GEOPHYSICS, VOL. 78, NO. 4 (JULY-AUGUST 2013); P. U41–U51, 21 FIGS.
10.1190/GEO2012-0240.1

A fast butterfly algorithm for generalized Radon transforms

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

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 have devised 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 butterfly algorithm. For 2D data, the algorithm runs in complexity $O(N^2 \log N)$, where N depends on the maximum frequency and offset in the data set and the range of parameters (intercept time and slowness) in the model space. From a series of studies, we found that this 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, relatively costly "velocity scan" (i.e., direct integration plus interpolation in the time domain), our method provides an accurate approximation in only $O(N^2 \log N)$ (all the logs in this paper refer to logarithm to base 2) operations for 2D data. Here, N depends on the maximum frequency and offset in the data set and the range of parameters (intercept time and slowness) in the model space, and can often be chosen small compared with 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 for its inversion, which is the main objective of many previous works on fast Radon transforms (Sacchi, 1996; Liu and Sacchi, 2002; Trad et al., 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 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

Manuscript received by the Editor 2 July 2012; revised manuscript received 12 February 2013; published online 21 June 2013.
Part of this work was presented at the 82nd Annual International Meeting, SEG.
[1]The University of Texas at Austin, Institute for Computational Engineering and Sciences (ICES), Austin, Texas, USA. E-mail: hu@ices.utexas.edu.
[2]The University of Texas at Austin, Bureau of Economic Geology and Department of Geological Sciences, Austin, Texas, USA. E-mail: sergey.fomel@beg.utexas.edu.
[3]Massachusetts Institute of Technology, Department of Mathematics, Cambridge, Massachusetts, USA. E-mail: laurent@math.mit.edu.
[4]Stanford University, Department of Mathematics and Institute for Computational and Mathematical Engineering (ICME), Stanford, California, USA. E-mail: lexing@math.stanford.edu.

smooth curves. The term was introduced by Beylkin (1984, 1985), who 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 approximations 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, and a complete derivation can be found in Appendix A. We present numerical examples using synthetic and field data 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

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

where t is time, h is offset, τ is intercept time, and p is slowness. Fixing (τ,p), 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,

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

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

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

(the area element is omitted; the same symbols f, h, τ, and p are used for continuous and discrete variables). The reason that hyperbolic RT is harder to compute than linear RT ($t=\tau+ph$) or 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 $[0,1]\times[0,1]$ ($[a,b]\times[c,d]$ represents a 2D rectangular domain in the xy-plane with $x\in[a,b]$ and $y\in[c,d]$), i.e., a point $(f,h)\in[f_{\min},f_{\max}]\times[h_{\min},h_{\max}]$ is mapped to $\mathbf{k}=(k_1,k_2)\in[0,1]\times[0,1]=K$ via

$$f=(f_{\max}-f_{\min})k_1+f_{\min},\quad h=(h_{\max}-h_{\min})k_2+h_{\min}; \tag{4}$$

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

$$\tau=(\tau_{\max}-\tau_{\min})x_1+\tau_{\min},\quad p=(p_{\max}-p_{\min})x_2+p_{\min}. \tag{5}$$

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 phase function $\Phi(\mathbf{x},\mathbf{k})=f(k_1)\sqrt{\tau(x_1)^2+p(x_2)^2h(k_2)^2}$, then equation 3 can be written as

$$u(\mathbf{x})=\sum_{\mathbf{k}\in K}e^{2\pi i\Phi(\mathbf{x},\mathbf{k})}g(\mathbf{k}),\quad \mathbf{x}\in X \tag{6}$$

(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). This form is the discrete version of an oscillatory integral of the type

$$u(\mathbf{x})=\int_K e^{2\pi i\Phi(\mathbf{x},\mathbf{k})}g(\mathbf{k})\,d\mathbf{k},\quad \mathbf{x}\in X, \tag{7}$$

whose fast evaluation has been considered in Candés et al. (2009). Our method for computing the summation in equation 6 follows the Fourier integral operator (FIO) butterfly algorithm introduced there.

Low-rank approximations

Clearly the range and possibly other factors such as 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 $w(A)w(B)\le 1/N$ — in which case the pair (A,B) is called admissible — then

$$\left|e^{2\pi i\Phi(\mathbf{x},\mathbf{k})}-\sum_{t=1}^{r_\epsilon}\alpha_t^{AB}(\mathbf{x})\beta_t^{AB}(\mathbf{k})\right|\le\epsilon,\quad \text{for } \mathbf{x}\in A,\quad \mathbf{k}\in B, \tag{8}$$

where r_ϵ is independent of N for a fixed error ϵ. Here and below, the subscript t is slightly abused: t should be understood as multiindices (t_1,t_2), and accordingly r_ϵ is the total number of terms in a double sum. Furthermore, Candés et al. (2009) showed 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 $w(A)\le 1/\sqrt{N}$, and in the $\mathbf{k}$ variable when $w(B)\le 1/\sqrt{N}$.

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

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

$$\beta_t^{AB}(\mathbf{k})=e^{-2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_t^B)}L_t^B(\mathbf{k})e^{2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k})}; \tag{10}$$

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

$$\alpha_t^{AB}(\mathbf{x})=e^{2\pi i\Phi(\mathbf{x},\mathbf{k}_0(B))}L_t^A(\mathbf{x})e^{-2\pi i\Phi(\mathbf{x}_t^A,\mathbf{k}_0(B))}, \tag{11}$$

$$\beta_t^{AB}(\mathbf{k}) = e^{2\pi i\Phi(\mathbf{x}_t^A,\mathbf{k})}. \tag{12}$$

