GEOPHYSICS, VOL. 74, NO. 6 (NOVEMBER-DECEMBER 2009); P. WCA129–WCA139, 12 FIGS., 2 TABLES.
10.1190/1.3223186

Accelerating 3D Fourier migration with graphics processing units

Jin-Hai Zhang[1], Shu-Qin Wang[1,2], and Zhen-Xing Yao[1]

ABSTRACT

Computational cost is a major factor that inhibits the practical application of 3D depth migration. We have developed a fast parallel scheme to speed up 3D wave-equation depth migration on a parallel computing device, i.e., on graphics processing units (GPUs). The third-order optimized generalized-screen propagator is used to take advantage of the built-in software implementation of the fast Fourier transform. The propagator is coded as a sequence of kernels that can be called from the computer host for each frequency component. Moving the wavefield extrapolation for each depth level to the GPUs allows handling a large 3D velocity model, but this scheme can be speeded up to a limited degree over the CPU implementation because of the low-bandwidth data transfer between host and device. We have created further speedup in this extrapolation scheme by minimizing the low-bandwidth data transfer, which is done by storing the 3D velocity model and imaged data in the device memory, and reducing half the memory demand by compressing the 3D velocity model and imaged data using integer arrays instead of float arrays. By incorporating a 2D tapered function, time-shift propagator, and scaling of the inverse Fourier transform into a compact kernel, the computation time is reduced greatly. Three-dimensional impulse responses and synthetic data examples have demonstrated that the GPU-based Fourier migration typically is 25 to 40 times faster than the CPU-based implementation. It enables us to image complex media using 3D depth migration with little concern for computational cost. The macrovelocity model can be built in a much shorter turnaround time.

INTRODUCTION

The one-way wave-equation method plays an important role in the fields of seismic modeling and depth migration (Claerbout, 1985; Wu, 1994; Xie and Wu, 2001). It can handle multipathing wavefields naturally (Le Rousseau and de Hoop, 2001) and has attractive storage demand with a depth-iterative algorithm (Claerbout, 1985). Unfortunately, 3D prestack depth migration is difficult to use routinely because of high computational cost. A typical 3D prestack depth migration might run for several weeks, even with powerful aid from high-performance computer (PC) clusters. Furthermore, building the macrovelocity model for depth migration requires many iterations of wavefield extrapolation (Shen and Symes, 2008). Therefore, computational efficiency of the wavefield propagator is still a major bottleneck in the practical application of one-way depth migration.

During the last three decades, various one-way wave-equation methods based on Fourier transforms have been developed for imaging complex media (Gazdag, 1978; Gazdag and Sguazzero, 1984; Stoffa et al., 1990; Wu, 1994; Huang et al., 1999a, Huang et al., 1999b; de Hoop et al., 2000; Huang and Fehler, 2000; Chen and Liu, 2004; Fu, 2005; Liu and Zhang, 2006; Zhang and Liu, 2007). Compared with the finite-difference method (Claerbout, 1985; Hale, 1991), the Fourier method is immune to the two-way splitting error (Brown, 1983) in 3D cases and has almost no numerical dispersion for coarse grids and high frequencies. The Fourier method also is relatively effective because of using fast Fourier transforms. The generalized-screen propagator (de Hoop et al., 2000; Le Rousseau and de Hoop, 2001) is a general form of the high-order Fourier method, but it is accurate only for weak velocity contrasts (Zhang et al., 2009). Liu and Zhang (2006) significantly improve the accuracy of the generalized-screen propagator by optimizing the constant coefficients while keeping the algorithm structure.

However, high-order terms are needed to handle wide-angle propagation in the presence of strong lateral velocity variations, which require many more 2D Fourier transforms to shuttle the wavefield between the spatial and wavenumber domains. Zhang et al. (2009) show that the third-order generalized-screen propagator (de Hoop et al., 2000) is slower than the Fourier finite-difference method (Ristow and Rühl, 1994), although the fastest CPU-based Fourier transform is used. Therefore, the speed of the Fourier transform is the major factor that impacts the computational efficiency of 3D Fourier migration. However, the 3D Fourier migration has great potential to

Manuscript received by the Editor 23 December 2008; revised manuscript received 10 July 2009; published online 15 December 2009.
[1]Chinese Academy of Sciences, Institute of Geology and Geophysics, Beijing, China. E-mail: zjh@mail.igcas.ac.cn; yaozx@mail.igcas.ac.cn.
[2]Central University of Nationalities, Institute of Information Engineering, Beijing, China. E-mail: cd-wx@163.com.

be accelerated in the presence of an enhanced fast algorithm of the Fourier transform.

In recent years, the computing capacities of graphics processing units (GPUs) have been improved enormously. With multiple cores driven by high-memory bandwidth, today's GPUs offer tremendous computational resources for graphics processing and general-purpose computations. The latest GPUs even can perform more than 500 billion floating-point operations per second, which is more than one order of magnitude faster than the latest Intel CPUs. The compiler and development tools, called compute unified device architecture (CUDA), provide an easily accessible way to generate parallel code for execution on GPUs. The CUDA-enabled GPU has been used in many fields of scientific computation (Stantchev et al., 2008), such as medical imaging (Muyan-Ozcelik et al., 2008) and molecular mechanics simulations (Stone et al., 2007; Yang et al., 2007). Recently, the CUDA-enabled GPU also has been used for explicit finite-difference depth migration (Hale, 1991) and Kirchhoff time migration (Li et al., 2009).

In this study, we present a GPU-based computing scheme to accelerate the 3D Fourier depth migration. We use the third-order optimized generalized-screen propagator (de Hoop et al., 2000; Liu and Zhang, 2006) to take advantage of the CUDA library of fast Fourier transform. For each frequency component, the propagator is coded as a sequence of CUDA kernels that can be called in terms of functions when the wavefield is downward extrapolated iteratively from the surface to the bottom of the velocity model. Each kernel is tailored for the specific architecture and parallel implementation on the GPUs. Numerical experiments show that our GPU-based parallel code runs 25 to 40 times faster than the traditional CPU-based serial code.

Our computing scheme is easy to use and implement. It performs fast Fourier transforms, the major parts of the Fourier depth propagator, by calling the CUDA library, and it involves using only the global memory of the graphics device, which can be used safely in an instant manner. This computing scheme provides us with a very simple way to accelerate depth migration using fashionable GPUs. It enables us to achieve a relatively high speedup ratio over the equivalent CPU-based algorithm but without expending too much effort in porting the existing code to the CUDA-enabled GPU.

First we review the generalized-screen propagator. Then we benchmark the speed of the GPU-based fast Fourier transform and the memory bandwidth of data transfer. Next we construct a computing scheme for 3D Fourier migration to cater to the CUDA implementation on GPUs. Finally, we demonstrate the proposed scheme using impulse responses and synthetic data of the SEG/EAGE salt model (Aminzadeh et al., 1996).

METHODOLOGY

Review of the generalized-screen propagator

The downward one-way wave equation for 3D depth migration reads (Claerbout, 1985)

$$\frac{\partial P(x,y,z;\omega)}{\partial z} = ik_z P(x,y,z;\omega), \tag{1}$$

with the square-root operator defined as $k_z = \sqrt{\omega^2 s^2 + \partial^2/\partial x^2 + \partial^2/\partial y^2}$, where the slowness s is the reciprocal of the velocity $v(x,y,z)$, $i = \sqrt{-1}$ is the imaginary unit, ω is the circular frequency, and $P(x,y,z;\omega)$ is the wavefield in frequency domain. The formal solution of equation 1 is

$$P(x,y,z+\Delta z;\omega) \approx \exp(ik_z \Delta z) P(x,y,z;\omega), \tag{2}$$

where Δz is the thickness of a thin horizontal slab (i.e., depth interval).

