Data Transposition and Helical Scheme in Prestack Finite-Difference Migration

Yu Zhang
Veritas DGC Inc., 10300 Town Park Drive, Houston, TX 77072, USA.
yu_zhang@veritasdgc.com

Jiandong "JD" Liang*
Veritas GeoServices, 715 - 5 Ave SW, #2200, Calgary, AB, Canada T2P 5A2
jd_liang@veritasgdc.com

and

Guanquan Zhang
Chinese Academy of Sciences, Beijing, China

ABSTRACT

Summary

We analyze the efficiency of implicit prestack finite-difference migration and focus on data transpositions in the two-way splitting algorithm. We discuss a tiled data transposition algorithm and how it achieves good efficiency by increasing access coherence. Then we introduce a helical scheme that eliminates the need for data transposition in common-shot migration. Numerical examples show that this helical algorithm can speed up downward extrapolation by 30-40% and produce comparable imaging quality as non-helical methods.

Introduction

The increasing demand for imaging complex geological structures has led to the exploration of wave equation based prestack depth migration methods, which do not suffer from such limitations as high frequency and few arrivals approximation as in the case of commonly used Kirchhoff migration. However, compared with Kirchhoff methods, these migration methods that are based on one-way wave equations are computationally more intensive, especially when 3-D prestack migrated imaging gathers are required. This has spurred researchers to seek various ways to speedup these algorithms.

In this abstract, we first focus on the data transposition in prestack finite-difference migration, which has a significant impact on migration efficiency but is often ignored in algorithm design. As we will show in the content, the traditional way of implementing implicit finite-difference scheme with two-way splitting (Brown, 1983) requires four x to y data transpositions in each downward continuation step in a prestack common-shot migration, and may take 30-40% of the total CPU time in wavefield extrapolation, depending on transposition

algorithm, data size and hardware. We introduce a tiled data transpose algorithm based on the principle of memory access coherence and investigate its performance impact for prestack migration.

Based on some previous work (Zhang et al. 2000, Zhang and Shan 2001), we then introduce a helical finite-difference scheme for 3-D common-shot migration that eliminates the need for data transposition. We discuss its implementation and efficiency in common-shot migration. This helical scheme was firstly proposed by Zhang in solving 3-D postack depth migration in t-x domain (Zhang and Hou, 1996). By re-formulating one-way wave equation and its finite-difference schemes, the helical method accesses wavefield data in memory sequentially during computation. This makes migration more efficient by saving memory operations of transposing data. Unlike the helical scheme proposed by Stanford Exploration Project Group (Rickett, et al., 1998), the method introduced here is a variant of traditional x-y splitting method and can easily handle lateral velocity variation. It also has splitting error as other 3-D implicit finite-difference migrations do and it relies on phase-correction (Graves and Clayton, 1990; Li, 1991; Zhou and McMechan, 1999), multi-way splitting (Collino and Joe, 1995; Ristow and Ruhl, 1997; Zhang et al., 2000) or other correction methods to improve its kinematic behavior.

Two-way splitting

The dominant calculation in wave equation migration is wavefield downward extrapolation. Given upgoing and downgoing wavefields U and D, the wave propagation is governed by following one-way wave equations

$$\frac{\partial U}{\partial z} = i\frac{\omega}{v}\left[1 + \frac{\alpha v^2\left(\partial_x^2 + \partial_y^2\right)}{\omega^2 + \beta v^2\left(\partial_x^2 + \partial_y^2\right)}\right]U\,, \qquad (1)$$

and

$$\frac{\partial D}{\partial z} = -i\frac{\omega}{v}\left[1 + \frac{\alpha v^2\left(\partial_x^2 + \partial_y^2\right)}{\omega^2 + \beta v^2\left(\partial_x^2 + \partial_y^2\right)}\right]D\,, \qquad (2)$$

where v is velocity, α and β are coefficients chosen to approximate the square-root operator (Lee and Suh, 1985). Directly solving above equations using stable implicit finite-difference schemes is both numerically difficult and time consuming. In production, implicit finite-difference with two-way splitting (Brown, 1983) is extensively used. Taking upgoing wavefield U for example, instead of solving *(1),* we actually calculate

$$\frac{\partial U}{\partial z} = i\frac{\omega}{v}\left(1 + \frac{\alpha v^2\partial_x^2}{\omega^2 + \beta v^2\partial_x^2} + \frac{\alpha v^2\partial_y^2}{\omega^2 + \beta v^2\partial_y^2}\right)U\,. \qquad (3)$$

The discrepancy between *(1)* and *(3)* generates so-called splitting error in migration. *Equation (3)* can be solved cascadedly

