A KOSLOFF/BASAL METHOD, 3D MIGRATION PROGRAM IMPLEMENTED ON THE CYBER 205 SUPERCOMPUTER

L. D. PYLE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF HOUSTON
HOUSTON, TEXAS

S. R. WHEAT
BELL TELEPHONE LABORATORIES
NAPERVILLE, ILLINOIS

Title: A Kosloff/Basal Method, 3D Migration Program Implemented on the CYBER 205 Supercomputer

Authors: L.D. Pyle* and S.R. Wheat**

ABSTRACT:

Conventional finite-difference migration has relied on approximations to the acoustic wave equation which allow energy to propagate only downwards. Although generally reliable, such approaches usually do not yield an accurate migration for geological structures with strong lateral velocity variations or with steeply dipping reflectors. An earlier study by D. Kosloff and E. Baysal (Migration with the Full Acoustic Wave Equation) examined an alternative approach based on the full acoustic wave equation. The 2D, Fourier-type algorithm which was developed was tested by Kosloff and Baysal against synthetic data and against physical model data. The results indicated that such a scheme gives accurate migration for complicated structures. This paper describes the development and testing of a vectorized, 3D migration program for the CYBER 205 using the Kosloff/Baysal method. The program can accept as many as 65,536 zero-offset (stacked) traces. In order to efficiently process a data cube of such magnitude, (65 million data values), data motion aspects of the program employ the CDC supplied subroutine SLICE4, which provides high speed input/output, taking advantage of the efficiency of the system-provided subroutines Q7BUFIN and Q7BUFOUT and of the parallelism achievable by distributing data transfer over four different input/output channels. The results obtained are consistent with those of Kosloff and Baysal. Additional investigations, based upon the work reported in this paper, are in progress.

This research was supported by the Control Data Corporation and the Allied Geophysical Laboratories at the University of Houston.

*Department of Computer Science, Unversity of Houston, Houston, Texas
**Bell Telephone Laboratories, Naperville, Illinois

I INTRODUCTION

1.1 THE KOSLOFF/BAYSAL STUDY

In an attempt to develop a migration technique that did not have the faults of conventional finite-difference migration techniques, Kosloff and Baysal introduced a migration technique based on the full acoustic wave equation [1]. While conventional finite-difference techniques used an approximation to the wave equation, they allowed energy to propagate only downwards. Although these techniques yield reliable migration in most cases, they usually do not yield an accurate migration for geological structures with strong lateral velocity variations or with steeply dipping reflectors. The results of the migration technique developed by Kosloff and Baysal showed their technique to be able to accurately migrate these complicated geological structures. Furthermore, they found that there was no need to invoke complicated schemes in an attempt to correct the deficiencies of one-way equations [2].

1.2 DESCRIPTION OF THE PRESENT STUDY

Although the technique developed by Kosloff and Baysal provides an excellent migration algorithm, it still is a two-dimensional migration technique. The object of this research was to extend the 2D migration technique of Kosloff and Baysal into a 3D migration technique that would migrate a cube of 65,536 (or less) traces, each of length 1,024 samples. This goal immediately imposed several problems that were much greater than extending the numerical methods of Kosloff and Baysal. Of these problems, execution time and data motion were the most significant. Although the 2D migration of Kosloff and Baysal was implemented on a Digital Equipment Corporation VAX-11/780 incorporating a FPS-100 array processor, with favorable processing time, it was observed that this hardware was much too small to expect it to handle the 3D technique in a reasonable amount of time. Consequently, for its high rate of computation, the CDC CYBER 205 located at Colorado State University (CSU) was chosen to be the target machine. In Chapters II, III and IV, the following aspects of the 3D migration technique are developed: (1) the numerical methods involved; (2) the major features of the program implementing the 3D migration technique; and (3) the results of numerical tests of the program.

II THE KOSLOFF/BAYSAL FOURIER TECHNIQUE

2.1 INTRODUCTION

Conventional finite-difference migration has relied on approximations to the wave equation which allow energy to propagate only downwards. Although generally reliable, such equations usually do not give accurate migration for structures with strong lateral velocity variations or with steep dips. The migration technique presented here is a three-dimensional extension of a two-dimensional migration technique developed earlier by Kosloff and Baysal [3]. The migration technique presented here, referred to in this paper as the KBF migration technique (for Kosloff/Baysal Fourier type), is based on the full acoustic wave equation, (2.1).

$$\frac{\delta}{\delta x}\left[\frac{1}{\rho}\frac{\delta P}{\delta x}\right]+\frac{\delta}{\delta y}\left[\frac{1}{\rho}\frac{\delta P}{\delta y}\right]+\frac{\delta}{\delta z}\left[\frac{1}{\rho}\frac{\delta P}{\delta z}\right]=\frac{1}{c^2\rho}\frac{\delta^2 P}{\delta t^2} \quad (2.1)$$

2.2 INPUT

It is assumed that input to the KBF program consists of a "cube" of zero-offset traces in (x,y,z=0,t) space. The KBF technique presented here is designed to handle Nx * Ny such traces corresponding to Nx * Ny uniformly spaced points in the x and the y directions. The implementation discussed is designed so that the following must be true:

$32 <= Nx <= 256$ and $Nx = 2^i$ for some integer i

$32 <= Ny <= 256$ and $Ny = 2^j$ for some integer j

These restrictions were chosen so as to test program efficiency; they do not apply, in general, to the KBF scheme.

