RTM using effective boundary saving: A staggered grid GPU implementation[a]

[a]Published in Computers & Geosciences, 68, 64-72, (2014)

Pengliang Yang[*], *Jinghuai Gao*[*], *and Baoli Wang*[†]
[*]*Xi'an Jiaotong University, National Engineering Laboratory for Offshore Oil Exploration, Xi'an, China, 710049*
[†]*CCTEG Xi'an Research Institute, Xi'an, China, 710077*

ABSTRACT

GPU has become a booming technology in reverse time migration (RTM) to perform the intensive computation. Compared with saving forward modeled wavefield on the disk, RTM via wavefield reconstruction using saved boundaries on device is a more efficient method because computation is much faster than CPU-GPU data transfer. In this paper, we introduce the effective boundary saving strategy in backward reconstruction for RTM. The minimum storage requirement for regular and staggered grid finite difference is determined for perfect reconstruction of the source wavefield. Particularly, we implement RTM using GPU programming, combining staggered finite difference scheme with convolutional perfectly matched layer (CPML) boundary condition. We demonstrate the validity of the proposed approach and CUDA codes with numerical example and imaging of benchmark models.

INTRODUCTION

One-way equation based imaging techniques are inadequate to obtain accurate images in complex media due to propagation direction changes in the background model (Biondi, 2006). These approaches are extremely limited when handling the problems of turning waves in the model containing sharp wave-speed contrasts and steeply dipping reflectors. As an advanced imaging technology without dip and extreme lateral velocity limitation, reverse time migration (RTM) was proposed early (Baysal et al., 1983; McMechan, 1983), but not practical in terms of stringent computation and memory requirement. However, it gained increasingly attention in recent years due to the tremendous advances in computer capability. Until recently, 3D prestack RTM is now feasible to obtain high fidelity images (Yoon et al., 2003; Guitton et al., 2006).

Nowadays, graphics processing unit (GPU) is a booming technology, widely used to mitigate the computational drawbacks in seismic imaging and inversion, from one-

way depth migration (Liu et al., 2012b; Lin and Wang, 2012) to two-way RTM (Hussain et al., 2011; Micikevicius, 2009; Clapp et al., 2010), from 2D to 3D (Micikevicius, 2009; Abdelkhalek et al., 2009; Foltinek et al., 2009; Liu et al., 2013a; Michéa and Komatitsch, 2010), from acoustic media to elastic media (Weiss and Shragge, 2013), from isotropic media to anisotropy (Guo et al., 2013; Suh and Wang, 2011; Liu et al., 2009). The investigators have studied many approaches: the Fourier integral method (Liu et al., 2012c), spectral element method (Komatitsch et al., 2010b), finite element method (Komatitsch et al., 2010a) as well as the rapid expansion method (REM) with pseudo-spectral approach (Kim et al., 2013). A variety of applications were conducted, for instance, GPU-based RTM denoising (Ying et al., 2013), iterative velocity model building (Ji et al., 2012), multi-source RTM (Boonyasiriwat et al., 2010), as well as least-square RTM (Leader and Clapp, 2012).

The superior speedup performance of GPU-based imaging and inversion has been demonstrated by numerous studies. One key problem of GPU-based RTM is that the computation is much faster while the data exchange between host and device always takes longer time. Many researchers choose to reconstruct the source wavefield instead of storing the modeling time history on the disk, just saving the boundaries. Unlike most GPU-based imaging and inversion studies, this paper is devoted to the practical technical issues instead of speedup performance. Starting from the computational strategies by Dussaud et al. (2008), we determine the minimum storage requirement in backward wavefield reconstruction for regular and staggered grid finite difference. We implement RTM with staggered finite difference scheme combined with convolutional perfectly matched layer (CPML) boundary condition using GPU programming. We demonstrate the validity of the proposed approach and CUDA codes with numerical test and imaging of benchmark models.

OVERVIEW OF RTM AND ITS COMPUTATION

In the case of constant density, the acoustic wave equation is written as

$$\frac{1}{v^2(\mathbf{x})}\frac{\partial^2 p(\mathbf{x},t;\mathbf{x}_s)}{\partial t^2} = \nabla^2 p(\mathbf{x},t;\mathbf{x}_s) + f(t)\delta(\mathbf{x}-\mathbf{x}_s), \tag{1}$$

where $p(\mathbf{x},t;\mathbf{x}_s)$ is the wavefield excited by the source at the position $\mathbf{x}=\mathbf{x}_s$, $v(\mathbf{x})$ stands for the velocity in the media, $\nabla^2 = \nabla\cdot\nabla = \partial_{xx}+\partial_{zz}$, $f(t)$ denotes the source signature. For the convenience, we eliminate the source term hereafter and use the notation $\partial_u = \frac{\partial}{\partial u}$ and $\partial_{uu} = \frac{\partial}{\partial u^2}$, $u = x, z$. The forward marching step can be specified after discretization as

$$p^{k+1} = 2p^k - p^{k-1} + v^2\Delta t^2\nabla^2 p^k. \tag{2}$$

Based on the wave equation, the principle of RTM imaging can be interpreted as the cross-correlation of two wavefields at the same time level, one computed by forward time recursion, the other computed by backward time stepping (Symes, 2007).

Mathematically, the cross-correlation imaging condition can be expressed as

$$I(\mathbf{x}) = \sum_{s=1}^{ns} \int_0^{t_{\max}} \mathrm{d}t \sum_{g=1}^{ng} p_s(\mathbf{x}, t; \mathbf{x}_s) p_g(\mathbf{x}, t; \mathbf{x}_g), \tag{3}$$

where $I(\mathbf{x})$ is the migrated image at point $\mathbf{x}$; and $p_s(\cdot)$ and $p_g(\cdot)$ are the source wavefield and receiver (or geophone) wavefield. The normalized cross-correlation imaging condition is designed by incorporating illumination compensation:

$$I(\mathbf{x}) = \sum_{s=1}^{ns} \frac{\int_0^{t_{\max}} \mathrm{d}t \sum_{g=1}^{ng} p_s(\mathbf{x}, t; \mathbf{x}_s) p_g(\mathbf{x}, t; \mathbf{x}_g)}{\int_0^{t_{\max}} \mathrm{d}t p_s(\mathbf{x}, t; \mathbf{x}_s) p_s(\mathbf{x}, t; \mathbf{x}_s)}. \tag{4}$$

There are some possible ways to do RTM computation. The simplest one may be just storing the forward modeled wavefields on the disk, and reading them for imaging condition in the backward propagation steps. This approach requires frequent disk I/O and has been replaced by wavefield reconstruction method. The so-called wavefield reconstruction method is a way to recover the wavefield via backward reconstructing or forward remodeling, using the saved wavefield snaps and boundaries. It is of special value for GPU computing because saving the data in device variables eliminates data transfer between CPU and GPU. By saving the last two wavefield snaps and the boundaries, one can reconstruct the wavefield of every time step, in time-reversal order. The checkpointing technique becomes very useful to further reduce the storage (Symes, 2007; Dussaud et al., 2008). Of course, it is also possible to avert the issue of boundary saving by applying the random boundary condition, which may bring some noises in the migrated image (Clapp, 2009; Clapp et al., 2010; Liu et al., 2013b,a).

