A fast butterfly algorithm for generalized Radon transforms

Jingwei Hu[1], **Sergey Fomel**[2], **Laurent Demanet**[3], **and Lexing Ying**[4]

[1]*Institute for Computational Engineering and Sciences (ICES)*

The University of Texas at Austin

1 University Station, C0200, Austin, TX 78712, USA

hu@ices.utexas.edu

[2]*Bureau of Economic Geology and Department of Geological Sciences*

Jackson School of Geosciences

The University of Texas at Austin

University Station, Box X, Austin, TX 78713, USA

sergey.fomel@beg.utexas.edu

[3]*Department of Mathematics*

Massachusetts Institute of Technology

77 Massachusetts Avenue, Cambridge, MA 02139, USA

laurent@math.mit.edu

[4]*Department of Mathematics*

and Institute for Computational and Mathematical Engineering (ICME)

Stanford University

450 Serra Mall, Bldg 380, Stanford, CA 94305, USA

lexing@math.stanford.edu

(February 11, 2013)

GEO-2013-

Running head: **Fast Generalized Radon Transforms**

ABSTRACT

Generalized Radon transforms such as the hyperbolic Radon transform cannot be implemented as efficiently in the frequency domain as convolutions, thus limiting their use in seismic data processing. We introduce a fast butterfly algorithm for the hyperbolic Radon transform. The basic idea is to reformulate the transform as an oscillatory integral operator and to construct a blockwise low-rank approximation of the kernel function. The overall structure follows the Fourier integral operator (FIO) butterfly algorithm. For two-dimensional data, the algorithm runs in complexity $O(N^2 \log N)$, where N depends on the maximum frequency and offset in the dataset and the range of parameters (intercept time and slowness) in the model space. Using a series of examples, we show that the proposed algorithm can be significantly more efficient than the conventional time-domain integration.

INTRODUCTION

In seismic data processing, the Radon transform (RT) (Radon, 1917), or slant stack, is a set of line integrals that map mixed and overlapping events in seismic gathers to a new transformed domain where they can be separated (Gardner and Lu, 1991). The integrals can also be taken along curves: parabolas (parabolic RT), or hyperbolas (hyperbolic RT or velocity stack) are most commonly used. A major difference between these transforms is that the former two are time-invariant (i.e., involve a convolution in time) whereas the latter is time-variant. When the curves are time-invariant, the transform can be performed efficiently in the frequency domain using the convolution theorem. However, this approach does not work for time-variant transforms. As a result, the hyperbolic Radon transform is usually thought of as requiring a computation in the time domain, which is computationally expensive due to the large size of seismic data. Nevertheless, the hyperbolic transform is often preferred as it better matches the true seismic events in common midpoint (CMP) gathers (Thorson and Claerbout, 1985).

In this work, we construct a fast butterfly algorithm to effectively evaluate time-variant transforms such as the hyperbolic Radon transform. As opposed to the conventional, costly *velocity scan* (i.e., direct integration + interpolation in the time domain), our method provides an accurate approximation in only $O(N^2 \log N)$ operations[1] for 2-D data. Here N depends on the maximum frequency and offset in the dataset and the range of parameters (intercept time and slowness) in the model space, and can often be chosen small compared to the grid size. The adjoint of the transform can be evaluated similarly without extra difficulty. Note that the algorithm introduced in this paper only deals with the fast implementation of a single integral operator (forward Radon transform or its adjoint), not an iteration process

[1]All the log in this paper refer to logarithm to base 2.

for its inversion which is the main objective of much previous work on fast Radon transforms (Sacchi, 1996; Trad et al., 2002; Liu and Sacchi, 2002; Wang and Ng, 2009).

Radon transforms have been widely used to separate and attenuate multiple reflections (Hampson, 1986; Yilmaz, 1989; Foster and Mosher, 1992; Herrmann et al., 2000; Moore and Kostov, 2002; Hargreaves et al., 2003; Trad, 2003). As having fast implementations of both forward and adjoint transforms is an essential component of least-squares minimization, our hope is that the current fast algorithm will help to make the hyperbolic Radon transform an accessible tool for improving the inversion process.

The term "generalized Radon transform" connotes a broader context where integrals are taken along arbitrary parametrized sets of smooth curves. The term was introduced by Beylkin in (Beylkin, 1984, 1985), where the author showed that an asymptotically-correct inverse follows from an amplitude correction to the adjoint. Kirchhoff migration and its (regularized) inverse can be expressed as generalized Radon transforms. The algorithm presented in this paper can in principle be applied in the context of Kirchhoff migration, although we do not attempt to do so here.

The rest of the paper is organized as follows. We first introduce the low-rank approximation and the butterfly structure of the hyperbolic Radon operator; then use these building elements to construct our fast algorithm. A brief description of the algorithm is given in the main text; while a complete derivation can be found in the appendix. Numerical examples of both synthetic and field data are presented to illustrate the accuracy and efficiency of the proposed algorithm.

ALGORITHM

Assume $d(t,h)$ is a function in the data space. The hyperbolic Radon transform R maps d to a function $(Rd)(\tau,p)$ in the model space (Thorson and Claerbout, 1985) through

$$(Rd)(\tau,p) = \int d(\sqrt{\tau^2+p^2h^2},h)\,dh. \tag{1}$$

Here t is time, h is offset, τ is intercept time, and p is slowness. Fixing (τ,p), the hyperbola $t=\sqrt{\tau^2+p^2h^2}$ describes the traveltime for the event. Hence integration along these curves can be used to identify different reflections.

Instead of approximating the integral in equation (1) directly, we reformulate it as a double integral,

$$(Rd)(\tau,p) = \iint \hat{d}(f,h)e^{2\pi i f\sqrt{\tau^2+p^2h^2}}\,df\,dh, \tag{2}$$

where f is the frequency, $\hat{d}(f,h)$ is the Fourier transform of $d(t,h)$ in t. A simple discretization of (2) yields[2]

$$(Rd)(\tau,p) = \sum_{f,h} e^{2\pi i f\sqrt{\tau^2+p^2h^2}}\hat{d}(f,h). \tag{3}$$

The reason that the hyperbolic RT is harder to compute than the linear RT ($t=\tau+ph$) or the parabolic RT ($t=\tau+ph^2$) should be clear from equation (3): product $f\tau$ in the phase cannot be decoupled from other terms.

To construct the fast algorithm, we first perform a linear transformation to map all discrete points in (f,h) and (τ,p) domains to points in the unit square[3] $[0,1]\times[0,1]$, i.e., a point $(f,h)\in[f_{\min},f_{\max}]\times[h_{\min},h_{\max}]$ is mapped to $\mathbf{k}=(k_1,k_2)\in[0,1]\times[0,1]$ via

$$f=(f_{\max}-f_{\min})k_1+f_{\min}, \quad h=(h_{\max}-h_{\min})k_2+h_{\min};$$

[2]For simplicity, the area element is omitted; the same symbols f, h, τ, and p are used for both continuous and discrete variables.

[3]We use $[a,b]\times[c,d]$ to represent a 2-D rectangular domain in xy-plane, with $x\in[a,b]$ and $y\in[c,d]$.

a point $(\tau, p) \in [\tau_{\min}, \tau_{\max}] \times [p_{\min}, p_{\max}]$ is mapped to $\mathbf{x} = (x_1, x_2) \in [0,1] \times [0,1]$ via

$$\tau = (\tau_{\max} - \tau_{\min})x_1 + \tau_{\min}, \quad p = (p_{\max} - p_{\min})x_2 + p_{\min}.$$[4]

If we define input $g(\mathbf{k}) = \hat{d}(f(k_1), h(k_2))$, output $u(\mathbf{x}) = (Rd)(\tau(x_1), p(x_2))$, and the phase function $\Phi(\mathbf{x}, \mathbf{k}) = f(k_1)\sqrt{\tau(x_1)^2 + p(x_2)^2 h(k_2)^2}$, then equation (3) can be written as

$$u(\mathbf{x}) = \sum_{\mathbf{k} \in K} e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})} g(\mathbf{k}), \quad \mathbf{x} \in X. \tag{4}$$

This form is the discrete version of an oscillatory integral of the type

$$u(\mathbf{x}) = \int_K e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})} g(\mathbf{k})\, d\mathbf{k}, \quad \mathbf{x} \in X,\text{[5]} \tag{5}$$

whose fast evaluation has been considered in Candès et al. (2009). Our method for computing the summation (4) follows the FIO butterfly algorithm introduced there.

Low-rank approximations

Clearly the range and gradient of phase $\Phi(\mathbf{x}, \mathbf{k})$ determine the degree of oscillations in the kernel $e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})}$. Let N be an integer power of two, which is on the order of the maximum value of $|\Phi(\mathbf{x}, \mathbf{k})|$ for $\mathbf{x} \in X$ and $\mathbf{k} \in K$ (the exact choice of N depends on the desired efficiency and accuracy of the algorithm, which will be made specific in numerical examples). The design of the fast algorithm relies on the key observation that the kernel $e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})}$, when properly restricted to subsets of X and K, admits accurate and low-rank separated approximations. More precisely, if A and B are two square boxes in X and K, with sidelengths $w(A)$, $w(B)$ obeying $w(A)w(B) \leq 1/N$ — in which case the pair (A, B) is

[4]The notations $\mathbf{x}$ and $\mathbf{k}$ are adopted to be consistent with the Candès et al. paper referenced below.

[5]Throughout the paper, K and X will either be used for sets of discrete points or square domains containing them. The meaning should be clear from the context.

called admissible — then

$$\left| e^{2\pi i \Phi(\mathbf{x},\mathbf{k})} - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(\mathbf{x}) \beta_t^{AB}(\mathbf{k}) \right| \leq \epsilon, \quad \text{for } \ \mathbf{x} \in A, \ \ \mathbf{k} \in B, \tag{6}$$

where r_ϵ is independent of N for a fixed error ϵ. Here (and in the sequel) the subscript t is slightly abused: t should be understood as multi-indices (t_1, t_2), and accordingly r_ϵ is the total number of terms in a double sum. Furthermore, Candès et al. (2009) show that this low-rank approximation can be constructed via a tensor-product Chebyshev interpolation of $e^{2\pi i \Phi(\mathbf{x},\mathbf{k})}$ in the $\mathbf{x}$ variable when $w(A) \leq 1/\sqrt{N}$, and in the $\mathbf{k}$ variable when $w(B) \leq 1/\sqrt{N}$.

Specifically, when $w(B) \leq 1/\sqrt{N}$, α_t^{AB} and β_t^{AB} are given by

$$\alpha_t^{AB}(\mathbf{x}) = e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_t^B)}, \tag{7}$$

$$\beta_t^{AB}(\mathbf{k}) = e^{-2\pi i \Phi(\mathbf{x_0}(A), \mathbf{k}_t^B)} L_t^B(\mathbf{k}) e^{2\pi i \Phi(\mathbf{x_0}(A), \mathbf{k})}; \tag{8}$$

and when $w(A) \leq 1/\sqrt{N}$, α_t^{AB} and β_t^{AB} are given by

$$\alpha_t^{AB}(\mathbf{x}) = e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_0(B))} L_t^A(\mathbf{x}) e^{-2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_0(B))}, \tag{9}$$

$$\beta_t^{AB}(\mathbf{k}) = e^{2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k})}. \tag{10}$$

The boldface letters $\mathbf{k}_t^B$, $\mathbf{x}_t^A, \mathbf{x}_0(A), \mathbf{k}_0(B)$ denote 2-D vectors. $\mathbf{k}_t^B$ is a point on the 2-D, $q_{k_1} \times q_{k_2}$ Chebyshev grid in box B centered at $\mathbf{k}_0(B)$: let $\mathbf{k}_t^B = (k_{t_1}^B, k_{t_2}^B)$, $\mathbf{k}_0(B) = (k_{0_1}^B, k_{0_2}^B)$, then

$$k_{t_1}^B = k_{0_1}^B + w(B) z_{t_1}, \quad 0 \leq t_1 \leq q_{k_1} - 1, \tag{11}$$

$$k_{t_2}^B = k_{0_2}^B + w(B) z_{t_2}, \quad 0 \leq t_2 \leq q_{k_2} - 1, \tag{12}$$

where

$$\left\{ z_{t_i} = \frac{1}{2} \cos\left(\frac{\pi t_i}{q_{k_i} - 1} \right) \right\}_{0 \leq t_i \leq q_{k_i} - 1, \ i=1,2} \tag{13}$$

is the 1-D Chebyshev grid of order q_{k_i} on $[-1/2, 1/2]$. See Figure 1 for an illustration. $L_t^B(\mathbf{k})$ is the 2-D Lagrange interpolation on this Chebyshev grid:

$$L_t^B(\mathbf{k}) = \left(\prod_{s_1=0, s_1 \neq t_1}^{q_{k_1}-1} \frac{k_1 - k_{s_1}^B}{k_{t_1}^B - k_{s_1}^B} \right) \left(\prod_{s_2=0, s_2 \neq t_2}^{q_{k_2}-1} \frac{k_2 - k_{s_2}^B}{k_{t_2}^B - k_{s_2}^B} \right). \tag{14}$$

Analogously, $\mathbf{x}_t^B$ is a point on the 2-D, $q_{x_1} \times q_{x_2}$ Chebyshev grid in box A centered at $\mathbf{x}_0(A)$, and $L_t^A(\mathbf{x})$ is the 2-D Lagrange interpolation defined on this grid. Based on the discussion above, the number r_ϵ of expansions in (6) is equal to $q_{k_1} q_{k_2}$ when $w(B) \leq 1/\sqrt{N}$, and $q_{x_1} q_{x_2}$ when $w(A) \leq 1/\sqrt{N}$.

A simple way of viewing expressions (7) – (10) is: when $w(B) \leq 1/\sqrt{N}$, plugging (7) into the approximation (6) (leaving $\beta_t^{AB}(\mathbf{k})$ as it is) yields

$$e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})} \approx \sum_t e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_t^B)} \beta_t^{AB}(\mathbf{k}), \quad \text{for } \mathbf{x} \in A, \quad \mathbf{k} \in B. \tag{15}$$

For fixed $\mathbf{x}$, the right hand side of (15) is just a special interpolation of function $e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})}$ in variable $\mathbf{k}$, where $\mathbf{k}_t^B$ are the interpolation points, $\beta_t^{AB}(\mathbf{k})$ are the basis functions. Likewise, when $w(A) \leq 1/\sqrt{N}$, plugging (10) into (6), we get

$$e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})} \approx \sum_t e^{2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k})} \alpha_t^{AB}(\mathbf{x}), \quad \text{for } \mathbf{x} \in A, \quad \mathbf{k} \in B. \tag{16}$$

For fixed $\mathbf{k}$, the right hand side of (16) is a special interpolation of $e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})}$ in variable $\mathbf{x}$: $\mathbf{x}_t^A$ are the interpolation points, $\alpha_t^{AB}(\mathbf{x})$ are the basis functions.