Boldface letters $\mathbf{k}_t^B$, $\mathbf{x}_t^A$, $\mathbf{k}_0(B)$, $\mathbf{x}_0(A)$ denote 2D vectors. Vector $\mathbf{k}_t^B$ is a point on the 2D, $q_{k_1} \times q_{k_2}$ Chebyshev grid in box B centered at $\mathbf{k}_0(B)$, i.e., 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

$$k_{t_1}^B = k_{0_1}^B + w(B)z_{t_1}, \quad 0 \le t_1 \le q_{k_1} - 1, \tag{13}$$

$$k_{t_2}^B = k_{0_2}^B + w(B)z_{t_2}, \quad 0 \le t_2 \le q_{k_2} - 1, \tag{14}$$

where

$$\left\{ z_{t_i} = \frac{1}{2} \cos\left(\frac{\pi t_i}{q_{k_i} - 1}\right)\right\}_{0\le t_i \le q_{k_i}-1, i=1,2} \tag{15}$$

is the 1D 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 2D Lagrange interpolation defined on the Chebyshev grid,

$$L_t^B(\mathbf{k}) = \left(\prod_{s_1=0,s_1\ne t_1}^{q_{k_1}-1} \frac{k_1 - k_{s_1}^B}{k_{t_1}^B - k_{s_1}^B}\right)\left(\prod_{s_2=0,s_2\ne t_2}^{q_{k_2}-1} \frac{k_2 - k_{s_2}^B}{k_{t_2}^B - k_{s_2}^B}\right). \tag{16}$$

Analogously, $\mathbf{x}_t^A$ is a point on the 2D, $q_{x_1} \times q_{x_2}$ Chebyshev grid in box A centered at $\mathbf{x}_0(A)$, and $L_t^A(\mathbf{x})$ is the 2D Lagrange interpolation defined on this grid. Based on the discussion above, the number r_ϵ in low-rank approximation 8 is equal to $q_{k_1}q_{k_2}$ when $w(B) \le 1/\sqrt{N}$, and $q_{x_1}q_{x_2}$ when $w(A) \le 1/\sqrt{N}$.

A simple way of viewing expressions 9–12 is: when $w(B) \le 1/\sqrt{N}$, plugging expression 9 into approximation 8 (leaving $\beta_t^{AB}(\mathbf{k})$ as it is) yields

$$e^{2\pi i\Phi(\mathbf{x},\mathbf{k})} \approx \sum_t e^{2\pi i\Phi(\mathbf{x},\mathbf{k}_t^B)}\beta_t^{AB}(\mathbf{k}), \quad \text{for } \mathbf{x} \in A, \quad \mathbf{k} \in B. \tag{17}$$

For fixed $\mathbf{x}$, the right-hand side of equation 17 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 $w(A) \le 1/\sqrt{N}$, plugging expression 12 into approximation 8, we get

$$e^{2\pi i\Phi(\mathbf{x},\mathbf{k})} \approx \sum_t e^{2\pi i\Phi(\mathbf{x}_t^A,\mathbf{k})}\alpha_t^{AB}(\mathbf{x}), \quad \text{for } \mathbf{x} \in A, \quad \mathbf{k} \in B. \tag{18}$$

For fixed $\mathbf{k}$, the right-hand side of equation 18 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 low-rank approximation 8 is known, computing the partial sum

$$u^B(\mathbf{x}) := \sum_{\mathbf{k}\in B} e^{2\pi i\Phi(\mathbf{x},\mathbf{k})}g(\mathbf{k}), \quad \text{for } \mathbf{x} \in A, \tag{19}$$

generated by points $\mathbf{k}$ inside a box B becomes

$$u^B(\mathbf{x}) \approx \sum_{\mathbf{k}\in B}\sum_t \alpha_t^{AB}(\mathbf{x})\beta_t^{AB}(\mathbf{k})g(\mathbf{k}) = \sum_t \alpha_t^{AB}(\mathbf{x})\delta_t^{AB}, \tag{20}$$

where

$$\delta_t^{AB} := \sum_{\mathbf{k}\in B}\beta_t^{AB}(\mathbf{k})g(\mathbf{k}). \tag{21}$$

The case that box B represents the whole domain, K is of particular interest because 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. The butterfly algorithm was introduced by Michielssen and Boag (1996), and generalized by O'Neil et al. (2010) and Candés et al. (2009). Different applications include sparse Fourier transform (Ying, 2009) and radar imaging (Demanet et al., 2012). Demanet et al. (2012) also provided a complete error analysis of the method introduced by Candés 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 boxes B in a multiscale fashion. In tandem, the sizes of 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

$$w(A)w(B) = \frac{1}{2^l}\frac{1}{2^{L-l}} = \frac{1}{N}; \tag{22}$$

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).

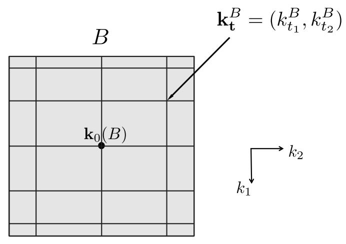

Figure 1. A 2D, $q_{k_1} \times q_{k_2}$ ($q_{k_1} = 7$, $q_{k_2} = 5$) Chebyshev grid in box B. Here, $\mathbf{k}_0(B)$ is the center of the box, and $\mathbf{k}_t^B = (k_{t_1}^B, k_{t_2}^B)$, $0 \le t_1 \le q_{k_1} - 1$, and $0 \le t_2 \le q_{k_2} - 1$ is a point on the grid.

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}, definition 21, so as to get $u^B(\mathbf{x})$, definition 19, 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 Appendix A, 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

$$\delta_t^{AB}=e^{-2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_t^B)}\sum_{\mathbf{k}\in B}\left(L_t^B(\mathbf{k})e^{2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k})}g(\mathbf{k})\right). \tag{23}$$

2) Recursion

At $l=1,2,\ldots,L/2$, 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. Update $\{\delta_t^{AB}\}$ from $\{\delta_{t'}^{A_pB_c}\}$ by

$$\delta_t^{AB}=e^{-2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_t^B)}\sum_c\sum_{t'}\left(L_t^B(\mathbf{k}_{t'}^{B_c})e^{2\pi i\Phi(\mathbf{x}_0(A),\mathbf{k}_{t'}^{B_c})}\delta_{t'}^{A_pB_c}\right). \tag{24}$$