For laterally varying media, a constant reference velocity function $v_0(z)$ can be introduced to handle the homogeneous background for each depth step (Stoffa et al., 1990; Wu, 1994). Rewriting the operator k_z as $k_z = k_{z0}\sqrt{1 - (\omega^2 s_0^2 - \omega^2 s^2)/k_{z0}^2}$ and expanding it by Taylor expansion, we obtain

$$k_z = k_{z0} + k_{z0} \sum_{n=1}^{\infty} a_n \left(\frac{\omega^2 s_0^2 - \omega^2 s^2}{k_{z0}^2} \right)^n, \tag{3}$$

where $k_{z0} = \sqrt{\omega^2 s_0^2 + \partial^2/\partial x^2 + \partial^2/\partial y^2}$; $s_0 = 1/v_0(z)$ is the reference slowness, and a_n are constant coefficients with the first four being $a_1 = -1/2$, $a_2 = -1/8$, $a_3 = -1/16$, and $a_4 = -5/128$, respectively. Considering

$$\Delta s = s - s_0 = s_0 \sum_{n=1}^{\infty} a_n \left(1 - \frac{s^2}{s_0^2} \right)^n, \tag{4}$$

we obtain $k_z \approx k_{z0} + \omega \Delta s + k_z^{\mathrm{GS}}$, where $\Delta s = s - s_0$ is the perturbation of slowness between the real slowness s and the reference slowness s_0, and

$$k_z^{\mathrm{GS}} = \omega \sum_{n=1}^{N} a_n (s_0^2 - s^2)^n \left(\frac{\omega^{2n-1}}{k_{z0}^{2n-1}} - \frac{1}{s_0^{2n-1}} \right) \tag{5}$$

is the Nth-order propagator of the generalized-screen method (de Hoop et al., 2000).

The formal solution of the one-way wave equation, equation 2, can be decomposed for laterally varying media into three cascaded equations,

$$P'(x,y,z+\Delta z;\omega) = \exp(ik_{z0}\Delta z) P(x,y,z;\omega), \tag{6}$$

$$P''(x,y,z+\Delta z;\omega) = \exp(i\omega \Delta s \Delta z) P'(x,y,z + \Delta z;\omega), \tag{7}$$

and

$$P(x,y,z+\Delta z;\omega) = \exp(ik_z^{\mathrm{GS}}\Delta z) P''(x,y,z+\Delta z;\omega). \tag{8}$$

Equation 6 performs the phase shift for the reference slowness s_0 in the wavenumber domain (Gazdag, 1978); equation 7 performs the time-delay correction for slowness perturbations in the spatial domain (Stoffa et al., 1990; Wu, 1994); and equation 8 handles the high-order corrections for large velocity contrasts and wide-angle propagations (Le Rousseau and de Hoop, 2001).

In the implementation, however, another Taylor expansion (i.e., $e^x \approx 1 + x$) is required on the exponential function in equation 8 to explicitly separate the spatial and wavenumber variations (Le Rousseau and de Hoop, 2001), i.e., the generalized-screen correction

$$P(x,y,z+\Delta z;\omega) \approx (1 + ik_z^{\mathrm{GS}}\Delta z) P''(x,y,z+\Delta z;\omega). \tag{9}$$

De Hoop et al. (2000) propose a normalization operator $\mathcal{N}$ to handle the stability and to reduce the phase error caused by the Taylor expansion used in equation 9, which reads

$$\mathcal{N}(1+p+iq)=\exp(iq)\left|1+\frac{p}{1+iq}\right|^{-1}\left(1+\frac{p}{1+iq}\right), \tag{10}$$

where p and q denote the real part and imaginary part of a complex number, respectively.

Accuracy of the third-order optimum split-step Fourier propagator

Phase error versus velocity contrast is shown in Figure 1 following the work by Zhang et al. (2009). Obviously, the low-order generalized-screen propagator is accurate enough for wide-angle propagation in media with weak velocity contrast; however, a high-order generalized-screen propagator is needed to handle the wide-angle propagation in media with moderate and strong velocity contrasts. Liu and Zhang (2006) significantly improve the accuracy of the generalized-screen propagator by optimizing the constant coefficients a_n while retaining the algorithm structure. The resultant propagator, named the optimum split-step Fourier propagator (OSP), enables us to image much steeper dips in complex media.

The fourth-order generalized-screen propagator tends to be unstable in the presence of strong velocity variations (Zhang et al., 2009). Consequently, we restrict the expansion used in this study to only the third order. We obtain the optimized parameters, $a_1=-0.3710$, $a_2=-0.1413$, and $a_3=-0.2311$, using the optimization scheme proposed by Liu and Zhang (2006). As shown in Figure 1, the third-order OSP is a significant improvement over the generalized-screen method and even is superior to the Fourier finite-difference method for a wide range of lateral velocity variations. The third-order generalized-screen method is superior to the Fourier finite-difference method only when the velocity contrast $(v\text{-}v_0)/v\times 100\%$ is smaller than 18% (see the vertical dotted line on the right in Figure 1).

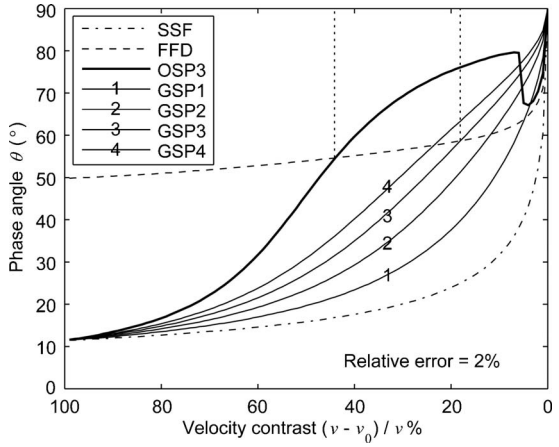

Figure 1. Velocity contrast versus phase angle of propagators under relative phase error of 2%. The velocity contrast is defined as $(v\text{-}v_0)/v\times 100\%$. A small velocity contrast denotes weak lateral velocity variations, and a big one denotes strong lateral velocity variations. The dashed-dot line denotes the split-step Fourier method (SSF); the dashed line denotes the Fourier finite-difference method (FFD) with alternating-direction-implicit (ADI) — also called two-way splitting — plus interpolation; the bold solid line denotes the third-order optimum split-step Fourier propagator (OSP) used in this study; the thin solid lines indicated by 1–4 denote the first four orders of the generalized-screen propagator (GSP). Above the line, the error is larger than the chosen relative error; below, it is smaller.

In contrast, the optimized third-order OSP is superior to the Fourier finite-difference method when the velocity contrast $(v\text{-}v_0)/v\times 100\%$ is smaller than 45% (see the vertical dotted line on the left in Figure 1). The maximum accurate dip angle of the third-order OSP is higher than that of the Fourier finite-difference method by as much as 18°. The third-order OSP enables us to handle wide-angle propagation in 3D complex media with moderate velocity contrasts.

Algorithm of the third-order OSP

Figure 2 shows a sketch map of downward extrapolation using the third-order OSP. For each frequency component ω, the wavefield extrapolation from depth z to $z+\Delta z$ is implemented as

$$\begin{aligned}\bar{P}(k_x,k_y,z;\omega)&=F^{+}_{x,y}[P''(x,y,z;\omega)]\\&\times\mathcal{N}\left\{1+i\Delta z\omega\sum_{n=1}^{3}\left(\frac{\omega^{2n-1}}{\bar{k}_{z0}^{2n-1}}-\frac{1}{s_0^{2n-1}}\right)\right.\\&\left.\times\frac{F^{+}_{x,y}[a_n(s_0^2-s^2)^nP''(x,y,z;\omega)]}{F^{+}_{x,y}[P''(x,y,z;\omega)]}\right\}\end{aligned} \tag{11}$$

and

$$\begin{aligned}P''(x,y,z+\Delta z;\omega)&=\exp(i\omega\Delta s\Delta z)F^{-}_{x,y}\\&\times[\exp(i\bar{k}_{z0}\Delta z)\bar{P}(k_x,k_y,z;\omega)],\end{aligned} \tag{12}$$

where $\bar{k}_{z0}=\sqrt{\omega^2s_0^2-k_x^2-k_y^2}$ is the vertical wavenumber in reference media with k_x and k_y being horizontal wavenumbers; and $F^{+}_{x,y}$ and $F^{-}_{x,y}$ denote 2D forward and inverse Fourier transforms along horizontal space, respectively. In the phase-screen method (equation 12) and the generalized-screen corrections (equation 11), the terms associated with spatial coordinates, $(s_0^2-s^2)^n$ or Δs, are explicitly separated from the terms associated with wavenumber variations $\bar{k}_{z0}$.