EFFECTIVE BOUNDARY SAVING

Here we mainly focus on finding the part of boundaries which is really necessary to be saved (referred to as the effective boundary in this paper), even though there are many other practical implementation issues in GPU-based RTM (Liu et al., 2012a). In what follows, we introduce the effective boundary saving for regular grid and staggered grid finite difference. All analysis will be based on 2D acoustic wave propagation in RTM. In other cases, the wave equation may change but the principle of effective boundary saving remains the same.

Which part of the wavefield should be saved?

To reconstruct the modeled source wavefield in backward steps rather than read the stored history from the disk, one can reuse the same template by exchanging the role of p^{k+1} and p^{k-1}, that is,

$$p^{k-1} = 2p^k - p^{k+1} + v^2 \Delta t^2 \nabla^2 p^k. \tag{5}$$

We conduct the modeling (and the backward propagation in the same way due to template reuse):

$$
\begin{aligned}
&for\ ix, iz... \quad p_0(:) = 2p_1(:) - p_0(:) + v^2(:)\Delta t^2 \nabla^2 p_1(:) \\
&ptr = p_0; p_0 = p_1; p_1 = ptr; //\text{exchange pointer}
\end{aligned}
$$

where $(:) = [ix, iz]$, p_0 and p_1 are p^{k+1}/p^{k-1} and p^k, respectively. When the modeling is finished, only the last two wave snaps (p^{nt} and p^{nt-1}) as well as the saved boundaries are required to do the backward time recursion.

As you see, RTM begs for an accurate reconstruction before applying the imaging condition using the backward propagated wavefield. The velocity model is typically extended with sponge absorbing boundary condition (ABC) (Cerjan et al., 1985) or PML and its variants (Komatitsch and Martin, 2007) to a larger size. In Figure 1, the original model size $A_1A_2A_3A_4$ is extended to $C_1C_2C_3C_4$. In between is the artificial boundary ($C_1C_2C_3C_4 \backslash A_1A_2A_3A_4$). Actually, the wavefield we intend to reconstruct is not the part in extended artificial boundary $C_1C_2C_3C_4 \backslash A_1A_2A_3A_4$ but the part in the original model zone $A_1A_2A_3A_4$. We can reduce the boundary load further (from whole $C_1C_2C_3C_4 \backslash A_1A_2A_3A_4$ to part of it $B_1B_2B_3B_4$) depending on the required grids in finite difference scheme, as long as we can maintain the correctness of wavefield in $A_1A_2A_3A_4$. We do not care about the correctness of the wavefield neither in $A_1A_2A_3A_4$ nor in the effective zone $B_1B_2B_3B_4$ (i.e. the wavefield in $C_1C_2C_3C_4 \backslash B_1B_2B_3B_4$). Furthermore, we only need to compute the imaging condition in the zone $A_1A_2A_3A_4$, no concern with the part in $C_1C_2C_3C_4 \backslash A_1A_2A_3A_4$.

Effective boundary for regular grid finite difference

Assume $2N$-th order finite difference scheme is applied. The Laplacian operator is specified by

$$
\begin{aligned}
\nabla^2 p^k &= \partial_{xx} p^k + \partial_{zz} p^k \\
&= \frac{1}{\Delta z^2} \sum_{i=-N}^{N} c_i p^k[ix][iz+i] + \frac{1}{\Delta x^2} \sum_{i=-N}^{N} c_i p^k[ix+i][iz]
\end{aligned}
\tag{6}
$$

where c_i is given by Table 1, see a detailed derivation in Fornberg (1988). The Laplacian operator has x and z with same finite difference structure. For x dimension only, the second derivative of order $2N$ requires at least N points in the boundary zone, as illustrated by Figure 2. In 2-D case, the required boundary zone has been plotted in Figure 3a. Note that four corners in $B_1B_2B_3B_4$ in Figure 1 are not needed. This is exactly the boundary saving scheme proposed by Dussaud et al. (2008).

Keep in mind that we only need to guarantee the correctness of the wavefield in the original model zone $A_1A_2A_3A_4$. However, the saved wavefield in $A_1A_2A_3A_4 \backslash B_1B_2B_3B_4$ is also correct. Is it possible to further shrink it to reduce number of points for saving? The answer is true. Our solution is: *saving the inner N layers on each side neighboring the boundary* $A_1A_2A_3A_4 \backslash D_1D_2D_3D_4$, as shown in Figure 3b. We call it the effective boundary for regular finite difference scheme.

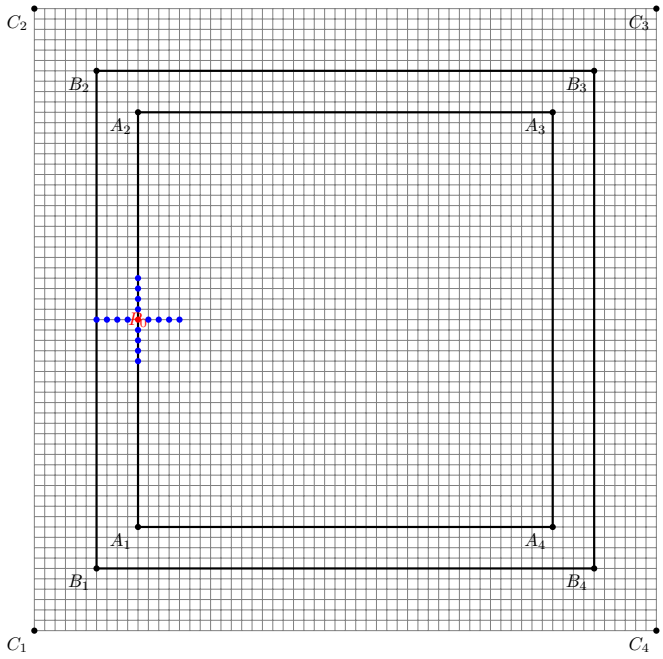


Figure 1: Extend the model size with artificial boundary. $A_1A_2A_3A_4$ indicates the original model size $(nz \times nx)$. $C_1C_2C_3C_4$ is the extended model size $(nz+2nb)(nx+2nb)$. $B_1B_2B_3B_4 \backslash A_1A_2A_3A_4$ is the effective boundary area.

Table 1: Finite difference coefficients for regular grid (Order-$2N$)

i	-4	-3	-2	-1	0	1	2	3	4
$N=1$				1	-2	1			
$N=2$			-1/12	4/3	-5/2	4/3	-1/12		
$N=3$		1/90	-3/20	3/2	-49/18	3/2	-3/20	1/90	
$N=4$	-1/560	8/315	-1/5	8/5	-205/72	8/5	-1/5	8/315	-1/560

After nt steps of forward modeling, we begin our backward propagation with the last 2 wavefield snap p^{nt} and p^{nt-1} and saved effective boundaries in $A_1A_2A_3A_4 \backslash D_1D_2D_3D_4$. At that moment, the wavefield is correct for every grid point. (Of course, the correctness of the wavefield in $A_1A_2A_3A_4$ is guaranteed.) At time k, we assume the wavefield in $A_1A_2A_3A_4$ is correct. One step of backward propagation means $A_1A_2A_3A_4$ is shrunk to $D_1D_2D_3D_4$. In other words, the wavefield in $D_1D_2D_3D_4$ is correctly reconstructed. Then we load the saved effective boundary of time k to overwrite the area $A_1A_2A_3A_4 \backslash D_1D_2D_3D_4$. Again, all points of the wavefield in $A_1A_2A_3A_4$ are correct. We repeat this overwriting and computing process from one time step to another ($k \rightarrow k-1$), in reverse time order. The wavefield in the boundary $C_1C_2C_3C_4 \backslash A_1A_2A_3A_4$ may be incorrect because the points here are neither saved nor correctly reconstructed from the previous step.