Once the expansion (6) is known, the partial sum

$$u^B(\mathbf{x}) := \sum_{\mathbf{k} \in B} e^{2\pi i \Phi(\mathbf{x}, \mathbf{k})} g(\mathbf{k}) \tag{17}$$

generated by points $\mathbf{k}$ inside a box B can be approximated by

$$u^B(\mathbf{x}) \approx \sum_{\mathbf{k} \in B} \sum_t \alpha_t^{AB}(\mathbf{x}) \beta_t^{AB}(\mathbf{k}) g(\mathbf{k}) = \sum_t \alpha_t^{AB}(\mathbf{x}) \delta_t^{AB}, \quad \text{for } \mathbf{x} \in A, \tag{18}$$

where

$$\delta_t^{AB} := \sum_{\mathbf{k}\in B} \beta_t^{AB}(\mathbf{k}) g(\mathbf{k}). \tag{19}$$

The case that the box B represents the whole domain K is of particular interest, since it corresponds to the original problem. Therefore, if we can find the set of interaction coefficients δ_t^{AB} relative to all admissible couples of boxes (A, B) with $B = K$, our problem will be solved.

Butterfly structure

The coefficients δ_t^{AB} for $B = K$ are however not readily available. The so-called butterfly algorithm turns out to be an appropriate tool. It was introduced by Michielssen and Boag (1996), and generalized by O'Neil et al. (2010); Candès et al. (2009). Different applications include Ying (2009); Demanet et al. (2012). The latter also provides a complete error analysis of the method in Candès et al. (2009).

The idea of the butterfly algorithm is to obtain δ_t^{AB} for $B = K$ at the last step of a hierarchical construction of *all* the coefficients δ_t^{AB} for *all* pairs of admissible boxes (A, B) belonging to a quad tree structure. The algorithm starts with very small boxes B, where δ_t^{AB} are easily computed by direct summation, and gradually increases the sizes of the boxes B in a multiscale fashion. In tandem, the sizes of the boxes A where u^B is evaluated must decrease to respect the admissibility of each couple (A, B). The computation then mostly consists in updating coefficients δ_t^{AB} from one scale to the next — from finer to coarser B boxes, and from coarser to finer A boxes.

The main data structure underlying the algorithm is a pair of quad trees T_X and T_K. The tree T_X has $[0, 1] \times [0, 1]$ as its root box (level 0) and is built by recursive, dyadic partitioning until level $L = \log N$, where the finest boxes are of sidelength $1/N$. The tree T_K is built similarly but in the opposite direction. Figure 2 shows such a partition for $N = 4$. A crucial property of this structure is that at arbitrary level l, the sidelengths of a box A in T_X and a box B in T_K always satisfy

$$w(A)w(B) = \frac{1}{2^l}\frac{1}{2^{L-l}} = \frac{1}{N}.$$

Thus a low-rank approximation of the kernel $e^{2\pi i\Phi(\mathbf{x},\mathbf{k})}$ is available at every level of the tree, for every couple of admissible boxes (A, B).

Fast butterfly algorithm

With the previously introduced low-rank approximations and the butterfly structure, we are ready to describe the fast algorithm. Our goal is to approximate δ_t^{AB} (19) so as to get $u^B(\mathbf{x})$ (18) by traversing the tree structure (Figure 2) from top to bottom on the X side, and from bottom to top on the K side. This can be done in five major steps. To avoid too much technical detail, we deliberately defer the complete derivation of the algorithm until the appendix, and only summarize here the final updating formulas for each step.

1. *Initialization.* At level $l = 0$, let A be the root box of T_X. For each leaf box $B \in T_K$, construct the coefficients $\{\delta_t^{AB}\}$ by

$$\delta_t^{AB} = e^{-2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_t^B)} \sum_{\mathbf{k}\in B} \left(L_t^B(\mathbf{k}) e^{2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k})} g(\mathbf{k}) \right). \tag{20}$$

2. *Recursion.* At $l = 1, 2, ..., L/2$, for each pair (A, B), let A_p be A's parent and $\{B_c, c = 1, 2, 3, 4\}$ be B's children from the previous level (see also Figure A-1). Update $\{\delta_t^{AB}\}$ from $\{\delta_{t'}^{A_pB_c}\}$ by

$$\delta_t^{AB} = e^{-2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_t^B)} \sum_c \sum_{t'} \left(L_t^B(\mathbf{k}_{t'}^{B_c}) e^{2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_{t'}^{B_c})} \delta_{t'}^{A_pB_c} \right). \tag{21}$$

3. *Switch.* At middle level $l = L/2$, for each (A, B) compute the new set of coefficients $\{\delta_t^{AB}\}$ from the old set $\{\delta_s^{AB}\}$ by

$$\delta_t^{AB} = \sum_s e^{2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_s^B)} \delta_s^{AB}. \tag{22}$$

4. *Recursion.* At $l = L/2+1, ..., L$, for each pair (A, B), update $\{\delta_t^{AB}\}$ from $\{\delta_{t'}^{A_p B_c}\}$ of the previous level by

$$\delta_t^{AB} = \sum_c e^{2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_0(B_c))} \sum_{t'} \left(L_{t'}^{A_p}(\mathbf{x}_t^A) e^{-2\pi i \Phi(\mathbf{x}_{t'}^{A_p}, \mathbf{k}_0(B_c))} \delta_{t'}^{A_p B_c} \right). \tag{23}$$

5. *Termination.* Finally at level $l = L$, B is the entire domain K. For every box A in X and every $\mathbf{x} \in A$, compute $u(\mathbf{x})$ by

$$u(\mathbf{x}) = e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_0(B))} \sum_t \left(L_t^A(\mathbf{x}) e^{-2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_0(B))} \delta_t^{AB} \right). \tag{24}$$

Discussion

To analyze the algorithm's numerical complexity, let us assume the numbers of Chebyshev points in every box and every dimension of K and X are all equal to a small constant q, i.e., $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = q$ and $r_\epsilon \equiv q^2$. The main workload of the fast butterfly algorithm is in Steps 2 and 4. For each level, there are N^2 pairs of boxes (A, B), and the operations between each A and B is $O(r_\epsilon^2)$, which can be further reduced to $O(r_\epsilon^{3/2})$ by performing Chebyshev interpolation one dimension at a time. Since there are $\log N$ levels, the total cost is $O(r_\epsilon^{3/2} N^2 \log N)$. It is not difficult to see that Step 3 takes $O(r_\epsilon^2 N^2)$, and Steps 1 and 5 take $O(r_\epsilon N_f N_h)$ and $O(r_\epsilon N_\tau N_p)$ operations. Considering the initial Fourier transform of preparing data in the (f, h) domain, we conclude that the overall complexity of the algorithm is $O(N_h N_t \log N_t + r_\epsilon^{3/2} N^2 \log N + r_\epsilon^2 N^2 + r_\epsilon(N_f N_h + N_\tau N_p))$. The analysis in Candès et al. (2009) shows that the relation between r_ϵ and error ϵ is $r_\epsilon = O(\log^4(1/\epsilon))$.

We would like to mention that this is only the worst case estimate. Numerical results in the same paper demonstrate that the dependence of r_ϵ on $\log(1/\epsilon)$ is rather moderate in practice.

In comparison, the conventional *velocity scan* requires at least $O(N_\tau N_p N_h)$ computations, which quickly becomes a burden as the problem size increases. Yet the efficiency of our algorithm is mainly controlled by $O(N^2 \log N)$ with a constant polylogarithmic in ϵ, where N is determined by the degree of oscillations in the kernel function $e^{2\pi i f\sqrt{\tau^2+p^2h^2}}$, i.e., roughly the range of f, h, τ, and p. In other words, N depends neither on data size nor on data content (here we mean the data after the Fourier transform), as the Chebyshev interpolation is only performed on the kernel.

NUMERICAL EXAMPLES

In this section we provide several numerical examples to illustrate the empirical properties of the fast butterfly algorithm. To check the results *qualitatively*, we compare with the *velocity scan* method (a piecewise constant interpolation is used to minimize the interpolation cost); to test the results *quantitatively*, however, it makes more sense to compare with the direct evaluation of equation (3), since the fast algorithm is to speed up this summation in the frequency domain, whereas the *velocity scan* computes a slightly different sum in the time domain, which may contain interpolation artifacts.

There is no general rule for selecting parameters N, q_{k_1}, q_{k_2}, ... The larger N is, the fewer Chebyshev points are needed, and vice versa. In practice, parameters can be tuned to achieve the best efficiency and accuracy trade-off. For simplicity, in the following examples N and q_{k_1}, q_{k_2}, q_{x_1}, q_{x_2} are chosen such that the relative error between the fast algorithm and the direct computation of (3) is about $O(10^{-2})$. These combinations are not necessarily optimal in terms of efficiency.

Synthetic data — square sampling

We start with a simple 2-D example of square sampling. Figure 3 is a synthetic CMP gather sampled on $N_t = N_h = 1000$. Figure 4 shows the absolute value of its Fourier transform on time axis. These band-limited data allow us to shorten the computational range for f, which can be crucial as N depends on this range. In model space, the sampling sizes are chosen as $N_\tau = N_p = 1000$. Figure 5 is the output of the fast butterfly algorithm for $N = 32$, $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$ (here the range of $\Phi = f\sqrt{\tau^2 + p^2h^2}$ is about 125). Figure 6 is the output of the *velocity scan*. The two methods yield nearly the same results. The fast algorithm runs in only 1.75 seconds of CPU time, while the *velocity scan* takes about 37 s. In Figure 7, we plot the difference between the results of the fast algorithm and the direct evaluation of (3), where the relative error is 0.0178. For reference, if we let $N = 64$ and run the same test, the error decreases to $O(10^{-3})$ and the running time is 3.63 s.

Synthetic data — rectangular sampling

We now make two synthetic datasets using rectangular sampling $N_t = 4000$, $N_h = 400$. The first one (Figure 8) has the same range as the previous example (Figure 3), while the second one (Figure 9) doubles the range of time and offset. The results of the fast algorithm are shown in Figures 10 and 11. The purpose of showing these two examples is to demonstrate that the choice of N does not depend on the problem size, but rather on the range of parameters — for the data in Figure 9, one has to increase N to preserve the same accuracy (the range of $\Phi = f\sqrt{\tau^2 + p^2h^2}$ is about 125 for the first dataset, and 250 for the second one).

Synthetic data — irregular sampling

Going back to the five steps of the butterfly algorithm, it is clear that the input data $g(\mathbf{k})$ is only involved at the very first step. Besides, for every (A, B) the operation connecting $g(\mathbf{k})$ and δ_t^{AB} amounts to a matrix-vector multiplication (see (20)), which does not at all require the input data to be uniformly distributed (the same argument applies to the output data $u(\mathbf{x})$). Therefore, our algorithm can be easily extended to handle the following problem:

$$(Rd)(\tau, p) = \iint d(\sqrt{\tau^2 + p^2(h_1^2 + h_2^2)}, h_1, h_2)\, dh_1\, dh_2, \tag{25}$$

where $d(t, h_1, h_2)$ is a 3-D function. All we need is to introduce a new variable for the absolute offset $h = \sqrt{h_1^2 + h_2^2}$, and reorder the values $d(t, h_1, h_2)$ according to h. Figure 12 shows such synthetic data sampled on $N_t = 1000$, $N_{h_1} = N_{h_2} = 128$. The output is obtained on $N_\tau = 1000$, $N_p = 128$. The fast algorithm (Figure 13) runs in only 1.67 s for $N = 64$, $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 5$ (here the range of $\Phi = f\sqrt{\tau^2 + p^2(h_1^2 + h_2^2)}$ is about 162), while the *velocity scan* (Figure 14) takes more than 125 s.

Field data

We now consider a 2-D field seismic gather in Figure 15. Its Fourier transform is shown in Figure 16. Due to the comparatively wide frequency bandwidth, N cannot be chosen too small (here the range of $\Phi = f\sqrt{\tau^2 + p^2h^2}$ is about 306). The input sampling sizes are $N_t = 1500$, $N_h = 240$, while the output sizes are chosen as $N_\tau = 1500$, $N_p = 800$. Although this small dataset is not very suitable for showcasing the fast algorithm, our method runs in 6.62 s for $N = 128$, $q_{k_1} = q_{x_1} = 7, q_{k_2} = q_{x_2} = 5$ (Figure 17), still outperforming the *velocity scan* which takes about 10 s (Figure 18). Note that the simplest interpolation is used in the *velocity scan*, any other higher order interpolation will presumably take longer computation time.

Computing the adjoint operator

The last example is concerned with the computation of the adjoint of the hyperbolic Radon transform. Assuming $m(\tau, p)$ and $d(t, h)$ are two arbitrary functions (in the discrete sense) in the model domain and data domain, if we require

$$\langle m(\tau, p), (Rd)(\tau, p)\rangle = \langle (R^* m)(t, h), d(t, h)\rangle, \tag{26}$$

where $(Rd)(\tau, p)$ is given by equation (3); the inner product $\langle \cdot, \cdot \rangle$ is defined as

$$\langle g_1(x, y), g_2(x, y)\rangle = \sum_{x,y} g_1(x, y)\, \overline{g_2(x, y)}, \quad \forall g_1(x, y),\ g_2(x, y), \tag{27}$$

then it is easy to verify that the adjoint operator R^* is given by

$$(R^* m)(t, h) = \mathcal{F}^{-1}_{f\to t}\left(\sum_{\tau,p} e^{-2\pi i f\sqrt{\tau^2+p^2h^2}} m(\tau, p)\right), \tag{28}$$

where $\mathcal{F}^{-1}_{f\to t}$ is the inverse Fourier transform from variable f to t. The summation in equation (28) again resembles an oscillatory integral operator, therefore the fast algorithm for computing R applies with minor modifications. The computational cost remains the same.

