Expanding domain methods in GPU based TTI reverse time migration

Sang Suh and Bin Wang*

Summary

Three expanding domain methods are studied which include: (1) constant velocity layer, (2) eikonal equation solver and (3) amplitude comparison. The layer method requires the least traveltime computing overhead but has a loose domain size. The eikonal equation solver method gives a relatively compact domain size but requires relatively expensive traveltime computation. The amplitude comparison method is accurate and less expensive. The expanding domain method is used for efficient finite-difference Reverse Time Migration (RTM).

A Graphics Processing Unit (GPU) kernel of cross-derivative computation is introduced. The algorithm attempts to minimize the global memory access count and is thought to be highly efficient because it minimizes global memory access and uses fast shared memory buffers as in the previously known GPU-based Laplacian kernel found in isotropic RTM. The new kernel is used in GPU-based Tilted Transverse Isotropic (TTI) RTM which implements a nested multi-thread technique efficiently utilizing CPU and GPU resources in parallel. The program is tested on Gulf of Mexico (GOM) field data to give a migrated image.

Introduction

RTM propagates wavefields in time through the use of the two-way wave equation (Baysal et al., 1984; Whitmore, 1983). It correctly handles both multi-arrivals and phase changes. Its main advantage over the one-way equation technique is that it has no dip limitation. Thus RTM enables imaging of a very complex subsurface. Two-way migration methods require significantly greater computational resources than one-way migration methods.

Robert et al. (2010) discussed various important issues on the efficient implementation of RTM. To reduce the wavefield modeling costs, they introduced four methods: (1) following the wavefield, (2) domain decomposition and resampling, (3) property compression and (4) cache-oblivious approaches.

We found that the first of the above four methods, following the wavefield, is very important and discuss it further in this paper.

Micikevicius (2009) introduced a 3D finite difference GPU kernel for isotropic RTM. His method computes a 3D wavefield Laplacian using a 2D horizontal stencil marching in the vertical direction. This turned out to be the most effective method of reducing the memory access redundancy. Extending his idea to TTI RTM, we introduce a GPU kernel algorithm for 3D cross-derivative computation.

Expanding Domain Methods

In the early stage of modeling, most of the wavefield is zero except near the source point. The method of "following the wavefield" involves skipping the wavefield computation where the wavefield is known to be zero, thereby reducing the computation cost.

To actually implement this idea, we must compute the non-zero wavefield area (i.e. the traveltime). The traveltime computation must be fast but accurate. The wavefield domain is expanded preceding the wavefront.

We discuss three methods of expanding the wavefield domain: (1) constant velocity layer, (2) eikonal equation solver and (3) amplitude comparison.

In the constant velocity layer method, the traveltime is computed from a simplified constant velocity layer model. For traveltimes along the z-axis, a horizontal layer model is constructed from the 3D velocity model, v(x,y,z), to 1D velocity, V(z), where V(z) is the maximum velocity for each depth. This procedure can be repeated for the x- and y-axis, producing traveltimes in the horizontal direction.

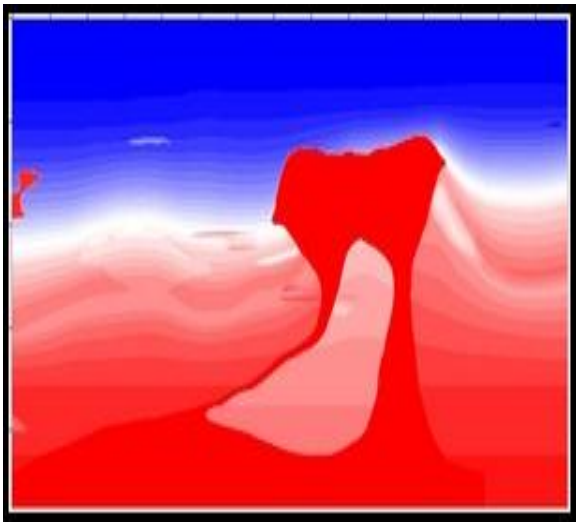

***Figure 1**. The 2004 BP synthetic velocity model.*

Figure 1 shows a part of the 2004 BP synthetic model (Billette et al., 2005). The left column in Figure 2 shows a series of expanding domain boundaries and snapshots computed by the constant velocity layer method. The traveltimes vary from 0.5 s to 3.0 s with an increment of 0.5 s from top to bottom. The figure shows too large of a domain boundary especially in the horizontal direction. We need more accurate traveltimes.