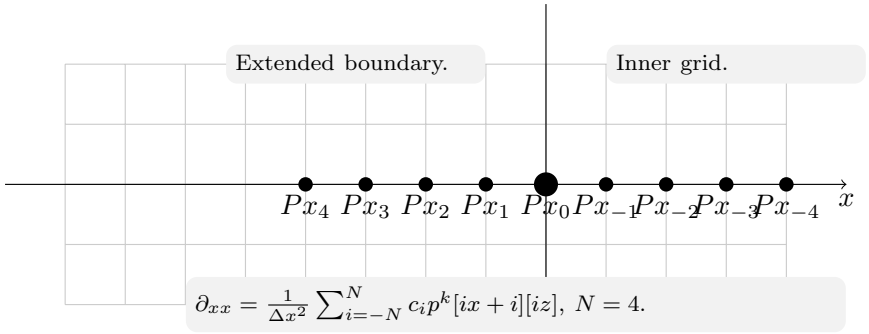


Figure 2: 1-D schematic plot of required points in regular grid for boundary saving. Computing the laplacian needs N points in the extended boundary zone, the rest $N+1$ points in the inner model grid. N points is required for boundary saving.

Effective boundary for staggered grid finite difference

The limitation of boundary saving strategy proposed in Dussaud et al. (2008) is that only regular grid finite difference scheme is considered in RTM. In the case of staggered grid, half grid points are employed to obtain higher accuracy for finite difference. Recursion from time k to $k+1$ (or $k-1$) may not be realized with ease due to the Laplacian operator, which involves the second derivative. An effective approach is to split Eq. (1) into several first derivative equations or combinations of first derivative and second derivative equations. The first derivative is defined as

$$\partial_u f = \frac{1}{\Delta u} \left(\sum_{i=1}^{N} c_i (f[u + i\Delta u/2] - f[u - i\Delta u/2]) \right), u = z, x \tag{7}$$

where the finite difference coefficients are listed in Table 2.

The use of half grid points in staggered grid makes the effective boundary a little different from that in regular grid. To begin with, we define some intermediate

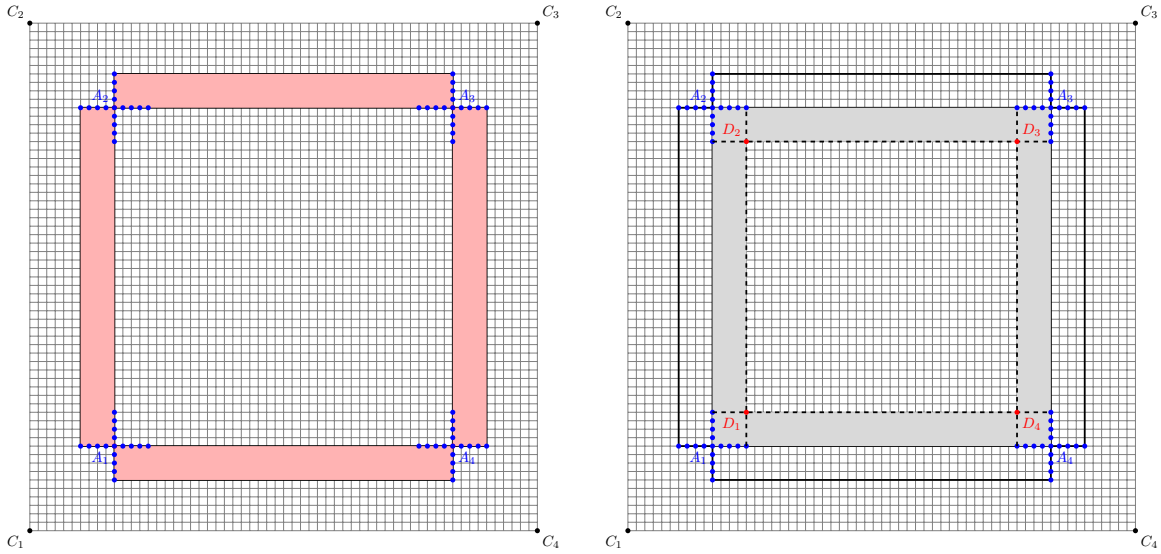


Figure 3: A 2-D sketch of required points for boundary saving for regular grid finite difference: (a) The scheme proposed by Dussaud et al. (2008) (red zone). (b) Proposed effective boundary saving scheme (gray zone).

Table 2: Finite difference coefficients for staggered grid (Order-2N)

i	1	2	3	4
$N = 1$	1			
$N = 2$	1.125	-0.0416667		
$N = 3$	1.171875	-0.0651041667	0.0046875	
$N = 4$	1.1962890625	-0.079752604167	0.0095703125	-0.000697544642857

auxiliary variables: $Ax := \partial_x p$, $Az := \partial_z p$, $Px := \partial_x Ax$ and $Pz := \partial_z Az$. Thus the acoustic wave equation reads

$$\begin{cases} \dfrac{\partial^2 p}{\partial t^2} = v^2 \left(Px + Pz\right) \\ Px = \partial_x Ax, Pz = \partial_z Az \\ Ax = \partial_x p, Az = \partial_z p \end{cases} \tag{8}$$

It implies that we have to conduct 2 finite difference steps (one for Ax and Az and the other for Px and Pz) to compute the Laplacian in one step of time marching. Take 8-th order ($2N = 8$) finite difference in x dimension for example. As can be seen from Figure 4, computing ∂_{xx} at Px_0 needs the correct values at Ax_4,Ax_3,Ax_2,Ax_1 in the boundary; computing Ax_1,Ax_2,Ax_3,Ax_4 needs the correct values at Px_4,Px_5,Px_6,Px_7 in the boundary. An intuitive approach is saving N points of Ax ($Ax_1, \ldots, Ax_4$) and N points of Px ($Px_4, \ldots, Px_7$). The saving procedure guarantees the correctness of these points in the wavefield. Another possible approach is just saving the $2N-1$ points of Px ($Px_1, \ldots, Px_7$). In this way, the values of $Ax_1, \ldots, Ax_4$ can be correctly obtained from the calculation of the first derivative. The latter method is preferable because it is much easier for implementation while requiring less points. Speaking two dimensionally, some points in the four corners at in $B_1B_2B_3B_4$ of Figure 1 may be still necessary to store, as shown in Figure 5a. The reason is that you are working with Laplacian, not second derivative in one dimension. Again, we switch our boundary saving part from out of $A_1A_2A_3A_4$ to $A_1A_2A_3A_4 \backslash D_1D_2D_3D_4$. Less grid points are required to guarantee correct reconstruction while points in the corner are no longer needed. Therefore, *the proposed effective boundary for staggered finite difference needs* $2N-1$ *points to be saved on each side*, see Figure 5b.