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

$$\delta_t^{AB}=\sum_s e^{2\pi i\Phi(\mathbf{x}_t^A,\mathbf{k}_s^B)}\delta_s^{AB}. \tag{25}$$

4) Recursion

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

$$\delta_t^{AB}=\sum_c e^{2\pi i\Phi(\mathbf{x}_t^A,\mathbf{k}_0(B_c))}\sum_{t'}\left(L_{t'}^{A_p}(\mathbf{x}_t^A)e^{-2\pi i\Phi(\mathbf{x}_{t'}^{A_p},\mathbf{k}_0(B_c))}\delta_{t'}^{A_pB_c}\right). \tag{26}$$

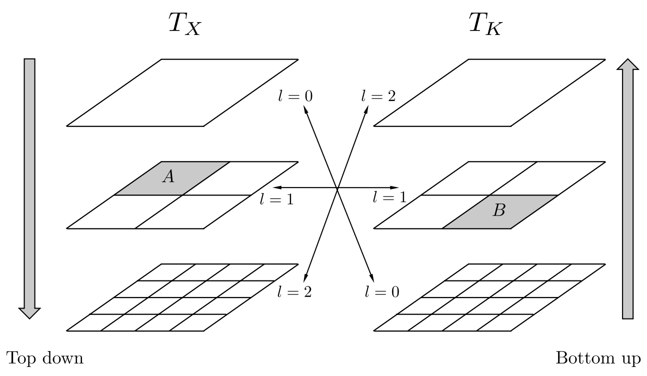

Figure 2. The butterfly quad tree structure for the special case of $N=4$.

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

$$u(\mathbf{x})=e^{2\pi i\Phi(\mathbf{x},\mathbf{k}_0(B))}\sum_t\left(L_t^A(\mathbf{x})e^{-2\pi i\Phi(\mathbf{x}_t^A,\mathbf{k}_0(B))}\delta_t^{AB}\right). \tag{27}$$

Numerical complexity and accuracy

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 1D at a time. Because 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 $O(r_\epsilon^2N^2)$, and steps 1 and 5 take $O(r_\epsilon N_fN_h)$ 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 $O(N_hN_t\log N_t+r_\epsilon^{3/2}N^2\log N+r_\epsilon^2N^2+r_\epsilon(N_fN_h+N_\tau N_p))$. The analysis in Candés et al. (2009) showed 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 demonstrated that the dependence of r_ϵ on $\log(1/\epsilon)$ is rather moderate in practice.

In comparison, the conventional velocity scan requires at least $O(N_\tau N_pN_h)$ 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 depends neither on data size nor on data content (here we mean the data after the Fourier transform). Because the Chebyshev interpolation is only performed on the kernel, our choice of parameters (N and number of Chebyshev points) relies on the preknowledge about the range of f, h, τ, and p. In other words, we need a general idea about how oscillate the kernel is. Recall that everything is mapped to a unit square, so the larger the range of $\Phi(\mathbf{x},\mathbf{k})$ is, the more oscillations occur in the unit square. If the original data (data before the Fourier transform) contain high-frequency information, the accuracy will be affected as the frequency bandwidth is now larger. A possible way to get around it is to divide the Fourier domain into two or three smaller subdomains (so the range of f in each subdomain is smaller than the original problem), and apply the fast algorithm to each part separately, finally add the results back together. This only increases the cost by a small factor, but presumably offers better accuracy.

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 (the nearest neighbor 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, because 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}, and q_{x_2} are chosen such that the relative error between the fast algorithm and the direct computation of equation 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 2D 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 s of CPU time, whereas 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 equation 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 data sets using rectangular sampling $N_t = 4000$, $N_h = 400$. The first one (Figure 8) has the same range as the previous example (Figure 3), whereas the second one (Figure 9) doubles the range of time and offset. Results of the fast

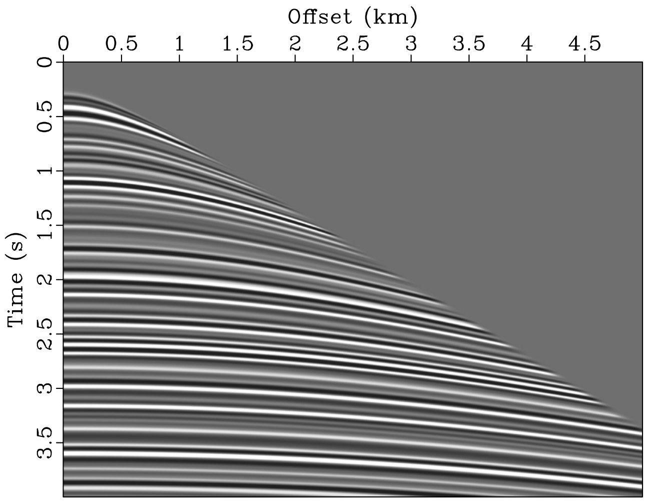

Figure 3. Two-dimensional synthetic CMP gather. Here, $N_t = N_h = 1000$, $\Delta t = 0.004$ s, and $\Delta h = 0.005$ km.

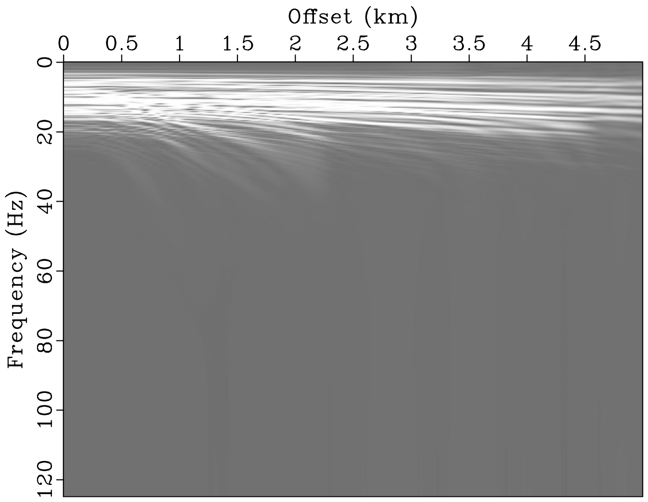

Figure 4. The Fourier transform (absolute value) on time axis of the synthetic data in Figure 3.