Phase shift: $$\frac{\partial U}{\partial z} = i\frac{\omega}{v}U\ , \qquad (4a)$$

x-pass: $$\frac{\partial U}{\partial z} = i\frac{\omega}{v}\frac{\alpha v^2\partial_x^2}{\omega^2 + \beta v^2\partial_x^2}U\ , \qquad (4b)$$

y-pass: $$\frac{\partial U}{\partial z} = i\frac{\omega}{v}\frac{\alpha v^2\partial_y^2}{\omega^2 + \beta v^2\partial_y^2}U\ . \qquad (4c)$$

In numerical computation, seismic data is stored along a certain direction, say, x-direction. In this case, operations along x direction are much faster than along y-direction. Therefore, in implementation, an x to y transposition is usually inserted between x and y pass splitting, and later the data is transposed back to original x-order. A typical computation data flow in downward extrapolation is shown in *Fig. 1*, which requires four data transpositions in each extrapolation step: two for upgoing wavefield and two for downgoing wavefield.

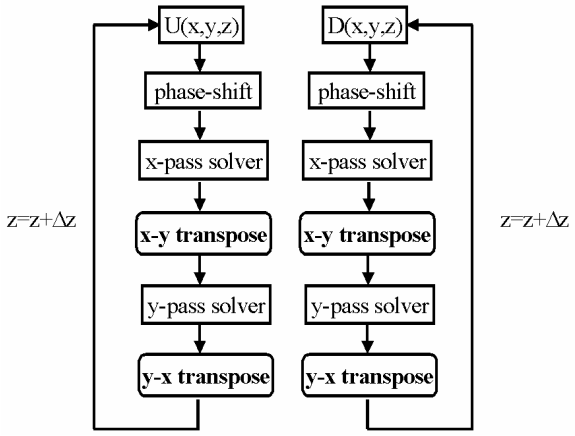

Fig. 1: data flow of downward extrapolation in common-shot migration. Usually it requires four data transpositions.

Data transposition

While data transposition does not involve significant arithmetic calculations, it accounts for a significant portion of the total migration time. Therefore, we would like to optimize its performance. Previous optimization efforts have focused upon minimizing the requirement for additional memory. In this particular case, it is not a major concern. Another aspect, namely access coherence, is more important.

Modern computer architecture relies on a small capacity high-speed buffer memory (called cache) to bridge the speed gap between CPU and primary memory. This mechanism dictates that the performance of an algorithm heavily depends on its memory access coherence. In particular, it severely penalizes algorithms with poor access coherence.

A direct data transposition algorithm happens to exhibit poor access coherence. Suppose we directly transpose an *n*-row by *m*-column matrix stored in row order. If we scan the source data in row order and assign them to the destination locations, we achieve good access coherence when reading the source data. However, we have poor coherence when writing to the destination locations, as each successive element is now *n* addresses apart. Likewise, if we scan the source data in column order, then we lose access coherence for the reading part even though we have access coherence for the writing part.

To achieve good coherence for both reading and writing, we designed a tiled data transposition algorithm. The idea is to group chunks of data that occupy coherent memory locations in both the source and destination matrix, and handle them as one unit at a time. We conceptually divide a matrix into sub-matrices, called tiles, such that two tiles (one as the source and the other as the destination) can fit into the data cache of the target computer architecture. We transpose one source tile into a destination tile at a time, as illustrated in *Fig. 2*, where A_{ij}, is a sub-matrix and $B_{ij} = \left(A_{ji}\right)^T$.

$$\begin{pmatrix} A_{11} & \cdots & A_{1q} \\ & & \\ A_{p1} & \cdots & A_{pq} \end{pmatrix} \Rightarrow \begin{pmatrix} B_{11} & \cdots & B_{p1} \\ & & \\ B_{1q} & \cdots & B_{pq} \end{pmatrix}$$

Fig. 2: illustration of tiled matrix transposition.

Assume the data elements are stored in row order. We transpose each tile in the order: $A_{11}\ A_{12}\ \ldots\ A_{1q}\ A_{21}\ \ldots\ A_{2q}\ \ldots\ A_{pq}$. This way, reading from each source tile A_{ij} has good memory coherence, since each row in A_{ij} is stored continuously. Since we transpose a tile at a time, each row of elements in the destination tile B_{ji} is also fully set before moving to the next one, therefore achieve good access coherence in the writing part as well.