We consider still the first example and apply the (discrete) adjoint operators of the fast butterfly algorithm and the *velocity scan* respectively to the data in Figures 5 and 6. The two methods produce very similar results (see Figures 19, 20). It is also clear that the adjoint is far from the inverse, at least for this geometry, hence some kind of least-squares implementation is needed for inversion process.

To further verify that the numerically computed R^* is the adjoint operator of R, one can compare the values of $\langle Rd, Rd\rangle$ and $\langle R^* Rd, d\rangle$ for arbitrary d. Indeed, the proposed algorithm passed this dot-product test with a relative error of $O(10^{-7})$ in single precision.

CONCLUSIONS

We constructed a fast butterfly algorithm for the hyperbolic Radon transform, a class of time-variant Radon transforms. Compared with the time-consuming integration in the time domain, the new method runs in only $O(N^2 \log N)$ operations, where N depends on the range of frequency and offset in the dataset and the range of intercept time and slowness in the model space, and can often be chosen smaller than the grid size. Our ongoing work is studying the performance of this fast solver on the sparse iterative inversion of the hyperbolic Radon transform applied to multiple attenuation.

Due to the generality of the butterfly algorithm, its application is obviously not limited to the hyperbolic transform considered here. Using a different phase function, one can easily extend the algorithm to higher-order transforms. If the slowness or velocity range is not constant but a corridor around a central function, then a sparse butterfly algorithm can be defined to save the cost by building the quad tree adaptively (Ying, 2009). Furthermore, many of the Radon-like integral operators, such as Kirchhoff migration, the apex-shifted Radon transform, the multi-parameter velocity analysis, etc., can be reformulated in a similar fashion as we did in this paper. To address these problems, a 3-D version of the butterfly algorithm might be more appropriate.

ACKNOWLEDGMENTS

We are grateful to Tariq Alkhalifah, Anatoly Baumstein, Ian Moore, Daniel Trad, and the anonymous reviewer for their valuable comments and suggestions. We thank Dr. Alexander Klokov for preprocessing the field data. We thank KAUST and sponsors of the Texas Consortium for Computational Seismology (TCCS) for financial support.

APPENDIX A

THE MATHEMATICAL DERIVATION OF THE FAST BUTTERFLY ALGORITHM

This appendix gives a complete derivation and description of the algorithm, which combines the low-rank approximations and the butterfly structure introduced in the main text. For more mathematical exposition, the reader is referred to Candès et al. (2009).

To facilitate the presentation, we add a new figure (Figure A-1) to illustrate the notations.

1. *Initialization.* At level $l = 0$, let A be the root box of T_X. For each leaf box $B \in T_K$, expressions (7) and (8) are valid as $w(B) \leq 1/\sqrt{N}$. Substituting β_t^{AB} (8) into the definition (19) of δ_t^{AB}, we get

$$\delta_t^{AB} = e^{-2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_t^B)} \sum_{\mathbf{k}\in B} \left(L_t^B(\mathbf{k}) e^{2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k})} g(\mathbf{k}) \right), \tag{A-1}$$

i.e. the equation (20) in the main text. In addition, for $\mathbf{x} \in A$, the partial sum $u^B(\mathbf{x})$ (18) is given by (with α_t^{AB} (7) plugged in)

$$u^B(\mathbf{x}) \approx \sum_t e^{2\pi i\Phi(\mathbf{x},\mathbf{k}_t^B)} \delta_t^{AB}. \tag{A-2}$$

Comparing the right hand sides of (17) and (A-2), if we call $g(\mathbf{k})$ the sources at $\mathbf{k}$, then the coefficients δ_t^{AB} are just like the *equivalent sources* at $\mathbf{k}_t^B$. This initial step is to redistribute the original sources $g(\mathbf{k})$ located at $\mathbf{k}$ (denoted by blue dots in Figure A-1) to equivalent sources δ_t^{AB} located at Chebyshev grid $\mathbf{k}_t^B$ (not shown in the figure). We next aim at updating δ_t^{AB} until the end level L.

2. *Recursion.* At $l = 1, 2, ..., L/2$, for each pair (A, B), let A_p be A's parent and $\{B_c, c = 1, 2, 3, 4\}$ be B's children from the previous level (see Figure A-1). For each child B_c, we have available from the previous level an approximation of the form

$$u^{B_c}(\mathbf{x}) \approx \sum_{t'} e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_{t'}^{B_c})} \delta_{t'}^{A_p B_c}, \quad \text{for } \ \mathbf{x} \in A_p. \tag{A-3}$$

Summing over all children gives

$$u^{B}(\mathbf{x}) \approx \sum_{c} \sum_{t'} e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_{t'}^{B_c})} \delta_{t'}^{A_p B_c}, \quad \text{for } \ \mathbf{x} \in A_p. \tag{A-4}$$

Since $A \subset A_p$, this is of course true for any $\mathbf{x} \in A$. Also we know that equation (15) holds for $\mathbf{k}_{t'}^{B_c} \in B$, i.e.,

$$e^{2\pi \Phi(\mathbf{x}, \mathbf{k}_{t'}^{B_c})} \approx \sum_{t} e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_{t}^{B})} \beta_t^{AB}(\mathbf{k}_{t'}^{B_c}), \quad \text{for } \ \mathbf{x} \in A. \tag{A-5}$$

Plugging it into (A-4) yields

$$u^{B}(\mathbf{x}) \approx \sum_{c} \sum_{t'} \sum_{t} e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_{t}^{B})} \beta_t^{AB}(\mathbf{k}_{t'}^{B_c}) \delta_{t'}^{A_p B_c}, \quad \text{for } \ \mathbf{x} \in A. \tag{A-6}$$

On the other hand, $u^B(\mathbf{x})$ admits a low-rank approximation of equivalent sources at the current level,

$$u^{B}(\mathbf{x}) \approx \sum_{t} e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_{t}^{B})} \delta_t^{AB}, \quad \text{for } \ \mathbf{x} \in A. \tag{A-7}$$

Equating (A-6) and (A-7) suggests that we can take

$$\delta_t^{AB} = \sum_{c} \sum_{t'} \beta_t^{AB}(\mathbf{k}_{t'}^{B_c}) \delta_{t'}^{A_p B_c}. \tag{A-8}$$

Substituting β_t^{AB} (8), we get

$$\delta_t^{AB} = e^{-2\pi i \Phi(\mathbf{x}_0(A), \mathbf{k}_t^B)} \sum_c \sum_{t'} \left(L_t^B(\mathbf{k}_{t'}^{B_c}) e^{2\pi i \Phi(\mathbf{x}_0(A), \mathbf{k}_{t'}^{B_c})} \delta_{t'}^{A_p B_c} \right), \tag{A-9}$$

i.e. the equation (21) in the main text.

3. *Switch.* A switch of the representation to expressions (9) and (10) is needed at the middle level $l = L/2$ since expressions (7) and (8) are no longer valid as soon as $l > L/2$ (boxes B are getting bigger and bigger so that $w(B) \leq 1/\sqrt{N}$ is no longer satisfied). Plugging (10) into the definition (19) of δ_t^{AB}, one has

$$\delta_t^{AB} = \sum_{\mathbf{k} \in B} e^{2\pi \Phi(\mathbf{x}_t^A, \mathbf{k})} g(\mathbf{k}) = u^B(\mathbf{x}_t^A). \tag{A-10}$$

From (A-7),

$$u^B(\mathbf{x}_t^A) \approx \sum_s e^{2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_s^B)} \delta_s^{AB}, \tag{A-11}$$

where we use $\{\delta_t^{AB}\}$ to denote the new set of coefficients and $\{\delta_s^{AB}\}$ the old set. Equating (A-10) and (A-11), we can set δ_t^{AB} as

$$\delta_t^{AB} = \sum_s e^{2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_s^B)} \delta_s^{AB}, \tag{A-12}$$

i.e. the equation (22) in the main text. This middle step is to switch from equivalent sources δ_s^{AB} located at Chebyshev grid $\mathbf{k}_s^B$ on the K side to equivalent sources δ_t^{AB} at Chebyshev grid $\mathbf{x}_t^A$ on the X side.

4. *Recursion.* The rest of the recursion is analogous to Step 2. For $l = L/2 + 1, ..., L$, we have

$$u^B(\mathbf{x}) \approx \sum_c \sum_{t'} \alpha_{t'}^{A_p B_c}(\mathbf{x}) \delta_{t'}^{A_p B_c}, \quad \text{for } \mathbf{x} \in A_p, \tag{A-13}$$

thus

$$u^B(\mathbf{x}_t^A) \approx \sum_c \sum_{t'} \alpha_{t'}^{A_p B_c}(\mathbf{x}_t^A) \delta_{t'}^{A_p B_c}; \tag{A-14}$$

recalling (A-10), one can simply set

$$\delta_t^{AB} = \sum_c \sum_{t'} \alpha_{t'}^{A_p B_c}(\mathbf{x}_t^A) \delta_{t'}^{A_p B_c}. \tag{A-15}$$

Substituting α_t^{AB} (9) gives the update

$$\delta_t^{AB} = \sum_c e^{2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_0(B_c))} \sum_{t'} \left(L_{t'}^{A_p}(\mathbf{x}_t^A) e^{-2\pi i \Phi(\mathbf{x}_{t'}^{A_p}, \mathbf{k}_0(B_c))} \delta_{t'}^{A_p B_c} \right), \tag{A-16}$$

i.e. the equation (23) in the main text.

5. *Termination.* Finally we reach the level $l = L$, and B is the entire domain K. For every box A in X and every $\mathbf{x} \in A$,

$$u(\mathbf{x}) = u^B(\mathbf{x}) \approx \sum_t \alpha_t^{AB}(\mathbf{x}) \delta_t^{AB}. \tag{A-17}$$

Plugging in α_t^{AB} (9), we get

$$u(\mathbf{x}) = e^{2\pi i \Phi(\mathbf{x}, \mathbf{k}_0(B))} \sum_t \left(L_t^A(\mathbf{x}) e^{-2\pi i \Phi(\mathbf{x}_t^A, \mathbf{k}_0(B))} \delta_t^{AB} \right), \tag{A-18}$$

i.e. the equation (24) in the main text. This final step is to transform the equivalent sources δ_t^{AB} at Chebyshev grid $\mathbf{x}_t^A$ back to the targets $u(\mathbf{x})$ located at $\mathbf{x}$ (denoted by red dots in Figure A-1).

In the above algorithm, $L = \log N$ is assumed to be an even number. If it is odd, one can either switch at level $(L-1)/2$ or $(L+1)/2$. Everything else remains unchanged.

REFERENCES

Beylkin, G., 1984, The inversion problem and applications of the generalized Radon transform: Commun. Pure Appl. Math., **37**, no. 5, 579–599.

———, 1985, Imaging of discontinuities in the inverse scattering problem by inversion of a causal generalized Radon transform: J. Math. Phys., **26**, no. 1, 99.

Candès, E., L. Demanet, and L. Ying, 2009, A fast butterfly algorithm for the computation of Fourier integral operators: Multiscale Model. Simul., **7**, 1727–1750.

Demanet, L., M. Ferrara, N. Maxwell, J. Poulson, and L. Ying, 2012, A butterfly algorithm for synthetic aperture radar imaging: SIAM J. Imaging Sciences, **5**, 203–243.

Foster, D. J., and C. C. Mosher, 1992, Suppression of multiple reflections using the Radon transform: Geophysics, **57**, 386–395.

Gardner, G. H. F., and L. Lu, eds., 1991, Slant-stack processing: Society of Exploration Geophysicists. Issue 14 of Geophysics reprint series.

Hampson, D., 1986, Inverse velocity stacking for multiple elimination: J. Can. Soc. Expl. Geophys., **22**, 44–55.

Hargreaves, N., B. verWest, R. Wombell, and D. Trad, 2003, Multiple attenuation using an apex-shifted Radon transform: EAGE 65th Conference and Exhibition, June 2 -5, 2003, Stavanger, Norway.

Herrmann, P., T. Mojesky, M. Magesan, and P. Hugonnet, 2000, De-aliased, high-resolution Radon transforms: SEG Annual Meeting, August 6 - 11, 2000, Calgary, Alberta.

Liu, Y., and M. Sacchi, 2002, De-multiple via a fast least squares hyperbolic Radon transform: SEG Annual Meeting, October 6 - 11, 2002, Salt Lake City, Utah.

Michielssen, E., and A. Boag, 1996, A multilevel matrix decomposition algorithm for analyzing scattering from large structures: IEEE Trans. Antennas and Propagation, **44**, 1086–1093.

Moore, I., and C. Kostov, 2002, Stable, efficient, high-resolution Radon transforms: EAGE 64th Conference and Exhibition, May 27 - 30, 2002, Florence, Italy.

O'Neil, M., F. Woolfe, and V. Rokhlin, 2010, An algorithm for the rapid evaluation of special function transforms: Appl. Comput. Harmon. Anal., **28**, 203–226.

Radon, J., 1917, Über die bestimmung von funktionen durch ihre integralwerte längs gewisser mannigfaltigkeiten: Berichte über die Verhandlungen der Sächsische Akademie der Wissenschaften (Reports on the proceedings of the Saxony Academy of Science), **69**, 262–277.

Sacchi, M., 1996, A bidiagonalization procedure for the inversion of time-variant velocity stack operator: CDSST report, 73–92.

Thorson, J. R., and J. F. Claerbout, 1985, Velocity-stack and slant-stack stochastic inversion: Geophysics, **50**, 2727–2741.

Trad, D., 2003, Interpolation and multiple attenuation with migration operators: Geophysics, **68**, 2043–2054.

Trad, D., T. Ulrych, and M. Sacchi, 2002, Accurate interpolation with high resolution time-variant Radon transforms: Geophysics, **67**, 644–656.

Wang, J., and M. Ng, 2009, Greedy least-squares and its application in Radon transforms: 2009 CSPG CSEG CWLS Convention, Calgary, Alberta, Canada.

Yilmaz, O., 1989, Velocity-stack processing: Geophysical Prospecting, **37**, 357–382.

Ying, L., 2009, Sparse Fourier transform via butterfly algorithm: SIAM J. Sci. Comput., **31**, 1678–1694.