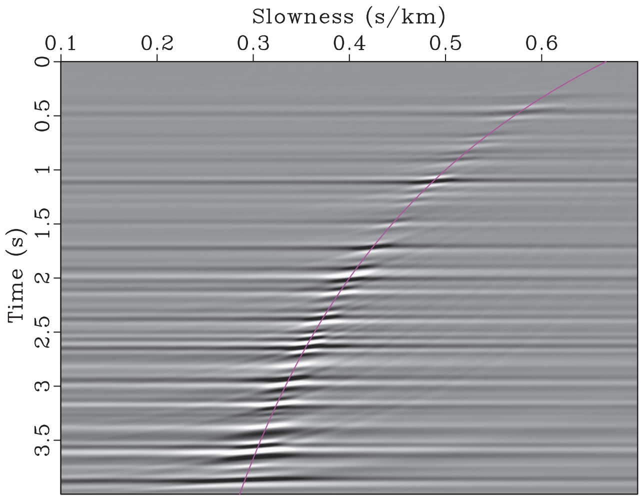

Figure 5. Output of the fast butterfly algorithm applied to the synthetic data in Figure 3. Here, $N_\tau = N_p = 1000$, $N = 32$, and $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$. CPU time: 1.75 s. Purple curve overlaid is the true slowness.

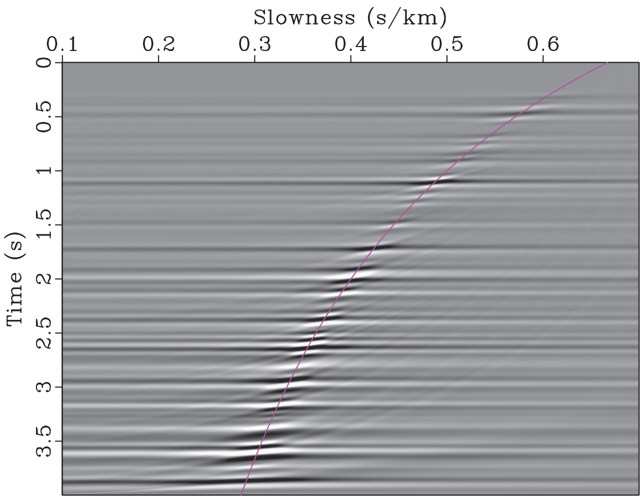

Figure 6. Output of the velocity scan applied to the synthetic data in Figure 3. Here, $N_\tau = N_p = 1000$. CPU time: 37.23 s. Purple curve overlaid is the true slowness.

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 data set, 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 equation 23), 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:

$$(Rd)(\tau, p) = \iint d\left(\sqrt{\tau^2 + p^2(h_1^2 + h_2^2)}, h_1, h_2\right) dh_1\, dh_2, \tag{28}$$

where $d(t, h_1, h_2)$ is a 3D 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)

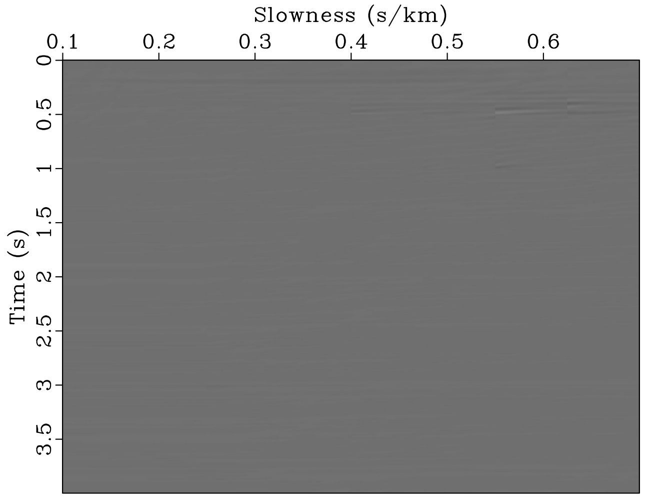

Figure 7. Difference between the results of the fast algorithm and the direct evaluation of equation 3 plotted at the same scale as in Figure 5.

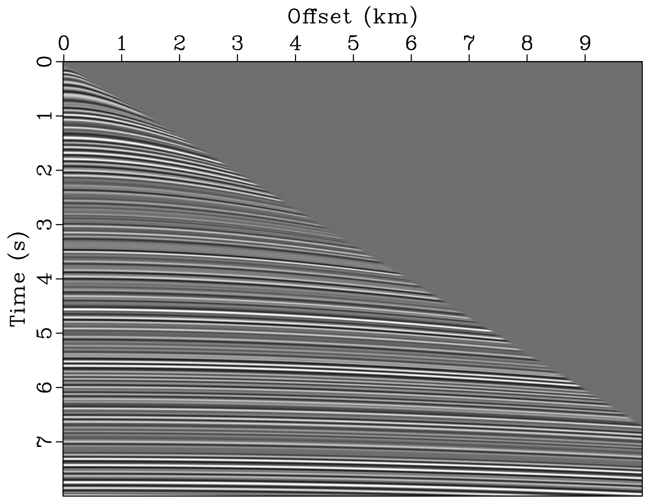

Figure 9. Two-dimensional synthetic CMP gather. Here, $N_t = 4000$, $N_h = 400$, $\Delta t = 0.002$ s, and $\Delta h = 0.025$ km.

