DE GRUYTER OPEN

Acta Geophysica
vol. 64, no. 4, Aug. 2016, pp. 1051-1063
DOI: 10.1515/acgeo-2016-0033

Practical Implementation of Prestack Kirchhoff Time Migration on a General Purpose Graphics Processing Unit

Guofeng LIU[1,2] and Chun LI[1]

[1]Key Laboratory of Geo-detection, China University of Geosciences, Ministry of Education, Beijing, China; e-mail: liugf@cugb.edu.cn

[2]State Key Laboratory of Continental Tectonics and Dynamics, Institute of Geology, Chinese Academy of Geological Sciences, Beijing, China

Abstract

In this study, we present a practical implementation of prestack Kirchhoff time migration (PSTM) on a general purpose graphic processing unit. First, we consider the three main optimizations of the PSTM GPU code, *i.e.*, designing a configuration based on a reasonable execution, using the texture memory for velocity interpolation, and the application of an intrinsic function in device code. This approach can achieve a speedup of nearly 45 times on a NVIDIA GTX 680 GPU compared with CPU code when a larger imaging space is used, where the PSTM output is a common reflection point that is gathered as $I[nx][ny][nh][nt]$ in matrix format. However, this method requires more memory space so the limited imaging space cannot fully exploit the GPU sources. To overcome this problem, we designed a PSTM scheme with multi-GPUs for imaging different seismic data on different GPUs using an offset value. This process can achieve the peak speedup of GPU PSTM code and it greatly increases the efficiency of the calculations, but without changing the imaging result.

Key words: GPGPU, offset splitting, parallelization, PSTM, texture memory.

1. INTRODUCTION

Seismic exploration is an important area of geophysical research, which aims at determining subsurface structures to detect where oil and gas can be found and recovered. Prestack Kirchhoff time migration (PSTM) is one of the most popular migration techniques used for seismic data processing because of its simplicity, efficiency, feasibility, and target-orientated properties (Bevc 1997). However, the practical application of PSTM to tasks during large 3D surveys is still computationally intensive. To accelerate the processing of migration, the parallel processing of prestack time migration has been implemented routinely on distributed parallel computers (Schleicher and Copeland 1993, Chen *et al.* 1993), as well as on PC clusters (Morton *et al.* 1999, Hellman 2000, Dai 2005). In recent years, many other devices have also been used to accelerate PSTM such as FPGAs (He *et al.* 2005).

Recently, programmable graphics processor units (GPUs) have evolved into a computing workhorse. GPUs possess multiple cores with a very high memory bandwidth, which makes them useful resources for graphics and non-graphics processing (NVIDIA 2012). Potentially, GPUs can achieve hundreds or even thousands of GFLOPS, whereas general CPUs are only capable of dozens of GFLOPS at present. NVIDIA's computed unified device architecture (CUDA) provides a C-like programming model for exploiting the massively parallel processing power of NVIDIA's GPU (NVIDIA 2013), and it is now employed widely for many parallel computation applications (Lu *et al.* 2013, Capuzzo-Dolcetta and Spera 2013, Westphal *et al.* 2014). Some studies have also used NVIDIA GPUs to accelerate PSTM. In particular, Liu *et al.* (2009) discussed the possibility of parallel computation with NVIDIA GPUs. Shi *et al.* (2011) proposed a method for accelerating PSTM on GPUs by splitting the PSTM procedure into four consequence kernels according to the GPU memory limitations, as well as considering the floating point error problem, which may lead to differences when comparing PSTM with CPU computations.

In this study, we propose the possible application of computing with PSTM on GPUs by combining the characteristics of GPU and PSTM algorithms. First, we consider three key optimization points, *i.e.*, designing a configuration that achieves a reasonable execution, using texture memory for velocity interpolation, and employing faster intrinsic functions. After analyzing the efficiency of this method, we also propose a multi-GPU scheme that employs a splitting offset to keep the GPU busy. Our test results demonstrate that the proposed method achieves a great speedup.