LIST OF FIGURES

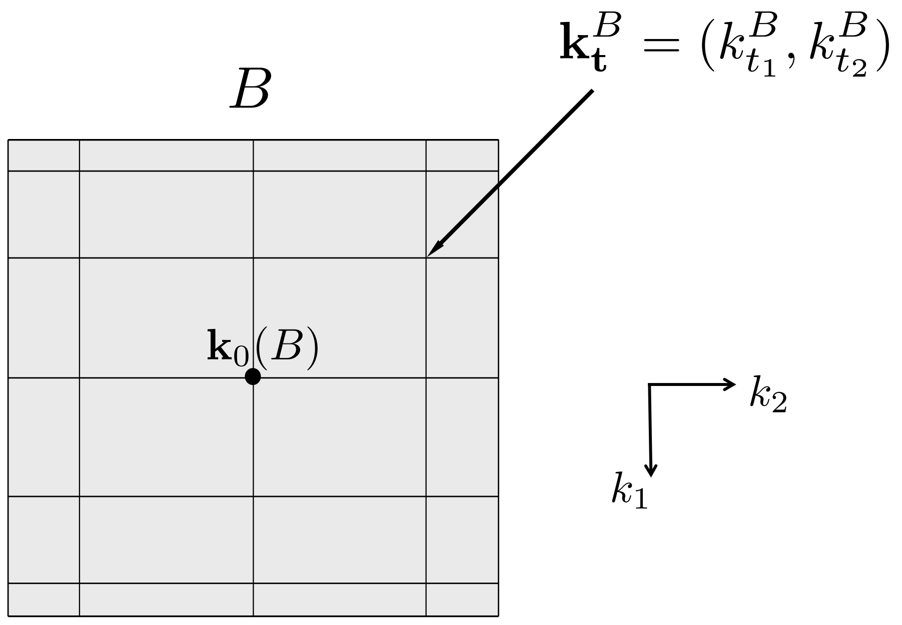

Figure 1: A 2-D, $q_{k_1} \times q_{k_2}$ ($q_{k_1} = 7$, $q_{k_2} = 5$) Chebyshev grid in box B. $\mathbf{k}_0(B)$ is the center of the box. $\mathbf{k}_t^B = (k_{t_1}^B, k_{t_2}^B),\ 0 \leq t_1 \leq q_{k_1} - 1,\ 0 \leq t_2 \leq q_{k_2} - 1$ is a point on the grid.

– **GEO-2013-**

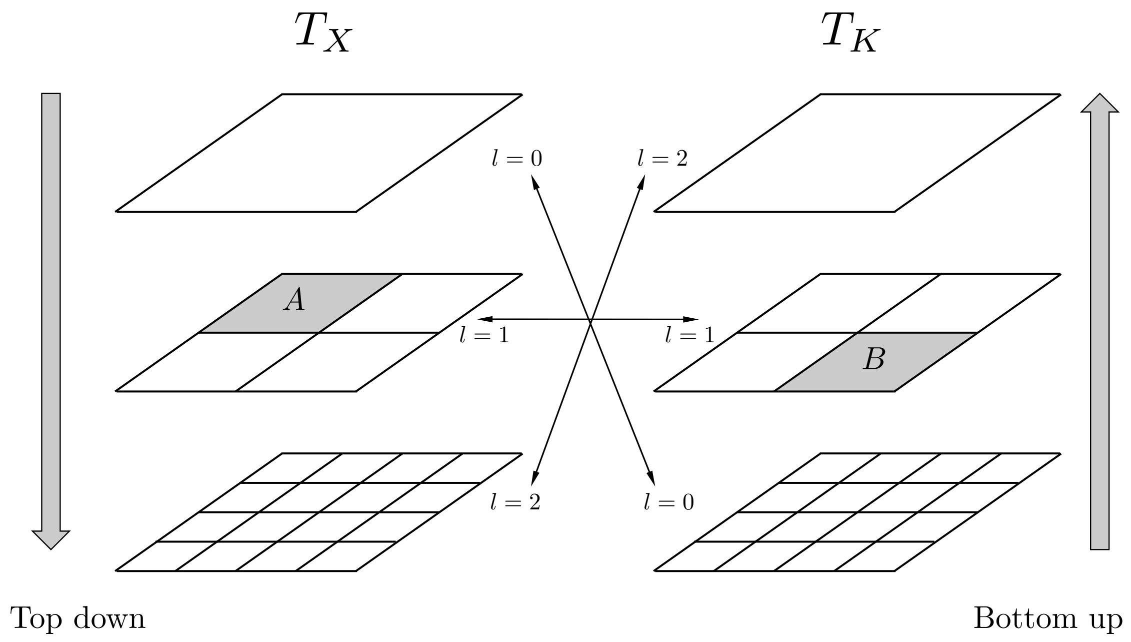

Figure 2: The butterfly quad tree structure for the special case of $N = 4$. – **GEO-2013-**

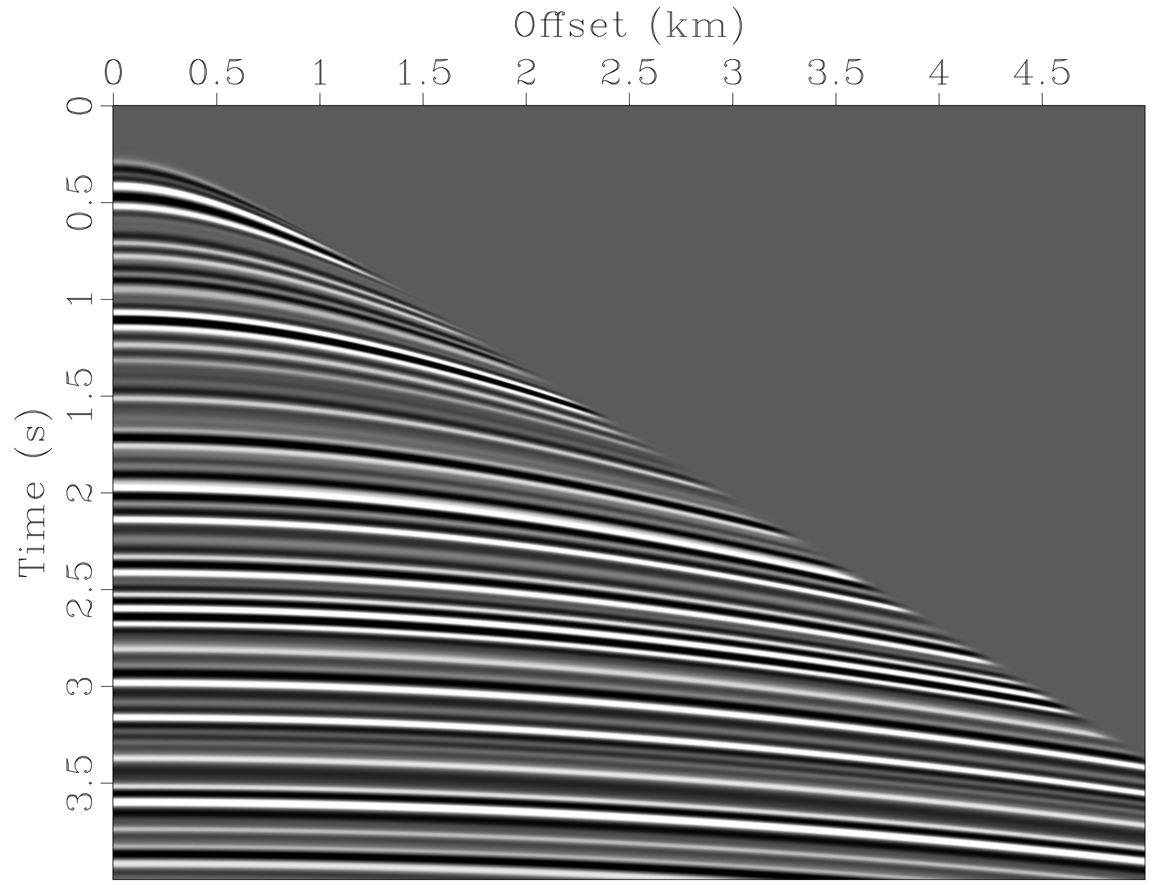

Figure 3: 2-D synthetic CMP gather. $N_t = N_h = 1000$. $\Delta t = 0.004$ s, $\Delta h = 0.005$ km.

– GEO-2013-

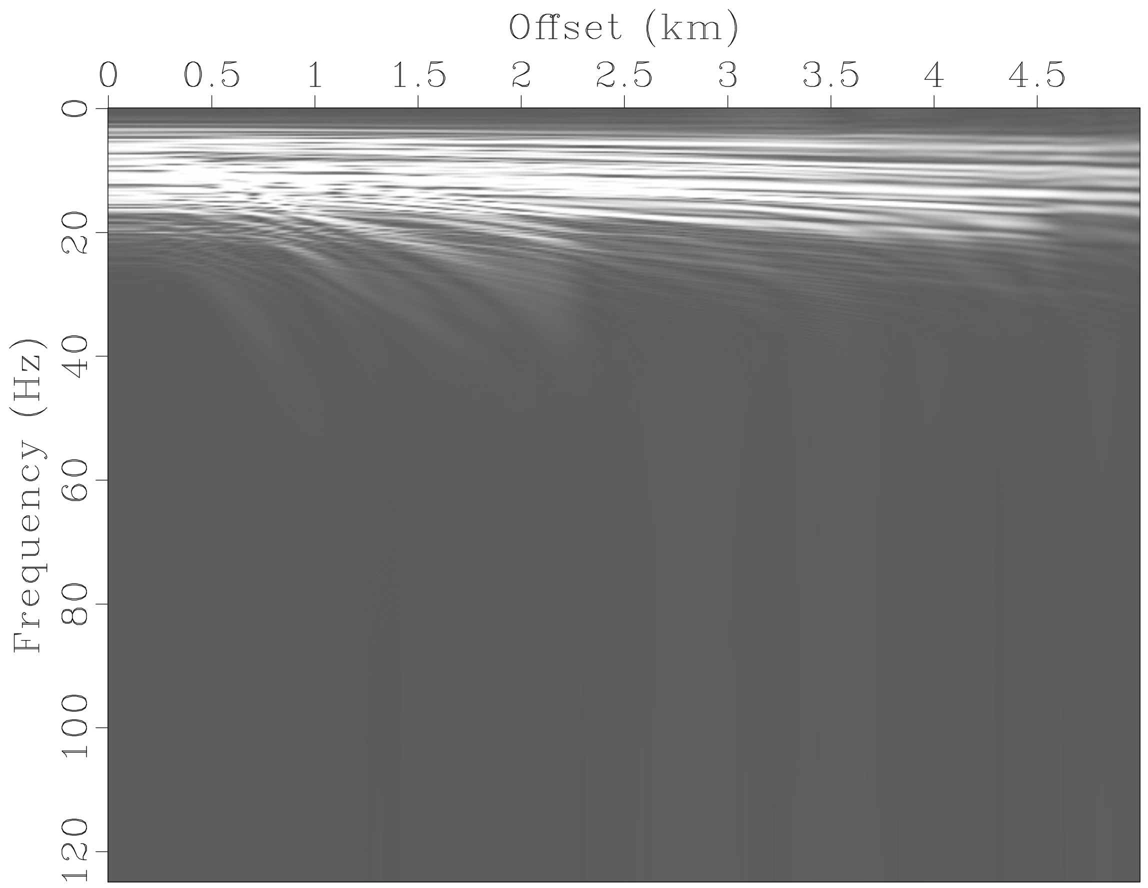

Figure 4: The Fourier transform (absolute value) on time axis of the synthetic data in Figure 3.

– GEO-2013-

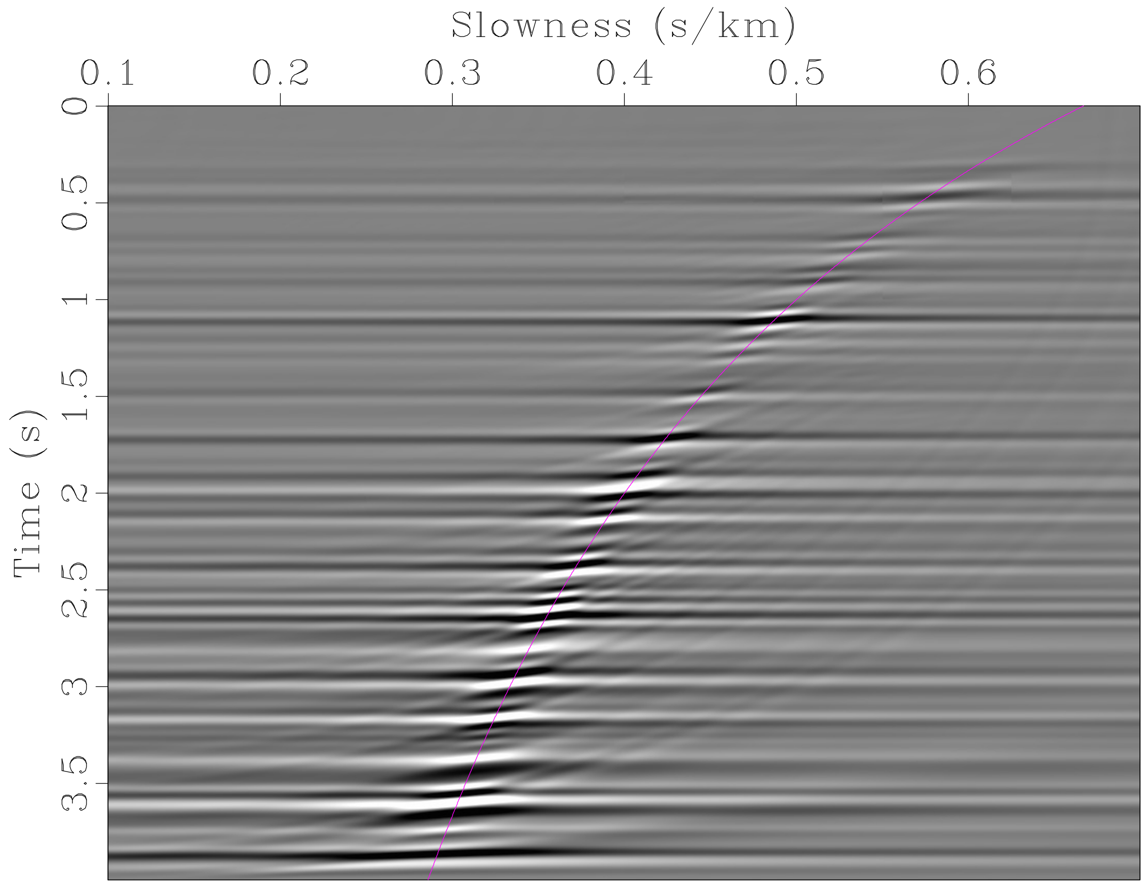

Figure 5: $N_\tau = N_p = 1000$. Output of the fast butterfly algorithm applied to the synthetic data in Figure 3. $N = 32$, $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$ (here the range of $\Phi = f\sqrt{\tau^2 + p^2h^2}$ is about 125). CPU time: **1.75 s**. Purple curve overlaid is the true slowness.

