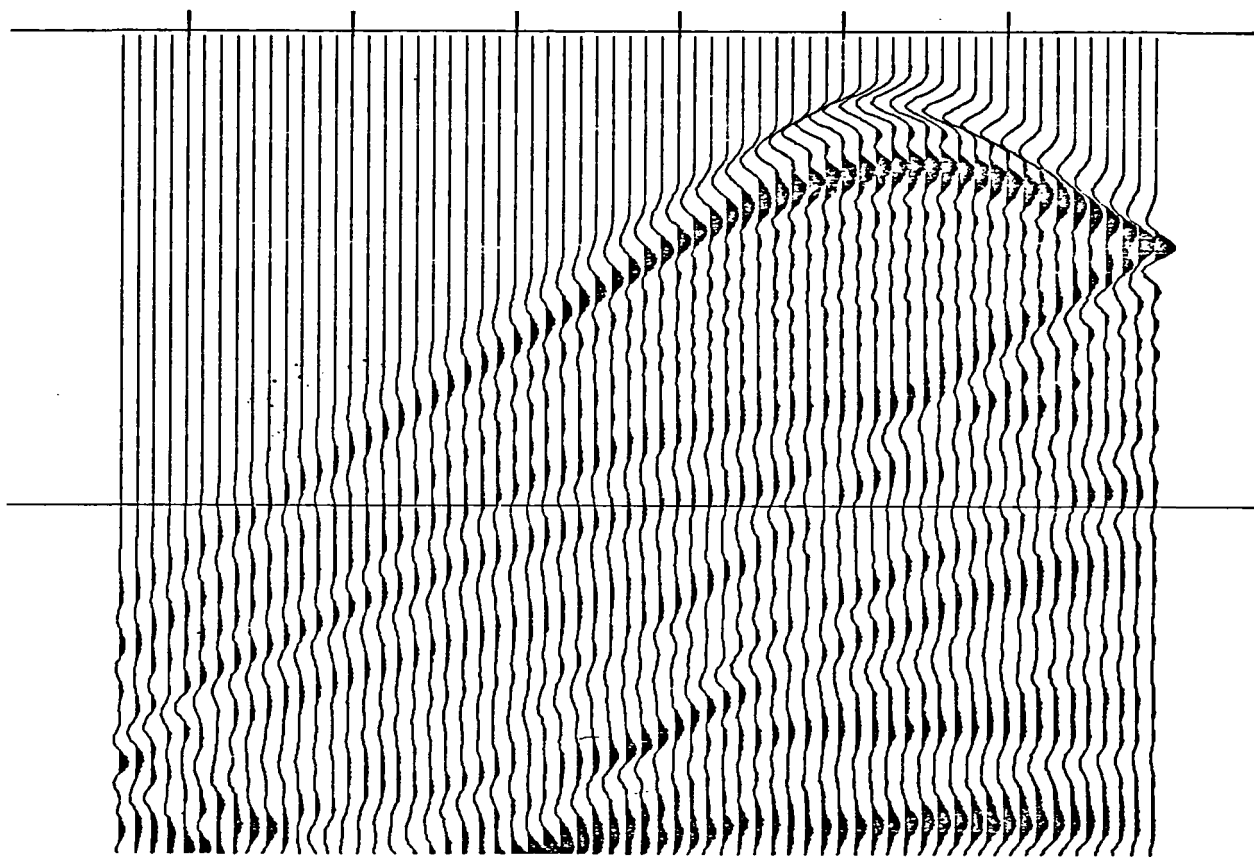


Figure 22. Synthetic time section recorded using a point source wave located at (16, 17, 8) in the horizontal two-layer model for 400 ms of data (350 samples).