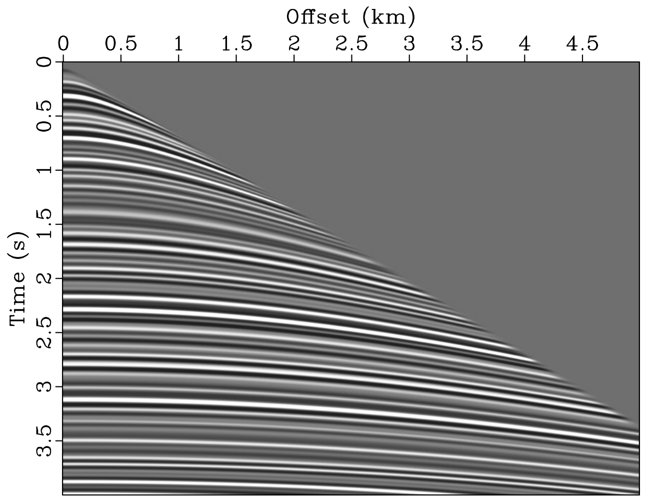

Figure 8. Two-dimensional synthetic CMP gather. Here, $N_t = 4000$, $N_h = 400$, $\Delta t = 0.001$ s, and $\Delta h = 0.0125$ km.

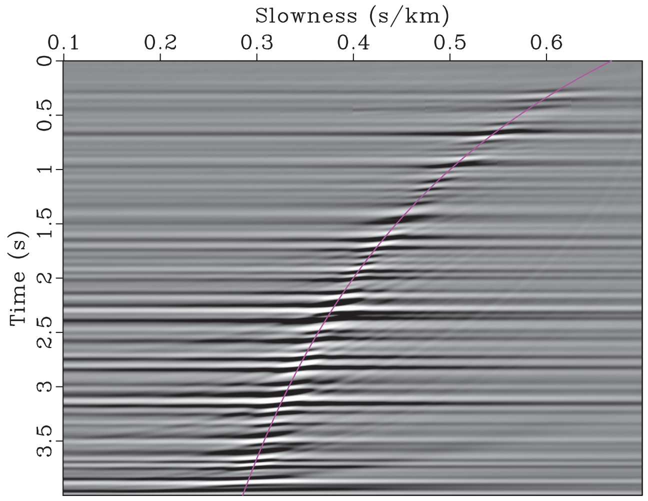

Figure 10. Output of the fast butterfly algorithm applied to the synthetic data in Figure 8. Here, $N_\tau = 4000$, $N_p = 400$, $N = 32$, and $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$. CPU time: 2.46 s. Ref: CPU time of velocity scan: 21.84 s. Purple curve overlaid is the true slowness.

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 2D field seismic gather shown 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 $N_t = 1500$, $N_h = 240$, whereas the output sizes are chosen as $N_\tau = 1500$, $N_p = 800$. Although this small data set 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 should take longer computation time.

Computing the adjoint operator

The last example is concerned with the computation of the adjoint of the hyperbolic RT. 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

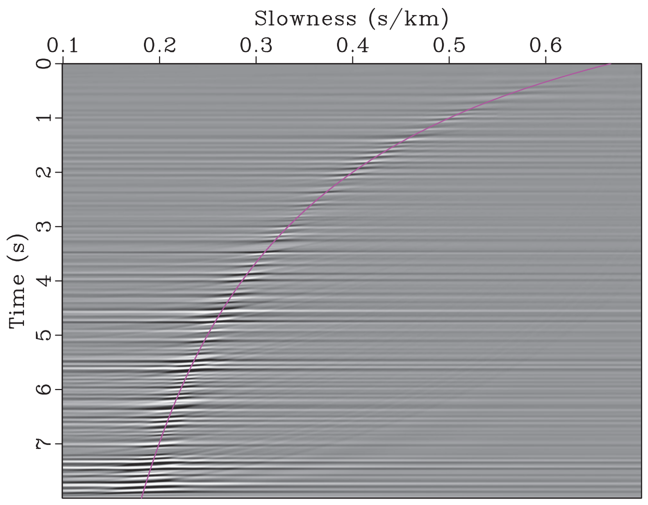

Figure 11. Output of the fast butterfly algorithm applied to the synthetic data in Figure 9. Here, $N_\tau = 4000$, $N_p = 400$, $N = 64$, and $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$. CPU time: 4.35 s. Ref: CPU time of velocity scan: 21.93 s. Purple curve overlaid is the true slowness.

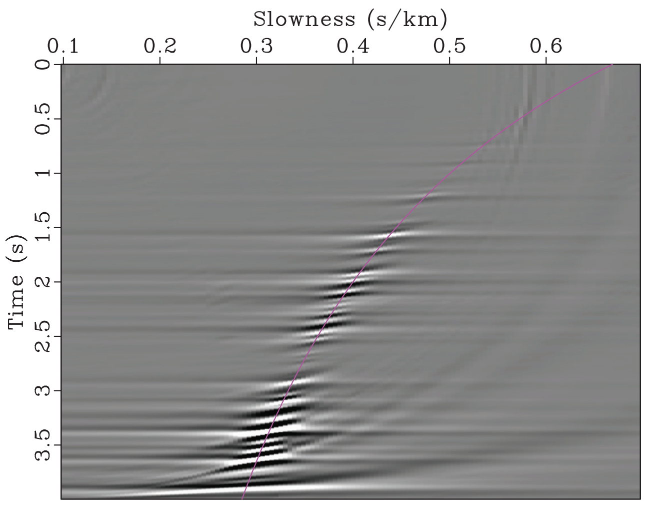

Figure 13. Output of the fast butterfly algorithm applied to the synthetic data in Figure 12. Here, $N_\tau = 1000$, $N_p = 128$, $N = 64$, and $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 5$. CPU time: 1.67 s. Purple curve overlaid is the true slowness.

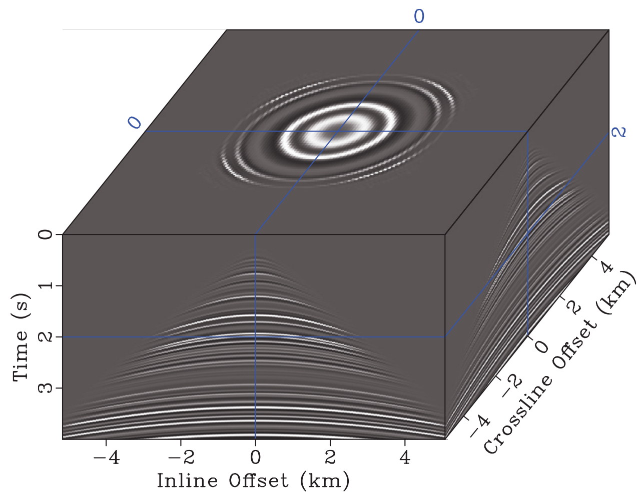

Figure 12. Three-dimensional synthetic CMP gather. Here, $N_t = 1000$, $N_{h_1} = N_{h_2} = 128$, $\Delta t = 0.004$ s, and $\Delta h_1 = \Delta h_2 = 0.08$ km.