– **GEO-2013-**

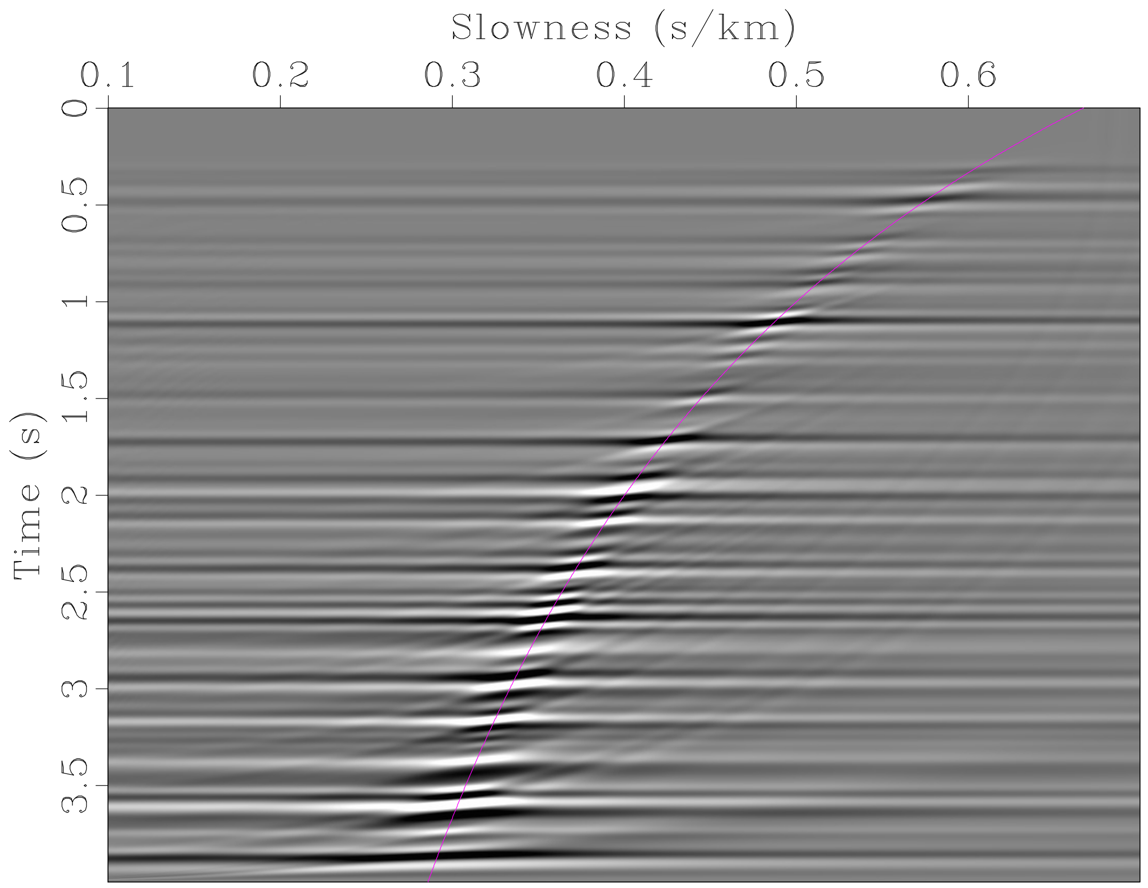

Figure 6: $N_\tau = N_p = 1000$. Output of the *velocity scan* applied to the synthetic data in Figure 3. CPU time: **37.23 s**. Purple curve overlaid is the true slowness.

– **GEO-2013-**

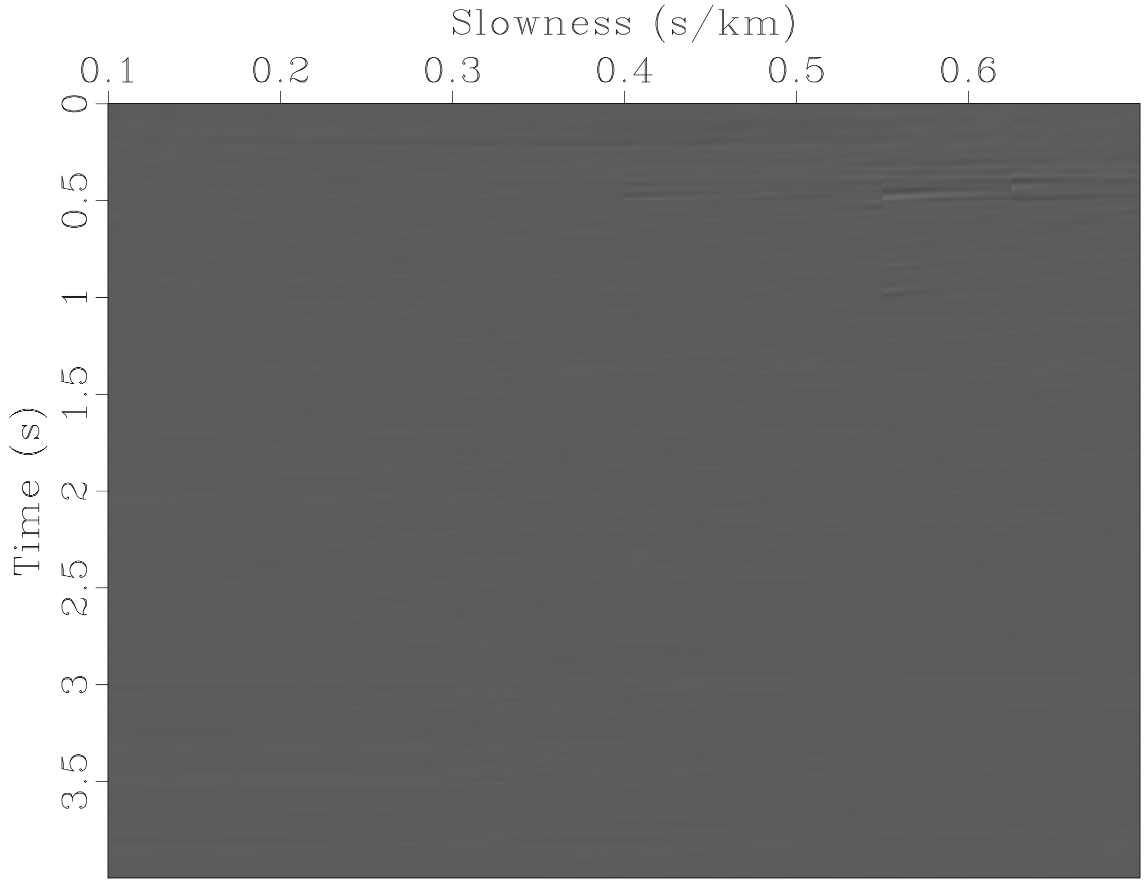

Figure 7: Difference between the results of the fast algorithm and the direct evaluation of equation (3) plotted at the same scale as in Figure 5.

– **GEO-2013-**

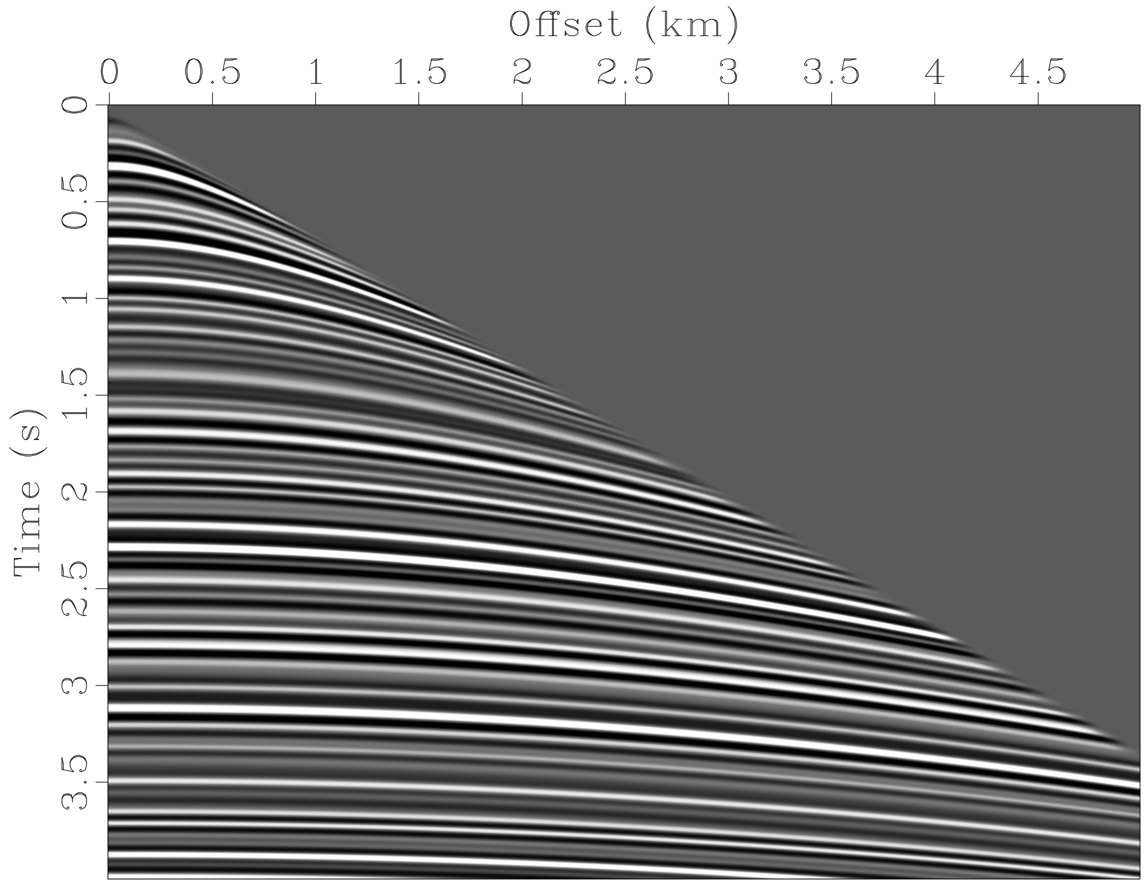

Figure 8: 2-D synthetic CMP gather. $N_t = 4000$, $N_h = 400$. $\Delta t = 0.001$ s, $\Delta h = 0.0125$ km.

– **GEO-2013-**

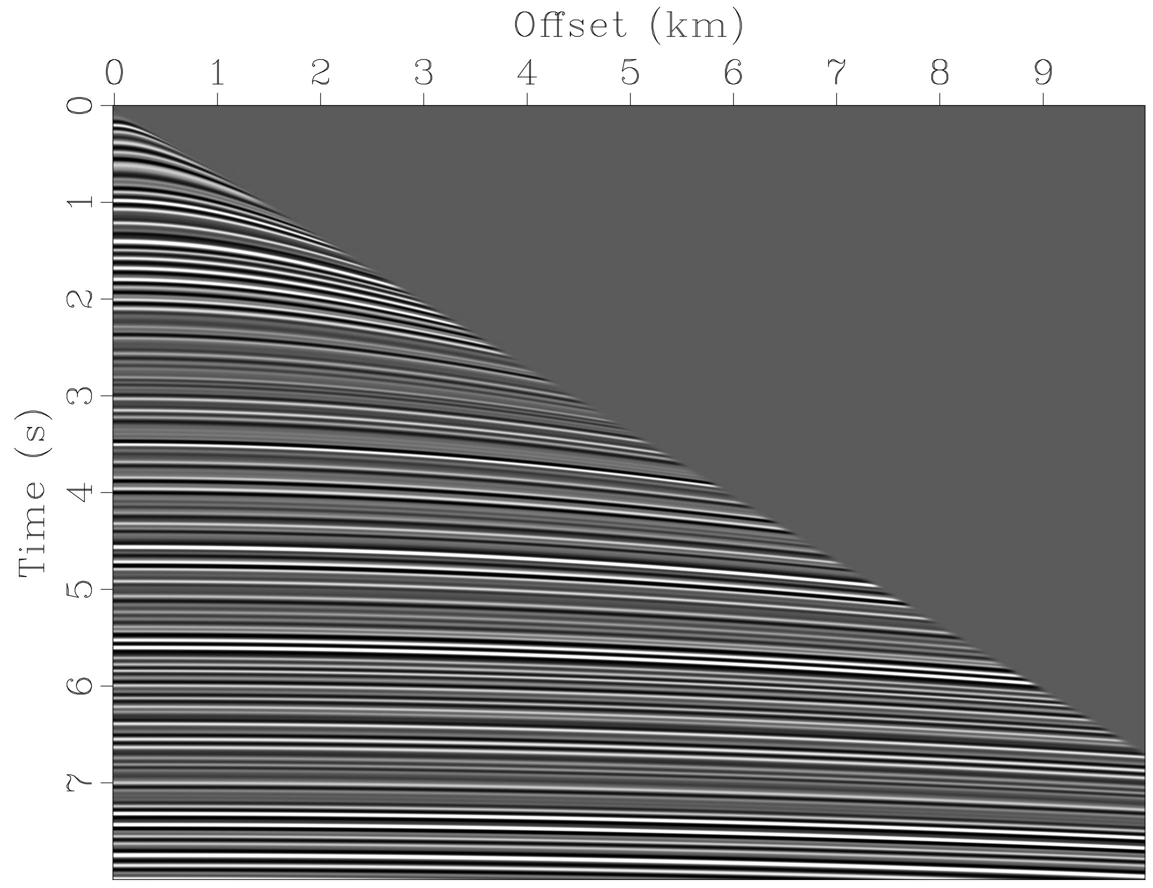

Figure 9: 2-D synthetic CMP gather. $N_t = 4000$, $N_h = 400$. $\Delta t = 0.002$ s, $\Delta h = 0.025$ km.