For each (x, y) pair, there will be N_t sample points in time, t_m, $m = 1, \ldots, N_t$, at which values of pressure, $P(x,y,z=0,t_m)$ are given. N_t must also be a power of two.

In equation (2.1) it is assumed that the density, ρ, is constant and that the velocity function, c(x,y,z), will be provided by the user. For testing purposes, velocity is given by a Fortran function subprogram in the code presented in Appendix. Other forms representing the velocities may be used to replace the supplied function.

2.3 THE KOSLOFF/BAYSAL TECHNIQUE, IN 3D

OBJECT OF THE PROGRAM

Given P(x, y, z=0, t) for t = 0, 1DT, 2DT, ..., TMAX

obtain P(x, y, z, t=0) for z = 0, 1DZ, 2DZ, ..., ZMAX

BASIC NUMERICAL METHOD

Equation (2.1) is Fourier transformed with respect to time, assuming density, ρ, is constant. The second order transformed equations can then be reduced to a system of first order equations in the usual manner. If density is constant, then we can write the following series of equations:

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} = \frac{1}{c^2}\frac{\partial^2 P}{\partial t^2}$$

$$P(x,y,z,t) = F^{-1}\tilde{P}(x,y,z,w)$$

$$\frac{\partial P}{\partial t} = jwF^{-1}\tilde{P}$$

$$\frac{\partial^2 P}{\partial t^2} = -w^2F^{-1}\tilde{P}$$

where

$$W = \begin{bmatrix} w_0 & & & \\ & w_1 & & \\ & & \ddots & \\ & & & w_{n-1} \end{bmatrix}$$

$$\Longrightarrow \quad \nabla^2 F^{-1}\bar{p} + \frac{\partial^2}{\partial z^2} F^{-1}\bar{p} = -\frac{W^2}{c^2} F^{-1} p$$

$$\Longrightarrow \quad \nabla^2 \bar{p} + \frac{\partial^2}{\partial z^2}\bar{p} = -\frac{W^2}{c^2}\bar{p}$$

$$\Longrightarrow \quad \frac{\partial}{\partial z}\begin{bmatrix} \bar{p} \\ \frac{\partial \bar{p}}{\partial z} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-W^2}{c^2} - \nabla^2 & 0 \end{bmatrix}\begin{bmatrix} \bar{p} \\ \frac{\partial \bar{p}}{\partial z} \end{bmatrix} \qquad (2.2)$$

where

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \qquad (2.3)$$

which is of the form

$$\frac{\partial v}{\partial z} = f(z,v) \qquad (2.4)$$

where

$$v = \begin{bmatrix} \bar{p} \\ \frac{\partial \bar{p}}{\partial z} \end{bmatrix} \qquad (2.5)$$

The expression "transformed with respect to time" means that the functions $P(x,y,z,t_m)$ are represented by Discrete Fourier Transforms:

$$P(x,y,z,t_m) = \sum_{i=1}^{N_t} \tilde{P}(x,y,z,w_i) e^{jw_i t_m} \qquad (2.6)$$

where

$$t_m = \begin{cases} (m-1)\, DT & \text{for } m = 1, 2, \ldots, \frac{N_t}{2} + 1 \\ (m-(N_t+1))DT & \text{for } m = \frac{N_t}{2} + 2, \ldots, N_t \end{cases}$$

$\tilde{P}$ is given by the Inverse Discrete Fourier Transform:

$$\tilde{P}(x,y,z,w_i) = \frac{1}{N_t} \sum_{m=1}^{N_t} P(x,y,z,t_m)\, e^{-jw_i t_m} \qquad (2.7)$$

where

$$w_i = \begin{cases} \frac{2\pi}{DTN_t} (i-1) & \text{for } i = 1, 2, \ldots, \frac{N_t}{2} + 1 \\ \frac{2\pi}{DTN_t} (i-(N_t+1)) & \text{for } i = \frac{N_t}{2} + 2, \ldots, N_t \end{cases}$$

DT is the sampling interval in time; $j = \sqrt{-1}$. Equation (2.6) is then substituted in (2.1). This results in (2.2), which must be satisfied for each w_i, for $i = 1, \ldots, \frac{N_t}{2} + 1$.