Thus the wavefield extrapolation can be implemented as a dual-domain procedure in the frequency-space and frequency-wavenum-

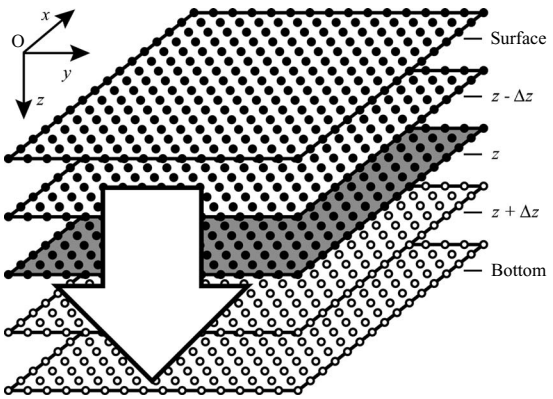

Figure 2. Sketch map of the wavefield extrapolation from surface to bottom for an independent frequency component. The big arrow shows the iterative direction of wavefield extrapolation. Five depth levels exist with the interval of Δz. The gray plane denotes the current extrapolation step at the depth of z. The filled circles denote the extrapolated wavefield on grids, and the hollow circles denote the wavefield to be extrapolated.

ber domains alternately. The terms that contain spatial coordinates are implemented in the spatial domain, and the terms that contain horizontal wavenumbers are implemented in the wavenumber domain. In each domain, only pointwise operations are involved. Fast Fourier transforms are used to transform wavefields between these two domains. The split-step Fourier propagator requires a forward and an inverse 2D Fourier transform for shuttling wavefields between these two domains as shown in equation 12. Each additional term of the generalized-screen correction requires an additional forward Fourier transform as shown in equation 11. Thus the Nth-order OSP requires $N + 2$ Fourier transforms for wavefield extrapolation at each depth level.

Figure 3 shows the flowchart of one-way depth migration for zero-offset records. The downward extrapolation is the core of one-way depth migration and is executed for each depth and for each frequency. Thus, the number of 2D Fourier transforms for 3D poststack migration using the third-order OSP is $5 \times N_\omega \times N_z$, where N_ω is the number of frequency components and N_z is the number of depth steps. For shot-gather prestack migration using the third-order OSP, the number of 2D Fourier transforms is $10 \times N_\omega \times N_z \times N_{\text{shot}}$, which is as high as several billions, where N_{shot} is the total number of shots. Furthermore, migration velocity analysis commonly is needed for high-accuracy depth migration, which means additional multiples of several tens (Shen and Symes, 2008).

ACCELERATING WITH GPU

Overview of GPU and CUDA

Graphics processing units consist of a cluster of processors attached to a graphics card for extremely fast processing of large graphics data sets. The GPUs feature optimized hardware architecture for simultaneously performing a large number of independent arithmetic operations in parallel mode. In contrast, the CPUs feature optimized hardware architecture for more general operations in serial mode, including data caching and flow control. The GPUs own many more transistors devoted to data processing than the CPUs do,

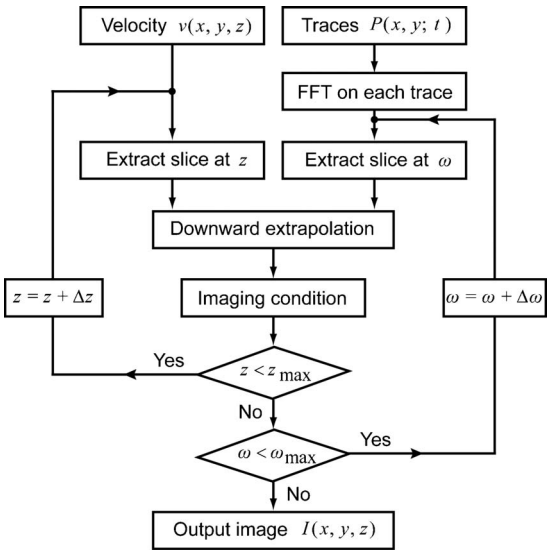

Figure 3. A CPU-based flowchart of 3D poststack depth migration. The velocity model has a maximum depth of $z_{\max}$ with a depth interval of Δz. The seismic record has a maximum frequency of $\omega_{\max}$ with a frequency interval of $\Delta\omega$.

and the highly parallel structure makes modern GPUs more attractive than general-purpose CPUs for intensive and highly parallel computations. In recent years, the computing capabilities of GPUs have been improved dramatically compared with general-purpose CPUs. The peak floating-point operations per second of GPUs now are about ten times those of CPUs. The graphics card with GPUs has been used successfully as a coprocessor to speed up nongraphics applications, especially for parallel scientific computations.

Originally, the programmability of GPUs was very limited because of involving graphics-oriented details (Stone et al., 2007; Muyan-Ozcelik et al., 2008). Recently, NVIDIA (2009) provided a friendly development environment, named CUDA, which allows the programmer to think in terms of memory and operation as in traditional CPU programs. Thus, the implementation of general-purpose applications on the GPU has become much easier. Figure 4 shows the software architecture of CUDA-enabled GPU programming. The CUDA uses the C programming language to define device functions, named kernels. These kernels are called by the host (i.e., the computer host), similar to calling as regular C functions, but are executed on the device (i.e., the graphics device) in parallel mode by multiple threads. A warp is the scheduling unit in the streaming multiprocessors, and it manages threads in groups of 32. Kernels run on a grid of blocks, and each block contains many warps. In implementation, each block is mapped to a multiprocessor, and each thread is mapped to a single processor.

Several types of memory exist on GPUs, and each type has its own benefits and limitations. In this study, we take advantage of only the global memory, as the global memory space can contain large-volume data sets. However, the global memory is not cached; thus it is important to follow the right access pattern to obtain maximum memory bandwidth. If memory accesses are coalesced (NVIDIA, 2009), all the threads of a half-warp will access the memory simultaneously so that the performance will increase significantly. Otherwise, with a noncoalesced pattern, the time consumption of global memory access is about one order of magnitude higher.

Third-dimensional depth migration based on the one-way wave equation is well suited to CUDA implementation on GPUs because it

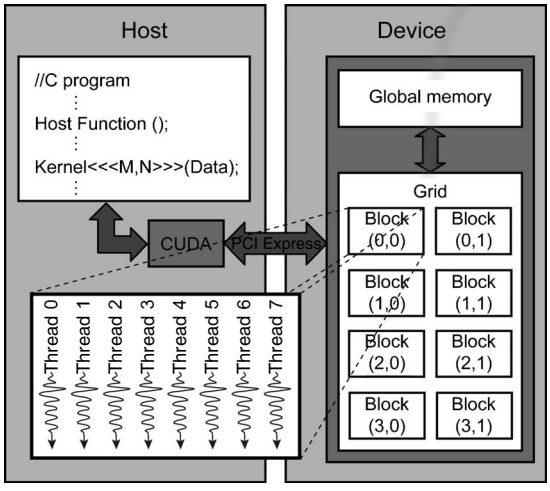

Figure 4. Software architecture of CUDA-enabled GPU programming. The left part of the figure denotes the computer host, and the right part denotes the graphics device. PCI Express denotes the PCI interface between the host and the device. The host and device specifications are listed in Tables 1 and 2, respectively.

is data parallel and computationally intensive. In addition, the efficiency of the OSP method is highly dependent on the speed of the Fourier transform, whereas a GPU-based parallel algorithm of the fast Fourier transform is available in the CUDA library. Tables 1 and 2 list the specifications of the host and the device used in our numerical experiments, respectively. We use only one core of the dual-core CPU, and the wall-clock time is measured without considering the disk input/output (I/O).

