RTM using effective boundary saving: A staggered grid GPU implementation

Pengliang Yang\textsuperscript{a,b,*}, Jinghuai Gao\textsuperscript{a,*}, Baoli Wang\textsuperscript{c}

\textsuperscript{a}Xi’an Jiaotong University, National Engineering Laboratory for Offshore Oil Exploration, Xi’an 710049, China
\textsuperscript{b}Bureau of Economic Geology, Jackson School of Geosciences, The University of Texas at Austin, TX 78713, USA
\textsuperscript{c}CCTEG Xi’an Research Institute, Xi’an 710077, China

\textbf{A R T I C L E  I N F O}

Article history:
Received 4 February 2014
Received in revised form 30 March 2014
Accepted 30 April 2014
Available online 18 April 2014

Keywords:
Reverse time migration
GPU
Absorbing boundary condition
Finite difference
Marmousi
Sigsbee

\textbf{A B S T R A C T}

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.

© 2014 Elsevier Ltd. All rights reserved.

1. 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.
2. Overview of RTM and its computation

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

\[
\frac{1}{v^2(x)} \frac{\partial^2 p(x,t;x_0)}{\partial t^2} = \nabla^2 p(x,t;x_0) + f(t) \delta(x-x_0),
\]

where \( p(x,t;x_0) \) is the wavefield excited by the source at the position \( x = x_0 \), \( \nabla^2 \) stands for the velocity in the media, \( \nabla^2 = \nabla \cdot \nabla = \delta_{x_0} + \delta_{x_2} \), \( f(t) \) denotes the source signature. For the convenience, we eliminate the source term hereafter and use the notation \( \delta_x = \delta_{x_1} + \delta_{x_2} \), \( \delta_{uu} = \delta_{u_1} + \delta_{u_2} \). The forward marching step can be specified after discretization as

\[
p^{k+1} = 2p^k - p^{k-1} + v^2 \Delta t^2 v^2 p^k.
\]

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

\[
I(x) = \frac{\sum_{s=1}^{N} \int_0^{t_{max}} dt \sum_{g=1}^{ng} p_s(x,t;x_0)p_g(x,t;x_1)}{\sum_{s=1}^{N} \int_0^{t_{max}} dt \sum_{g=1}^{ng} |p_s(x,t;x_0)p_g(x,t;x_1)|}
\]

where \( I(x) \) is the migrated image at point \( x \); 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:

\[
I(x) = \frac{\sum_{s=1}^{N} \int_0^{t_{max}} dt \sum_{g=1}^{ng} p_s(x,t;x_0)p_g(x,t;x_1)}{\sum_{s=1}^{N} \int_0^{t_{max}} dt |p_s(x,t;x_0)p_g(x,t;x_1)|}
\]

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).

3. 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.

3.1. 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,

\[
p^{k-1} = 2p^k - p^{k+1} + v^2 \Delta t^2 v^2 p^k.
\]

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

\[
\text{for } ix, iz... \quad p_0(\cdot) = 2p_1(\cdot) - p_2(\cdot) + v^2(\cdot) \Delta t^2 v^2 p_1(\cdot)
\]

where \( \cdot = [ix, iz] \), \( p_0 \) 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 Fig.1, the original model size \( A_1A_2A_3A_4 \) is extended to \( C_1C_2C_3C_4 \). In between is the artificial boundary \( (C_1C_2C_3A_1A_2A_3A_4) \). Actually, the wavefield we intend to reconstruct is not the part in extended artificial boundary \( C_1C_2C_3A_1A_2A_3A_4 \) but the part in the original model zone \( A_1A_2A_3A_4 \). We can reduce the boundary load further (from whole \( C_1C_2C_3A_1A_2A_3A_4 \) 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 \( C_1C_2C_3A_1A_2A_3A_4 \)). Furthermore, we only need to compute the imaging condition in the zone \( A_1A_2A_3A_4 \), no concern with the part in \( C_1C_2C_3A_1A_2A_3A_4 \).

3.2. Effective boundary for regular grid finite difference

Assume 2Nth order finite difference scheme is applied. The Laplacian operator is specified by

\[
\nabla^2 p^k = \delta_{x_0} p^k + \delta_{x_2} p^k = \frac{1}{\Delta x^2} \sum_{i=-N}^{N} c_i p^k[ix+i]
\]

\[
+ \frac{1}{\Delta x^2} \sum_{i=-N}^{N} c_i p^k[ix+i][iz]
\]
where \(c_i\) is given by Table 1, see a detailed derivation in Fornberg (1988). The Laplacian 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 Fig. 2. In 2-D case, the required boundary zone has been plotted in Fig. 3a. Note that four corners in \(B_1B_2B_3B_4\) in Fig. 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 \(A_1A_2A_3A_4B_1B_2B_3B_4\) 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 \(A_1A_2A_3A_4D_1D_2D_3D_4\), as shown in Fig. 3b. We call it the effective boundary for regular finite difference scheme.