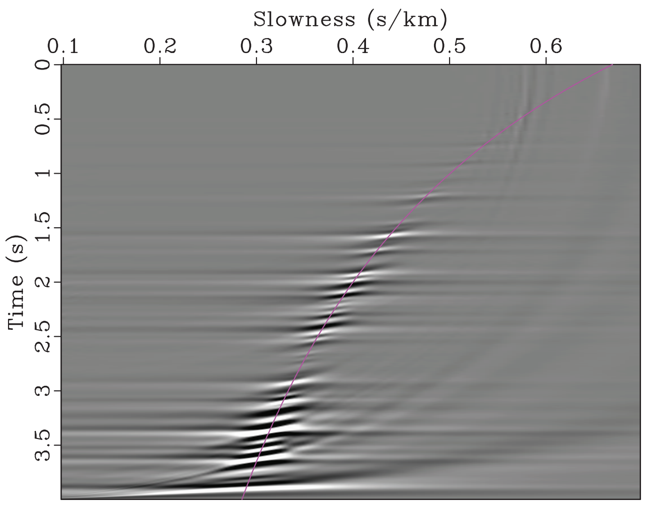

Figure 14. Output of the velocity scan applied to the synthetic data in Figure 12. Here, $N_\tau = 1000$ and $N_p = 128$. CPU time: 125.54 s. Purple curve overlaid is the true slowness.

$$\langle m(\tau, p), (Rd)(\tau, p)\rangle = \langle (R^* m)(t, h), d(t, h)\rangle, \tag{29}$$

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

$$\langle g_1(x,y), g_2(x,y)\rangle = \sum_{x,y} g_1(x,y)\overline{g_2(x,y)}, \ \forall g_1(x,y), \ g_2(x,y); \tag{30}$$

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

$$(R^* m)(t, h) = \mathcal{F}^{-1}_{f\to t}\left(\sum_{\tau,p} e^{-2\pi i f\sqrt{\tau^2+p^2h^2}} m(\tau, p)\right), \tag{31}$$

where $\mathcal{F}^{-1}_{f\to t}$ is the inverse Fourier transform from variable f to t. The summation in equation 31 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 similar results (see Figures 19 and 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.

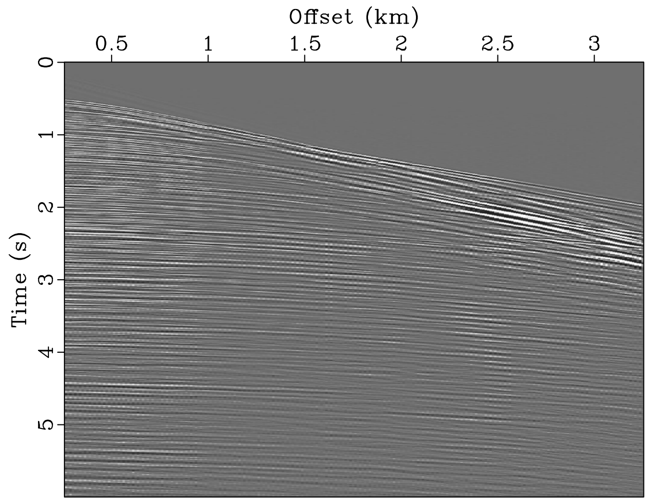

Figure 15. Two-dimensional field CMP gather. Here, $N_t = 1500$, $N_h = 240$, $\Delta t = 0.004$ s, and $\Delta h = 0.0125$ km.

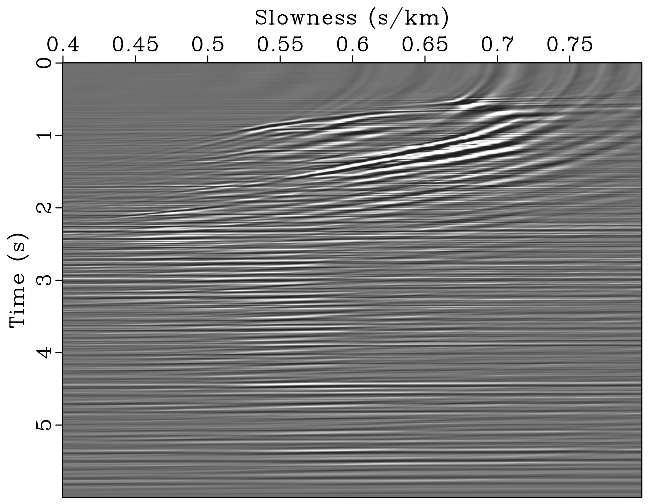

Figure 17. Output of the fast butterfly algorithm applied to the field data in Figure 15. Here, $N_\tau = 1500$, $N_p = 800$, $N = 128$, $q_{k_1} = q_{x_1} = 7$, and $q_{k_2} = q_{x_2} = 5$. CPU time: 6.62 s.

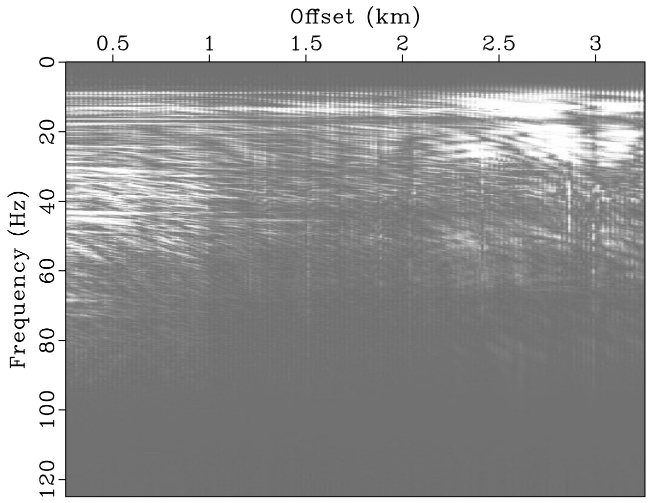

Figure 16. The Fourier transform (absolute value) on time axis of the field data in Figure 15.