2. REVIEW OF PSTM

PSTM (Fig. 1) uses seismic traces that originate from a source S, which are received at R as input data. The variable, h, is an offset of the distance be-

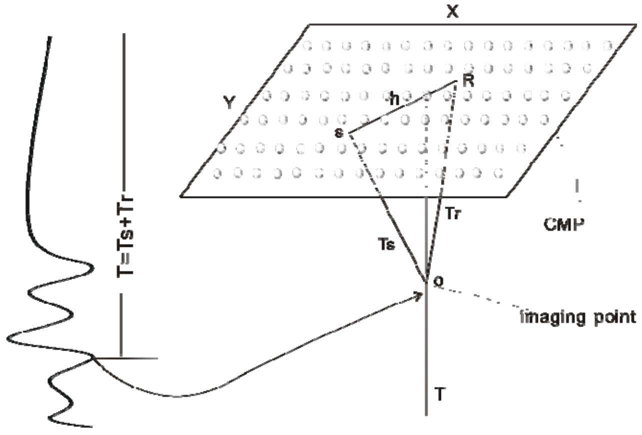

Fig. 1. PSTM scheme.

tween the source and receiver pair, where the 3D imaging space is a uniform common midpoint (CMP) on the ground surface and an imaging point in the depth direction. However, it expresses the depth using a two-way travel time, *T*. For an imaging point, *O*, in the *T* direction, the imaging value for a certain input trace is the amplitude of the input trace at time *t*, *i.e.*, the seismic wave travel time from *S* to *O* and then back to the surface for *O* to *R*. The travel time can be calculated by the straight ray equation:

$$T^2\left(x,y,t_0\right)=t_0^2(x,y)+\frac{x^2}{v^2\left(x,y,t_0\right)}\ , \tag{1}$$

where *T* is the two-way travel time, t_0 is the vertical time from the surface to the imaging point, *x* is the distance from the source or receiver to the imaging CMP, and *v* is the root squared mean velocity of the imaging point, which is obtained by the velocity analysis method in seismic processing and it is interpolated to all of the imaging points.

The output of the PSTM for an imaging CMP is called a common reflection gather (CRP), which is used to update *v* in Eq. 1 until we obtain a better image of the subsurface. In Fig. 2 panel (a) is the CRP of a CMP. Stacking all of the traces in this gather can yield the imaging trace of the CMP, while combining all of the CMPs produces the final imaging profile (Fig. 2c), which is a 2D seismic PSTM imaging profile that depicts the geological structure of the subsurface.

In addition to this PSTM kernel, all input seismic traces require an anti-aliasing process (Lumley *et al.* 1993) and an amplitude-preserving weight (Sun and Martinez 2002). The pseudo-code for Algorithm 1 (Table 1) shows how one seismic trace is processed for PSTM imaging.

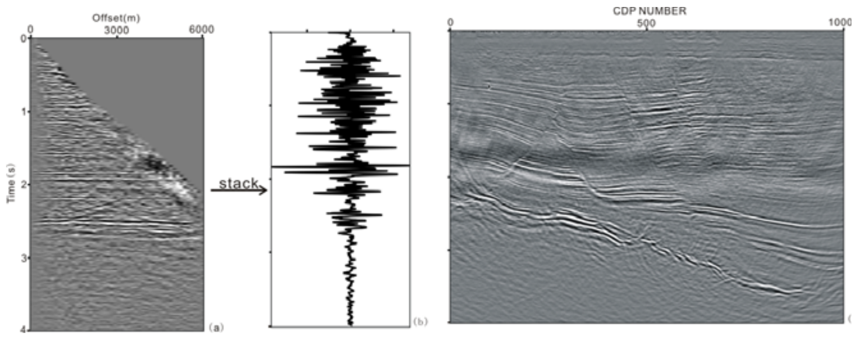

Fig. 2: (a) CRP gather of a midpoint, (b) imaging trace of a midpoint after stacking the CRP gather, and (c) final imaging section obtained by PSTM.