– GEO-2013-

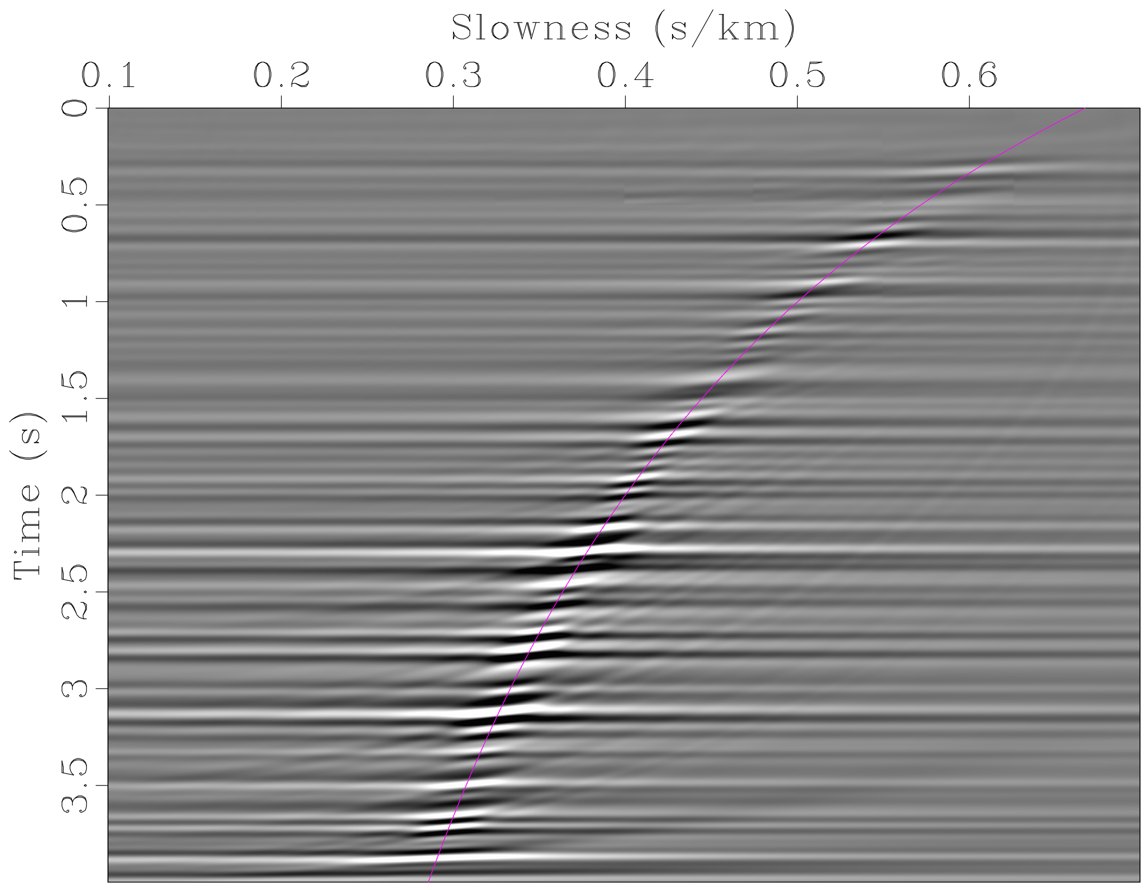

Figure 10: $N_\tau = 4000$, $N_p = 400$. Output of the fast butterfly algorithm applied to the synthetic data in Figure 8. $N = 32$, $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$ (here the range of $\Phi = f\sqrt{\tau^2 + p^2h^2}$ is about 125). CPU time: **2.46 s**. Ref: CPU time of *velocity scan*: **21.84 s**. Purple curve overlaid is the true slowness.

– GEO-2013-

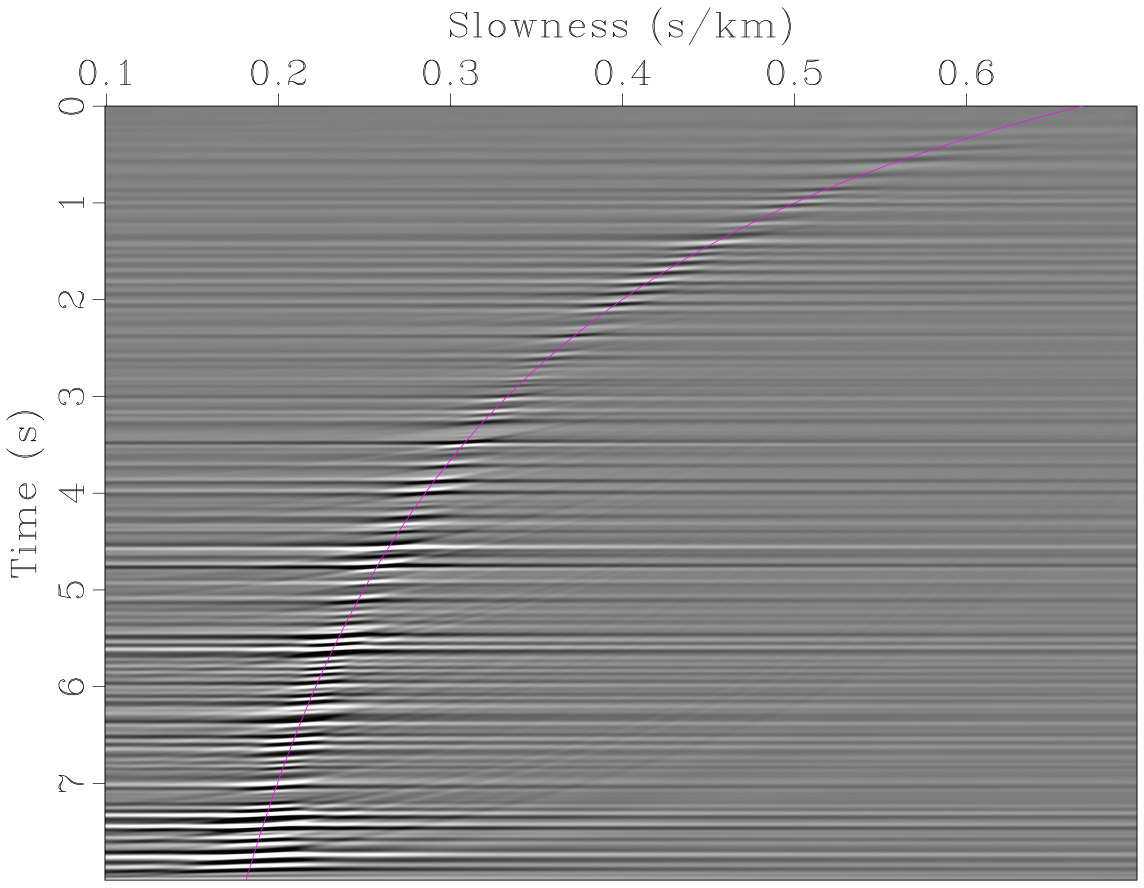

Figure 11: $N_\tau = 4000$, $N_p = 400$. Output of the fast butterfly algorithm applied to the synthetic data in Figure 9. $N = 64$, $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$ (here the range of $\Phi = f\sqrt{\tau^2 + p^2h^2}$ is about 250). CPU time: **4.35 s**. Ref: CPU time of *velocity scan*: **21.93 s**. Purple curve overlaid is the true slowness.

– GEO-2013-

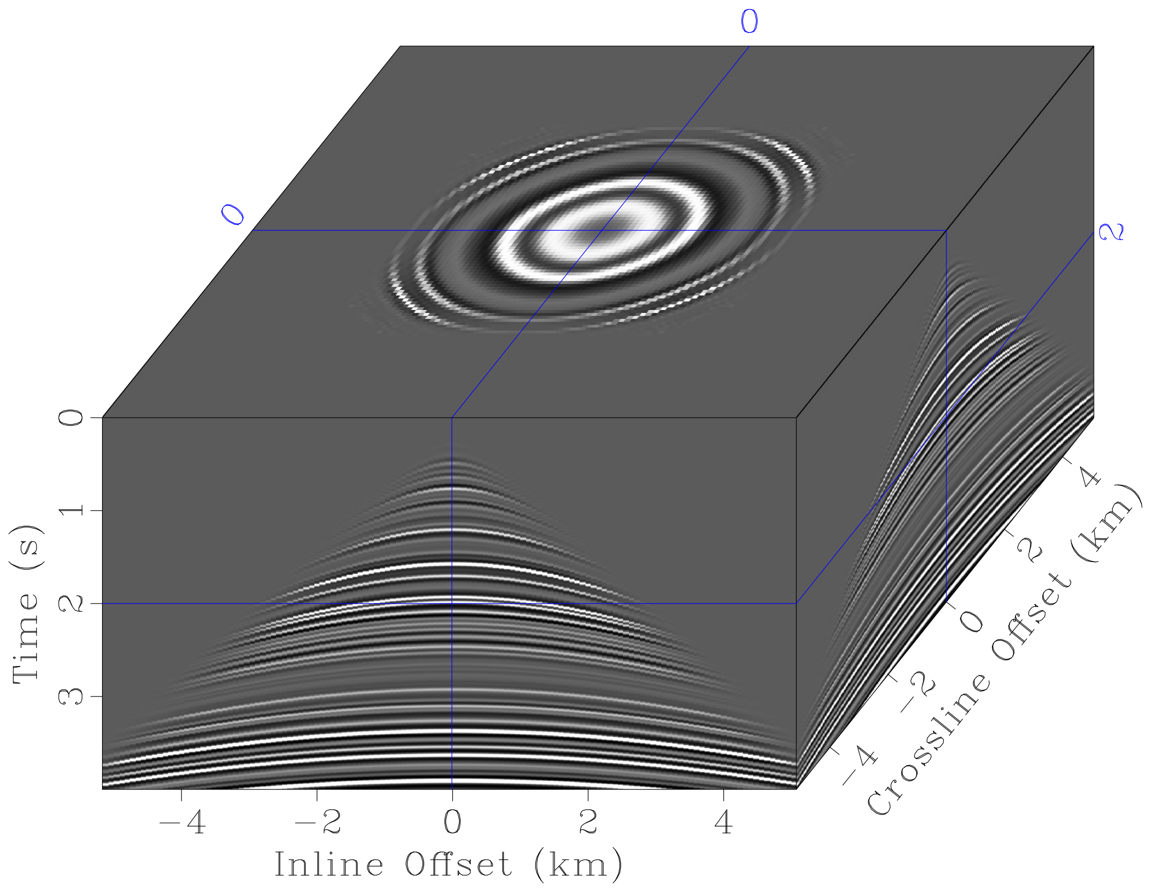

Figure 12: 3-D synthetic CMP gather. $N_t = 1000$, $N_{h_1} = N_{h_2} = 128$. $\Delta t = 0.004$ s, $\Delta h_1 = \Delta h_2 = 0.08$ km.

– GEO-2013-

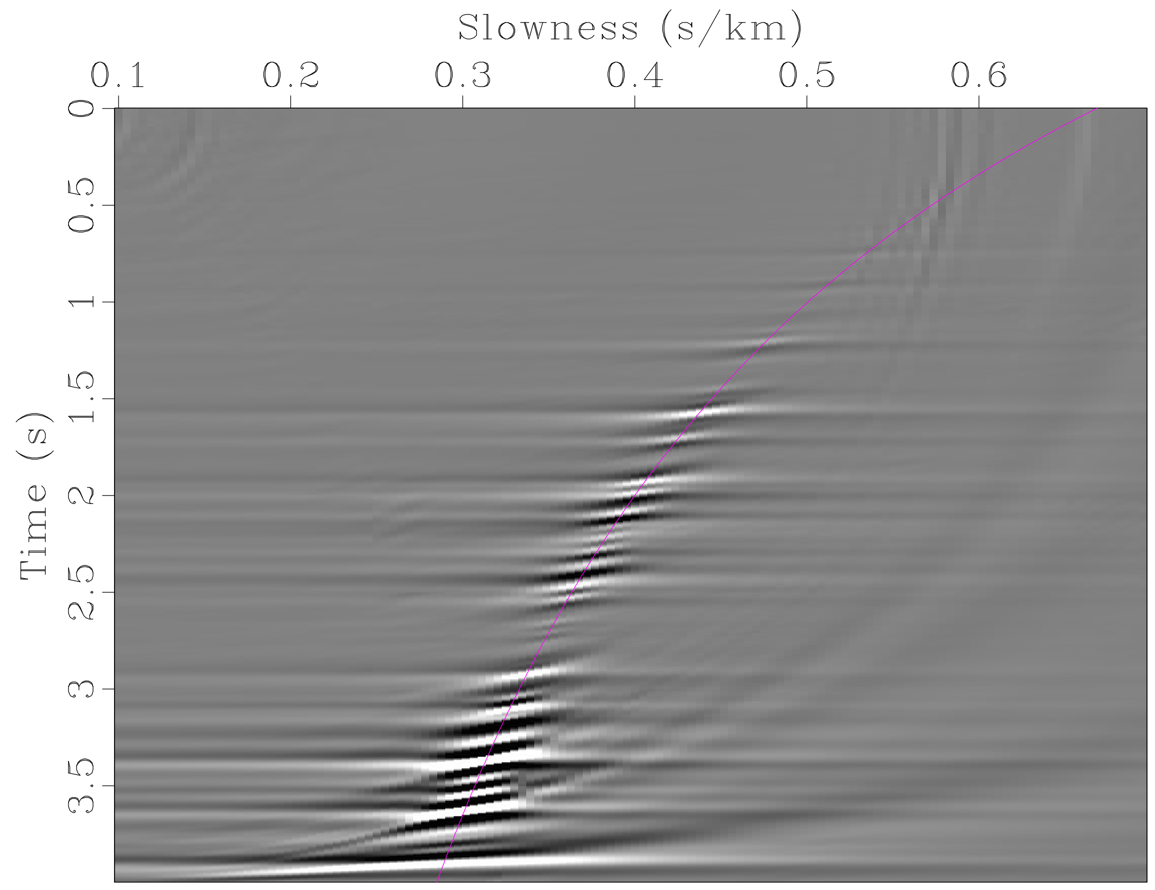

Figure 13: $N_\tau = 1000$, $N_p = 128$. Output of the fast butterfly algorithm applied to the synthetic data in Figure 12. $N = 64$, $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 5$ (here the range of $\Phi = f\sqrt{\tau^2 + p^2(h_1^2 + h_2^2)}$ is about 162). CPU time: **1.67 s**. Purple curve overlaid is the true slowness.