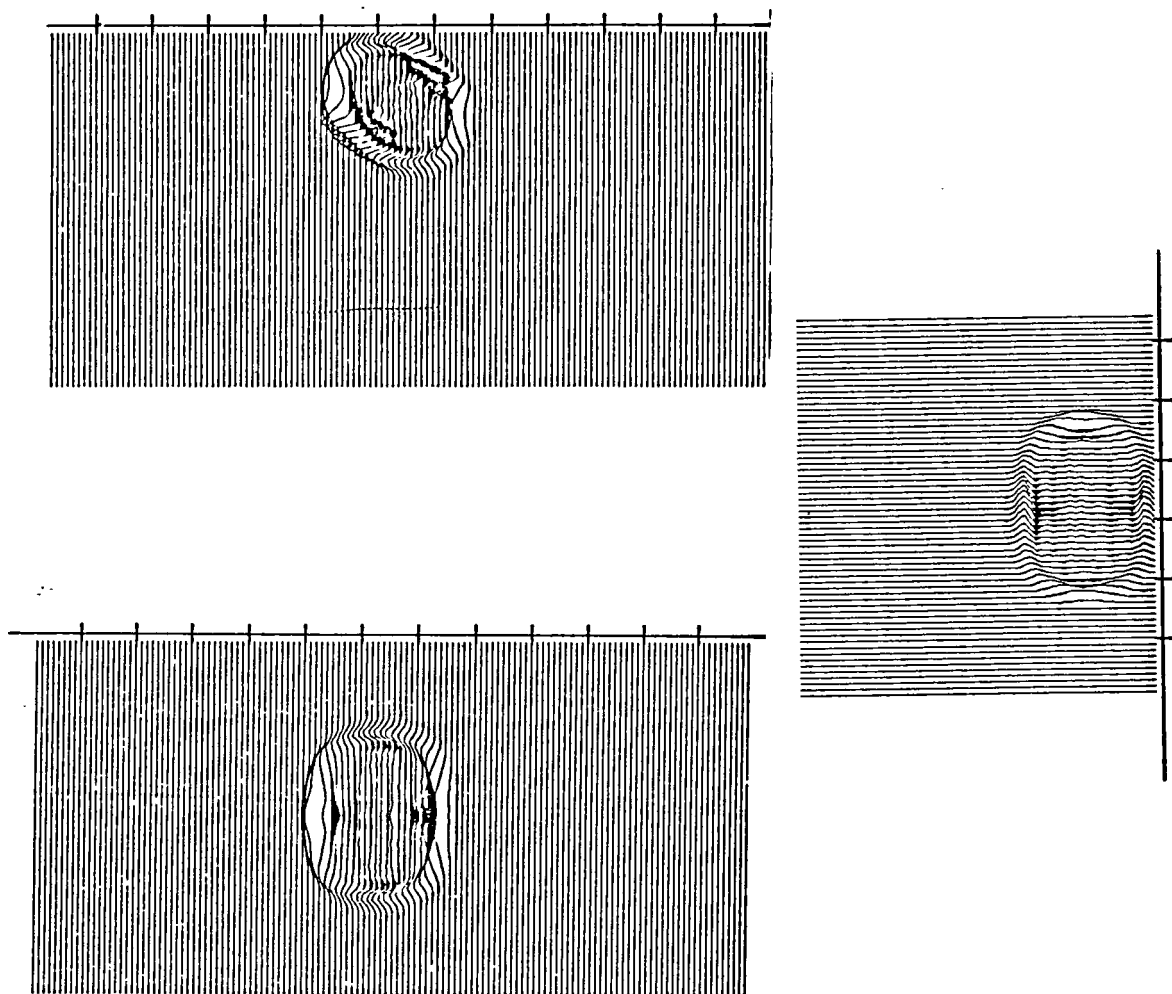
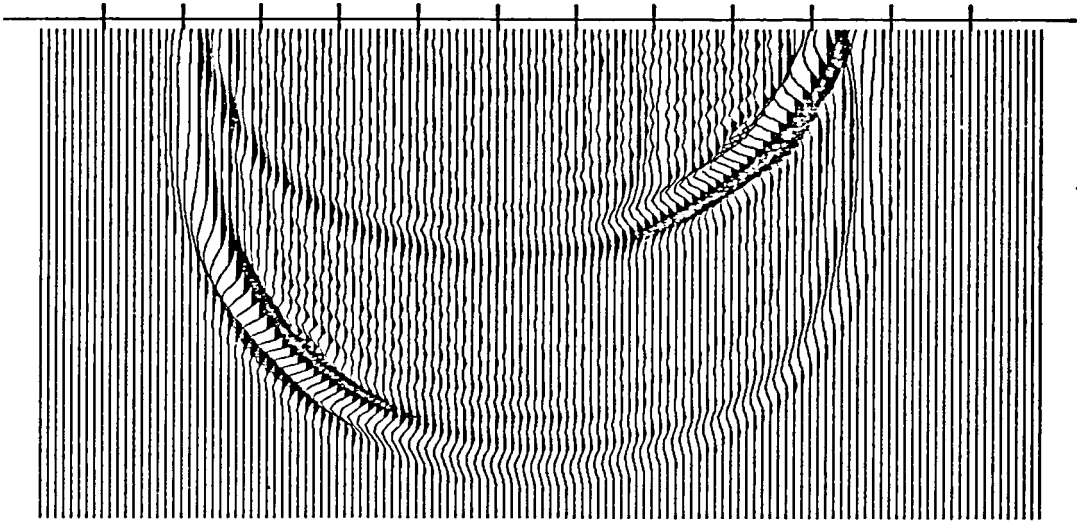
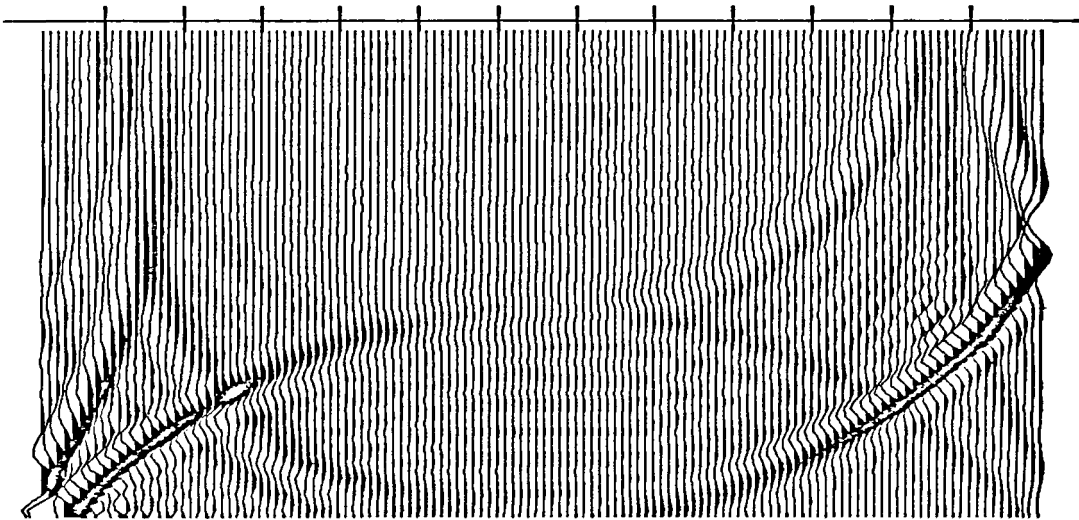


Figure 23. Snapshot of plane wave located between planes
 $(Z = 5 \text{ and } Z = 8, Y = 13 \text{ and } Y = 19, \text{ and } X = 32 \text{ and } X = 35)$
 at 35 samples in planes XY, YZ, XZ.



a) 85 samples



b) 155 samples

Figure 24. Snapshot of plane wave at 85 and 155 samples in plane YZ.