Thus, the N_t partial differential equations which provide a discrete approximation to (2.1), involving unknown functions $P(x,y,z,t_m)$ are replaced by $\frac{N_t}{2}$ +1 partial differential equations involving unknown functions $\tilde{P}(x,y,z,w_i)$. Note that in the transformed equations, dependence on time, t, has been eliminated.

With an appropriate approximation to $\nabla^2\tilde{P} \equiv \frac{\partial^2\tilde{P}}{\partial x^2} + \frac{\partial^2\tilde{P}}{\partial y^2}$

the "classical" 4th order Runge-Kutta algorithm is applied to integrate equation (2.2) numerically in z. The (vector) computational equations are summarized below:

$K1 = Dz * f(z, v_{old})$

$K2 = Dz * f(z + \frac{Dz}{2}, v_{old} + \frac{K1}{2})$

$K3 = Dz * f(z + \frac{Dz}{2}, v_{old} + \frac{K2}{2})$

$K4 = Dz * f(z + Dz, v_{old} + K3)$

$v_{new} = v_{old} + (K1 + 2K2 + 2K3 + K4) / 6$

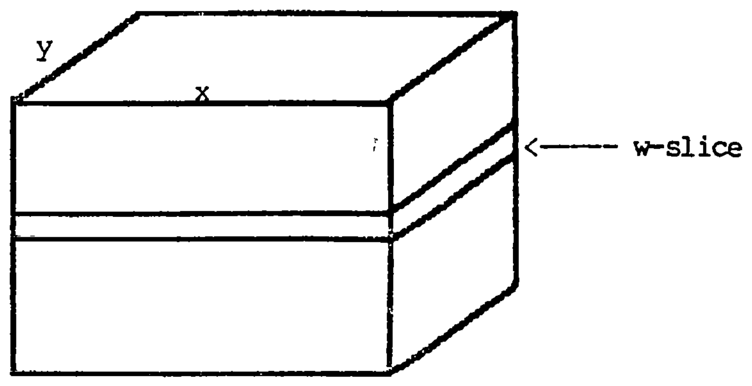
2.4 KBF DESIGN OUTLINE

The program has four main subdivisions, whose tasks are summarized below:

Part I: For each pair of (x,y) values, the corresponding zero-offset trace of P(x,y,0,t) values is converted to another "trace" of $\tilde{P}(x,y,0,w)$ values by application of the discrete Fourier transform (2.7).

Part II: For each w_i value $(i=1,2,\ldots,N_t)$ the $\tilde{P}(x,y,0,w_i)$ values are re-ordered into w_i-slices organized either sequentially in y for each x, or sequentially in x for each y, as appropriate for further transformations.

Part III: Each w_i-slice, from the transformed input cube of $\tilde{P}(x,y,0,w_i)$ values (see Figure 2.1), is developed into an (x,y,z,w_i) cube of $\tilde{P}(x,y,z,w_i)$ values. This development is performed by integrating equation (2.2) numerically. The resulting $\tilde{P}(x,y,z,w_i)$ values are accumulated for all w_i for each (x,y,z) combination. Since all the related exponential multipliers $e^{jm_it_1}$ equal 1 in magnitude (see equation (2.6)), this results in the generation of P(x,y,z,t=0) values, as required. (Note: $t_1 = 0$)



Note: z = 0 throughout this data cube.

Figure 2.1
Transformed Input Cube

There are two sub-problems of Part III:

Part III.1: Initial values for $\frac{\partial \tilde{\tilde{P}}}{\partial z}$ are obtained by the application of a two-dimensional Fourier transform to $\tilde{P}$ followed by multiplication by SQRT[-1 * $(\frac{W^2}{c^2} - \nabla^2)$]. Evanescent energy components are then eliminated and $\frac{\partial \tilde{P}}{\partial z}$ is obtained by the application of a 2-dimensional inverse Fourier transform to $\frac{\partial \tilde{\tilde{P}}}{\partial z}$.

Part III.2: $\tilde{P}(x,y,z,w)$ and $\frac{\partial\tilde{P}}{\partial z}(x,y,z,w)$ are propagated from z to z+ ∇z using the Runge-Kutta 4th order method to integrate equation (2.2) numerically. To do this

$$\nabla^2\tilde{P} = \frac{\partial^2\tilde{P}}{\partial x^2} + \frac{\partial^2\tilde{P}}{\partial y^2}$$

must be approximated four times for each ∇z. This is achieved by the use of a two-dimensional Fourier transform, followed by multiplication by $-(k_x^2 + k_y^2)$. Evanescent energy is eliminated from $\tilde{P}$ by applying a two-dimensional Fourier transform to $\tilde{P}$, obtaining $\tilde{\tilde{P}}$. For all (K_x, K_y) pairs such that $K_x^2 + K_y^2 > w_i/c(x,y,z)$, $\tilde{\tilde{P}}$ is set to zero. Then a two-dimensional inverse Fourier transform is applied to yield $\tilde{P}'$, which is input to the next step of numerical integration. Evanescent energy is also removed from $\frac{\partial\tilde{P}}{\partial z}$ in the same manner.