Table 1

Algorithm 1: PKTM kernel

```
Input: velocity model $v[ix][iy][it]$
    seismic data trace $[ih][it]$ source at $S(x,y)$ and receiver at $R(x,y)$
    other migration parameters: time interval $dt$, etc.
Output: common reflection gather $I[ix][iy][ih][it]$

Anti-aliasing processing for input trace;
    for ($ix = 0$; $ix < nx$; $ix$++) // CMP in $x$ direction
    for ($iy = 0$; $iy < ny$; $iy$++) // CMP in $y$ direction
    for ($it = 0$; $it < nt$; $it$++)
{ //imaging point in $t$ direction
    calculate the travel time $t$ with Eq. 1;
    $I[ix][iy][ih][it]$+ = weight * trace [$t/dt$]; }
```

The conventional approach implements parallel processing of Algorithm 1 on a low-cost PC cluster using the message passing interface, where each node calculates all of the imaging points that belong to the same CMPs and they share the same input trace each time. The processing time is longer than the communication time, so the time elapsed is inversely proportional to the number of CPUs, and thus using more CPU nodes can reduce the time elapsed and improve the efficiency (Dai 2005). In this study, we propose a complete GPU solution for PSTM.

3. PARALLELIZATION OF PSTM ON GPU

3.1 Hardware and real seismic field data

We used a 3.07 MHz Intel (R) core (TM) i7 CPU with 24 GB of DDR3 memory, which was connected to an NVIDIA GTX 680 GPU, where the main system comprised 2 GB of global memory, eight multiprocessors (MP) and 192 CUDA cores per MP, the total number of registers available per block was 65 536, the maximum number of threads per MP was 1024, and we used CUDA version 5.0. The field seismic data used for testing comprised a 20 GB subse+t of a real 3D dataset with 1 681 920 traces, where each trace had 3000 samples with intervals of 2 ms. The 3D imaging volume had 3000 imaging points with a 2 ms time interval for a CMP. The CMP number in the *X* direction (the crossline) ranged from 0 to 1000, and the CMP number in the *Y* direction (the inline) was defined as necessary.

3.2 Profiling the PSTM CPU code

Profiling the PSTM CPU code can help to find hotspots and bottlenecks. We used the GUN tool gprof to generate the profile of the seismic field data where the imaging space was equal to 1000 CMPs in the crossline direction, with four inlines. The profiling results (Fig. 3) showed that the PSTM kernel completed nearly 98% of the work with a relatively small amount of code. This value increased when the imaging space was larger. Based on Amdahl's law (Amdahl 1967), the maximum speedup, *S*, of PSTM is:

$$S = \frac{1}{(1-P)+\frac{P}{N}}\ , \tag{2}$$

where *P* is the fraction of the total serial execution time required by the portion of code that can be parallelized and *N* is the number of processors on which the code runs in parallel. When 97% of the running time for PSTM was parallelized, the maximum speedup was > 50, thereby demonstrating that GPU parallelization is a worthwhile procedure.

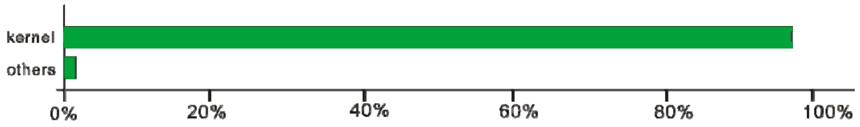

Fig. 3. PSTM kernel runtime percentage for a specific imaging space.

3.3 Parallelization strategy for PSTM on a GPU

According to Algorithm 1, the processing of each imaging point is independent. Thus, we can parallelize the three loops, where the CMPs in the *X* and *Y* directions can be computed in parallel by blocks in the grid. The CMPs in the *X* direction are calculated by blockidx.x and the CMPs in the *Y* direction are calculated by blockidx.y. The imaging point of each CMP is parallelized by threads for each block, where the thread structure of the PSTM is shown in Fig. 4.