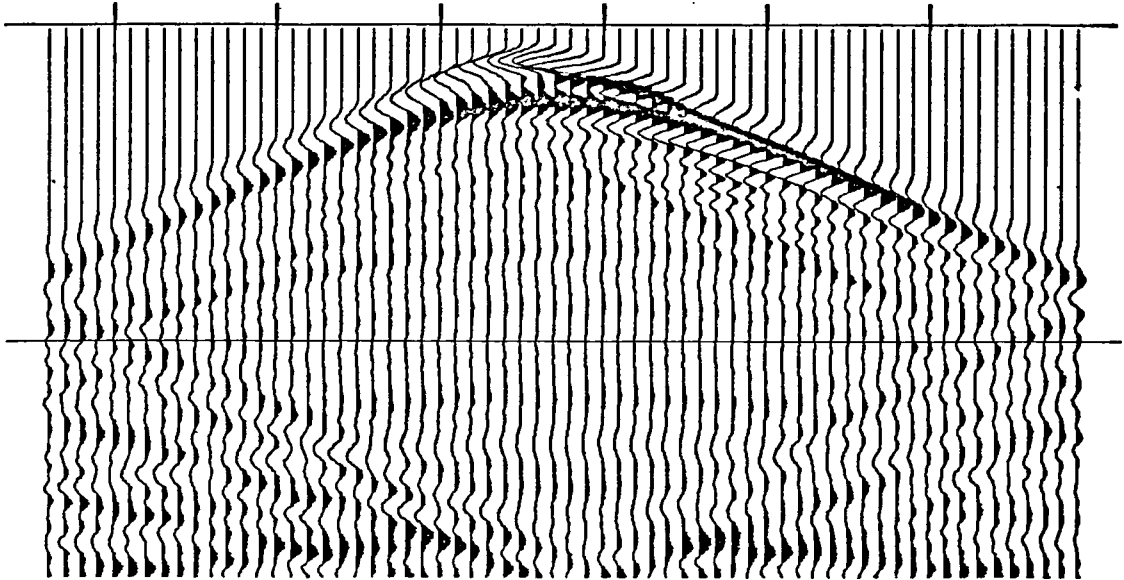


Figure 25. Synthetic time section recorded using a plane wave located between planes ($Z = 5$ and $Z = 8$, $Y = 13$ and $Y = 19$, and $X = 32$ and $X = 35$) in the horizontal two-layer model for 420 ms of data (350 samples).

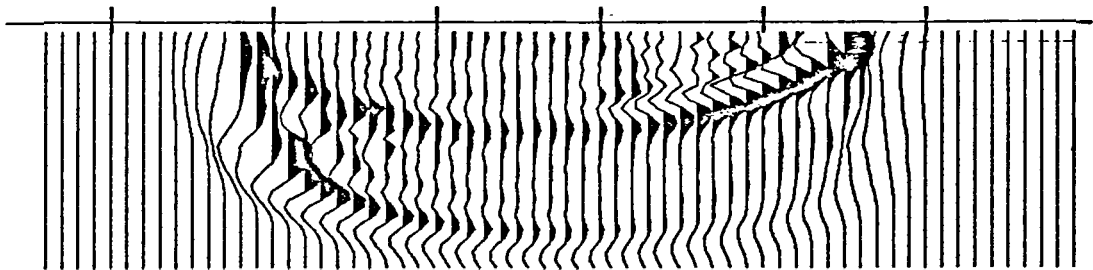


Figure 26. Snapshot of the horizontal two-layer at 100 samples for the plane wave case. (Relate to Figure 25)