Part IV: For each (x,y), the $P(x,y,z,t=0)$ values in Part III are retrieved so as to be contiguous in Z. These space traces are each Fourier transformed and the downgoing energy is eliminated by filtering out components with negative wave numbers K_z. The resulting filtered traces are inverse Fourier transformed, retaining only the real part of the result, which is the desired 3D depth migration.

III PROGRAM DESIGN FEATURES

3.1 INTRODUCTION

The speed and capacity of the computer available to an individual researcher imposes certain restrictions on the types of problems that can be solved. The CYBER 205's vector features and high speed scalar processor provide a tool for solving problems in a matter of minutes that would take on the order of days on a conventional scalar machine (this speed increase depends, to a considerable extent, on the degree to which it is possible to "vectorize" the scalar code). Of the problems that can now be solved using the CYBER 205, the migration application presented here makes extensive use of the CYBER 205's vector facilities. This chapter contains an overview of vector processing on the CYBER 205 and an in-depth discussion of the data-flow required by the KBF migration algorithm.

3.2 CONCEPTS OF VECTOR PROCESSING

This section deals primarily with the concept of vector machines; however, it is not within the scope of this paper to bring the novice up-to-date on vector computing. Several texts and papers have been written to perform that task. Hockney and Jesshope [4] present a comprehensive text covering vector and parallel processors as well as vector and parallel algorithms. Section 2.3 of Hockney and Jesshope [5] is dedicated to the CDC CYBER 205. For more information on the CYBER 205, see also Kascic [6].

THE CDC CYBER 205, HISTORY

The CYBER 205, announced in 1980, replaced its predecessor, the CYBER 203. In turn, the CYBER 203, introduced in 1979, was a re-engineered version of the STAR 100. Conceived in 1964, the first STAR 100 became operational in 1973. The instruction set for the vector operations in the STAR 100 were based, primarily, on the APL language. The STAR 100 was designed to execute at a rate of 100 Mega-flops (1 Mega-flop = one million floating point instructions executed per second).

THE CDC CYBER 205, DESIGN

The CYBER 205 is a member of the family of "pipelined" machines. Pipeline refers to an assembly-line style of performing certain operations; thus more than one set of operands can be operated upon at a time. The vector processor of the CYBER 205 has what are known as vector pipes. These vector pipes are designed to stream contiguous data elements (vectors) through their pipelines. Presently, the CYBER 205 can have as many as four vector pipes, all of which can operate concurrently. A four pipe CYBER 205, processing 32-bit words, can operate at a peak rate of 800 mega-flops.

The various data types utilized by the CYBER Fortran 2.0 language include the following:

Type		Comments
Bit	:	the machine is bit addressable
Half-word	:	32-bit floating point
Full-word	:	64-bit floating point; 64-bit integer
Double-precision	:	128-bit floating point
Complex	:	two consecutive 64-bit words

Vectors on the CYBER 205 are "pointed to" by vector descriptors. A vector descriptor is a 64-bit entity with the following two fields: (1) Vector length, which consists of 16 bits and (2) Virtual address of the first vector element, which consists of the remaining 48 bits. Thus, a vector can have a length ranging from 0 to 65,535. Note that a bit vector can be no longer than 65,535 elements even though it consists of only 1024 64-bit memory words.

Vector operations come in a variety of forms on the CYBER 205, some of which are displayed in Table 3.1.

Table 3.1. Vector Operation Examples.

```
DIMENSION A(100), B(100), C(100)

L = 100
```

EXAMPLE NUMBER	VECTOR CODE	EQUIVALENT SCALAR CODE
(1)	A(1; L) = Q8VINTL(0, 1; L)	DO 10 I = 1, L 10 A(I) = I - 1
(2)	B(1; L) = A(1; L) * 20.0	DO 20 I = 1, L 20 B(I) = A(I) * 20.0
(3)	C(1; L) = A(1; L)*2.0+B(1; L)	DO 30 I = 1, L 30 C(I)=A(I)*2.0+B(I)

The examples in Table 3.1 are rather simple but resemble many operations in scientific programs. Examples 1 and 2 show a vector function call and a vector-scalar operation. Example 3 shows a "linked triad" operation. A linked triad operation takes advantage of CYBER 205 hardware which supports such operations. As one can see in Table 3.2, the linked triad operations are quite efficient. An operation is generally considered a linked triad when it consists of two vector operands and one scalar operand.