Helical scheme

Based on the idea of memory coherence, we introduce a helical scheme in common-shot finite-difference migration. By discretization, the finite-difference scheme of Eq. (4b) can be written as

$$\left(I + r_x\delta_x\right)\left(I + r_y\delta_y\right)U_{i,j}^{n+1}(\omega) = \left(I + \bar{r}_x\delta_x\right)\left(I + \bar{r}_y\delta_y\right)U_{i,j}^{n}(\omega), \qquad (5)$$

where $U_{i,j}^{n}(\omega) = U(i\Delta x, j\Delta y, n\Delta z; \omega)$, I is the identical operator and δ is the second-order finite-difference operator, for example

$$\delta_x U_{i,j} = U_{i+1,j} - 2U_{i,j} + U_{i-1,j}.$$

Defining forward and backward first-order finite-difference operators:

$$\Delta_x^+ U_{i,j} = U_{i+1,j} - U_{i,j}, \quad \Delta_x^- U_{i,j} = U_{i,j} - U_{i-1,j}$$

and noticing following properties between operators Δ_x^+, Δ_x^- and δ_x:

$$\delta_x = \Delta_x^+\Delta_x^- = \Delta_x^-\Delta_x^+ = \Delta_x^+ - \Delta_x^-,$$

we have

$$I + r_x\delta_x = (I + q_x\Delta_x^+)(I - q_x\Delta_x^-), \qquad (6a)$$

where $q_x - q_x^2 = r_x$. Similarly, for y direction we have

$$I + r_y\delta_y = (I + q_y\Delta_y^+)(I - q_y\Delta_y^-). \qquad (6b)$$

Combining *(6a)* with *(6b),* we get

$$\begin{aligned} & \left(I + r_x\delta_x\right)\left(I + r_y\delta_y\right) \\ = \;& (I + q_x\Delta_x^+)(I - q_x\Delta_x^-)(I + q_y\Delta_y^+)(I - q_y\Delta_y^-) \\ = \;& (I + q_x\Delta_x^+)(I + q_y\Delta_y^+)(I - q_x\Delta_x^-)(I - q_y\Delta_y^-) \\ = \;& (I + q_x\Delta_x^+ + q_y\Delta_y^+ + q_xq_y\Delta_x^+\Delta_y^+)(I - q_x\Delta_x^- - q_y\Delta_y^- + q_xq_y\Delta_x^-\Delta_y^-) \end{aligned} \qquad (7)$$

The operator factorization and exchange in *(7)* are valid only when velocity is a constant. For a general velocity, the operator re-orgnization in *(7)* does not change the kinematics of wave propagation, therefore does not affect the purpose of seismic imaging. Applying finite-difference relation *(7)* to discretized one-way wave *equation (3),* we achieve the following wavefield extrapolation algorithm:

Phase shift: $$U_{i,j}^{n+1/3} = e^{i\frac{\omega}{v}\Delta z}U_{i,j}^n, \qquad (8a)$$

Forward:

$$(I + q_x\Delta_x^+ + q_y\Delta_y^+ + q_xq_y\Delta_x^+\Delta_y^+)U_{i,j}^{n+2/3} = (I + \overline{q}_x\Delta_x^+ + \overline{q}_y\Delta_y^+ + \overline{q}_x\overline{q}_y\Delta_x^+\Delta_y^+)U_{i,j}^{n+1/3}, \qquad (8b)$$

Backward:

$$(I - q_x\Delta_x^- - q_y\Delta_y^- + q_xq_y\Delta_x^-\Delta_y^-)U_{i,j}^{n+1} = (I - \overline{q}_x\Delta_x^- - \overline{q}_y\Delta_y^- + \overline{q}_x\overline{q}_y\Delta_x^-\Delta_y^-)U_{i,j}^{n+2/3}. \qquad (8c)$$

Let us define a helical coordinate system (Claerbout, 1998),

$$u_{i+(j-1)N_x}^n = U_{i,j}^n, \quad i = 1,\cdots N_x, \quad j = 1,\cdots N_y,$$

where N_x and N_y are the numbers of grid points along the x and y directions, respectively. In the helical coordinate, the implicit finite-difference schemes *(8b)* and *(8c)* become

$$au_k^{n+2/3} + bu_{k-1}^{n+2/3} + cu_{k-N_x}^{n+2/3} + du_{k-N_x-1}^{n+2/3} = \overline{a}u_k^{n+1/3} + \overline{b}u_{k-1}^{n+1/3} + \overline{c}u_{k-N_x}^{n+1/3} + \overline{d}u_{k-N_x-1}^{n+1/3}, \qquad (9a)$$