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 \(A_1A_2A_3A_4D_1D_2D_3D_4\). 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 \(D_1D_2\) is correctly reconstructed. Then we load the saved effective boundary of time \(k\) to overwrite the area \(A_1A_2A_3A_4D_1D_2D_3D_4\). Again, all points of the wavefield in \(A_1A_2A_3A_4\) are correct. We repeat this overwriting and computing process from one step to another \((k \rightarrow k-1)\), in reverse time order. The wavefield in the boundary \(C_1C_2C_3C_4A_1A_2A_3A_4\) may be incorrect because the points here are neither saved nor correctly reconstructed from the previous step.

### 3.3. 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

\[
\partial_z u = \frac{1}{\Delta x} \sum_{i=-N}^{N} c_i^z [u[i+1] - u[i]], \quad N = 4
\]

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 auxiliary variables: \(Ax = A_x p, Az = \partial_z p,\) etc.
Table 2
Finite difference coefficients for staggered grid (Order-2N).

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>N−1</td>
<td>1</td>
<td></td>
<td>−0.0416667</td>
<td></td>
</tr>
<tr>
<td>N−2</td>
<td>1.125</td>
<td></td>
<td>0.0065104667</td>
<td>0.0046875</td>
</tr>
<tr>
<td>N−3</td>
<td>1.171875</td>
<td></td>
<td>0.0095703125</td>
<td>−0.000697544642857</td>
</tr>
<tr>
<td>N−4</td>
<td>1.1962890625</td>
<td></td>
<td>−0.079752604167</td>
<td></td>
</tr>
</tbody>
</table>

According to Dussaud et al. (2008), for 2Nth 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

\[ 2N \cdot nx + 2N \cdot nx = 2N(nx + nx). \] (10)

In the proposed effective boundary saving strategy, the number becomes

\[ 2N \cdot nx + 2N \cdot nx - 4N^2 = 2N(nx + nx) - 4N^2. \] (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

\[ 2(2N - 1)nx + 2(2N - 1)nx - 4(2N - 1)^2 = 2(2N - 1)(nx + nx) - 4(2N - 1)^2 \] (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 . size-of(float) = 4nt. Table 3 lists this memory requirement for different boundary saving strategies.

In principle, the proposed effective boundary saving will reduce 4nt · 4N² 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 ≤ nb ≤ nx, nx. 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.

4. GPU implementation using CPML boundary condition

4.1. 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:

\[ \frac{\partial^2 p}{\partial t^2} = v^2 (P_x + P_z) \]
\[ P_x = \partial_t Ax + \Psi_x \]
\[ P_z = \partial_t Az + \Psi_z \]
\[ Ax = \partial_t p + \Phi_x \]
\[ Az = \partial_t p + \Phi_z \]

where \( \Psi_x, \Psi_z \) are the convolutional terms of Ax and Az; \( \Phi_x, \Phi_z \) are the convolutional terms of P_x and P_z. These convolutional terms can be computed via the following relation:

\[ \Psi_x = b_x \Psi_x^{n-1} + (b_x - 1) \Phi_x^{n+1/2} \]
\[ \Psi_z = b_z \Psi_z^{n-1} + (b_z - 1) \Phi_z^{n+1/2} \]
\[ \Phi_x = b_x \Phi_x^{n-1} + (b_x - 1) \Phi_x^{n-1/2} \]
\[ \Phi_z = b_z \Phi_z^{n-1} + (b_z - 1) \Phi_z^{n-1/2} \] (14)

According to Dussaud et al. (2008), for 2Nth 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

\[ 2N \cdot nx + 2N \cdot nx = 2N(nx + nx). \] (10)

In the proposed effective boundary saving strategy, the number becomes

\[ 2N \cdot nx + 2N \cdot nx - 4N^2 = 2N(nx + nx) - 4N^2. \] (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

\[ 2(2N − 1)nx + 2(2N − 1)nx − 4(2N − 1)^2 = 2(2N − 1)(nx + nx) − 4(2N − 1)^2 \] (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 . size-of(float) = 4nt. Table 3 lists this memory requirement for different boundary saving strategies.

In principle, the proposed effective boundary saving will reduce 4nt · 4N² 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 ≤ nb ≤ nx, nx. 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.