In certain situations, the results of some elements of a vector operation need not be saved. In this case, there is a mechanism for avoiding storage which involves a control vector. A control vector is a bit vector that specifies the storage of vector results. The control vector will be the same length as the result vector and where it has a value of one the corresponding result vector element will be saved and where it has a value of zero the corresponding result vector element will not be saved. The programmer also has the choice of reversing the meaning of the one's and zero's in the control vector.

A certain number of clock cycles are needed to set up the vector pipes. As this setup time is constant for a given operation, it is more efficient, in terms of total execution time, to reduce the number of vector operations by increasing the vector lengths whenever possible. Table 3.2 shows the set-up times, as well as the timings for the actual operations for various operations on the CYBER 205.

Table 3.2. Vector Timing Information

Vector Instruction	Number of Set-up Cycles	Number of Operating Cycles
Addition, Subtraction	51	N / 4
Multiplication	52	N / 4
Division, Square root	80	N / .61
Linked triad	84	N / 4

Where:
N = Vector length
1 Cycle = 20 nano-seconds
The vector operations are on 32-bit words

3.3 A NOTE ON THE APPLICATION OF VECTOR PROCESSING TO THE KBF METHOD

The KBF migration technique is such that almost all of the necessary operations can be vectorized. When working with a particular w-slice, all of the operations, including the two-dimensional FFT's, are vector operations. The computations performed at any given point of the omega-slice must be performed at all of the points. If there is a certain criteria that causes something different to occur at a given omega-slice point, a control vector can be created, dynamically, and the operation can still be performed in a vector manner. An example of this may be found in the routine CUTOFF where the evanescent energy is eliminated. In summary, there is no particular operation in the KBF migration scheme that can not be treated as a vector operation. To emphasize this point, one should examine the technique presented in chapter 2 and notice that there are no tricky operations that would prevent vectorization. In particular, it is important to note that there are no operations that have the following structure:

```
      DO 100 I = 1, N
         X(I) = F(Y(I))
         IF (X(I) .LT. VAL) GO TO 200
100      CONTINUE
200   CONTINUE
```

The above code can not be efficiently vectorized because of the inherently sequential nature of the computations.

3.4 DATA CONSIDERATIONS

As previously discussed, a program implementing the KBF migration technique, extended into three dimensions, is easily expressed in terms of vector operations. The program developed here contains very few scalar operations, many of which are operations needed in order to control various vector instructions or vector subroutine calls. Having such a match of software to hardware, one might conclude that there are no remaining barriers to running the program. There are, however, a few major items that one tends to overlook, being overwhelmed by the computational power of the CYBER 205. The greatest of these is the data motion required to keep the CYBER 205 vector pipes busy.

One penalty for the use of vector operations is that the data must be contiguous in memory for greatest efficiency (let alone for some vector operations to run at all). Furthermore, the vectors must reside in main memory as much as possible in order to prevent sure death from thrashing. With this in mind, one must realize that the memory requirement for the vectors that are necessary to perform a single step of the integration of one omega slice is quite large. For example, a (256 by 256) complex XY plane will require eleven vectors of length 131,072 half-words. These, along with various support vectors, comprise 12 large pages (1 large page = 65,536 full-words). This is slightly less than half of the memory available to a user on a

2-megaword 205, however it is about all one can expect to get for any reasonable period in a time-sharing environment. But this is really just the tip of the iceberg - these are just the work arrays. The total data set consists of the input data cube, the work arrays, and the output data cube.

Continuing with the previous example, the input cube could very well be of size 256*256*1024 half-words and the output cube could be as much as twice the size of the input cube (the size of the output cube depends upon the number of ZSTEPS in the migration). This would be a total of 201,326,592 half-words, which is equivalent to 1536 large pages. Obviously, this is much more data than any CYBER 205 can have in memory at any given time. Consequently, the question of how to handle the data-flow arises. A solution that one may consider is to declare the data cubes to be huge arrays and to let the virtual memory mechanism handle the data cubes.

To consider declaring the two data cubes as arrays, one must realize that access to these two arrays would have to be in a contiguous manner. Otherwise severe thrashing would result. In the case of the KBF migration algorithm, access to the data cubes must be done in several ways that would break the rule of contiguous access. Thus, it would be wise to check into at least one alternate method of handling these data cubes as large arrays.

Before presenting the data motion method used in this study, the need for efficiency must be established. Continuing with the previous example and without discussing the code in detail, the subroutine RHS3 takes on the order of 100 milli-seconds to run, each time it is called. In this example, RHS3 would be called on the order of 4*512*512 (1,048,576) times. The time needed for all of these calls is approximately 29 hours. Thus, any time for performing the data-motion is added onto the 29 hours. Therefore, one needs to find a mechanism to perform the data-motion without making the program run for an unacceptable amount of time.