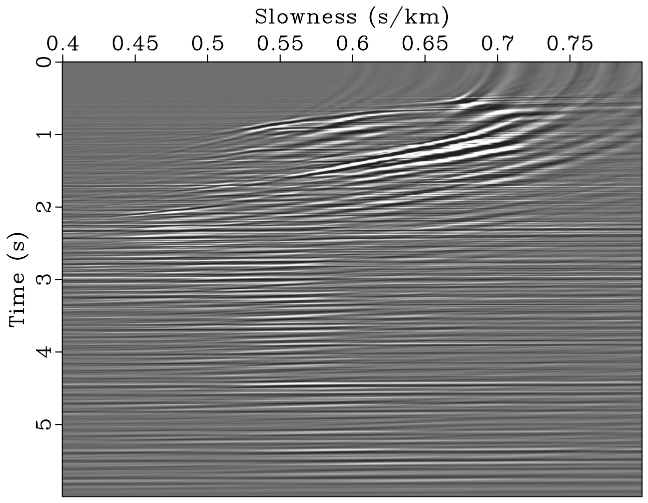

Figure 18. Output of the velocity scan applied to the field data in Figure 15. Here, $N_\tau = 1500$ and $N_p = 800$. CPU time: 9.91 s.

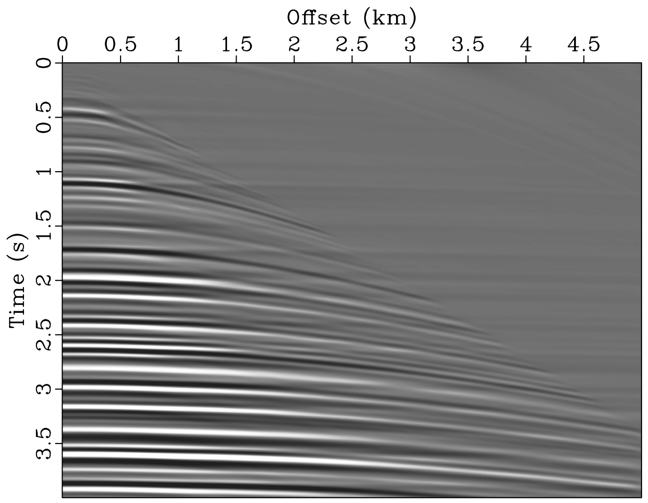

Figure 19. Output of the adjoint fast butterfly algorithm applied to the data in Figure 5. Here, $N = 32$ and $q_{k_1} = q_{k_2} = q_{x_1} = q_{x_2} = 9$.

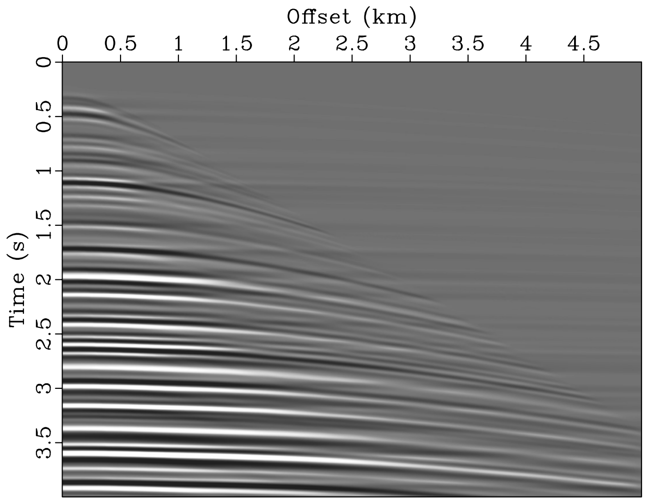

Figure 20. Output of the adjoint velocity scan applied to the data in Figure 6.

CONCLUSIONS

We constructed a fast butterfly algorithm for the hyperbolic Radon transform, a type of generalized Radon transforms. Compared with expensive 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 data set and the range of intercept time and slowness in the model space, and can often be chosen smaller than the grid size. In practice, this may lead to speedup of several orders of magnitude. Our ongoing work is studying the performance of this fast solver on the sparse iterative inversion of the hyperbolic RT applied to multiple attenuation.

Due to the generality of the butterfly algorithm, its application is 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 designed to save the cost by building the quad tree adaptively. Furthermore, many of the Radon-like integral operators, such as Kirchhoff migration, the apex-shifted Radon transform, the anisotropic multiparameter velocity scan, etc., can be reformulated in a similar fashion as we did in this paper. To address these extensions, a 3D 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 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 butterfly 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 Candés et al. (2009).

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

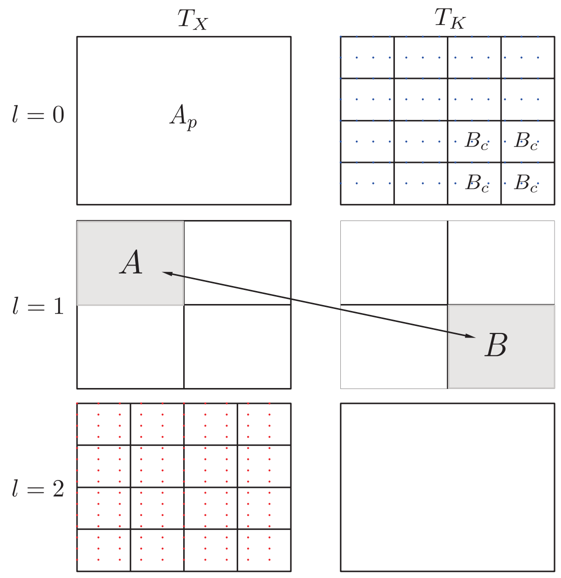

Figure A-1. The butterfly structure for the special case of $N = 4$. The top right panel represents the input domain K with sources $g(\mathbf{k})$ located at $\mathbf{k}$ (blue dots). The bottom left panel represents the output domain X with targets $u(\mathbf{x})$ located at $\mathbf{x}$ (red dots). 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.