Benchmarking the fast Fourier transform

The CUDA distribution package includes a built-in software implementation of the fast Fourier transform, named the cuFFT library, which is a parallel implementation of the widely used CPU-based fast Fourier transform, named the FFTW library (Frigo and Johnson, 1998). We measure the time consumption of the cuFFT and the FFTW to evaluate the potential speedup. For simplicity, only results for a square data set are shown in Figure 5. The integer power of 2 varies from 7 through 11; that is, the number of samples in both inline and crossline directions is 128, 256, 512, 1024, or 2048 points, respectively. The average time consumption of each fast Fourier transform is obtained by executing forward and inverse fast Fourier transforms for 1000 times.

As shown in Figure 5, the cuFFT always is faster than the FFTW for all listed data sets, and this trend is more significant for a data set of larger size. For data sets of the dimensions 128×128 and 256×256, the cuFFT is only several times faster than the FFTW; however, for data sets of the dimensions 512×512, 1024×1024, and 2048×2048, it is as much as 50 times faster than the FFTW. The FFTW has relatively high speed when the size of the data set is not bigger than 256×256 (Frigo and Johnson, 1998), whereas a small-size cuFFT has too low computational intensity to develop the high potential of the parallelism on the GPUs. Therefore, the cuFFT for a small-size data set has less speedup than the FFTW compared with a large-size data set.

Benchmarking the memory bandwidth of data transfer

The host and the device are connected using PCI Express, which has a maximum bandwidth of 6.4 gigabytes per second (GB/s). In contrast, the global memory (DDR3) on the device has a maximum bandwidth of 102 GB/s. Our benchmarking of memory bandwidth shows that small-size data sets have a low bandwidth compared with large-size data sets for data transfer either between the host and the device or within the device. For example, the bandwidth using PCI Express is about 2 GB/s for the data size of 64 K and is about 5 GB/s for 32 megabytes (MB) (a float array of 128×128, e.g., a velocity slice, requires 64 K of memory, and a complex array of 2048×2048 requires 32 MB of memory). In contrast, the bandwidth of data transfer within the device is 22 GB/s for the data size of 64 K and is more than 100 GB/s for 32 MB.

As shown in Figure 5, the cuFFT is 39 to 51 times faster than the FFTW for a large data set when the data transfer between the host and the device is not involved. However, the cuFFT is only 18 to 19 times faster than the FFTW when the data transfer between the host and the device is involved; that is, the time consumption caused by the data transfer between the host and the device is larger than the time consumption caused by the cuFFT. Therefore, we should minimize data transfer between the host and the device to achieve a high speedup. In addition, the data set should be as large as possible to obtain a high bandwidth if data transfer is necessary between the host and the device. We also should create intermediate data in the device memory without ever being visited by the host.

Table 1. Host specifications

Processor (CPU)	Intel Core2 Duo 2.53 GHz
Memory	2 GB, 800 MHz DDR2
Motherboard	Colorful C.P35 X7 Ver2.0
PCI interface	PCI Express GEN2×16

Table 2. Device specifications (GPUs)

Model	NVIDIA GTX280
Global memory	1 GB, 1107 MHz DDR3
Number of multiprocessors	30
Threads per multiprocessor	1024

CUDA kernels of the third-order OSP

For one-way depth migration, the whole 3D model is divided into a serial of 2D horizontal slabs along the depth direction, and the generation of the wavefield in each slab requires the wavefield in the latest slab and the velocity in the current slab. The corresponding GPU implementation consists of the following four stages:

1) Upload the depth slice of the velocity and the frequency slice of the wavefield from the host to the device.
2) Perform wavefield extrapolation on the device by calling a serial of kernels on the host.
3) Download the extrapolated wavefield from the device to the host.
4) Apply imaging conditions on the host.

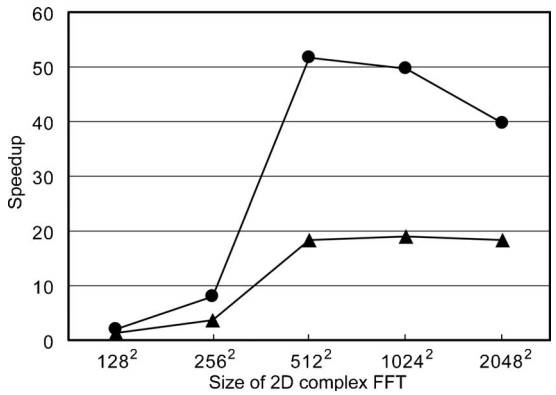

Figure 5. Speedup of the GPU-based fast Fourier transform (i.e., cuFFT) over the CPU-based one (i.e., FFTW). The average time consumption of 2D complex fast Fourier transform on a square data set is obtained by executing forward and inverse fast Fourier transforms for 1000 times. The filled circles on the upper line denote the speedup of the cuFFT over the FFTW when there is no data transfer between the host and the device for the cuFFT. The filled triangles on the bottom line denote the speedup of the cuFFT over the FFTW when data transfer between the host and the device is applied before the forward transform and after the inverse transform.

This implementation is capable of handling various sizes of 3D velocity models because only several 2D slices of both velocity model and wavefield are stored in the device memory. However, this implementation is less improved in efficiency because there are too many data transfers of the small-size data set between the host and the device. The efficiency can be improved if the whole 3D velocity model and the imaged data are stored in the device memory, where data transfer of the small-size data set between the host and the device would be minimized. Unfortunately, the available amount of device memory is limited; thus we should strive to minimize the memory occupied by the 3D velocity model and imaged data. A feasible way is to apply data compression.

Many advanced algorithms of data compression exist in digital image processing. However, most of them are costly in either compression or decompression procedures. Consequently, they are not applicable to the data compression of the velocity model and image data unless their GPU-accelerated algorithms are provided. In fact, the values of velocities in the model always are positive and usually vary within a fixed small range (e.g., 1000–6000 m/s), and the values of image data always vary within a range of −1 to 1. We suggest two extremely efficient data compression/decompression schemes to reduce the memory demand on the GPU by using data-type conversion.

Data compression

The data type of the velocity array usually is defined as float, which requires four bytes for each element. The unsigned integer type defined by CUDA requires only two bytes for each element. Thus we compress the 3D velocity model in the device memory by declaring its array as the unsigned integer type instead of the float type. The compression/decompression scheme consists of five steps:

1) Load the 3D velocity model into a float array $v(x,y,z)$ on the host.
2) Define a scale factor as $r = 65{,}535/(v_{\max} - v_{\min})$, where the constants $v_{\min}$ and $v_{\max}$ are the minimum and maximum values of the 3D velocity model, respectively.
3) Compress the float array into an unsigned integer array by first subtracting the minimum velocity $v_{\min}$ of the whole model and then rounding the float number to the nearest integer:

$$v3i(x,y,z) \leftarrow Int\{[v(x,y,z) - v_{\min}] \times r\}.$$

4) Upload the constants $v_{\min}$, r, and the integer array $v3i(x,y,z)$ to the device.
5) Extract the 2D velocity slice at depth z from $v3i(x,y,z)$ before performing wavefield extrapolation on the device and recover as

$$v2f(x,y) \leftarrow v3i(x,y,z)/r + v_{\min}.$$

This compression to a 3D velocity model maps the velocity variations of $0 \sim (v_{\max} - v_{\min})$ to the unsigned integer range of 0–65,535. The error, caused by rounding the float to the integer in the third step, is proportional to the range of velocity variations $(v_{\max} - v_{\min})$. In practice, the absolute error caused by rounding is smaller than 0.05 m/s for velocities ranging from 1500 to 4500 m/s, and the relative error is smaller than 0.0033%. Therefore, this compression of the velocity model is feasible for most practical applications.