$$au_k^{n+1} + bu_{k+1}^{n+1} + cu_{k+N_x}^{n+1} + du_{k+N_x+1}^{n+1} = \overline{a}u_k^{n+2/3} + \overline{b}u_{k+1}^{n+2/3} + \overline{c}u_{k+N_x}^{n+2/3} + \overline{d}u_{k+N_x+1}^{n+2/3}, \qquad (9b)$$

where $a = (1 - q_x)(1 - q_y)$, $b = q_x(1 - q_y)$, $c = (1 - q_x)q_y$ and $d = q_xq_y$. (9a) and (9b) are like 1-D finite-difference schemes, therefore, they can be solved explicitly and do not need data transposition. The same algorithm can be applied to downgoing wavefield D. Coding the helical scheme *(9a)* and *(9b)* is simple, as shown by the following psuedo code:

if (scheme == forward) {

for (k=1 to N_x) $u_k^{n+1} = 0$;

increment=1;

k= N_x;

}

else (scheme == backward) {

for (k= $N_x N_y$ to $N_x N_y - N_x$) $u_k^{n+1} = 0$;

increment=-1;

k= $N_x N_y - N_x$;

}

for (i=1 to $N_x N_y$) {

k=k+increment;

k1=k-increment;

k2=k-increment* N_x;

k3=k2-increment;

Read velocity v and calculate a, b, c, d;

$u_k^{n+1} = \left(\bar{a}u_k^n + \bar{b}u_{k1}^n + \bar{c}u_{k2}^n + \bar{d}u_{k3}^n - bu_{k1}^{n+1} - cu_{k2}^{n+1} - du_{k3}^{n+1}\right)/a$;

}

Numerical tests

We first tested tiled data transposition versus direct transposition. In *Fig. 3*, we plot the average time required to transpose a data element as a function of the data width (i.e. the number of rows in the source data matrix). With direct transposition, this time increases with data width, as can be expected when access coherence degrades. With tiled transposition, this time stays constant, as access coherence maintains. *Fig. 4* shows the performance ratio of the two algorithms.

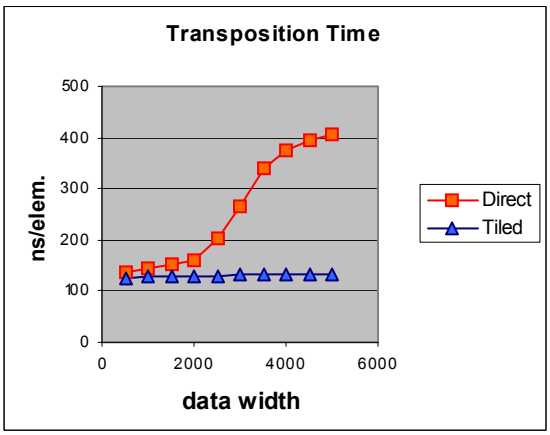

Fig. 3: average transposition time per data element.

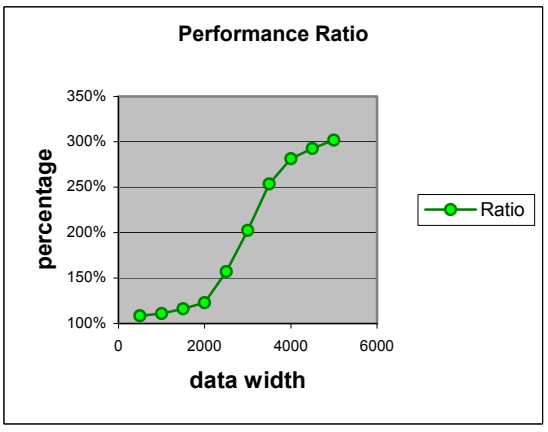

Fig. 4: performance ratio of tiled vs. direct transposition.

In the second test, we migrate a single shot record using implicit x-y splitting algorithm on a grid with $N_x = 480$ and $N_y = 320$. For non-helical program using tiled data transposition, the total CPU time for downward extrapolation is 2,126 seconds, and transposition takes 665 seconds, about 31% of the total time. We also tested directly transposition, which takes about 760 seconds for data transposition. By using helical scheme, CPU time for data transposition is saved, and the scheme *(9a-b)* takes about the same time as traditional x-y splitting solver. Therefore, in this test, helical scheme speeds up the migration by about 32%.

In the third test, we migrated SEG/EAGE model, C3NA dataset, which is a standard synthetic dataset for testing prestack depth imaging. To obtain steep dips and attenuate splitting error and numerical dispersion, we apply Zhiming Li's correction after x-y splitting in each downward continuation step (Li, 1991). *Fig. 5* is a crossline section of our migration output at $x = 5040m$. The dipping salt bottom and the flat reflector bellow the salt at depth 3500m are imaged clearly. The result is comparable to those obtained by traditional two-way splitting algorithms.