4. GPU implementation using CPML boundary condition

4.1. 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:

\[ \frac{\partial^2 p}{\partial t^2} = v^2 (P_x + P_z) \]
\[ P_x = \partial_t Ax + \Psi_x \]
\[ P_z = \partial_t Az + \Psi_z \]
\[ Ax = \partial_t p + \Phi_x \]
\[ Az = \partial_t p + \Phi_z \]

where \( \Psi_x, \Psi_z \) are the convolutional terms of Ax and Az; \( \Phi_x, \Phi_z \) are the convolutional terms of P_x and P_z. These convolutional terms can be computed via the following relation:

\[ \Psi_x = b_x \Psi_x^{n-1} + (b_x - 1) \Phi_x^{n+1/2} \]
\[ \Psi_z = b_z \Psi_z^{n-1} + (b_z - 1) \Phi_z^{n+1/2} \]
\[ \Phi_x = b_x \Phi_x^{n-1} + (b_x - 1) \Phi_x^{n-1/2} \]
\[ \Phi_z = b_z \Phi_z^{n-1} + (b_z - 1) \Phi_z^{n-1/2} \] (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):

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

(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 $10^{-3} - 10^{-6}$. 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).

### 4.3. 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 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_wavel(....)` with a dominant frequency $f_m$ and delayed wavelength. Adding a shot can be done by a smooth bell transition `cuda_add_bellwlt(....)`. We implement RTM (of order $N_f=2, 4, 6, 8, 10$) 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

---

1 Generally, a computer has same or larger amount of resource on host compared with GDDR memory on device.
applying the muting function `cuda_mute(...)` and the Laplacian filtering `cuda_laplace_filter(...)`.

5. Numerical examples

5.1. Exact reconstruction

To make sure that the proposed effective boundary saving strategy does not introduce any kind of error/artifact for the source wavefield, the first example is designed using a constant velocity model: velocity $= 2000$ m/s, $n_z = n_x = 320$, $\Delta z = \Delta x = 5$ m. The source position is set at the center of the model. The modeling process is performed with $n_t = 1000$ time samples. We record the modeled wavefield at $k=420$ and $k=500$, as shown in the top panels of Fig. 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 Fig. 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.

5.2. Marmousi model

The second example is GPU-based RTM for Marmousi model (Fig. 7) using our effective boundary saving. The spatial sampling interval is $\Delta x = \Delta z = 4$ m. 51 shots are deployed. In each shot, 301 receivers are placed in the split shooting mode. The parameters we use are listed as follows: $n_t = 13000$, $\Delta t = 0.3$ ms. Due to the limited resource on our computer, we store 65% boundaries using page-locked memory. Fig. 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

![Fig. 6. The wavefield snaps with a constant velocity model: velocity $= 2000$ m/s, $n_z = n_x = 320$, $\Delta z = \Delta x = 5$ m, source at the center. The forward modeling is conducted with $n_t = 1000$ time samples. (a–b) Modeled wavefield snaps at $k=420$ and $k=500$. The backward propagation starts from $k=1000$ and ends at $k=1$ and (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).](image-url)
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.

5.3. Sigsbee model

The last example is Sigsbee model shown in Fig. 9. The spatial interval is $\Delta x = \Delta z = 25$ m. 55 shots are evenly distributed on the surface of the model. We still perform $n_t = 13,000$ 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 Fig. 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

![Fig. 7. The Marmousi velocity model.](image)

![Fig. 8. RTM result of Marmousi model using effective boundary saving scheme (staggered grid finite difference). (a) Result of cross-correlation imaging condition and (b) result of normalized cross-correlation imaging condition.](image)

![Fig. 9. The Sigsbee velocity model.](image)
have a much lower amplitude or are even completely lost in the image of cross-correlation imaging condition.

6. 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 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 led 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).

Appendix A. Supplementary data

Supplementary data associated with this article can be found in
the online version at http://dx.doi.org/10.1016/j.cageo.2014.04.004.