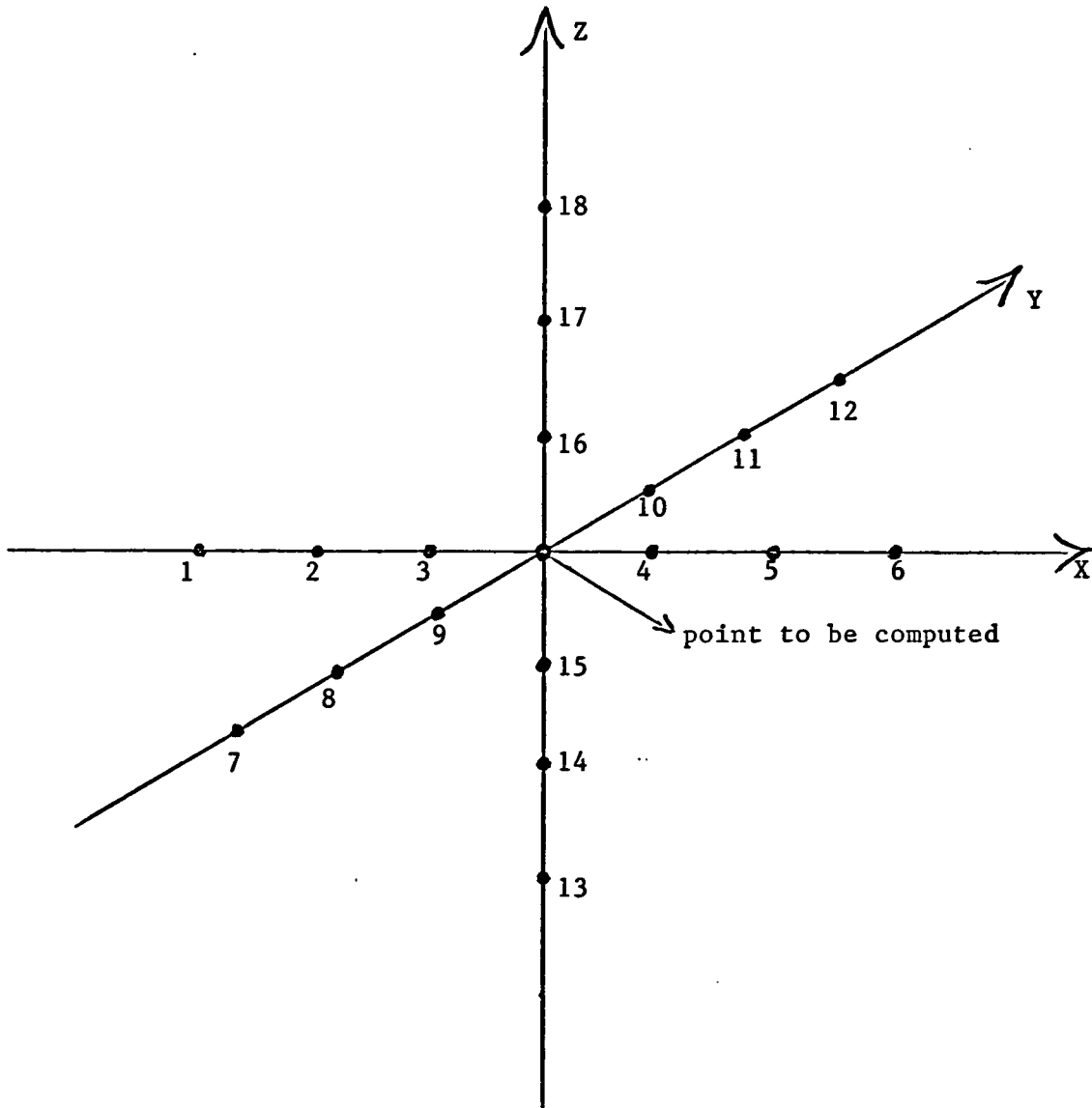


Figure 27. Point configuration and numbering used in the point elimination case

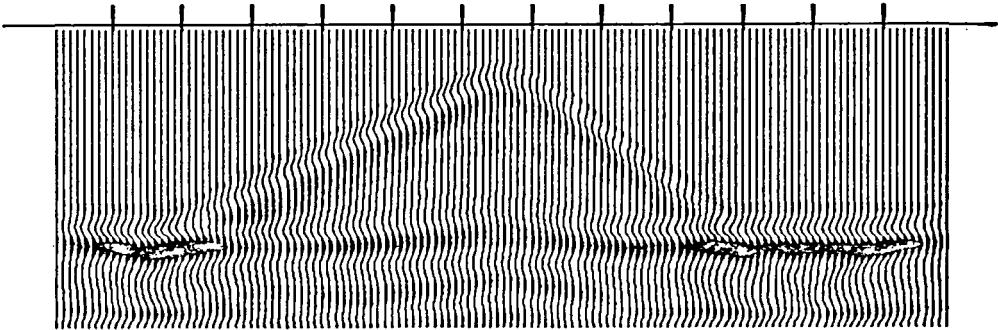


Figure 28. Synthetic time section of the triangular block model for the point elimination case. (Compare with Figure 14)

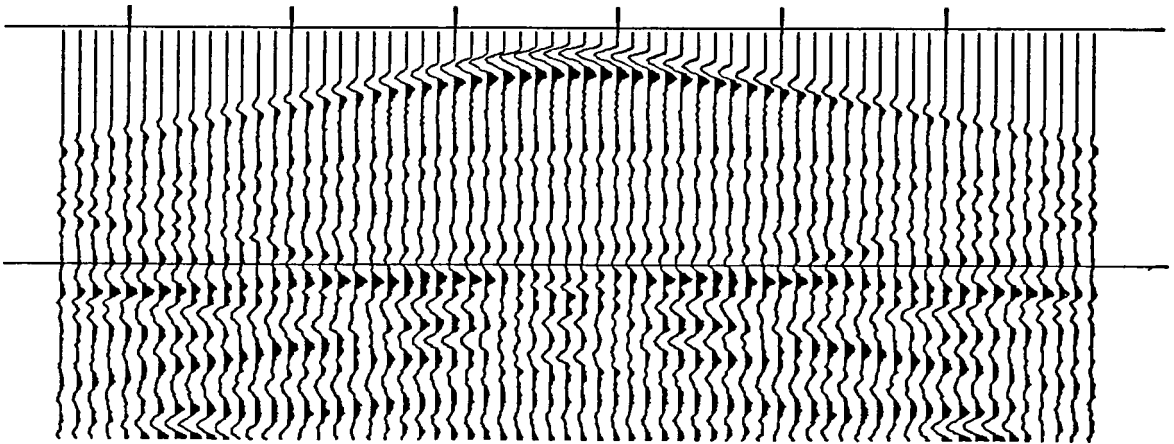


Figure 29. Synthetic time section of the horizontal two-layer model for the point elimination case. (Compare with Figure 20)

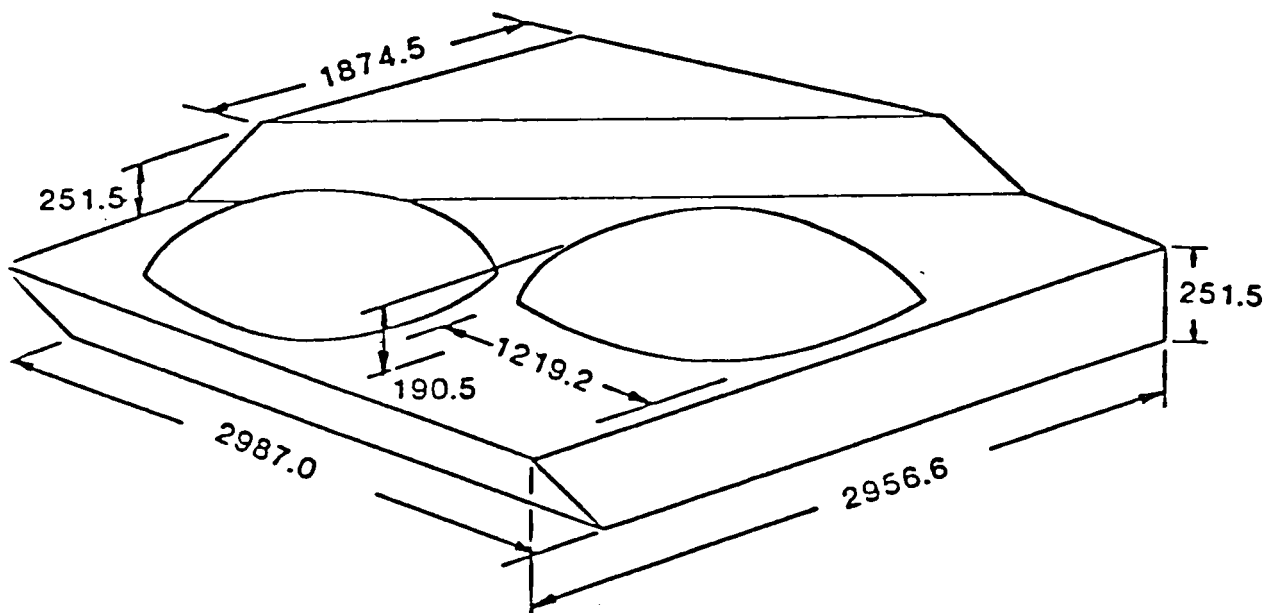


Figure 30. FRH physical model. All sizes scaled.