	Nonhelical scheme (sec)	helical scheme (sec)
Phase-shift	221	221
x-y splitting solver	1,240	1,189
Data transpositions	665	0
total	2,126	1,410

Table 1: CPU time of downward extrapolating a single shot record, with $N_x = 480$ and $N_y = 320$. In this test, helical scheme saves about 33%.

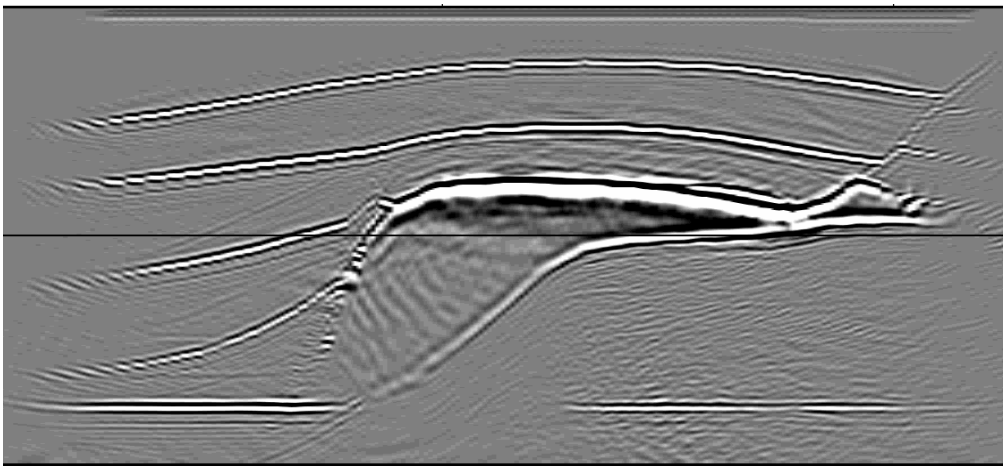

Fig. 5: Prestack depth migration of SEG/EAGE model, C3NA dataset, x=5040m

Conclusions

Directly implementation of data transposition exhibits poor access coherence and therefore needs to be optimized. The tiled data transposition algorithm performs better. Depending on data size, it can achieve acceleration between a few percent to a few times. We also introduce a helical scheme, which is designed to eliminate data resorting. Numerical tests show helical scheme is efficient and produces good depth migration results.

References

Brown, D. L., 1983, Applications of operator separation in reflection seismology: Geophysics, 48, 288-294.

Claerbout, J., 1985, Imaging the earth's interior: Blackwell Scientific Publication, Inc.

Claerbout, J., 1998, Multidimensional recursive filters via a helix with application to velocity estimation and 3-D migration: 68th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1995-1998.

Collino, F. and P. Joly, 1995, Splitting of operators, alternate directions, and paraxial approximations for the three-dimensional wave equation: SIAM J. Sci. Comput., 16, 1019-1048.

Graves, R.W. and Clayton, R.W., 1990, Modeling acoustic waves with paraxial extrapolators: Geophysics, 55, 306-319.

Lee, M. and Suh, S., 1985, Optimization of One-way Equation, Geophys., 50, 1634-1637.

Li, Z., 1991, Compensating finite-difference errors in 3-D migration and modeling: Geophysics, 56, 1650--1660.

Rickett, J., Claerbout, J. and Fomel, S. B., 1998, Implicit 3-D depth migration by wavefield extrapolation with helical boundary conditions, 68th Ann. Internat. Mtg: Soc. of Expl. Geophys., 1124-1127.

Ristow, D. and Ruhl, T., 1997, 3-D implicit finite-difference migration by multiway splitting: Geophysics, 62, 554--567.

Zhang, G. and Hou, W., 1996, Factorization algorithm for 3-D postack finite-difference migration, J. Geophysics (Chinese), 39, 382-391.

Zhang, G., Zhang, Y. and Zhou, H., 2000, Helical finite-difference schemes for 3-D depth migration, 70th Ann. Internat. Mtg: Soc. of Expl. Geophys., 862-865.

Zhang, G. and Shan, G., 2001, Helical scheme for 2-D prestack migration based on double-square-root equation, 71st Ann. Internat. Mtg: Soc. of Expl. Geophys., 1057-1060.

Zhou, H. and McMechan, G.A., 1999, Parallel Butterworth and Chebyshev dip filters with applications to 3-D seismic migration: Geophysics, 64, 1573-1578.