Another large-volume data set stored in the device memory is the image array. The amplitudes of 3D image data range from −1 to 1. The short integer type (from −32,767 to 32,767) defined by CUDA requires only two bytes for each element, rather than four bytes for the float type. Thus we use the short integer array to store the image data in the device memory. First the image data in each depth slice are scaled by 32,767. Then they are accumulated into the short integer array of 3D image data when applying imaging conditions. Finally, the 3D image data are divided by 32,767 after transferring back to the host. The relative error caused by this procedure is smaller than 0.0031%.

Figure 6 shows the GPU-based flowchart of depth migration by reusing 3D data sets on GPUs with data compressions. By using these two compressions to the 3D velocity model and image data, we save half the memory demand for large-volume data stored in the limited device memory; thus the capable model size is doubled. More importantly, they enable us to minimize the time consumption caused by the low-bandwidth data transfer between the host and the device for the larger 3D velocity model.

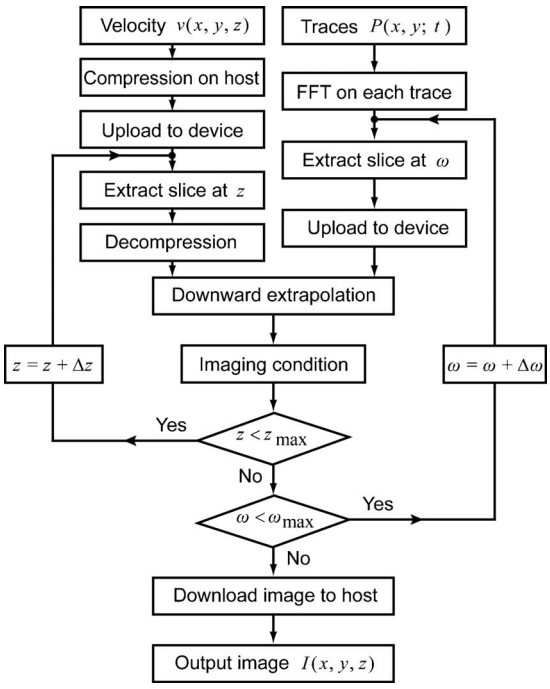

Figure 6. A GPU-based flowchart of 3D poststack depth migration by reusing 3D data sets on GPUs with data compressions. The maximum depth of the velocity model is $z_{\max}$ with a depth interval of Δz. The maximum frequency of the seismic record is $\omega_{\max}$ with a frequency interval of $\Delta\omega$.

2D tapered function

An absorbing boundary condition is required to reduce numerical artifacts caused by the boundaries of a limited model. Masking with a tapered function to each side of the wavefield (Cerjan et al., 1985) is popular and necessary to satisfy the periodicity requirement inherent in the Fourier-based migration (Wild et al., 2000). Commonly, only the samples of the attenuation function are stored in the memory and are applied to each side of the individual row and column. As a passband, the center area is excluded to reduce computational cost. This procedure is cost-effective for the CPU-based implementation; however, it is costly for the GPU-based implementation for three

reasons. First, this procedure consists of fewer parallelisms but more serial operations. Second, it requires too many flow control instructions on judging array indices; the GPU-based code is less efficient in flow control compared with the CPU-based code. Third, it leads to a random memory access pattern that is much slower than a coalesced memory access pattern.

To reduce the cost of applying absorbing boundary conditions on GPUs, we should adapt to the parallel-computation architecture on the device and obey the special rules of the memory access patterns. We use a 2D tapered function, shown in Figure 7, instead of the commonly used 1D function. Samples in passband and attenuation band are stored in a 2D array. This 2D tapered function is masked to the wavefield by pointwise multiplication when the absorbing boundary condition is applied. It allows fast parallel implementation on the GPU without any flow control or a random memory access pattern; thus it is very efficient, although some fruitless computations are performed in the passband.

Compact architecture

Numerical experiments show that a compact kernel containing more instructions is significantly faster than a sequence of kernels, although they fulfill the same function. Therefore, we should incorporate several scattered kernels into a compact kernel as much as possible. We incorporate the 2D tapered function and the scaling of inverse Fourier transform into the time-shift kernel. In addition, we incorporate the phase shift into the wavenumber-associated high-order correction kernel. Unfortunately, the cuFFT could not be called in the kernel produced by the programmer. Consequently, we have to exclude each cuFFT from the phase-shift kernel and the time-shift kernel, although this leads to redundant and less effective code.

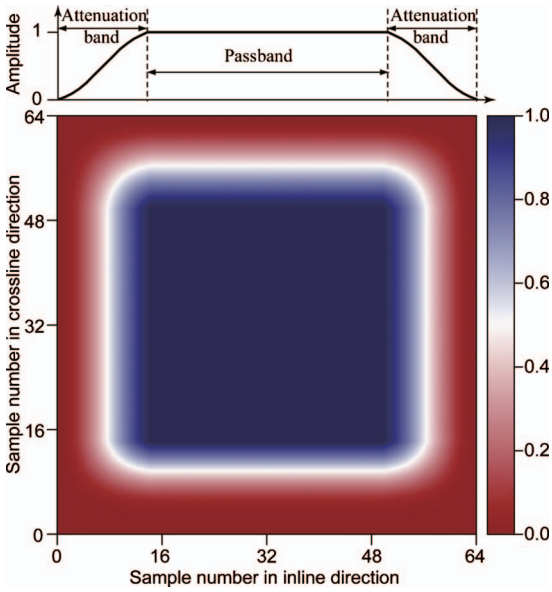

Figure 7. A 2D tapered function for the parallel implementation of the absorbing boundary condition. This function is generated in the shape of a Hanning window. A unitary 2D array is scaled by the 1D tapered function (shown in the upper part of Figure 7) first along the inline direction and then along the crossline direction. The samples in inline and crossline directions number 64 with the attenuation band of 15 samples on each side. Masking with this 2D tapered function executes much faster than separately applying the row- and column-based 1D tapered function because the former actually is parallel when executing on the GPU, but the latter involves a costly random memory access pattern.

Figure 8 shows the pseudocode that applies the data compressions, the 2D tapered function, and the compact scheme. The pseudocode of copying each slice without the data compressions is not shown because it can be obtained easily from Figure 8. This pseudocode is consistent with the GPU-based flowchart shown in Figure 6. The outer loop is over the independent frequency components. The inner loop performs depth extrapolation iteratively from the surface to the bottom of the model. Twelve kernels are included in the inner loop for the third-order optimum split-step Fourier propagator. Five kernels associated with forward or inverse Fourier transforms are implemented by directly calling the CUDA library

```
// Compress velocity on host and upload to device
r = 65535/(v_max − v_min);                                   // Initialize scale factor
v3i(x,y,z) ⇐ Int{[V(x,y,z) − v_min]×r};                      // Compress velocity
i(x,y,z) <<< 0;                                              // Initialize image array
ω ← 0;
for each frequency ω do                                      // Loop over frequency
    p(x,y) ⇐ P(x,y;ω);                                       // Upload to device
    z ← 0;
    for each depth z do                                      // Loop over depth
        // Extract and decompress velocity slice
        d(x,y) <<< 1/v_0^2 − 1/[v3i(x,y,z)/r + v_min]^2;
        p_1(x,y) <<< a_1 p(x,y) d(x,y);                      // First order
        p̄_1(k_x,k_y) <<< F^+_{x,y}[p_1(x,y)];                // 2D forward FFT
        p_2(x,y) <<< a_2 p(x,y) d^2(x,y);                    // Second order
        p̄_2(k_x,k_y) <<< F^+_{x,y}[p_2(x,y)];                // 2D forward FFT
        p_3(x,y) <<< a_3 p(x,y) d^3(x,y);                    // Third order
        p̄_3(k_x,k_y) <<< F^+_{x,y}[p_3(x,y)];                // 2D forward FFT
        p̄(k_x,k_y) <<< F^+_{x,y}[p(x,y)];                    // 2D forward FFT
        //Normalization and phase shift
        p̄_1(k_x,k_y) <<< p̄(k_x,k_y) exp(ik̄_{z0}Δz)×
        N[1 + iΔzω Σ_{n=1}^{3} p̄_n(k_x,k_y)(ω^{2n−1}/k̄_{z0}^{2n−1} − v_0^{2n−1})/p̄(k_x,k_y)];
        p(x,y) <<< F^−_{x,y}[p̄_1(k_x,k_y)];                  // 2D inverse FFT
        // Scaling, absorbing boundary, and time shift
        p(x,y) <<< p(x,y) exp(iωΔsΔz) t(x,y)/(N_x N_y);
        // Scaling and imaging
        i(x,y,z) <<< i(x,y,z) + Int[p(x,y)×32767];
        z ← z + Δz;                                          // Depth increase
    end
    ω ← ω + Δω;                                              // Frequency increase
end
I(x,y,z) ⇐ i(x,y,z)/32767;                                   // Download to host
```