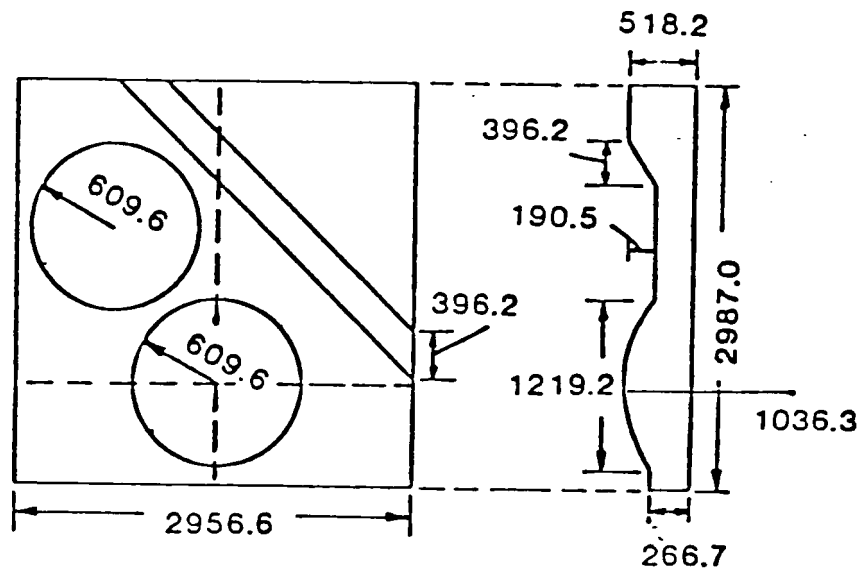


Figure 31. Cross-section of FRH physical model.

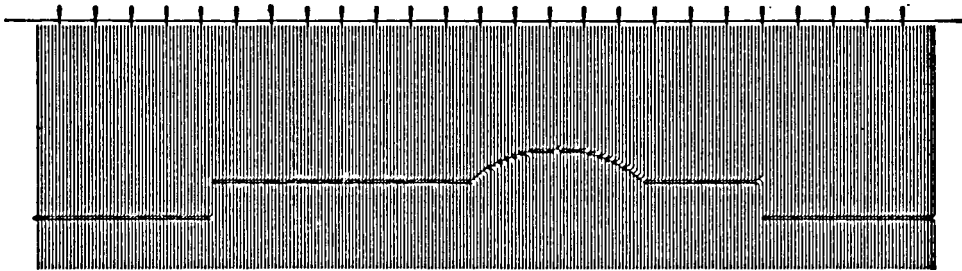


Figure 32. Snapshot of SALFRH at Time = 1 and $X = 90$.

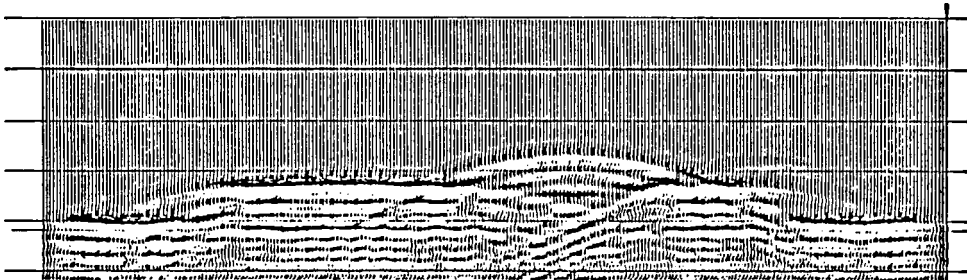


Figure 33. Synthetic time section using fourth order method of the exploding reflector model. (Compare with Figure 32)

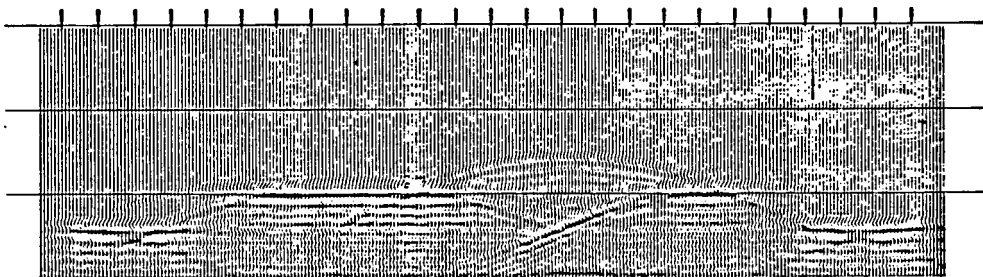


Figure 34. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 32)

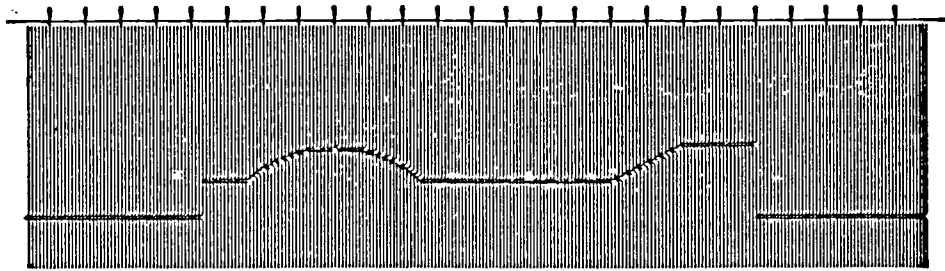


Figure 35. Snapshot of SALFRH at Time = 1 and X = 128.

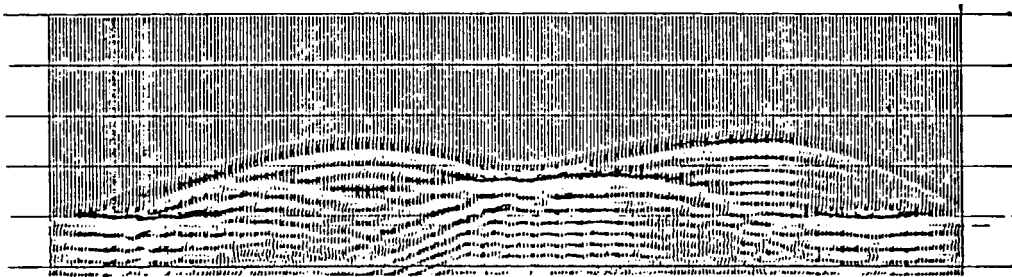


Figure 36. Synthetic time section using fourth order method of the exploding reflector model. (Compare with Figure 35)