– GEO-2013-

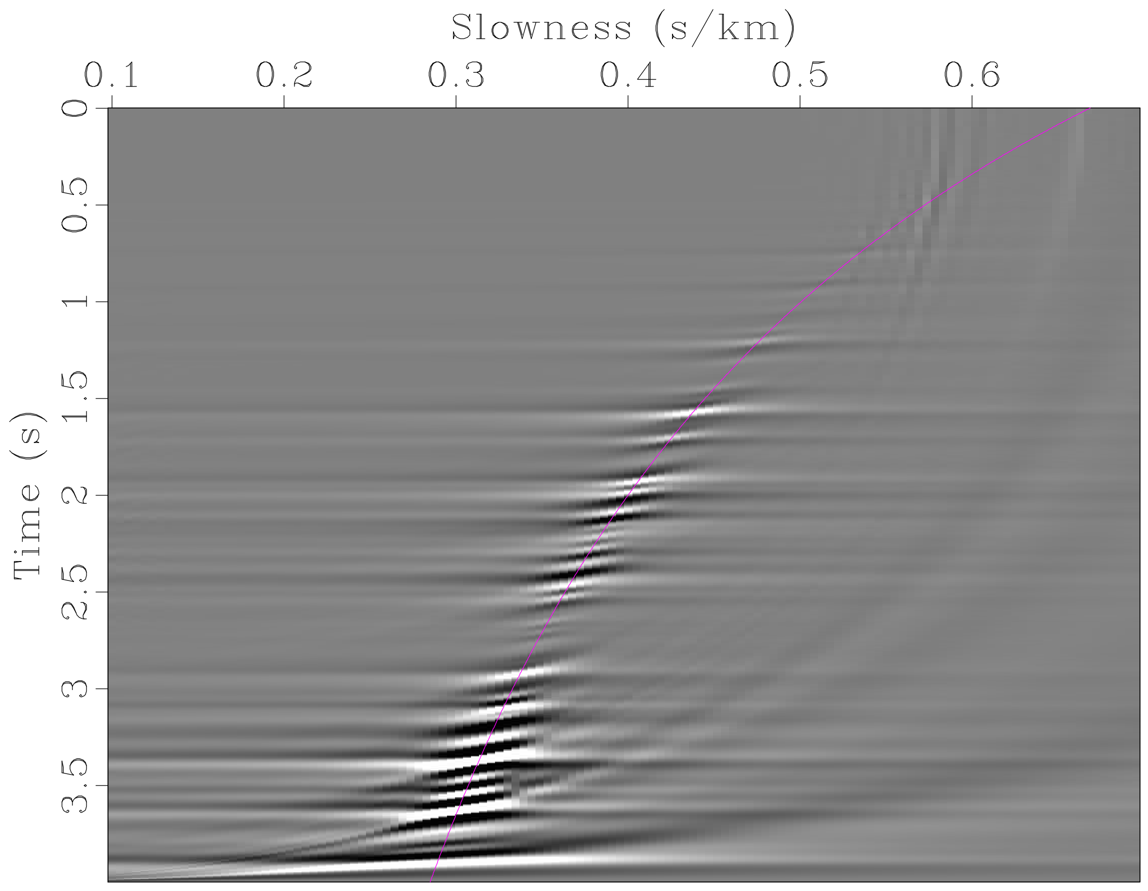

Figure 14: $N_\tau = 1000$, $N_p = 128$. Output of the *velocity scan* applied to the synthetic data in Figure 12. CPU time: **125.54 s**. Purple curve overlaid is the true slowness.

– **GEO-2013-**

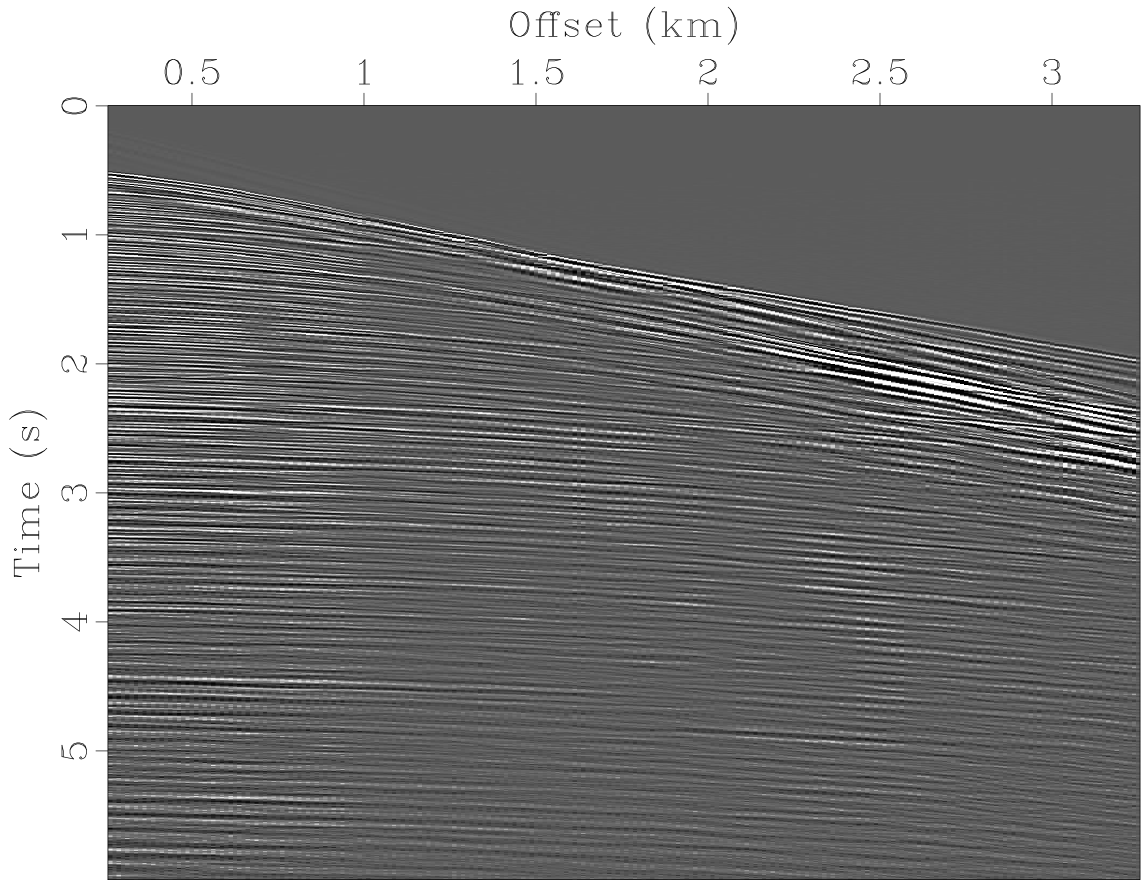

Figure 15: 2-D field CMP gather. $N_t = 1500$, $N_h = 240$. $\Delta t = 0.004$ s, $\Delta h = 0.0125$ km.

– GEO-2013-

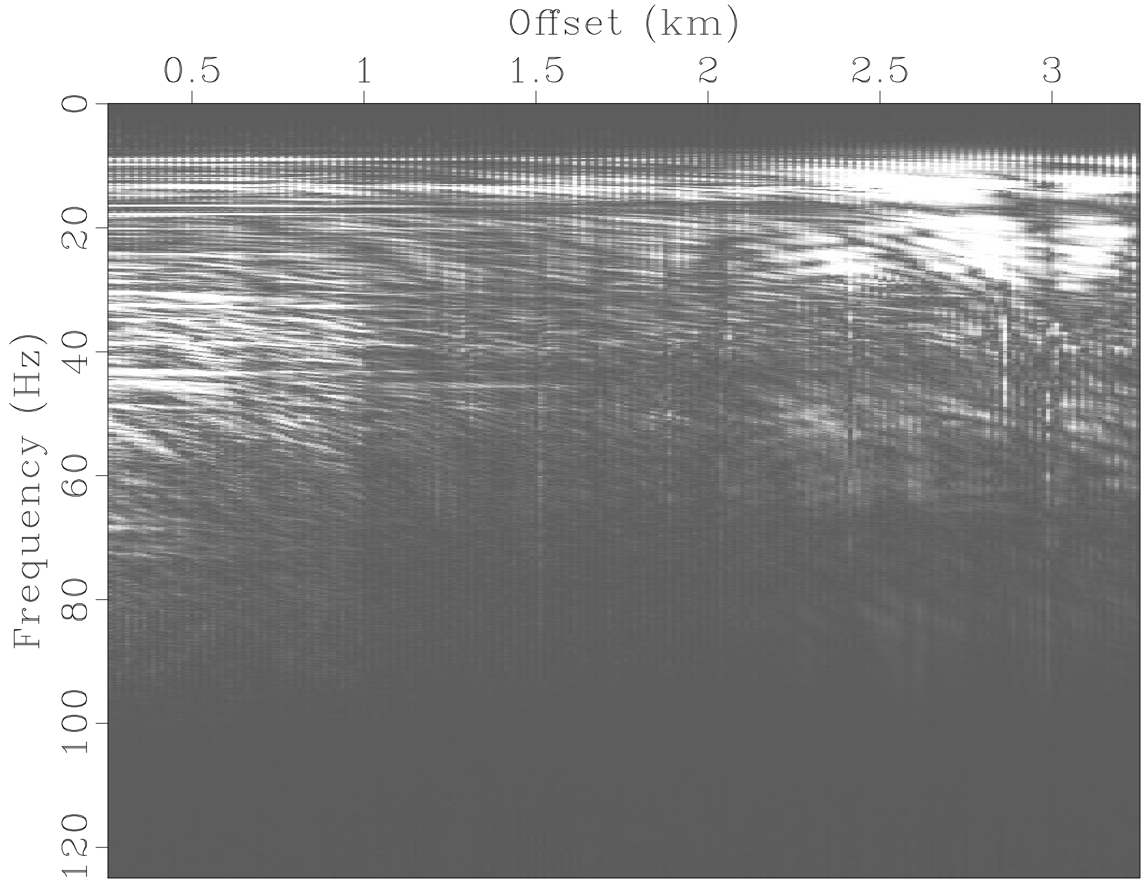

Figure 16: The Fourier transform (absolute value) on time axis of the field data in Figure 15.

– **GEO-2013-**

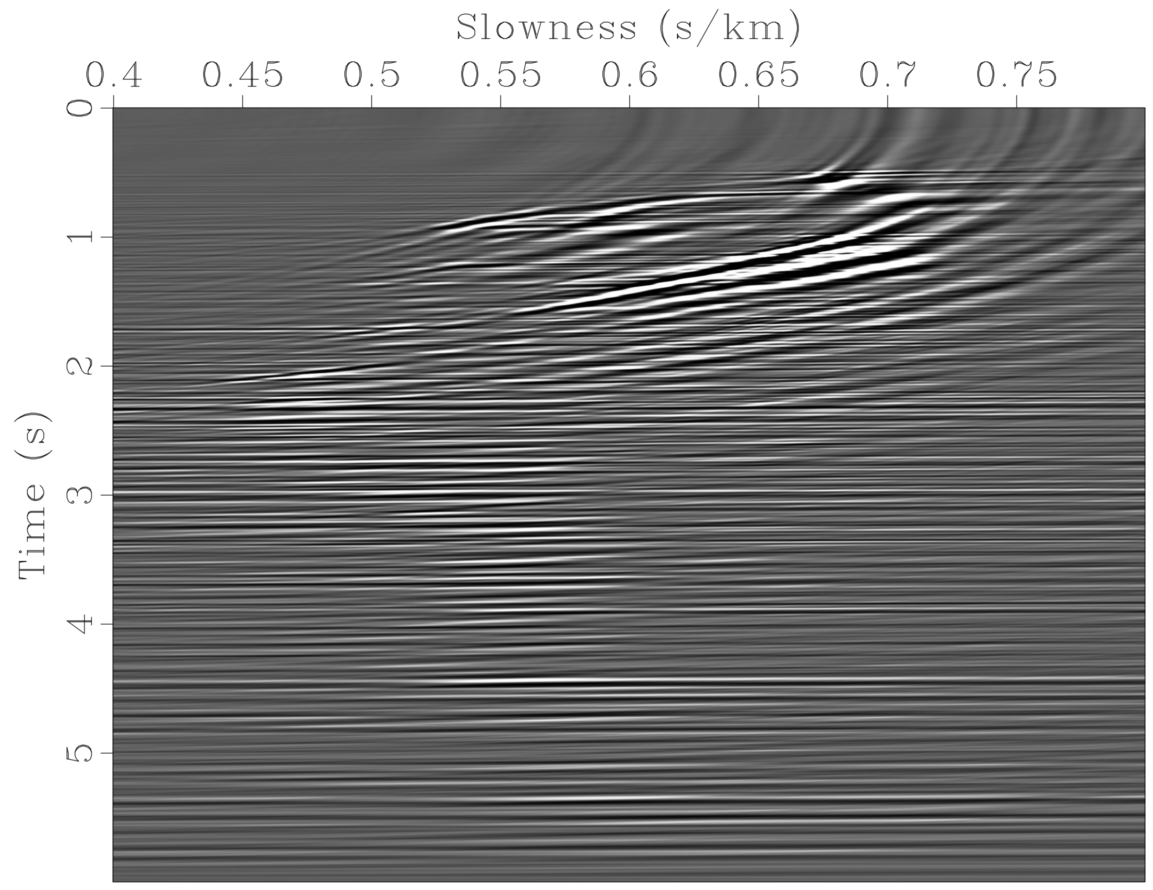

Figure 17: $N_\tau = 1500$, $N_p = 800$. Output of the fast butterfly algorithm applied to the field data in Figure 15. $N = 128$, $q_{k_1} = q_{x_1} = 7, q_{k_2} = q_{x_2} = 5$ (here the range of $\Phi = f\sqrt{\tau^2 + p^2h^2}$ is about 306). CPU time: **6.62 s**.