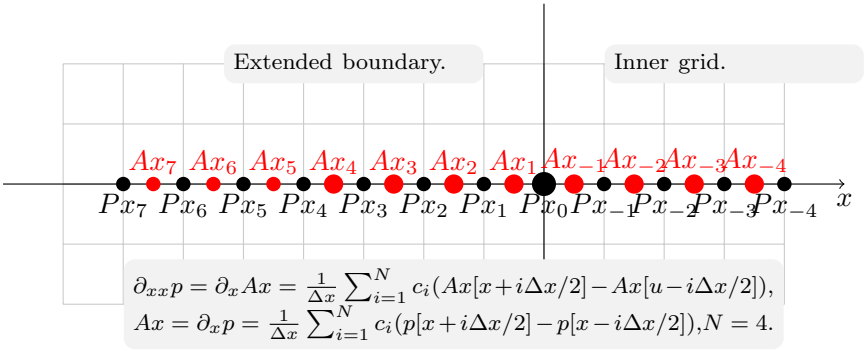


Figure 4: $2N$-th order staggered grid finite difference: correct backward propagation needs $2N-1$ points on one side. For $N = 4$, computing ∂_{xx} at Px_0 needs the correct values at Ax_4, Ax_3, Ax_2, Ax_1 in the boundary; computing Ax_4,Ax_3, Ax_2, Ax_1 needs the correct values at Px_4, Px_5, Px_6, Px_7 in the boundary. Thus, $2N-1 = 7$ points in boundary zone is required to guarantee the correctness of the inner wavefield.

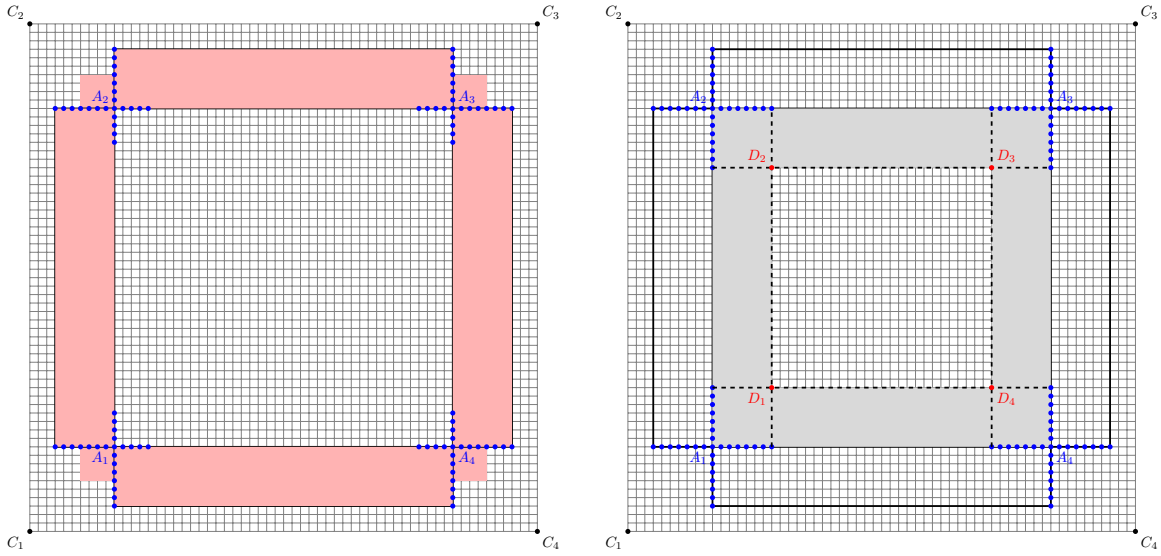


Figure 5: A 2-D sketch of required points for boundary saving for staggered grid finite difference: (a) Saving the points outside the model (red region). (b) Effective boundary, saving the points inside the model zone (gray region).

Storage analysis

For the convenience of complexity analysis, we define the size of the original model as $nz \times nx$. In each direction, we pad the model with the nb points on both sides as the boundary. Thus, the extended model size becomes $(nz + 2nb)(nx + 2nb)$. Conventionally one has to save the whole wavefield within the model size on the disk. The required number of points is

$$nz \cdot nx. \tag{9}$$

According to Dussaud et al. (2008), for $2N$-th order finite difference in regular grid, N points on each side are added to guarantee the correctness of inner wavefield. The saving amount of every time step is

$$2N \cdot nz + 2N \cdot nx = 2N(nz + nx). \tag{10}$$

In the proposed effective boundary saving strategy, the number becomes

$$2N \cdot nz + 2N \cdot nx - 4N^2 = 2N(nz + nx) - 4N^2. \tag{11}$$

In the case of staggered grid, there are $2N - 1$ points on each side. Allowing for four corners, the number for the effective boundary saving is

$$2(2N-1)nz + 2(2N-1)nx - 4(2N-1)^2 = 2(2N-1)(nz+nx) - 4(2N-1)^2 \tag{12}$$

Assume the forward modeling is performed nt steps using the floating point format on the computer. The saving amount will be multiplied by $nt \cdot \text{sizeof(float)} = 4nt$. Table 3 lists this memory requirement for different boundary saving strategies.

Table 3: Storage requirement for different saving strategy

Boundary saving scheme	Saving amount (Unit: Bytes)
Conventional saving strategy	$4nt \cdot nz \cdot nx$
Dussaud's: regular grid	$4nt \cdot 2N(nz + nx)$
Effective boundary: regular grid	$4nt \cdot (2N(nz + nx) - 4N^2)$
Effective boundary: staggered grid	$4nt \cdot (2(2N-1)(nz + nx) - 4(2N-1)^2)$

In principle, the proposed effective boundary saving will reduce $4nt \cdot 4N^2$ bytes for regular grid finite difference, compared with the method of Dussaud et al. (2008). The storage requirement of staggered grid based effective boundary saving is about $(2N-1)/N$ times of that in the regular grid finite difference, by observing $2N \ll nb \ll nx, nz$. For the convenience of practical implementation, the four corners can be saved twice so that the saving burden of the effective boundary saving has no difference with the method of Dussaud et al. (2008) in regular grid finite difference. Since the saving burden for staggered grid finite difference has not been touched in Dussaud et al. (2008), it is still of special value to minimize its storage requirement for GPU computing.