3.5 A FOUR-WAY PARALLEL DATA MOTION TECHNIQUE

CYBER 205 Fortran provides several routines that may be used to implement I/O that runs concurrently with other instructions being executed as well as with other I/O. These routines include Q7BUFIN, Q7BUFOUT, and Q7WAIT. For detailed information on these routines, see the CDC CYBER 200 FORTRAN VERSION 2 manual [7]. A typical use for these routines would be as follows:

```
                    .
                    .
                    .
      CALL Q7BUFOUT(...............)
      CALL WORK(............)
                    .
                    .
                    .
```

In this example where the programmer wishes to write information out to a unit and have the routine WORK run concurrently with the I/O. In general, as long as WORK does not use the I/O unit referred to in the Q7BUFOUT call, it can do anything it wishes. Thus, there is CPU activity concurrent to I/O activity.

Another example where two I/O requests cause concurrent I/O, is as follows:

```
                    .
                    .
                    .
      CALL Q7BUFIN(..............)
      CALL Q7BUFOUT(..............)
                    .
                    .
                    .
```

According to the CDC CYBER FORTRAN 2 manual [8], these calls are legal, so long as they do not access the same data block on the same disk. Also, two Q7BUFIN, two Q7BUFOUT calls, or a Q7BUFIN and a Q7BUFOUT call can be active at one time for a given unit.

It should be obvious that these "Q7" calls are the basis of a solution to the problem of data-flow that was presented in the previous section. Indeed, they are; yet they are only the basis of the method used in this study. Dr. Bjorn Mossberg [9], of Control Data Corporation, wrote a utility known as SLICE4. Mossberg used the "Q7" utilities; however, the scheme he developed is much more elaborate than a series of Q7 calls to a particular I/O unit.

SLICE4

It is not within the scope of this paper to duplicate Mossberg's documentation of SLICE4. However, the concept and the terminology of SLICE4 will be presented as it applies to this study. For efficient operation, SLICE4 must be tightly integrated into the master program. Therefore, its terminology affects the view that one takes of the master program.

In this study, two implementations of SLICE4 were needed and used; one for the input data cube and one for the output data cube. To explain the use of SLICE4, only the input data cube will be treated. The output data cube is handled in a similar manner.

SLICE4 TERMINOLOGY

The first step in using SLICE4 is to impose a coordinate system upon the data cube such that the cube is N1 by N2 by N3 elements in size, where N1 is the number of elements in what one normally considers the Z direction, N2 is the number of elements in the X direction, and N3 is the number of elements in the Y direction. The next step is to define a second coordinate system on the data cube. Instead of being coordinates of individual data items, this second coordinate system gives coordinates of "super-blocks." Super-blocks are small cubes of the original data set. The super-block coordinate system has NS1 super-blocks in the 1-direction, NS2 in the 2-direction, and NS3 in the 3-direction, where NS1 and NS2 must be multiples of four. NS3 does not have this restriction; however, for greatest efficiency, it should be one or a multiple of four. The reason for the multiple of four rule is that the super-blocks will reside on four different I/O units. No matter which direction the cube is accessed, each I/O unit will have one quarter of the super-blocks accessed. This is not the case when only a partial row or column of super-blocks is accessed; thus, it is most efficient to access a complete row or column. If it should happen that more than one I/O unit be controlled by a given controller, then SLICE4 will still execute, but in a less efficient manner (i.e. the parallelism is partially inhibited). Thus, one may access any four adjacent super-blocks at a cost which is one fourth the cost of accessing the same data with conventional techniques.

The super-blocks themselves have a coordinate structure imposed upon them. This coordinate structure is L1 by L2 by L3. Where L1 is the number of elements from the data cube in the 1-direction; L2 and L3 are defined in the same manner for their individual directions.

Summarizing the terminology presented so far, the original data cube is broken up into NS1 by NS2 by NS3 super-blocks. Each super-block has L1 by L2 by L3 data elements. Thus the following rules must apply:

```
N1 = NS1 * L1    with    NS1 = 4 * i,  i => 1
N2 = NS2 * L2    with    NS2 = 4 * j,  j => 1
N3 = NS3 * L3
```

SUPER-BLOCK ACCESS

The rows and columns of super-blocks are referred to as slices. A 1-slice is some column of super-blocks in the 1-direction, a 2-slice is some row of super-blocks in the 2-direction, and a 3-slice is some row of super-blocks in the 3-direction. One may access all, or just some, of the super-blocks of a slice via SLICE4. However, in this study, only the most efficient access is performed - accessing all super-blocks of a given slice. As access can be by any given slice, SLICE4 must have the super-blocks all formatted in the same manner. Thus, when accessing a given slice, the slice is written into a buffer by SLICE4 and the user must re-format the data from the buffer into a work array in the format that corresponds to the direction of access.