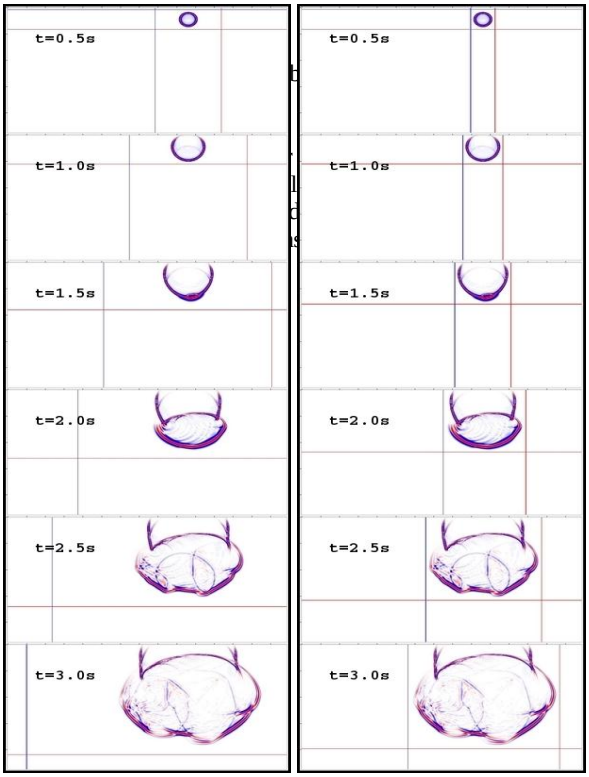

Figure 2*. The left column shows expanding domain by the constant velocity method, and the right column shows expanding domain by the eikonal solver method.*

The eikonal equation solver is the classical method of computing traveltimes (Vidale, 1990). Recently, Fomel (2010) published source code for eikonal equation solver. There are two programs, sfeikonal and sfeikonalvti. The first is for isotropic media, and the second is for vertical transverse isotropic (VTI) media. The programs use the fast marching method (Sethian et al., 1999) exploiting the priority queue to sort the wavefront traveltimes efficiently. For TTI media, we simply use the isotropic eikonal solver replacing the velocity along the symmetry plane.

The right column in Figure 2 shows a series of expanding domain boundaries and snapshots computed by the eikonal equation solver method. The traveltimes vary from 0.5 s to 3.0 s with an increment of 0.5 s from top to bottom. The domain boundary of this figure is more compact than the previous one. Figure 3 shows the source field traveltimes with a Contour Interval (CI) of 0.5 s. The problem with this method is that traveltime computation consumes a significant amount of CPU time, especially in 3D, large-aperture applications.

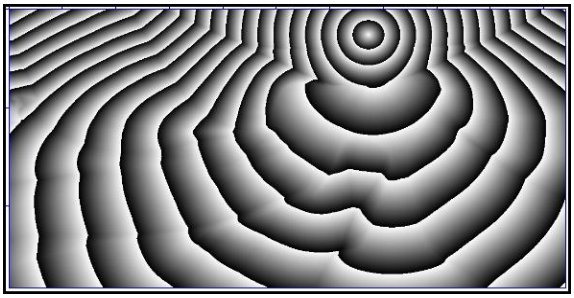

Figure 3. *Traveltimes by eikonal solver (CI = 0.5 s).*

In contrast to the previous two methods, which compute traveltimes before wavefield modeling, the amplitude comparison method does not pre-compute traveltimes. Instead, the domain boundary of the next time-step is computed based on the current wavefield for each time-step. For this, the amplitude in front of the wavefront is compared to the maximum wavefield amplitude. If the ratio exceeds certain limits, the domain is expanded; otherwise, the domain size does not change.