GPU IMPLEMENTATION USING CPML BOUNDARY CONDITION

CPML boundary condition

To combine the absorbing effects into the acoustic equation, CPML boundary condition is such a nice way that we merely need to combine two convolutional terms into the above equations:

$$\begin{cases} \dfrac{\partial^2 p}{\partial t^2} = v^2 \left(Px + Pz\right) \\ Px = \partial_x Ax + \Psi_x \\ Pz = \partial_z Az + \Psi_z \\ Ax = \partial_x p + \Phi_x \\ Az = \partial_z p + \Phi_z \end{cases} \tag{13}$$

where Ψ_x, Ψ_z are the convolutional terms of Ax and Az; Φ_x, Φ_z are the convolutional terms of Px and Pz. These convolutional terms can be computed via the following relation:

$$\begin{cases} \Psi_x^n = b_x \Psi_x^{n-1} + (b_x - 1)\partial_x^{n+1/2} Ax \\ \Psi_z^n = b_z \Psi_z^{n-1} + (b_z - 1)\partial_z^{n+1/2} Az \\ \Phi_x^n = b_x \Phi_x^{n-1} + (b_x - 1)\partial_x^{n-1/2} p \\ \Phi_z^n = b_z \Phi_z^{n-1} + (b_z - 1)\partial_z^{n-1/2} p \end{cases} \tag{14}$$

where $b_x = e^{-d(x)\Delta t}$ and $b_z = e^{-d(z)\Delta t}$. In the absorbing layers, the damping parameter $d(u)$ we used is (Collino and Tsogka, 2001):

$$d(u) = d_0(\frac{u}{L})^2, d_0 = -\frac{3v}{2L}\ln(R), \tag{15}$$

where L indicates the PML thickness; u represent the distance between current position (in PML) and PML inner boundary. R is always chosen as $10^{-3} \sim 10^{-6}$. For more details about the derivation of CPML, the interested readers are referred to Collino and Tsogka (2001) and Komatitsch and Martin (2007). The implementation of CPML boundary condition is easy to carry out: in each iteration the wavefield extrapolation is performed according to the first equation in (13); it follows by adding the convolutional terms in terms of (14).

Memory manipulation

Consider the Marmousi model (size=751x2301) and the Sigsbee model (size=1201x3201). Assume $nt = 10000$ and the finite difference of order $2N = 8$. Conventionally, one have to store 64.4 GB for Marmousi model and 143.2 GB for Sigsbee model on the disk of the computer. Using the method of Dussaud et al. (2008) or regular grid based effective boundary saving, the storage requirement will be greatly reduced, about 0.9 GB and 1.3 GB for the two models. Staggered grid finite difference is preferable due to higher accuracy, however, the saving amount of effective boundary needs 1.6 GB and 2.3 GB for the two models, much larger than regular grid. Besides the additional variable allocation, the storage requirement may still be a bottleneck to save all boundaries on GPU to avert the CPU saving and data exchange for low-level hardware, even if we are using effective boundary saving.

Fortunately, page-locked (also known as pinned) host memory provides us a practical solution to mitigate this conflict. Zero-copy system memory has identical coherence and consistency to global memory. Copies between page-locked host memory and device memory can be performed concurrently with kernel execution (Nvidia, 2011). * Therefore, we store a certain percentage of effective boundary on the page-locked host memory, and the rest on device. A reminder is that overuse of the pinned memory may degrade the bandwidth performance.

Code organization

Allowing for the GPU block alignment, the thickness of CPML boundary is chosen to be 32. Most of the CUDA kernels are configured with a block size 16x16. Some special configurations are related to the initialization and calculation of CPML boundary area. The CPML variables are initialized along x and z axis with CUDA

*Generally, a computer has same or larger amount of resource on host compared with GDDR memory on device.

kernels `cuda_init_abcz(...)` and `cuda_init_abcx(...)`. When `device_alloc(...)` is invoked to allocate memory, there is a variable `phost` to control the percentage of the effective boundary saved on host and device memory by calling the function `cudaHostAlloc(...)`. A pointer is referred to the pinned memory via `cudaHostGetDevicePointer(...)` The wavelet is generated on device using `cuda_ricker_wavelet(...)` with a dominant frequency `fm` and delayed wavelength. Adding a shot can be done by a smooth bell transition `cuda_add_bellwlt(...)`. We implement RTM (of order `NJ=2, 4, 6, 8, 10`) with forward and backward propagation functions `step_forward(...)` and `step_backward(...)`, in which the shared memory is also used for faster computation. The cross-correlation imaging of each shot is done by `cuda_cross_correlate(...)`. The final image can be obtained by stacking the images of many shots using `cuda_imaging(...)`. Most of the low-frequency noise can be removed by applying the muting function `cuda_mute(...)` and the Laplacian filtering `cuda_laplace_filter(...)`.

NUMERICAL EXAMPLES

Exact reconstruction

To make sure that the proposed effective boundary saving strategy does not introduce any kind of error/artifacts for the source wavefield, the first example is designed using a constant velocity model: velocity=2000 m/s, $nz = nx = 320$, $\Delta z = \Delta x = 5m$. The source position is set at the center of the model. The modeling process is performed $nt = 1000$ time samples. We record the modeled wavefield snap at $k = 420$ and $k = 500$, as shown in the top panels of Figure 6. The backward propagation starts from $k = 1000$ and ends up with $k = 1$. In the backward steps, the reconstructed wavefield at $k = 500$ and $k = 420$ are also recorded, shown in the bottom panels of Figure 6. We also plot the wavefield in the boundary zone in both two panels. Note that the correctness of the wavefield in the original model zone is guaranteed while the wavefield in the boundary zone does not need to be correct.

Marmousi model

The second example is GPU-based RTM for Marmousi model (Figure 7) using our effective boundary saving. The spatial sampling interval is $\Delta x = \Delta z = 4m$. 51 shots are deployed. In each shot, 301 receivers are placed in the split shooting mode. The parameters we use are listed as follows: $nt = 13000$, $\Delta t = 0.3$ ms. Due to the limited resource on our computer, we store 65% boundaries using page-locked memory. Figure 8 gives the resulting RTM image after Laplacian filtering. As shown in the figure, RTM with the effective boundary saving scheme produces excellent image: the normalized cross-correlation imaging condition greatly improves the deeper parts of the image due to the illumination compensation. The events in the central part of the model, the limits of the faults and the thin layers are much better defined.