DIMENSION CONSIDERATIONS

One needs to be careful to have enough array and buffer space to access the data cube in all the necessary directions. Thus, the size of the super-block comes into question. The larger the super-block, the fewer accesses to the data cube are needed and vica versa. In this study, the L1 dimension was set permanently to the value of 2. The reason for this is that, as one recalls from the migration technique, a complete XY plane is processed at any given time and there is only enough memory space to have two input planes in memory at the same time.

IV RESULTS AND CONCLUSIONS

4.1 EXECUTION TESTS

As discussed in section 3.4, it would take over 29 hours of execution time to migrate the maximum (assumed) data cube; thus for testing purposes, an input cube of size (64x64x64) was used. For both of the test runs discussed here, all of the traces consisted completely of zeros, except the center trace that had a single wavelet peaking at sample 16 (in time). The correctly migrated result, in this case, consists of a hemisphere. The first run (Figures 1 and 2) incorporated a padding in the time direction to delay the wrap-around effect inherent in Fourier algorithms. The second run (Figures 3 and 4) did not incorporate a padding - thus, wrap-around effects appeared. The first run took 240 CPU seconds and the second run took 115 CPU seconds.

Test Run 1: The migration of the input cube described above, using a constant velocity of 3000 m/s, a Dz interval of 6.0 meters, a Dx interval of 12.0 meters, a Dy interval of 12.0 meters, and a time interval of 4.0 milli-seconds, yields the results shown in Figures 1 and 2. Figures 1 and 2 are slices of the output cube in the XZ and in the YZ directions, respectively, intersecting at the center of the output cube (Note the absence of the wrap-around effect).

Test Run 2: The migration of the same input cube used in Test Run 1 using the same sampling rates in all dimensions, but with a velocity interface (see Figure 3; V1 = 4000 m/s; V2 = 3000 m/s), yields the results displayed in Figures 3 and 4. Note the wrap-around effect present in these figures.

4.2 FACTORS AFFECTING SPEED OF COMPUTATION

Until a superior algorithm for performing the I/O required by the KBF migration algorithm appears, SLICE4 will remain the most efficient method available to perform the I/O task. However, should a CYBER 205 ever be equipped with 8, or even 16, I/O channels, SLICE4 should easily be adapted to create SLICE8 and SLICE16 versions. Until then, there is little chance of decreasing the time required to perform the I/O.

Other than I/O, the Runge-Kutta 4th order algorithm employed in the KBF migration technique is the most expensive feature. Consequently, use of a less costly method for numerical integration (e.g., a multi-point method, using the Runge-Kutta method to get started) might result in increased computational efficiency.

4.3 CONCLUSIONS

The 3D KBF migration program, implemented on the CYBER 205 Supercomputer presented in this thesis, yields results that are consistent with those of Kosloff and Baysal [10]. This was confirmed by Kosloff [11]. Thus, a 3D migration program, using the KBF migration technique (based on the full acoustic wave equation) permitting lateral velocity variations is now available for use on the CYBER 205.