In the actual implementation, we do not examine amplitudes of domain boundaries directly. Instead we examine a few grid points, i.e. four, inside of each domain boundary. If any of the amplitude exceeds certain limit, the domain boundary is expanded by one grid point. We use 1/1024 of the maximum amplitude as the limit. This method produced an almost identical expanding domain boundary as in the eikonal equation solver method with very little computing overhead.

The three expanding domain methods are tested for 3D TTI RTM of a 16x16x16 km aperture shot. The elapsed times normalized by the times of the non-expanding method are 0.78 s, 0.82 s, and 0.75 s, respectively. This shows that the amplitude comparison method is the most efficient, while the eikonal solver method is inefficient especially on large aperture migrations because of its very large computing overhead.

GPU Kernel for TTI RTM

The recent development in Graphics Processing Units (GPU) technologies with unified architecture and general purpose languages coupled with the high and rapidly increasing performance throughput of these computers have made General Purpose Graphics Processing Units (GPGPU) an attractive solution to speed up diverse applications (Abdelkhalek et al., 2009).

Micikevicius (2009) introduced a 3D finite difference GPU kernel for reverse time migration. His method computes 3D wavefield Laplacian using a 2D horizontal stencil marching in vertical direction. This turned out to be the most effective method of reducing the memory access redundancy. The 8th order stencil-only prototype implementation gives more than 4,000 MPoints/s of finite difference performance on a C1060 GPU device.

Besides the wavefield Laplacian, wavefield modeling in TTI media requires computation of wavefield cross-derivatives such as:

$$\partial^2 P / \partial x \partial y, \ \partial^2 P / \partial x \partial z, \ \partial^2 P / \partial y \partial z$$

(Fletcher et al., 2009; Yoon et al., 2010). We discuss an algorithm that computes the cross-derivatives using GPU. Our method extends Micikevicius' algorithm so that the number of global memory accesses can be minimized with the help of faster shared memory buffers.

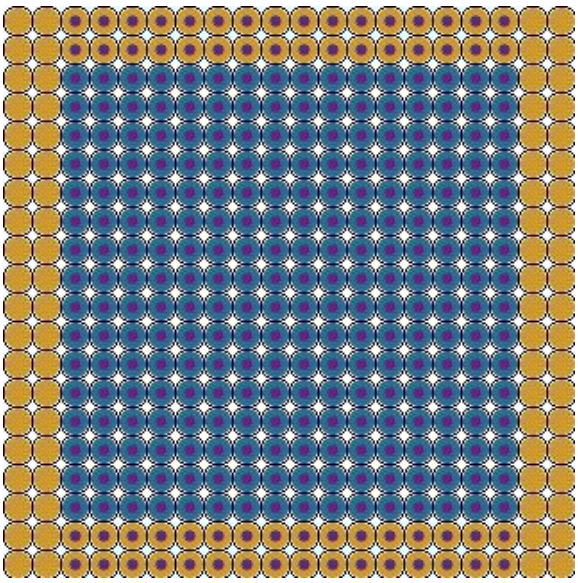

Figure 4. *A 16x16 data plus 2x2 halo tile.*

Let us suppose the number of threads in the x- and y-direction in a thread block is BDIMX by BDIMY. We allocate two shared memory arrays, s_pp and s_px as follow:

```
__shared__ float s_pp[BDIMY_RR][BDIMX_RR];
__shared__ float s_px[BDIMY_RR][BDIMX];
```

where BDIMX_RR = BDIMX + 2*R, and BDIMY_RR = BDIMY + 2*R, with R being the arm length of the first derivative operator. The arm length, R, of order N centered finite-difference first derivative is N/2, because N/2 adjacent values are used in the left and right directions, respectively.