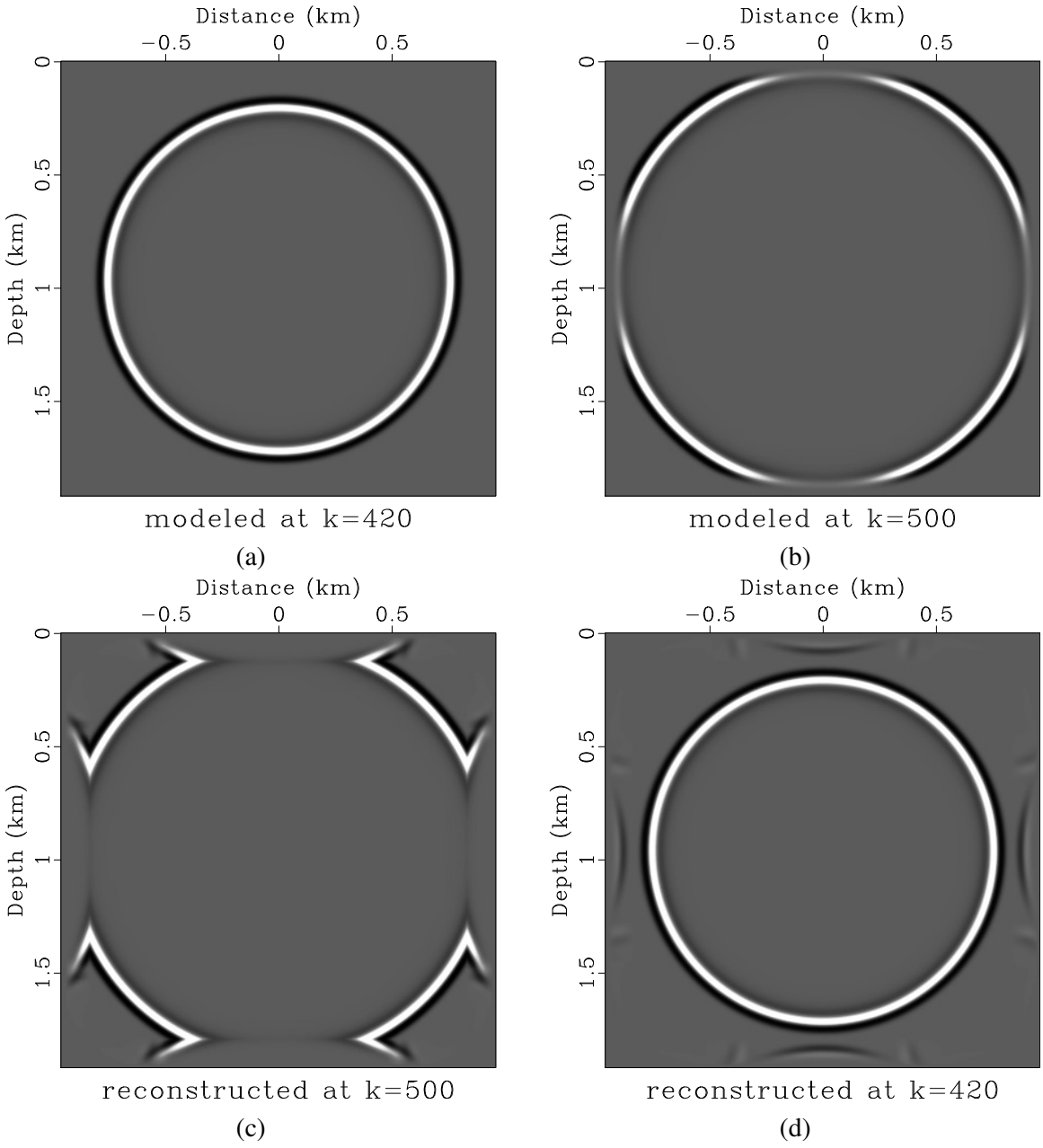


Figure 6: The wavefield snaps with a constant velocity model: velocity=2000 m/s, $nz = nx = 320$, $\Delta z = \Delta x = 5m$, source at the center. The forward modeling is conducted with $nt = 1000$ time samples. (a–b) Modeled wavefield snaps at $k = 420$ and $k = 500$. The backward propagation starts from $k = 1000$ and ends at $k = 1$. (c–d) Reconstructed wavefield snaps at $k = 500$ and $k = 420$. Note the correctness of the wavefield in the original model zone is guaranteed while the wavefield in the boundary zone may be incorrect (32 layers of the boundary on each side are also shown in the figure).

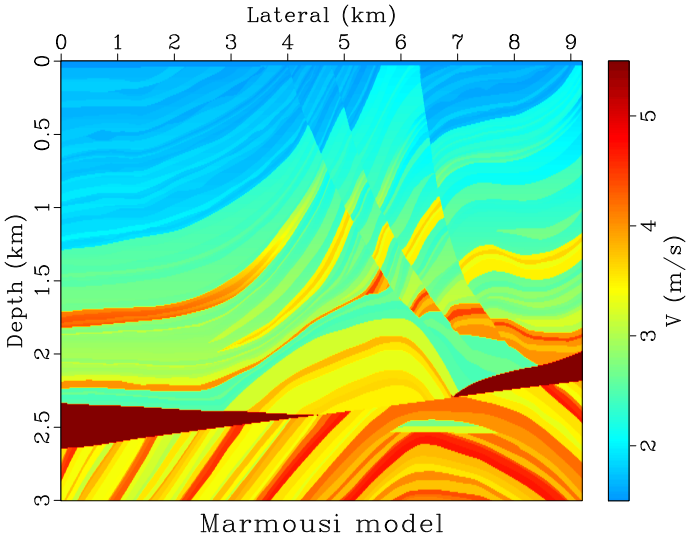


Figure 7: The Marmousi velocity model.

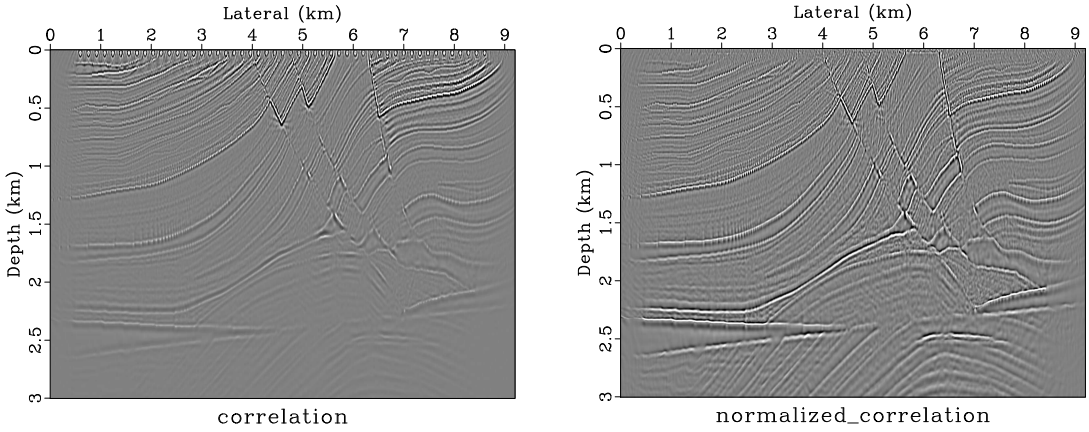


Figure 8: RTM result of Marmousi model using effective boundary saving scheme (staggered grid finite difference). (a) Result of cross-correlation imaging condition. (b) Result of normalized cross-correlation imaging condition.

Sigsbee model

The last example is Sigsbee model shown in Figure 9. The spatial interval is $\Delta x = \Delta z = 25m$. 55 shots are evenly distributed on the surface of the model. We still perform $nt = 13000$ time steps for each shot (301 receivers). Due to the larger model size, 75% boundaries have to be stored with the aid of pinned memory. Our RTM result is shown in Figure 10. Again, the resulting image obtained by normalized cross-correlation imaging condition exhibits better resolution for the edges of the salt body and the diffraction points. Some events in the image using normalized cross-correlation imaging condition are more visible, while they have a much lower amplitude or are even completely lost in the image of cross-correlation imaging condition.