– **GEO-2013-**

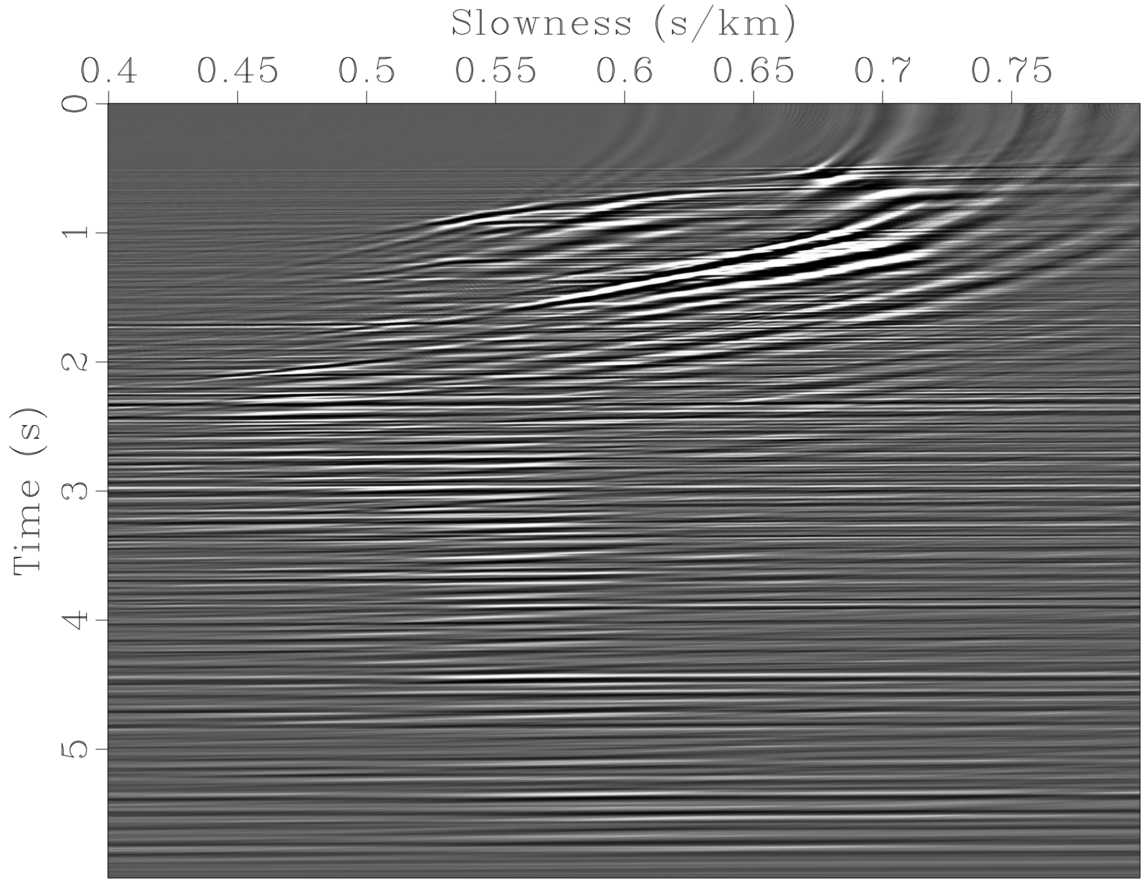

Figure 18: $N_\tau = 1500$, $N_p = 800$. Output of the *velocity scan* applied to the field data in Figure 15. CPU time: **9.91 s**.

– **GEO-2013-**

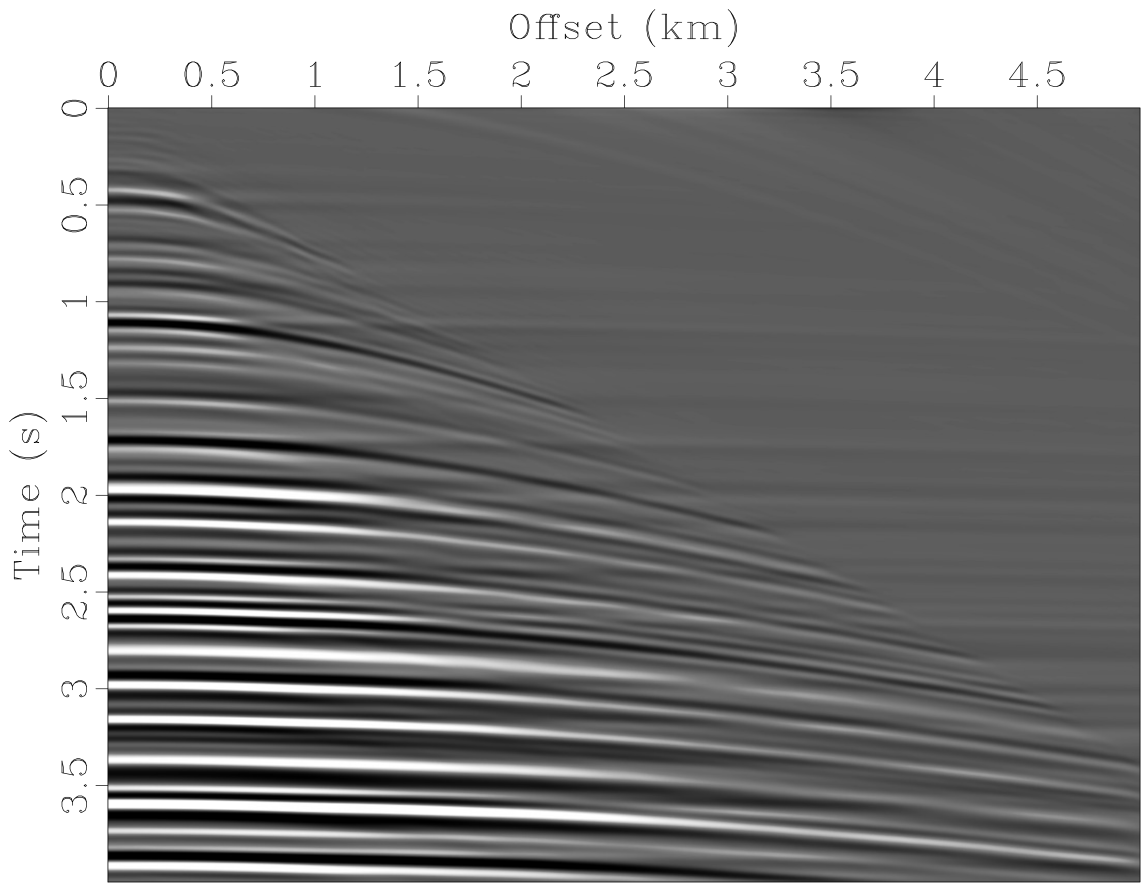

Figure 19: Output of the adjoint fast butterfly algorithm applied to the data in Figure 5. $N = 32$, $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$.

– **GEO-2013-**

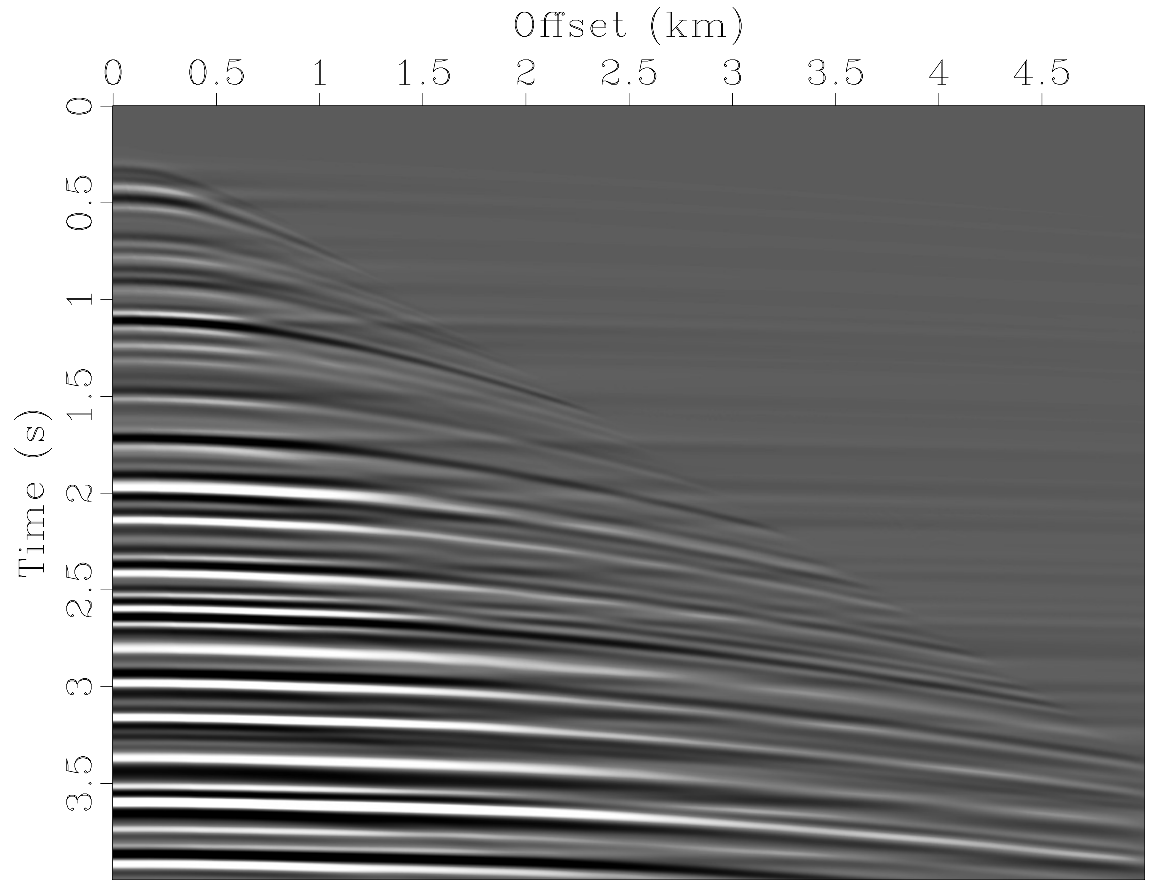

Figure 20: Output of the adjoint *velocity scan* applied to the data in Figure 6.

– GEO-2013-

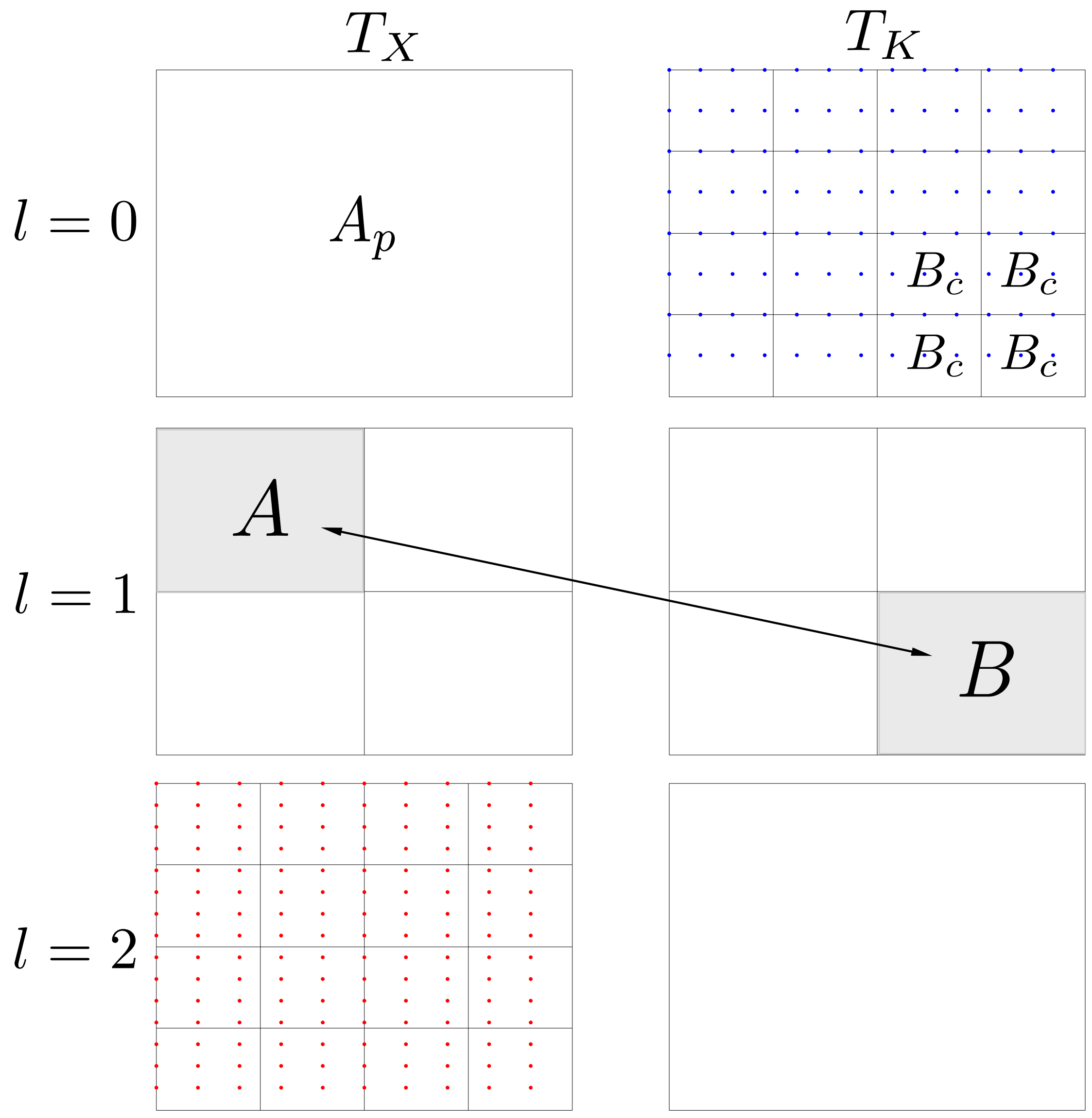

Figure A-1: The butterfly structure for the special case of $N = 4$. The top right panel represents the input domain K with sources $g(\mathbf{k})$ located at $\mathbf{k}$ (blue dots). The bottom left panel represents the output domain X with targets $u(\mathbf{x})$ located at $\mathbf{x}$ (red dots). For the pair of boxes (A, B) at level $l = 1$, box A_p is called A's parent at the previous level; four small boxes B_c are called B's children at the previous level.

– **GEO-2013-**