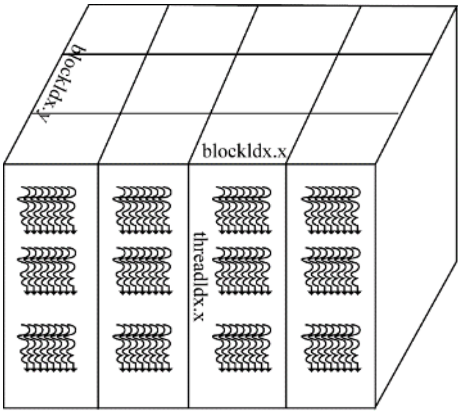

Fig. 4. Thread structure of PSTM.

One approach that improves the GPU performance is keeping the multiprocessors on the device as busy as possible, which has two main features. First, when choosing the number of blocks per grid, the grid size should be larger than the number of multiprocessors so all of the multiprocessors have at least one block to execute. Furthermore, there should be multiple active blocks per multiprocessor so the blocks can keep the hardware busy. This condition can be fulfilled easily, because the number of imaging CMPs in the *X* and *Y* directions is much greater than the number of device multiprocessors. Second, when choosing the block size, it is important to remember that multiple concurrent blocks can reside on the multiprocessor, so the occupancy (the ratio of the number of active warps per multiprocessor relative to the maximum number of possible active warps) is not determined by the block size alone. In particular, a large block size does not imply a higher occupancy. For example, it may lead to pressure on a number of registers per multiprocessors and shared memory usage. In the PSTM kernel, the main pressure is the number of registers, which is 31 in the PSTM kernel. If the block size is 128 threads, then based on the GTX 680's maximum register number per multiprocessor, there are 15 active blocks, which is greater than the eight maximum active blocks calculated using the maximum threads per block of 1024; therefore, we can achieve an occupancy of > 99%. According to the NVIDIA programming guide, many factors are involved in selecting the block size and thus some experimentation is required. Therefore, we provide the block size as a program parameter in the PSTM, which is decided by the user. Furthermore, we can apply –maxrregcount at the compilation time to balance the usage of registers and local memory, which is more expensive to access. Figure 5 shows the results of an experiment using GTX

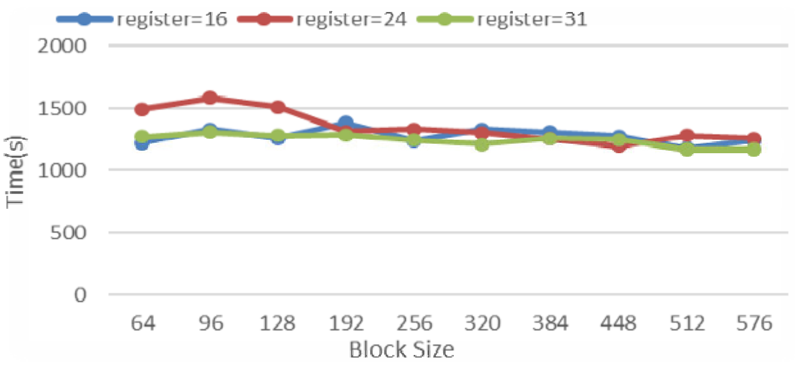

Fig. 5. Comparison of the computing time with different threads and registers.

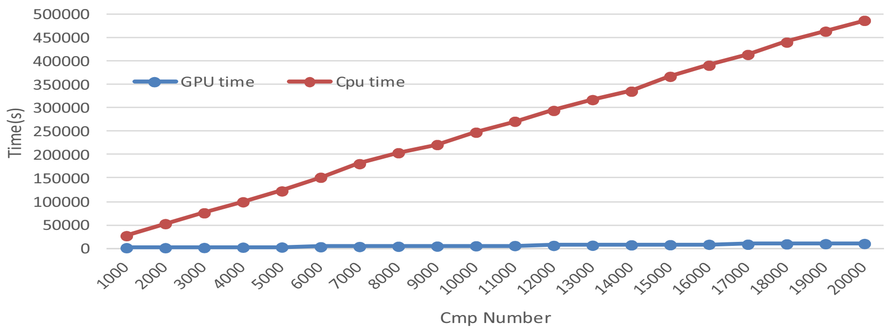

Fig. 6. Comparison of the GPU and CPU computing times with different imaging spaces.