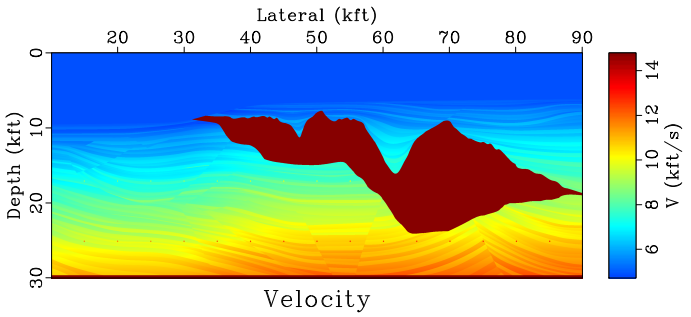
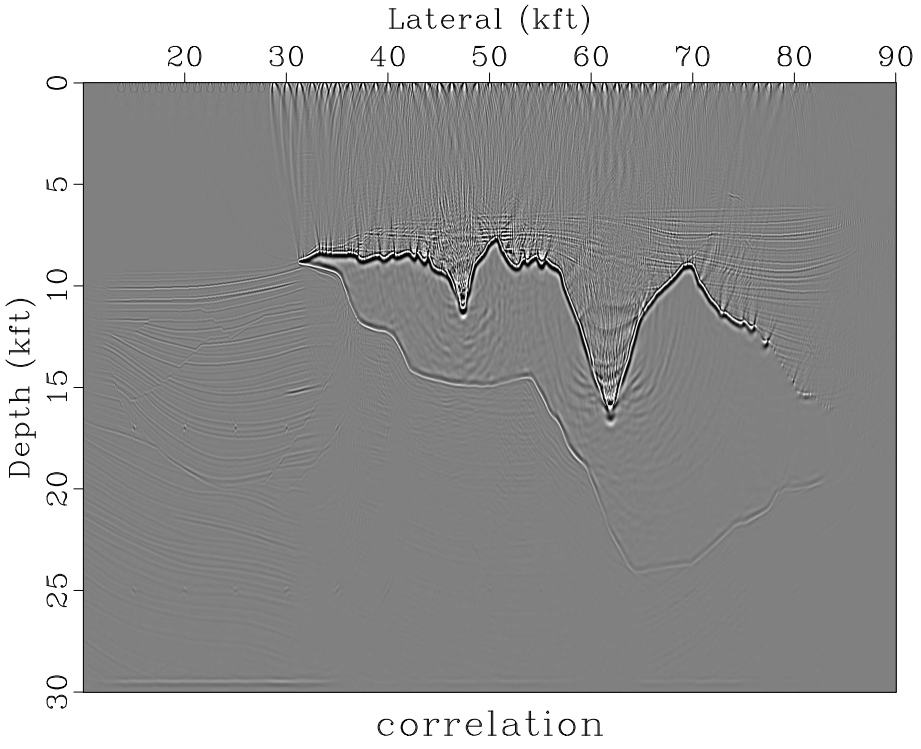


Figure 9: The Sigsbee velocity model.

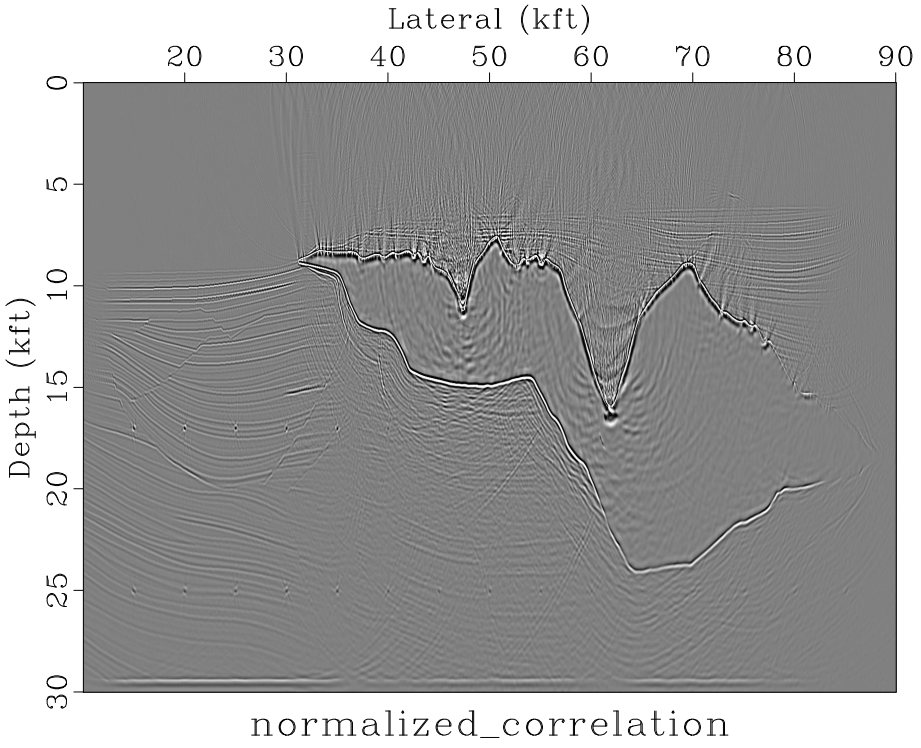
CONCLUSION AND DISCUSSION

In this paper, we introduce the effective boundary saving strategy for GPU-based RTM imaging. Compared with the method of Dussaud et al. (2008), the saving amount of effective boundary with regular grid finite difference scheme is slightly reduced. The RTM storage of effective boundary saving for staggered finite difference is first explored, and then implemented with CPML boundary condition. We demonstrate the validity of effective boundary saving strategy by numerical test and imaging of benchmark models.

The focus of this paper is RTM implementation using effective boundary saving in staggered grid instead of GPU acceleration. A limitation of this work is that the numerical examples are generated with NVS5400M GPU on a laptop (compute capability 2.1, GDDR3). It is easy to do performance analysis for different dataset size and higher stencil orders if the latest GPU card and CUDA driver are available. It is also possible to obtain improved speedup by incorporating MPI with GPU programming using advanced clusters with larger GDDR memory (Komatitsch et al., 2010a; Suh et al., 2010) or FPGA optimization (Fu and Clapp, 2011; Medeiros et al., 2011). Unfortunately, higher stencil orders of staggered grid RTM using effective boundary implementation in 3D is still a problem. 3D RTM using the 2nd order regular grid



(a)



(b)

Figure 10: RTM result of Sigsbee model using effective boundary saving scheme (staggered grid finite difference). (a) Result of cross-correlation imaging condition. (b) Result of normalized cross-correlation imaging condition.

finite difference with Clayton and Enquist boundary condition (only 1 layer on each side to save) needs tens of GBs (Liu et al., 2013b). It implies that 3D RTM with higher stencil orders will definitely exceed the memory bound of current and next generation GPUs. For GPU implementation of 3D RTM, the practical way is using the random boundary condition (Liu et al., 2013a) or saving on the disk. A deeper discussion of the practical issues for GPU implementation of RTM can be found in Liu et al. (2012a).

ACKNOWLEDGMENTS

The work of the first author is supported by China Scholarship Council during his visit to The University of Texas at Austin. This work is sponsored by National Science Foundation of China (No. 41390454). We wish to thank Sergey Fomel and two anonymous reviewers for constructive suggestions, which lead to massive amount of revision and improvement in this paper. The code of even-order GPU-based prestack RTM (combined with CPML boundary condition) using effective boundary saving strategy is available alongside this paper. The RTM examples are reproducible with the help of Madagascar software package (Fomel et al., 2013).

REFERENCES

Abdelkhalek, R., H. Calandra, O. Coulaud, J. Roman, and G. Latu, 2009, Fast seismic modeling and reverse time migration on a gpu cluster: International Conference on High Performance Computing & Simulation, HPCS'09., IEEE, 36–43.