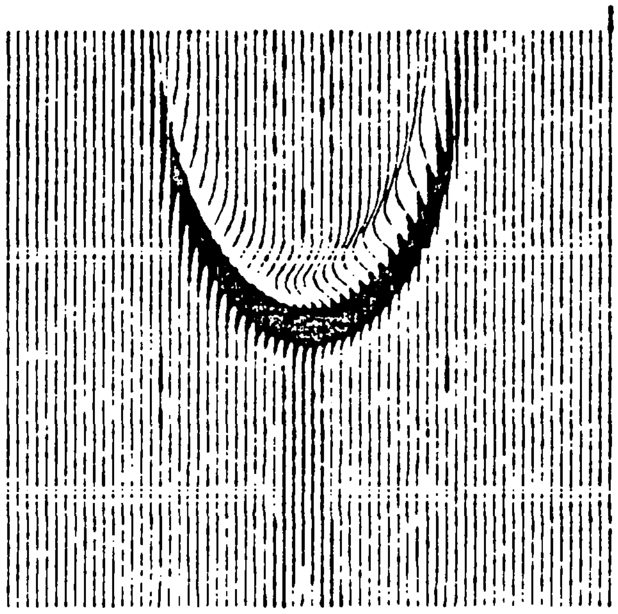


Figure 1

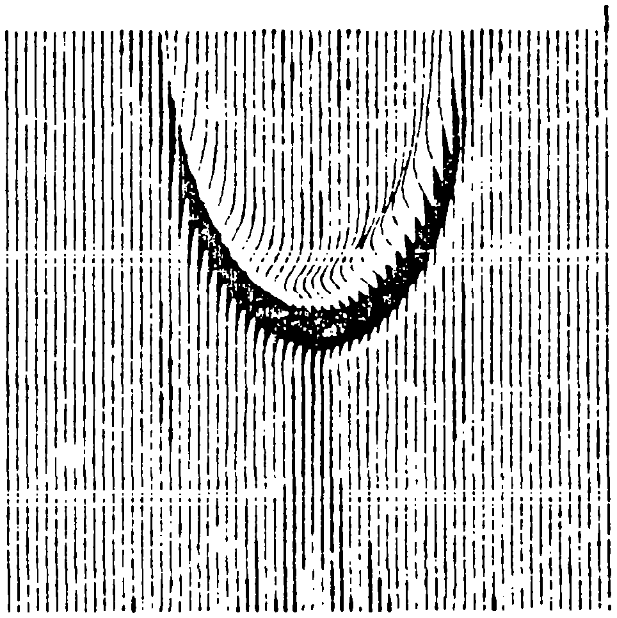


Figure 2

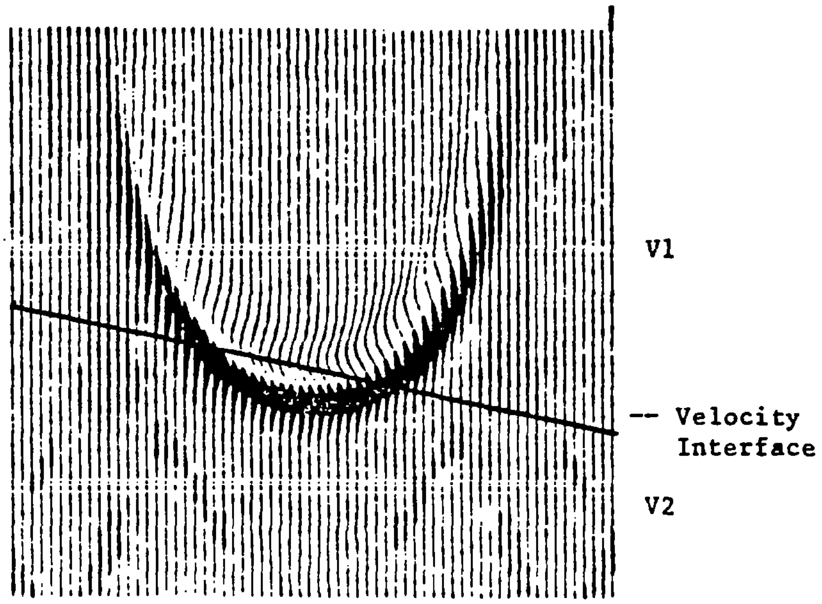


Figure 3

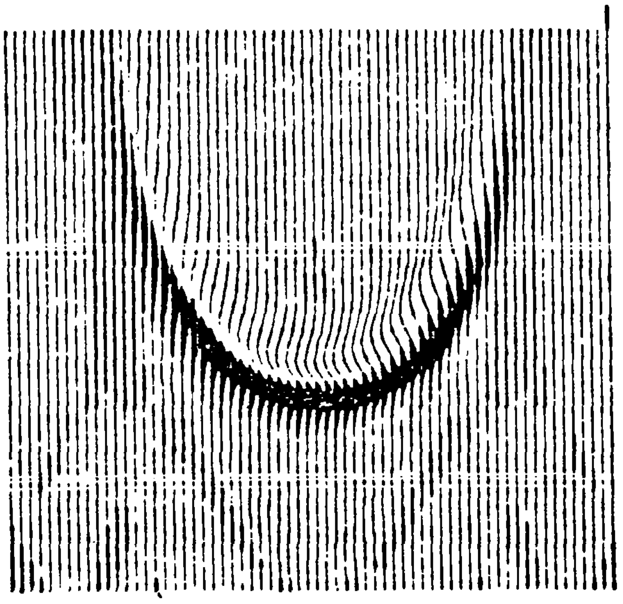


Figure 4

REFERENCES

1. Kosloff, D., and E. Baysal, "Migration With the Full Acoustic Wave Equation," Seismic Acoustics Labortory Fifth Year Semi-Annual Progress Review, No. 9 (1982), pp. 151-165.
2. Kosloff and Baysal, p. 152.
3. Kosloff and Baysal, pp. 151-165.
4. Hockney, R. W., and C. R. Jesshope, Parallel Computers: Architecture, Programming, and Algorithms (Bristol: Adam Hilger Ltd., 1981).
5. Hockney and Jesshope, pp. 95-126.
6. Kascic, M. J. Jr., Vector Processing On the Cyber 200 (St. Paul: Control Data Corporation, 1978).
7. Control Data Corp., CDC Cyber 200 Fortran Version 2 (St. Paul: Control Data Corporation, 1981).
8. Control Data Corp.
9. Control Data Corp., MAGEV Library Utility.
10. Kosloff and Baysal, p. 155.
11. Personal interview with Dan Kosloff, 25 August 1983.