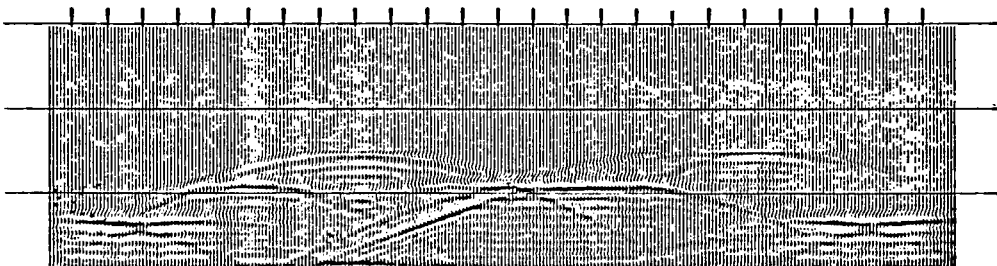


Figure 37. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 35)

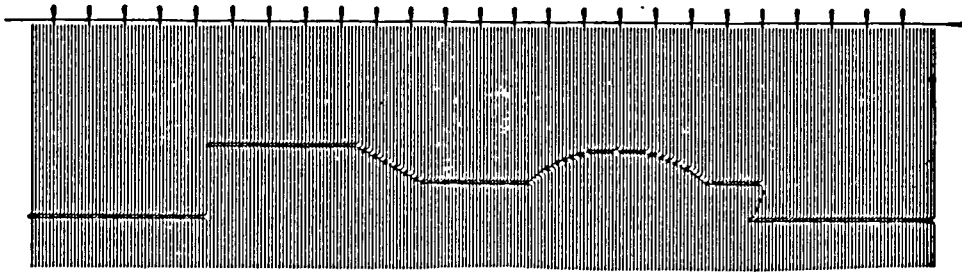


Figure 38. Snapshot of SALFRH at Time = 1 and Y = 106.

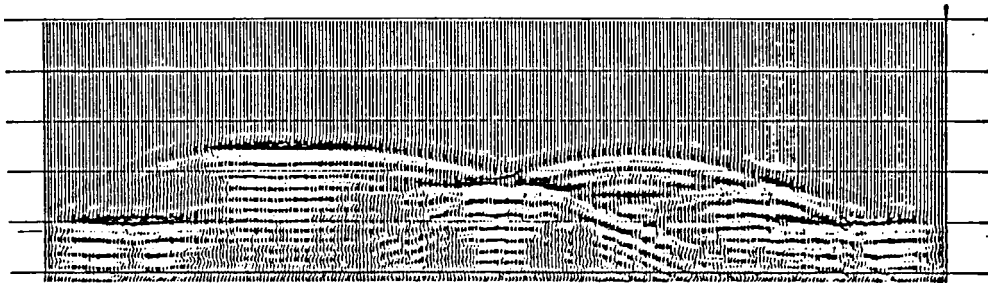


Figure 39. Synthetic time section using fourth order method of the exploding reflector model. (Compare with Figure 38)

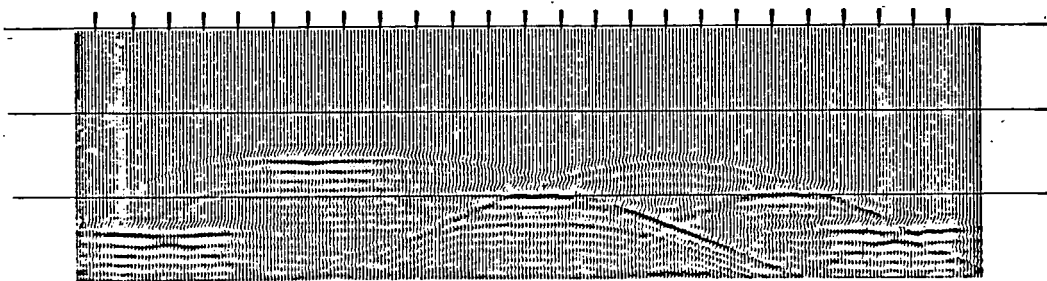


Figure 40. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 38)

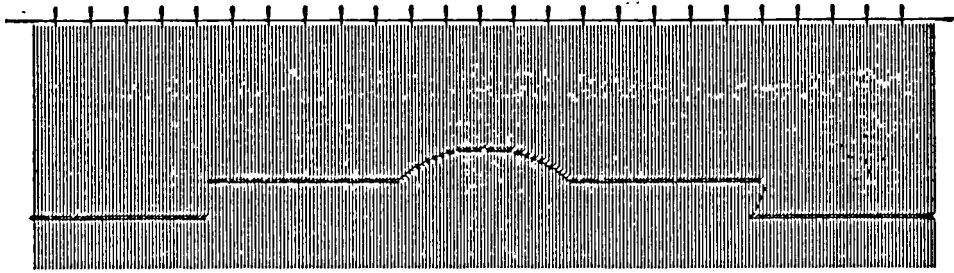


Figure 41. Snapshot of SALFRH at Time = 1 and Y = 168.

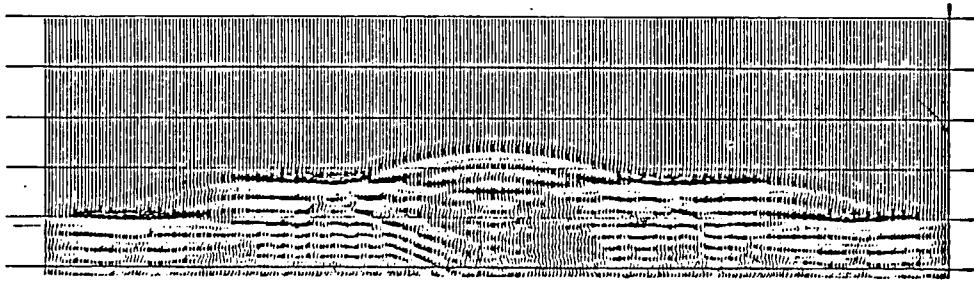


Figure 42. Synthetic time section using fourth order method of the exploding reflector model. (Compare with Figure 41)