References

modeling and reverse time migration on a GPU cluster. In: International
Conference on High Performance Computing & Simulation, HPCS'09, IEEE,
pp. 36–43.
48, 1514–1524.
Multisource reverse-time migration and full-waveform inversion on a GPGPU.
In: 72nd EAGE Conference & Exhibition.
condition for discrete acoustic and elastic wave equations. Geophysics 50,
705–708.
Clapp, R.G., Fu, H., Lindgren, O., 2010. Selecting the right hardware for reverse
model to the linear elastodynamic problem in anisotropic heterogeneous
Dussaud, E., Symes, W.W., Williamson, P., Lemaistre, L., Singer, P., Denel, B., Cherrett,
A., 2008. Computational Strategies for Reverse-Time Migration. SEG Annual
meeting. Las Vegas.
Foltinek, D., Eaton, D., Mahovsky, J., Moghaddam, P., McGarry, R., 2009. Industrial-
scale reverse time migration on GPU hardware. In: 2009 SEG Annual Meeting.
software project for multidimensional data analysis and reproducible compu-
solution for 3D reverse time migration. In: Proceedings of the 19th ACM/SIGDA
International Symposium on Field Programmable Gate Arrays, ACM, pp. 65–74.
Gutruf, A., Kaelin, B., Biondi, B., 2006. Least-squares attenuation of reverse-time-
Guo, M., Chen, Y., Wang, H., 2013. The application of GPU-based TTI RTM in a
complex area with shallow gas and fault shadow-a case history. In: 75th EAGE
Conference & Exhibition.
time migration kernel using the HCE high level synthesis tool. In: International
Conference & Exhibition.
Ji, Qi, Suh, S., Wang, B., 2012. Iterative velocity model building using GPU based
layer-stripping TTI RTM. In: SEG Technical Program Expanded Abstracts 2012,
Kim, Y., Cho, Y., Jang, U., Shin, C., 2013. Acceleration of stable tti p-wave reverse-
Komatitsch, D., Erlebacher, G., Goddeke, D., Michéa, D., 2010a. High-order finite-
element seismic wave propagation modeling with MPI on a large GPU cluster.
Komatitsch, D., Martin, R., 2007. An unsplit convolutional perfectly matched layer
improved at grazing incidence for the seismic wave equation. Geophysics 72,
SM155–SM167.
Komatitsch, D., Michéa, D., Erlebacher, G., Goddeke, D., 2010. Running 3d finite-
difference or spectral-element wave propagation codes 25× to 50× faster using
a GPU cluster. In: 72nd EAGE Conference & Exhibition.
Leader, C., Clapp, R., 2012. Least squares reverse time migration on GPUS-balancing
IO and computation. In: 74th EAGE Conference & Exhibition.
EAGE Conference & Exhibition.
Lin, C., Liu, Y., Ren, L., Meng, X., 2013a. 3D seismic reverse time migration on GPUS.
time migration. J. Geophys. Eng. 10, 015004.
reverse time migration and solutions with graphic processing unit implementa-
tion. Geophysics 76, 906–918.
Liu, H., Liu, H., Shi, X., Ding, R., Liu, J., 2012b. GPU based PSPI one-way wave high
resolution migration. In: SEG Technical Program Expanded Abstracts 2012,
CPU collaborative implementation for one-way wave equation migration.
Liu, W., Nemeth, T., Loddoch, A., Stefani, J., Ergas, R., Zhuo, L., Volz, B., Pell, O.,
Huggett, J., 2009. Anisotropic reverse-time migration using co-processors. In:
SEG Houston International Exposition. SEG.
McMechan, G., 1983. Migration by extrapolation of time-dependent boundary
Medeiros, V., Rocha, R., Ferreira, A., Correia, J., Barbosa, J., Silva-Filho, A., Lima, M.,
Gandra, R., Bragança, R., 2011. FPGA-based accelerator to speed-up seismic
IEEE, p. 9.
Michéa, D., Komatitsch, D., 2010. Accelerating a three-dimensional finite-difference
Mickievicius, P., 2009. 3D finite difference computation on GPUs using CUDA.
In: Proceedings of 2nd Workshop on General Purpose Processing on Graphics
Processing Units, ACM, pp. 79–84.
Suh, S., Wang, B., 2011. Expanding domain methods in GPU based TTI reverse
time migration. In: SEG Technical Program Expanded Abstracts 2011, Society of
Symes, W.W., 2007. Reverse time migration with optimal checkpointing. Geophy-
sics 72, F15.
Weiss, R.M., Shragge, J., 2013. Solving 3D anisotropic elastic wave equations on
Ying, S., Dong-sheng, T., Xuan, K., 2013. Denoise investigation on prestack reverse
time migration based on GPU/CPU collaborative parallel accelerating computa-
tion. In: Fifth International Conference on Computational and Information
Sciences (ICCIS), IEEE, pp. 30–33.
Yoon, K., Shin, C., Suh, S., Lines, L.R., Hong, S., 2003. 3D reverse-time migration using
the acoustic wave equation: an experience with the SEG/EAGE data set. Lead.
Edge 22, 38–41.