Figure 8. Pseudocode of the GPU-based third-order optimum split-step Fourier propagator (OSP). The array on the host is named using a capital letter, and the array on the device is named using lowercase letters. The sign "⇐" denotes data transfer between the host and the device, and the sign "←" denotes assigning the value on the right side to the variable on the left side. The part on the right side of the sign "⋘" denotes the kernel execution on the device, and the array on the left side stores the results. The operator $Int(\cdot)$ rounds a float number to the nearest integer, and the operator $\mathcal{N}(\cdot)$ denotes the normalizing operator. The constants $v_{\min}$ and $v_{\max}$ are the minimum and maximum values of the 3D velocity model, respectively. The reference velocity $v_0(z)$ is the minimum in the slice of 3D velocity at depth z, which is used for the phase shift in a homogeneous background.

(i.e., the cuFFT), and other kernels are cascaded to perform pointwise operations either in the frequency-space domain or in the frequency-wavenumber domain.

NUMERICAL EXAMPLES

Migration impulse responses

In this section, we illustrate the proposed GPU-based scheme on three aspects using impulse responses: first, the numerical precision; second, the performance of the third-order OSP in handling strong velocity contrast; and third, the speedup over the CPU-based scheme. A 3D homogeneous medium is defined on a grid system of the dimensions $256 \times 256 \times 128$ with grid spacing of 10 m. The real velocity is $v = 3000$ m/s with the reference velocity being $v_0 = 1500$ m/s, i.e., velocity contrast $(v\text{-}v_0)/v = 50\%$. All input traces are zeros except that the central trace has a Ricker wavelet with the dominant frequency of 25 Hz. The time delay of the wavelet is 375 ms with the sampling interval of 2 ms. Eighty frequency components are calculated. The 2D tapered function is used in the GPU-based code, and the 1D tapered function is used in the CPU-based code. The attenuation band has 15 samples on each side of the 2D wavefield.

Figure 9 shows the difference in the normalized images obtained

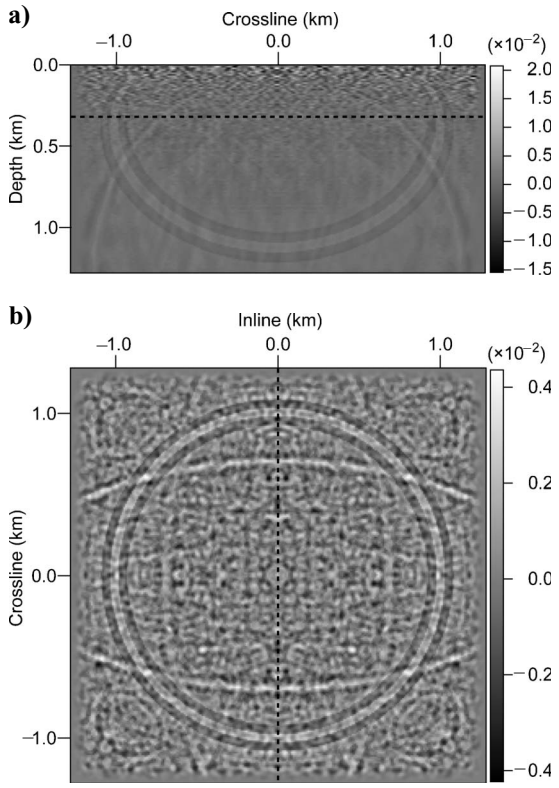

Figure 9. The difference in normalized images obtained by the CPU and GPU implementations: (a) vertical slice along the crossline direction at the inline position of 0 m; (b) depth slice at 640 m. The intersection of the vertical and horizontal slices is shown by a dashed line in each slice. The CPU implementation uses the FFTW, the 1D tapered function but without data compression, and the GPU implementation uses the cuFFT, the 2D tapered function, as well as the data compressions of the 3D velocity model and imaged data on the device.

by CPU and GPU implementations. In the vertical slice (a), most parts have small errors that are smaller than 0.5%, but apparent errors arise in the upper side, and the maximum error even reaches 2%. No wavenumber filter is used in our codes. If a wavenumber filter is applied, the apparent error at the upper side of vertical slice (a) would be reduced greatly. In the horizontal slice (b), the maximum error is 0.4%. Apparent errors exist at the positions of boundary reflections in the crossline direction (see the two circular arcs) but not the inline direction. This shows that the performance of the 2D tapered function is slightly different from that of the 1D one.

In the CPU-based implementation, the 1D tapered function is applied after the wavefield extrapolation at depth z has finished. In the GPU-based implementation, however, the 2D tapered function $t(x,y)$ shown in Figure 8 is applied when the wavefield extrapolation is performing in parallel mode, and thus the boundary absorbing on some positions might have finished before the wavefield extrapolation. Thus some differences exist between the 1D and 2D tapered functions in their actual performances. Fortunately, the differences are negligible, as they are much smaller than 0.4%. In general, only some distortions exist at the background or at the boundary reflections, and the error generally is smaller than 0.4%. Therefore, the accuracy is well kept after using our computing scheme compared with the original CPU-based computation.

The Fourier finite-difference method (Ristow and Rühl, 1994; Biondi, 2002) is well known in imaging complex media with strong velocity contrast. It is selected as a reference to evaluate the relative performance of the third-order OSP. The two-way splitting error (Brown, 1983) is removed using the wavenumber interpolation technique (Wang, 2001; Zhang et al., 2008). Figure 10 contains the slices obtained from four methods: the Fourier finite-difference method, the third-order OSP, the second-order generalized-screen method, and the fourth-order generalized-screen method. Both depth and vertical slices show that the optimized parameters can significantly improve the accuracy of the generalized-screen propagator. The third-order OSP is even more accurate than the fourth-order generalized-screen propagator. The accurate angle of the third-order OSP is as high as 50° when the velocity contrast is 50%, which is slightly lower than that of the Fourier finite-difference propagator. These analyses are consistent with the previous relative error analyses shown in Figure 1.

To check the speedup of the GPU-accelerated third-order OSP over the CPU-based one, we tested on three additional models with the dimensions $128 \times 128 \times 64$, $512 \times 512 \times 256$, and $1024 \times 1024 \times 512$, respectively. The speedup of the GPU implementation over the equivalent CPU implementation is measured by the ratio of wall-clock times without considering the disk I/O (see Tables 1 and 2 for CPU and GPU configurations). Figure 11 shows the results. Obviously, the scheme that copies slices of the velocity model and imaged data to the device allows a much larger size of 3D model running on the GPU. However, its speedup generally is lower than that of a scheme that uploads the whole 3D velocity model to the device before migration and downloads the whole set of 3D imaged data back to the host after migration. For example, the speedup of copying each slice is only 17 for the $256 \times 256 \times 128$ model; in contrast, the speedup of copying the whole is about 27 for the same model. For another example, the speedup of copying each slice is only 25 for the $512 \times 512 \times 256$ model; in contrast, the speedup of copying the whole is about 37 for the same model.

The speedup of migration impulse response is somewhat different from the speedup of the cuFFT over the FFTW shown in Figure 5.