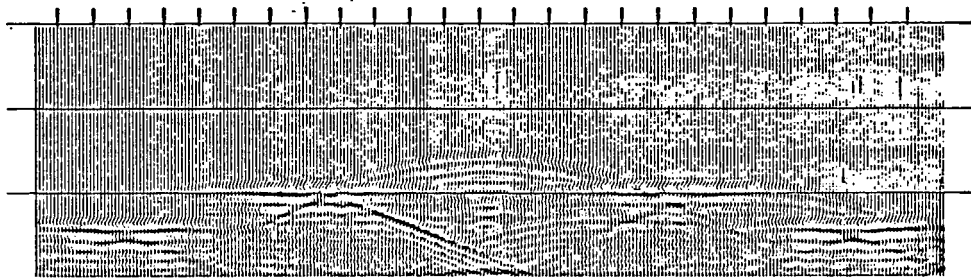


Figure 43. Synthetic time section using sixth order method of the exploding reflector model. (Compare with Figure 41)

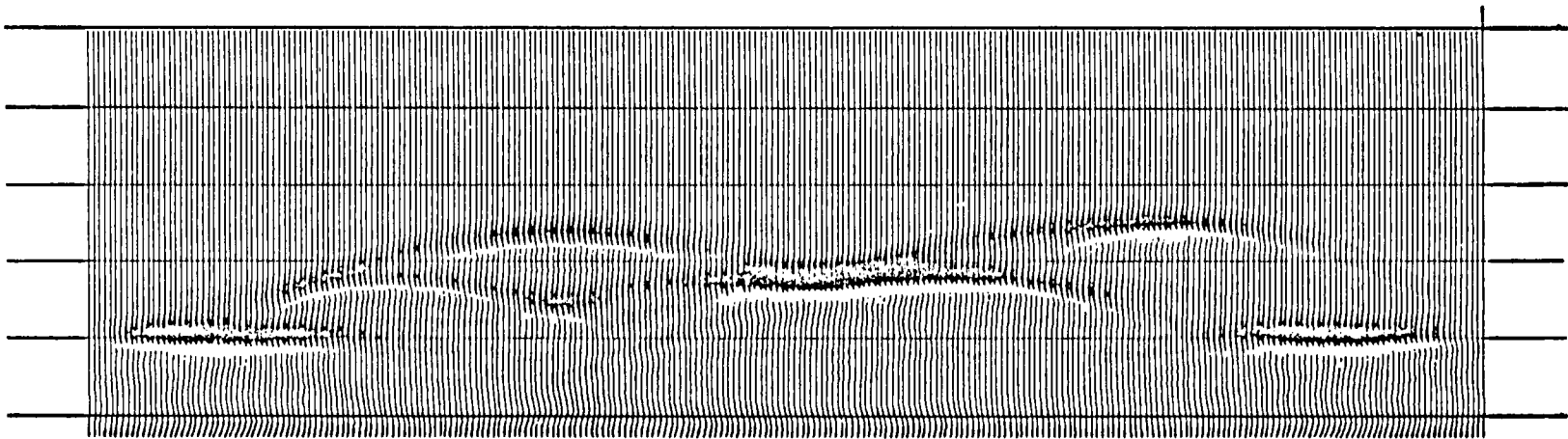


Figure 44. Synthetic time section generated along $X = 128$ using the exploding reflector sixth order program for the absorbing boundaries case. (Compare with Figure 35)

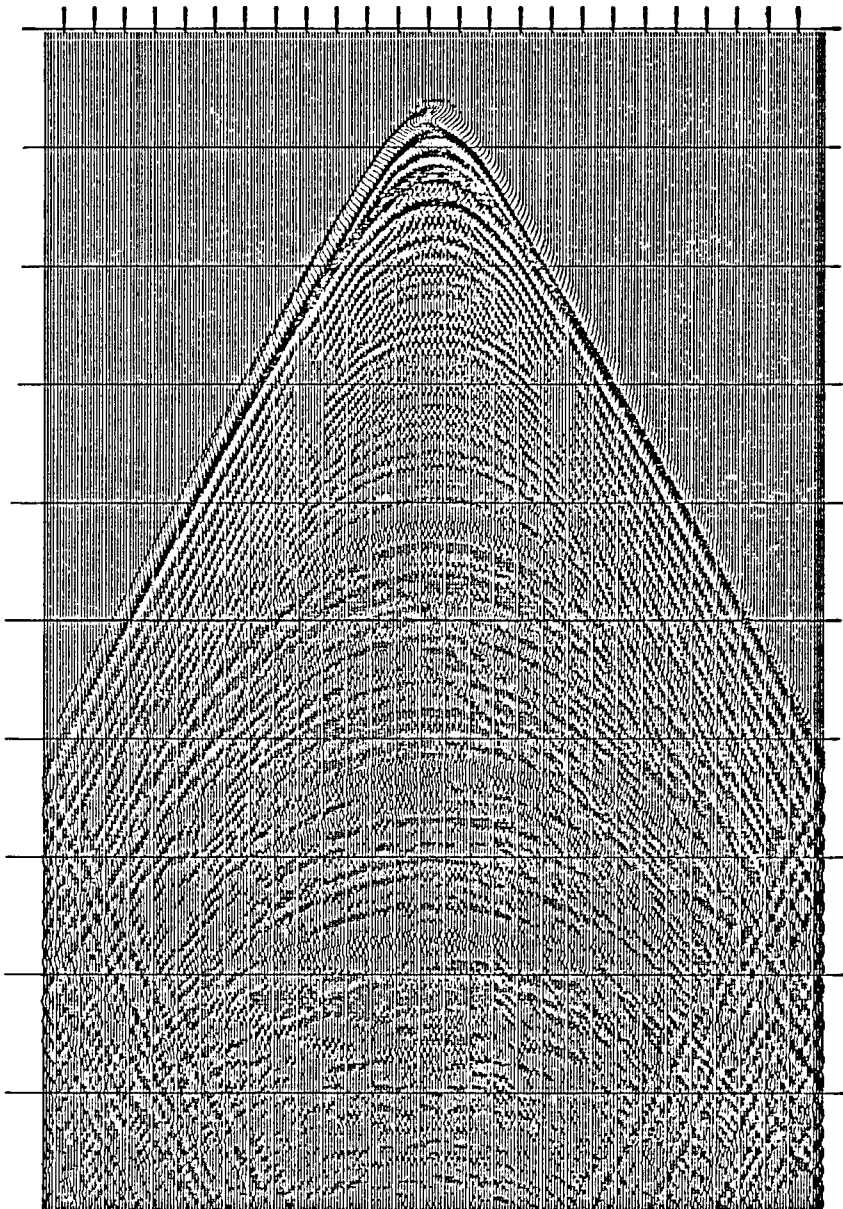


Figure 45. Synthetic time section recorded at $Z = 5$ for the point source wave case. Point source location: (128, 128, 20).