For a given output depth Z, s_pp[][] holds input data of a 2D horizontal tile of depth Z, where Z=z+R. Figure 4 shows a 16x16 horizontal tile and 2x2 halos of the cross-derivative scheme. This assumes BDIMX=16, BDIMY=16, and R=2. The blue colored 16x16 points are the main input area. Each thread reads one value from the global memory and writes to the tile. The surrounding brown colored two-point wide area is the halo. Because the number of points and number of threads do not match, only a subset of the threads transfers data from global memory to this halo. For efficient computation, the size ratio between the halo and main area must be minimized as long as the GPU resource permits.

Also, we allocate three thread local arrays, local_px[DIA], local_py[DIA], and local_pxy[R] where DIA = R+R+1. These arrays store x-derivative, y-derivative and xy-derivatives, respectively, of a sliding window in the z-direction. After loading the data of depth Z, we compute local_px[Z], local_py[Z] using the 2R-th order centered finite difference method.

The second shared memory buffer, s_px[][], is shown as purple dots in Figure 4. The buffer stores x-derivatives of size BDIMY_RR by BDIMX at depth Z. The main area values are already computed and are copied from the thread local array, local_px[Z]. The halo area values are computed by a subset of the threads from the first shared memory buffer using 2R-th order centered finite-difference. Synchronizing all threads in a block, each thread can share s_px[][] from adjacent threads to compute its y-derivative. This is xy-derivative at depth Z, which we store at local_pxy[Z].

The final three derivatives, pxy[z], pyz[z], pxz[z] are computed as follow:

(1) pxy[z] is read from the local_pxy[] array.
(2) pyz[z] is computed by z-derivative of local_py[]
(3) pxz[z] is computed by z-derivative of local_px[]

Other aspects of GPU implementation of TTI RTM

A GPU based TTI reverse time migration program is written. The program uses GPU for compute intensive wavefield modeling while the CPU takes care of snapshot i/o to and from disk and imaging. To maximize the snapshot i/o performance, the wavefield is compressed using three stages: (1) dynamic range reduction, (2) quantization and (3) Hauffman coding. The compression algorithm is designed to minimize the CPU time and not to exceed the GPU wavefield modeling time.

The fully parallel, nested multithread implementation of the compression is so successful that snapshot i/o and imaging times are completely hidden by the GPU wavefield modeling time in a typical TTI reverse time migration.

Application of GPU RTM

In complex geological areas such as the GOM, TTI RTM has been routinely used for velocity model building. In a typical GOM imaging project, multiple iterations of TTI RTM imaging are required for velocity modeling. To reduce the turnaround time, we need to dramatically improve the RTM efficiency. We have successfully deployed GPU racks in our processing center, and the GPU RTM program has been actively used in RTM production. Figure 5 is an example of our GPU RTM image from one of our WAZ surveys in GOM. GPU RTM has not only dramatically reduced iteration turnaround time, but also produced a high quality RTM image.

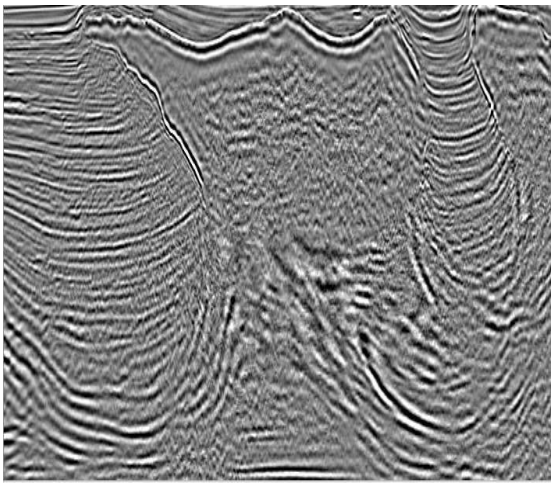

Figure 5. *GPU RTM image of a WAZ data set from GOM.*

Conclusions

The expanding domain method reduces the total number of operations required for wavefield modeling. We studied three expanding domain methods: (1) constant velocity layer method, (2) eikonal equation solver method and (3) amplitude comparison method. The layer method requires almost no computing overhead in traveltime computation. However, its domain size is too loose compared to the other method. The eikonal equation solver method gives a relatively compact domain size. However, the traveltime computation overhead is too large especially in a large aperture 3D application. The amplitude comparison method gives an almost identical domain size as in the eikonal equation solver method with very little overhead. The expanding domain method can be used for efficient finite-difference RTM.