680 with an imaging space of 1000 CMPs for the seismic data mentioned above, where we employed 16, 24, and 31 registers of PSTM elapsed time *versus* different thread numbers per block. This first test was required to obtain the most efficient register and thread pairs.

In PSTM, the root squared mean velocity at the imaging point t_0 plays an important role in the imaging accuracy. Thus, a velocity analysis is performed with sparse CMP points and some other points such as the red dots shown in Fig. 6. Traditionally, during the imaging process, the discrete velocity must be interpolated first for all of the imaging points and then passed to the GPU using global memory, which is expensive to access because it needs a relatively large memory space. In our method, we use texture to perform the velocity model transfer and the read-only texture memory space is cached. Therefore, a texture fetch costs only one device memory read for a cache miss; otherwise, it costs one read from the texture cache. Furthermore, threads in the same warp that read texture addresses located close together will obtain the best performance. Using a function, such as tex3D(), can also provide other capabilities, such as interpolation, and thus based on this characteristic, we can use the texture memory directly for velocity interpolation. This is less expensive to access than global memory and it can save more memory space for the imaging results.

The method used to execute instructions often permits low-level optimizations, which can be useful, especially in code that is run frequently, such as the PSTM kernel. This may involve trading precision for speed when it does not affect the end result, such as using intrinsic instead of regular functions, and including runtime math operations with prepended underscores, *e.g.*, __sinf(),__fmul_rz(). These types of function are mapped directly to the hardware level. They are faster but they have somewhat lower accuracy. When the imaging space of PSTM is 1000, there are 128 threads for a block. We found that if we do not use the intrinsic functions, the number of registers increases to 55 without increasing any variables, which is also a problem that affects the efficiency.

Many possible optimizations can be considered, such as having all read-only input data using the texture memory and employing a constant memory for the transparency function parameters of the device kernel. The final aim is to maximize the use of hardware by maximizing bandwidth and keeping the multiprocessors on the device as busy as possible. Following GPU parallelization, the PSTM kernel can be summarized according to the pseudo-code of Algorithm 2 (Table 2).

Table 2

Algorithm 2: GPU PTKM kernel

```
Input: seismic data trace $[ih][it]$ source at $S(x,y)$ and receiver at $R(x,y)$
    other migration parameters: time interval $dt$, etc.
Output: common reflection gather $I[ix][iy][ih][it]$

Anti-aliasing processing for input trace;
    $ix$ = blockIdx.x // CMP in $x$ direction
    $iy$ = blockIdx.y // CMP in $y$ direction
    for ($it$ = threadidx.x; $it < nt$; $it = it$+blockDim.x){ //imaging point in $t$ direction
        $v$ = tex3D(vel_tex, $it$, blockidx.x, blockidx,y);
        calculate the travel time $t$ with Eq. 1;
        $I[ix][iy][ih][it]$+ = weight * trace [$t/dt$]; }
```

The final GPU PSTM codes were tested with real seismic field data. The number of CMPs in the crossline direction was 1000 and the CMPs in the in-line direction were selected from 1 to 20, where each CMP had 3000 imag-