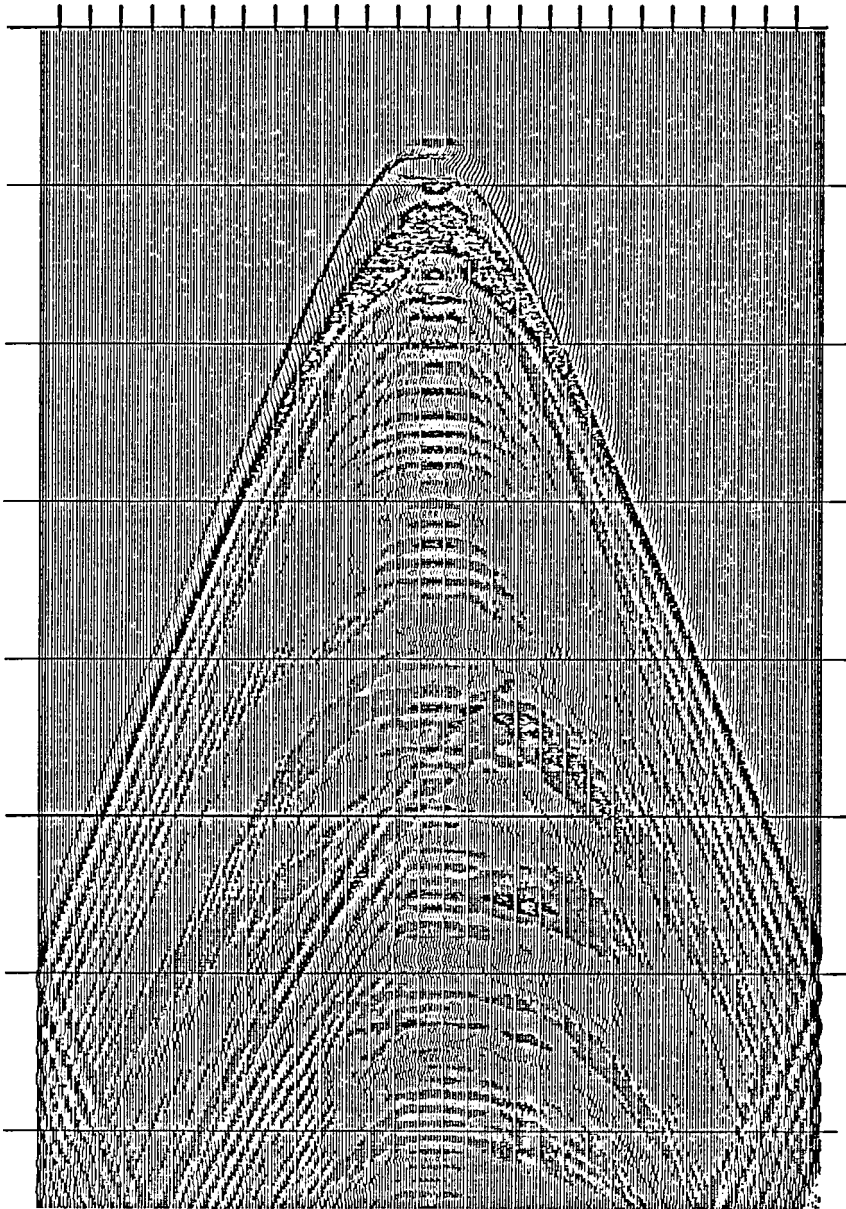


Figure 46. Synthetic time section recorded at $Z = 5$ for the plane wave case. Plane wave located between the planes: $X = 125$ and $X = 131$; $Y = 122$ and $Y = 134$; and $Z = 17$ and $Z = 23$.

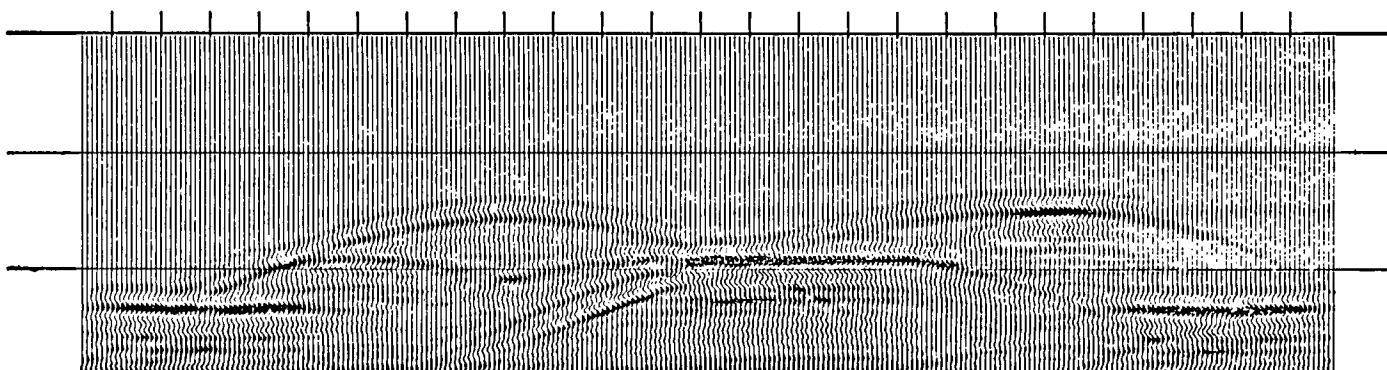


Figure 47. Synthetic time section generated along $X = 128$ using the exploding reflector sixth order program for the point elimination case. (Compare with Figure 34)