1) Initialization

At level $l = 0$, let A be the root box of T_X. For each leaf box $B \in T_K$, expressions 9 and 10 are valid as $w(B) \leq 1/\sqrt{N}$. Substituting β_t^{AB} (in equation 10) into the definition of δ_t^{AB}, equation 21, we get

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

i.e., equation 23 in the main text. In addition, for $\mathbf{x} \in A$, the partial sum $u^B(\mathbf{x})$ in equation 20 is given by (with α_t^{AB} [in equation 9] plugged in)

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

Comparing the right-hand sides of equations 19 and A-2, if we call $g(\mathbf{k})$ the sources at $\mathbf{k}$, then 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 $l = 1, 2, \ldots L/2$, 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

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

Summing over all children gives

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

Because $A \subset A_p$, this is of course true for any $\mathbf{x} \in A$. Also we know that equation 17 holds for $\mathbf{k}_{t'}^{B_c} \in B$, i.e.,

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

Inserting it into expression A-4 yields

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

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

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

Equating expressions A-6 and A-7 suggest that we can take

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

Substituting β_t^{AB} (in equation 10), we get

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

i.e., equation 24 in the main text.

3) Switch

A switch of the representation to expressions 11 and 12 is needed at the middle level $l = L/2$ because expressions 9 and 10 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 β_t^{AB} (in equation 12) into the definition of δ_t^{AB}, equation 21, one has

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

from expression A-7,

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

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

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

i.e., equation 25 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} located at Chebyshev grid $\mathbf{x}_t^A$ on the X side.

4) Recursion

The rest of the recursion is analogous to step 2. For $l = L/2 + 1, \ldots, L$, we have

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

thus,

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

recalling expression A-10, one can set

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

Inserting α_t^{AB} (in equation 11) 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., equation 26 in the main text.

5) Termination

We finally 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} (in equation 11), 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., equation 27 in the main text. This final step is to transform the equivalent sources δ_t^{AB} located 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 L 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: Communications on Pure and Applied Mathematics, **37**, 579–599, doi: 10.1002/cpa.3160370503.

Beylkin, G., 1985, Imaging of discontinuities in the inverse scattering problem by inversion of a causal generalized Radon transform: Journal of Mathematical Physics, **26**, 99–108, doi: 10.1063/1.526755.

Candés, E., L. Demanet, and L. Ying, 2009, A fast butterfly algorithm for the computation of Fourier integral operators: Multiscale Model and Simulation, **7**, 1727–1750, doi: 10.1137/080734339.

Demanet, L., M. Ferrara, N. Maxwell, J. Poulson, and L. Ying, 2012, A butterfly algorithm for synthetic aperture radar imaging: SIAM Journal on Imaging Sciences, **5**, 203–243, doi: 10.1137/100811593.

Foster, D. J., and C. C. Mosher, 1992, Suppression of multiple reflections using the Radon transform: Geophysics, **57**, 386–395, doi: 10.1190/1.1443253.

Gardner, G. H. F., and L. Lu, eds., 1991, Slant-stack processing: SEG.

Hampson, D., 1986, Inverse velocity stacking for multiple elimination: 56th Annual International Meeting, SEG, Expanded Abstracts, 422–424.

Hargreaves, N., B. verWest, R. Wombell, and D. Trad, 2003, Multiple attenuation using an apex-shifted Radon transform: 65th Conference and Exhibition, EAGE, Extended Abstracts.

Herrmann, P., T. Mojesky, M. Magesan, and P. Hugonnet, 2000, De-aliased, high-resolution Radon transforms: 70th Annual International Meeting, SEG, Expanded Abstracts, 1953–1956.

Liu, Y., and M. Sacchi, 2002, De-multiple via a fast least squares hyperbolic Radon transform: 72nd Annual International Meeting, SEG, Expanded Abstracts, 2182–2185.

Michielssen, E., and A. Boag, 1996, A multilevel matrix decomposition algorithm for analyzing scattering from large structures: IEEE Transactions on Antennas and Propagation **44**, 1086–1093, doi: 10.1109/8.511816.

Moore, I., and C. Kostov, 2002, Stable, efficient, high-resolution Radon transforms: 64th Conference and Exhibition, EAGE, Extended Abstracts, F-34.

O'Neil, M., F. Woolfe, and V. Rokhlin, 2010, An algorithm for the rapid evaluation of special function transforms: Applied and Computational Harmonic Analysis, **28**, 203–226, doi: 10.1016/j.acha.2009.08.005.

Radon, J., 1917, Über die bestimmung von funktionen durch ihre integralwerte langs gewisser mannigfaltigkeiten: Berichte über die Verhandlungen der Sachsische 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: Consortium for the Development of Specialized Seismic Techniques report, 73–92.

Thorson, J. R., and J. F. Claerbout, 1985, Velocity-stack and slant-stack stochastic inversion: Geophysics, **50**, 2727–2741, doi: 10.1190/1.1441893.

Trad, D., 2003, Interpolation and multiple attenuation with migration operators: Geophysics, **68**, 2043–2054, doi: 10.1190/1.1635058.

Trad, D., T. Ulrych, and M. Sacchi, 2002, Accurate interpolation with high resolution time-variant Radon transforms: Geophysics, **67**, 644–656, doi: 10.1190/1.1468626.

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

Yilmaz, O., 1989, Velocity-stack processing: Geophysical Prospecting, **37**, 357–382, doi: 10.1111/j.1365-2478.1989.tb02211.x.

Ying, L., 2009, Sparse Fourier transform via butterfly algorithm: SIAM Journal on Scientific Computing, **31**, 1678–1694, doi: 10.1137/08071291X.