For a small-size data set (e.g., 128×128 or 256×256), the total speedup of the algorithm is higher than the speedup of the cuFFT over the FFTW; whereas, for a large-size data set (e.g., 512×512 or 1024×1024), the total speedup of the algorithm is lower than the speedup of the cuFFT over the FFTW. This shows that the Fourier transform is the most time-consuming part for the CPU-based algorithm of Fourier depth migration but not for the GPU-based algorithm, and other parts besides the Fourier transform would have more effect on the total speedup. The speedup of other parts lies between the speedup of the cuFFT over the FFTW for a small-size data set and that for a large-size data set.

Migration for the SEG/EAGE salt model

To verify accuracy and efficiency of the GPU-based third-order OSP on imaging 3D complex structures, we ran tests on zero-offset records (Ober et al., 1997) of the SEG/EAGE salt velocity model (Aminzadeh et al., 1996). The 3D grid system used here is of the dimensions 250×250×210 with a spacing of 40 m along the transversal direction and 20 m along the depth direction. Eighty frequency components are calculated. The 2D tapered function is used for the GPU-based code, and the 1D tapered function is used for the CPU-based code.

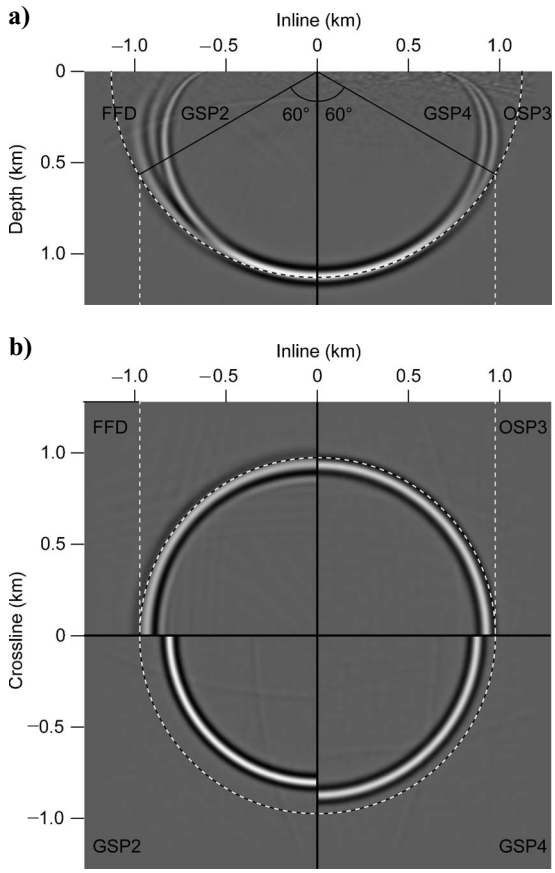

Figure 10. (a) Vertical slice, and (b) depth slice, from 3D migration impulse response. The dashed circle (or semicircle) denotes the exact position. The left part of Figure 10a shows the superposition of the vertical slices obtained by the Fourier finite-difference method with alternating-direction-implicit (ADI) plus interpolation (indicated by FFD) and the second-order generalized screen propagator (indicated by GSP2). The right part of Figure 10a shows the superposition of the vertical slices obtained by the fourth-order generalized screen propagator (indicated by GSP4) and the third-order optimum split-step Fourier propagator (indicated by OSP3). The horizontal slice consists of four equivalent parts: the upper left quadrant shows the Fourier finite-difference method with ADI plus interpolation (indicated by FFD); the bottom left quadrant shows the second-order generalized screen propagator (indicated by GSP2); the bottom right quadrant shows the fourth-order generalized screen propagator (indicated by GSP4); the upper right quadrant shows the third-order optimum split-step Fourier propagator (indicated by OSP3).

Figure 12 shows the vertical slice (at the inline position of 5000 m) and horizontal slice (at the depth of 2010 m) of the 3D velocity model and corresponding slices of the image obtained by the GPU-based third-order OSP. Obviously, salt boundaries and the structures under the salt body are well imaged except that some artifacts still exist in the salt body. Of course, the sharp peaks on the salt boundary are not well focused. This is because velocity contrasts and the dip angle at those positions exceed the upper limit of the third-order OSP's capabilities (see the left part of the bold black line in Figure 1). This result is comparable to the result obtained by the Fourier finite-difference method (Zhang et al., 2009).

The code of the CPU-based third-order OSP runs 595.02 s, whereas the GPU-accelerated code runs 18.52 s. The latter runs 32 times faster than the former does. This speedup overall is consistent with the results shown in Figure 11.

DISCUSSION

We accelerate the wavefield extrapolation using GPUs by fully taking advantage of the coalesced global memory access and the CUDA library of Fourier transforms. A very attractive point for our computing scheme is that it is easy to use and implement because only the global memory is used to pursue safely porting in an instant manner. Of course, great potential still exists to improve the speedup ratio by correctly using shared memory and registers within a block. However, the effective use of shared memory typically requires a complete overhaul of the algorithm and its mapping to the GPUs. This might be impractical for most geophysicists; thus we need a trade-off between the speedup ratio and the feasibility for practical applications. We tend to achieve a relatively high speedup ratio over the equivalent CPU-based algorithm, but without expending too

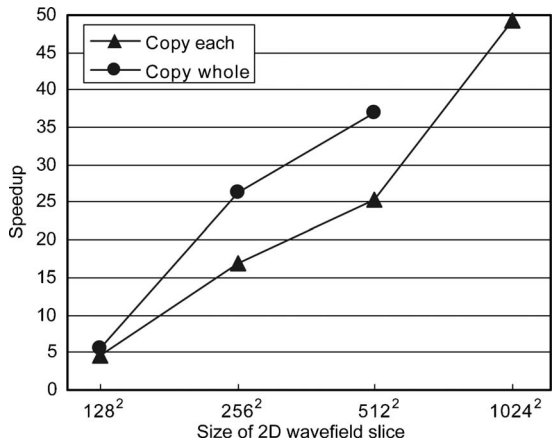

Figure 11. Speedup of the GPU-based third-order OSP over the CPU-based one. Only the velocity models with dimensions 128×128×64, 256×256×128, and 512×512×256 are tested for the scheme with data compressions (see circles on the upper line) because of the memory limitation. The triangles on the lower line denote the scheme copying each slice separately.

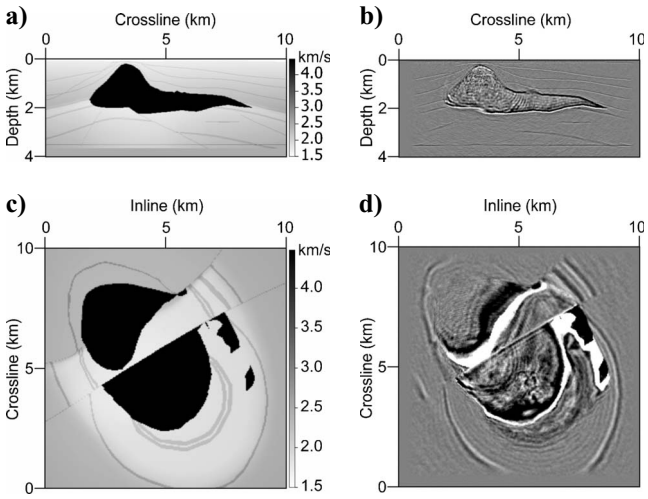

Figure 12. Migration test on the 3D SEG/EAGE salt model using the GPU-based third-order optimum split-step Fourier propagator (OSP). (a) Vertical profile of model along the crossline direction at the inline position of 5000 m, and (b) corresponding image. (c) Depth slice of model, and (d) corresponding image, at the depth of 2010 m.

much effort in porting the existing code to CUDA-enabled GPUs.

Although only the third-order OSP is illustrated, most elements of the proposed scheme can be applied easily to other kinds of Fourier-based 3D migrations. It is easy to extend all techniques to prestack migration because the prestack migration (e.g., shot-gather prestack migration) contains two similar parts (only the sign and the input data are different between downward and upward extrapolations). It is easy also to extend our scheme to multi-GPU implementation to obtain a much higher speedup ratio.