Baysal, E., D. D. Kosloff, and J. W. Sherwood, 1983, Reverse time migration: Geophysics, **48**, 1514–1524.

Biondi, B., 2006, 3d seismic imaging: Society of Exploration Geophysicists.

Boonyasiriwat, C., G. Zhan, M. Hadwiger, M. Srinivasan, and G. Schuster, 2010, Multisource reverse-time migration and full-waveform inversion on a gpgpu: Presented at the 72nd EAGE Conference & Exhibition.

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.

Clapp, R. G., 2009, Reverse time migration with random boundaries: 79th Annual International Meeting, SEG Expanded Abstracts, 2809–2813.

Clapp, R. G., H. Fu, and O. Lindtjorn, 2010, Selecting the right hardware for reverse time migration: The Leading Edge, **29**, 48–58.

Collino, F., and C. Tsogka, 2001, Application of the perfectly matched absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media: Geophysics, **66**, 294–307.

Dussaud, E., W. W. Symes, P. Williamson, L. Lemaistre, P. Singer, B. Denel, and A. Cherrett, 2008, Computational strategies for reverse-time migration: SEG Annual meeting.

Foltinek, D., D. Eaton, J. Mahovsky, P. Moghaddam, and R. McGarry, 2009,

Industrial-scale reverse time migration on gpu hardware: Presented at the 2009 SEG Annual Meeting.

Fomel, S., P. Sava, I. Vlad, Y. Liu, and V. Bashkardin, 2013, Madagascar: open-source software project for multidimensional data analysis and reproducible computational experiments: Journal of Open Research Software, **1**, e8.

Fornberg, B., 1988, Generation of finite difference formulas on arbitrarily spaced grids: Mathematics of computation, **51**, 699–706.

Fu, H., and R. G. Clapp, 2011, Eliminating the memory bottleneck: an fpga-based solution for 3d reverse time migration: Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays, ACM, 65–74.

Guitton, A., B. Kaelin, and B. Biondi, 2006, Least-squares attenuation of reverse-time-migration artifacts: Geophysics, **72**, S19–S23.

Guo, M., Y. Chen, and H. Wang, 2013, The application of gpu-based tti rtm in a complex area with shallow gas and fault shadow-a case history: Presented at the 75th EAGE Conference & Exhibition.

Hussain, T., M. Pericas, N. Navarro, and E. Ayguadé, 2011, Implementation of a reverse time migration kernel using the hce high level synthesis tool: International Conference on Field-Programmable Technology (FPT), IEEE, 1–8.

Ji, Q., S. Suh, and B. Wang, 2012, Iterative velocity model building using gpu based layer-stripping tti rtm, *in* SEG Technical Program Expanded Abstracts 2012: Society of Exploration Geophysicists, 1–5.

Kim, Y., Y. Cho, U. Jang, and C. Shin, 2013, Acceleration of stable {TTI} p-wave reverse-time migration with {GPUs}: Computers & Geosciences, **52**, 204 – 217.

Komatitsch, D., G. Erlebacher, D. Göddeke, and D. Michéa, 2010a, High-order finite-element seismic wave propagation modeling with mpi on a large gpu cluster: Journal of Computational Physics, **229**, 7692–7714.

Komatitsch, D., and R. Martin, 2007, An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation: Geophysics, **72**, SM155–SM167.

Komatitsch, D., D. Michéa, G. Erlebacher, and D. Göddeke, 2010b, Running 3d finite-difference or spectral-element wave propagation codes 25x to 50x faster using a gpu cluster: Presented at the 72nd EAGE Conference & Exhibition.

Leader, C., and R. Clapp, 2012, Least squares reverse time migration on gpus-balancing io and computation: Presented at the 74th EAGE Conference & Exhibition.

Lin, C., and H. Wang, 2012, Application of gpus in seismic depth migration: Presented at the 74th EAGE Conference & Exhibition.

Liu, G., Y. Liu, L. Ren, and X. Meng, 2013a, 3d seismic reverse time migration on gpgpu: Computers & Geosciences, **59**, 17 – 23.

Liu, H., R. Ding, L. Liu, and H. Liu, 2013b, Wavefield reconstruction methods for reverse time migration: Journal of Geophysics and Engineering, **10**, 015004.

Liu, H., B. Li, H. Liu, X. Tong, Q. Liu, X. Wang, and W. Liu, 2012a, The issues of prestack reverse time migration and solutions with graphic processing unit implementation: Geophysical Prospecting, **60**, 906–918.

Liu, H., H. Liu, X. Shi, R. Ding, and J. Liu, 2012b, Gpu based pspi one-way wave

high resolution migration, *in* SEG Technical Program Expanded Abstracts 2012: Society of Exploration Geophysicists, 1–5.

Liu, H.-w., H. Liu, X.-L. Tong, and Q. Liu, 2012c, A fourier integral algorithm and its gpu/cpu collaborative implementation for one-way wave equation migration: Computers & Geosciences, **45**, 139–148.

Liu, W., T. Nemeth, A. Loddoch, J. Stefani, R. Ergas, L. Zhuo, B. Volz, O. Pell, and J. Huggett, 2009, Anisotropic reverse-time migration using co-processors: Presented at the SEG Houston International Exposition. SEG.

McMechan, G., 1983, Migration by extrapolation of time-dependent boundary values: Geophysical Prospecting, **31**, 413–420.

Medeiros, V., R. Rocha, A. Ferreira, J. Correia, J. Barbosa, A. Silva-Filho, M. Lima, R. Gandra, and R. Bragança, 2011, Fpga-based accelerator to speed-up seismic applications: 2011 Simpasio em Sistemas Computacionais (WSCAD-SSC), IEEE, 9–9.

Michéa, D., and D. Komatitsch, 2010, Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards: Geophysical Journal International, **182**, 389–402.

Micikevicius, P., 2009, 3d finite difference computation on gpus using cuda: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, ACM, 79–84.

Nvidia, C., 2011, Nvidia cuda c programming guide.

Suh, S., and B. Wang, 2011, Expanding domain methods in gpu based tti reverse time migration, *in* SEG Technical Program Expanded Abstracts 2011: Society of Exploration Geophysicists, 3460–3464.

Suh, S. Y., A. Yeh, B. Wang, J. Cai, K. Yoon, and Z. Li, 2010, Cluster programming for reverse time migration: The leading edge, **29**, 94–97.

Symes, W. W., 2007, Reverse time migration with optimal checkpointing: Geophysics, **72**, SM213–SM221.

Weiss, R. M., and J. Shragge, 2013, Solving 3d anisotropic elastic wave equations on parallel gpu devices: Geophysics, **78**, F7–F15.

Ying, S., T. Dong-sheng, and K. Xuan, 2013, Denoise investigation on prestack reverse time migration based on gpu/cpu collaborative parallel accelerating computation: Fifth International Conference on Computational and Information Sciences (ICCIS), IEEE, 30–33.

Yoon, K., C. Shin, S. Suh, L. R. Lines, and S. Hong, 2003, 3d reverse-time migration using the acoustic wave equation: An experience with the seg/eage data set: The Leading Edge, **22**, 38–41.