A GPU kernel of cross-derivative computation is introduced. The algorithm attempts to minimize the global memory access count and is thought to be highly efficient because it minimizes global memory access and uses fast shared memory buffers as in the previously known GPU-based Laplacian kernel found in isotropic RTM.

The new kernel is used in the GPU-based TTI RTM program, which is a nested multithread program efficiently utilizing CPU and GPU resources. The program is actively used in RTM production.

Acknowledgements

The authors thank Paulius Micikevicius of NVIDIA for sharing his CUDA implementation of the high-order finite difference code. We thank BP for providing access to their synthetic model data. We would like to thank the following TGS colleagues for their contributions and support: Xinyi Sun, Alex Yeh, Kwangjin Yoon, James Cai, Xuening Ma, Gary Rodriguez and Zhiming Li. We also thank Laurie Geiger, Simon Baldock and Chuck Mason for reviewing and proof-reading this paper. Final thanks go to TGS management for the permission of publishing this paper.

EDITED REFERENCES

Note: This reference list is a copy-edited version of the reference list submitted by the author. Reference lists for the 2011 SEG Technical Program Expanded Abstracts have been copy edited so that references provided with the online metadata for each paper will achieve a high degree of linking to cited sources that appear on the Web.

REFERENCES

Abdelkhalek, R., H. Canandra, O. Coulaud, G. Latu, and J. Roman, 2009, Fast seismic modeling and reverse time migration on a GPU cluster: 2009 High Performance Computing Symposium, Proceedings, http://hal.inria.fr/docs/00/40/39/33/PDF/hpcs.pdf, accessed 2 June, 2011.

Billette, F., and S. Brandsberg-Dahl, 2005, The 2004 BP velocity benchmark: 67th Conference and Exhibition, EAGE, Extended Abstracts, B035.

Clapp, R. G., H. Fu, and O. Lindtjorn, 2010, Selecting the right hardware for reverse time migration: The Leading Edge, **29**, 48–58, doi:10.1190/1.3284053.

Fletcher, R. P., X. Du, and P. J. Fowler, 2009, Reverse time migration in tilted transversely isotropic (TTI) media: Geophysics, **74**, no. 6, WCA179–WCA187.

Foltinek, D., D. Eaton, J. Mahovsky, P. Moghaddam, and R. McGarry, 2009, Industrial-scale reverse time migration on GPU hardware: 79th Annual International Meeting, SEG, Expanded Abstracts, 2789–2793.

Fomel, S., 2010, Fast marching eikonal solver (3-D): http://www.reproducibility.org/RSF/sfeikonal.html, accessed 2 June 2011.

Micikevicius, P., 2009, 3D finite difference computation on GPUs using CUDA: GPGPU-2: Proceedings of 2nd workshop on general purpose processing on graphics processing units, 79–84.

Sethian, J. A., and A. M. Popovici, 1999, 3-D traveltime computation using the fast marching method: Geophysics, **64**, 516–523.

Sun, X., and S. Suh, 2011, Maximizing throughput for high productive RTM: From CPU-RTM to GPU-RTM: 81st Annual International Meeting, SEG, Expanded Abstracts, in press.

Vidale, J. E., 1990, Finite-difference calculation of traveltimes in three dimensions: Geophysics, **55**, 521–526.

Yoon, K., S. Suh, J. Ji, J. Cai, and B. Wang, 2010, Stability and speedup issues in TTI RTM implementation: 80th Annual International Meeting, SEG, Expanded Abstracts, 3221–3224