CONCLUSIONS

Cost is historically a major factor that inhibits the routine use of 3D wave-equation migration in practice. For example, the velocity updating requires several tens of iterations of 3D depth migration, and each iteration could run for several days even with high-performance PC clusters. However, the geologic interpretations must wait until the final migration results are obtained. Consequently, we must drop some traces or shots occasionally to obtain the result in a reasonable time. This could lead to low resolution, although we have had enough field data to produce good results. Only with a magnitude of speedup is it practical to achieve much higher resolution by taking more field data into account.

In this study, we present a computing scheme to speed up high-order Fourier migration using a GPU-based library of the fast Fourier transform. We copy the whole 3D velocity model to the device memory before migration and copy the whole set of 3D imaged data back to the host after migration. This scheme greatly reduces the time consumption caused by the low bandwidth of data transfer between the host and the device. We reduce half the memory demand by applying data compressions to the 3D velocity model and the 3D imaged data. This scheme is feasible for most scales of current 3D explorations. We also suggest a 2D tapered function for boundary conditions, which is suitable for parallel implementation on GPUs. We incorporate both boundary conditions using the 2D tapered function and the scaling of inverse Fourier transform into the time-shift kernel. This scheme reduces the time consumption caused by the random memory access pattern involved and by scattered kernels.

The proposed GPU-accelerated scheme speeds up the third-order OSP over the CPU-based implementation by 25 to 40 times. A task that would have run for a whole month before now will run for only about a day, which means the overall computational cost has been reduced by more than 95%. The proposed scheme allows us to produce a satisfactory image in a much shorter turnaround time when updating the migration velocity. This scheme is consistent also with prevailing systems of PC clusters. The combination of GPU-accelerated Fourier propagators and PC clusters would, in terms of computational efficiency, make the wave-equation migration comparable to Kirchhoff migration.

ACKNOWLEDGMENTS

We thank Chen Ji for his encouragement and insightful discussions. We are grateful to Christof Mueller and two anonymous reviewers for their kind and valuable comments on the original manuscript. We also thank Xiaofei Chen and Marilyn Perlberg for improving the English in our revised manuscript. This research was supported by the Knowledge Innovation Program of the Chinese Academy of Sciences (Grant No.KZCX2-YW-101) and by the National Major Project of China (Grant No. 2008ZX05008-006), and partially by Open Fund (No. GDL0702) of Key Laboratory of Geo-detection (China University of Geosciences, Beijing), Ministry of Education.

REFERENCES

Aminzadeh, F., N. Burkhard, J. Long, T. Kunz, and P. Duclos, 1996, Three dimensional SEG/EAEG models — An update: The Leading Edge, **15**, 131–134.

Biondi, B., 2002, Stable wide-angle Fourier finite-difference downward extrapolation of 3-D wavefields: Geophysics, **67**, 872–882.

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

Cerjan, C., D. Kosloff, R. Kosloff, and M. Reshef, 1985, A nonreflecting boundary condition for discrete acoustic and elastic wave equations: Geophysics, **50**, 705–708.

Chen, J.-B., and H. Liu, 2004, Optimization approximation with separable variables for the one-way operator: Geophysical Research Letters, **31**, L06613.

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

de Hoop, M. V., J. H. Le Rousseau, and R.-S. Wu, 2000, Generalization of the phase-screen approximation for the scattering of acoustic waves: Wave Motion, **31**, 285–296.

Frigo, M., and S. G. Johnson, 1998, FFTW: An adaptive software architecture for the FFT: Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing, **3**, 1381–1384, doi: 10.1109/ICASSP.1998.681704.

Fu, L. Y., 2005, Broadband constant-coefficient propagators: Geophysical Prospecting, **53**, 299–310.

Gazdag, J., 1978, Wave equation migration with the phase-shift method: Geophysics, **43**, 1342–1351.

Gazdag, J., and P. Sguazzero, 1984, Migration of seismic data by phase shift plus interpolation: Geophysics, **49**, 124–131.

Hale, D., 1991, 3-D depth migration via McClellan transformations: Geophysics, **56**, 1778–1785.

Huang, L.-J., and M. C. Fehler, 2000, Quasi-Born Fourier migration: Geophysical Journal International, **140**, 521–534.

Huang, L.-J., M. C. Fehler, P. M. Roberts, and C. C. Burch, 1999a, Extended local Rytov Fourier migration method: Geophysics, **64**, 1535–1545.

Huang, L.-J., M. C. Fehler, and R.-S. Wu, 1999b, Extended local Born Fourier migration method: Geophysics, **64**, 1524–1534.

Le Rousseau, J. H., and M. V. de Hoop, 2001, Modeling and imaging with the scalar generalized-screen algorithms in isotropic media: Geophysics, **66**, 1551–1568.

Li, B., G. F. Liu, and H. Liu, 2009, A method of using GPU to accelerate seismic pre-stack time migration: Chinese Journal of Geophysics [in Chi-

nese], **52**, 245–252.
Liu, L., and J. Zhang, 2006, 3D wavefield extrapolation with optimum split-step Fourier method: Geophysics, **71**, no. 3, T95–T108.
Muyan-Ozcelik, P., J. D. Owens, J. Xia, and S. S. Samant, 2008, Fast deformable registration on the GPU: A CUDA implementation of demons: Proceedings of the 2008 International Conference on Computational Science and Its Applications (ICCSA), IEEE Computer Society Press.
NVIDIA, 2009, NVIDIA CUDA Programming Guide, version 2.2.1, http: www.nvidia. com/object/cuda_develop. html, accessed July 10, 2009.
Ober, C. C., R. A. Oldfield, D. E. Womble, and C. C. Mosher, 1997, Seismic imaging on massively parallel computers: 67th Annual International Meeting, SEG, Expanded Abstracts, 1418–1421.
Ristow, D., and T. Rühl, 1994, Fourier finite-difference migration: Geophysics, **59**, 1882–1893.
Shen, P., and W. W. Symes, 2008, Automatic velocity analysis via shot profile migration: Geophysics, **73**, no. 5, VE49–VE59.
Stantchev, G., W. Dorland, and N. Gumerov, 2008, Fast parallel Particle-To-Grid interpolation for plasma PIC simulations on the GPU: Journal of Parallel and Distributed Computing, **68**, 1339–1349.
Stoffa, P. L., J. T. Fokkema, R. M. de Luna Freir, and W. P. Kessinger, 1990, Split-step Fourier migration: Geophysics, **55**, 410–421.
Stone, J. E., J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, 2007, Accelerating molecular modeling applications with graphics processors: Journal of Computational Chemistry, **28**, 2618–2640.
Wang, Y., 2001, ADI plus interpolation: Accurate finite-difference solution to 3D paraxial wave equation: Geophysical Prospecting, **49**, 547–556.
Wild, A. J., R. W. Hobbs, and L. Frenje, 2000, Modeling complex media: An introduction to the phase-screen method: Physics of the Earth and Planetary Interiors, **120**, 219–225.
Wu, R. S., 1994, Wide-angle elastic wave one-way propagation in heterogeneous media and an elastic wave complex-screen method: Journal of Geophysics Research, **99**, 751–766.
Xie, X. B., and R. S. Wu, 2001, Modeling elastic wave forward propagation and reflection using the complex screen method: Journal of the Acoustical Society of America, **109**, 2629–2635.
Yang, J., Y. Wang, and Y. Chen, 2007, GPU accelerated molecular dynamics simulation of thermal conductivities: Journal of Computational Physics, **221**, 799–804.
Zhang, J., and L. Liu, 2007, Optimum split-step Fourier 3D depth migration: Developments and practical aspects, Geophysics, **72**, no. 3, S167-S175.
Zhang, J. H., W. M. Wang, L. Y. Fu, and Z. X. Yao, 2008, 3D Fourier finite-difference migration by ADI plus interpolation: Geophysical Prospecting, **56**, 95–103.
Zhang, J. H., W. M. Wang, and Z. X. Yao, 2009, Comparison between the Fourier finite-difference method and the generalized-screen method: Geophysical Prospecting, **57**, 355–365.