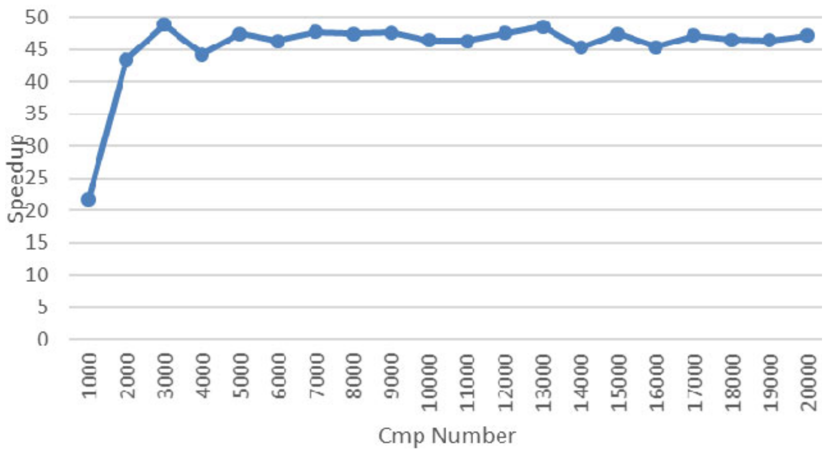

Fig. 7. Speedup in the time elapsed on a GPU compared with one thread on a CPU with different image spaces.

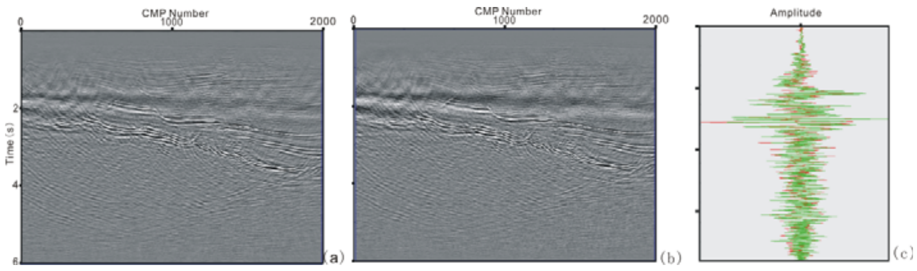

Fig. 8. PSTM section based on real data: (a) migrated section with CPU; (b) migrated section with GPU, and (c) difference in a migrated trace.

ing points. The calculation times obtained with the GPU and CPU are shown in Fig. 6. Figure 7 shows the speedup of the GPU compared with the CPU code where only one thread was used on 3.07 GHz intel i7 processor with different imaging spaces. These two figures demonstrate that the GPU code achieved a speedup of 45 times compared with that of the CPU code, and a larger imaging space reduced the time required significantly.

Figure 8 shows the inline migration results with 1000 CMPs, where panel (a) depicts the CPU migrated result and panel (b) illustrates the GPU migrated result, where it is difficult to see any difference between the two sections. Panel (c) shows the difference in one trace with the same phase and amplitude trend, which may be due to the floating point errors during GPU computation and different velocity interpolation methods, although the results are acceptable in each case.

4. MULTI-GPU SCHEME FOR PSTM

The output of the PSTM is a CRP gather in the form of the matrix *I*[*nx*]*ny*][nh][*nt*], where *nx* and *ny* are the surface CMP numbers in the cross-line and sub-line directions, respectively, and *nt* are the imaging points in the depth direction. The final imaging result includes another dimension, *nh*, which is related to the input trace offset. In the imaging parameters, we usually define the offset bin with a minimum and maximum offset, and an offset interval. Therefore, if the input trace is *h*, then *ih* is (*h*-minimum offset)/(offset interval).

The GPU global memory is limited, where the GTX 680 has 2 GB, so it must be split onto multi-GPUs when the imaging space is larger, as shown in Fig. 9.

For the seismic field data tested in this study, if $nx = 1000$, $nt = 3000$, and the offset number = 48, then the GTX 680 can calculate the sub-line number for $ny = 4$ only once with each GPU. As shown in Figs. 7 and 8, it is better to image a larger space to obtain a higher efficiency. Therefore, we propose an alternative method to maximize the GPU efficiency using multi-GPUs. Before image processing, we split the seismic data onto different GPU nodes according to the offset range. When imaging, assuming that *nh* is equal to the GPU node number, the inline *ny* of the imaging space is more than 150 and different GPU nodes are calculated using different offset parts of the same sub-line. After each node finishes imaging, all of the imaging results are collected from the different GPU nodes and the traces are sorted for the CRP gather. Figure 10 shows a flowchart of our proposed method.

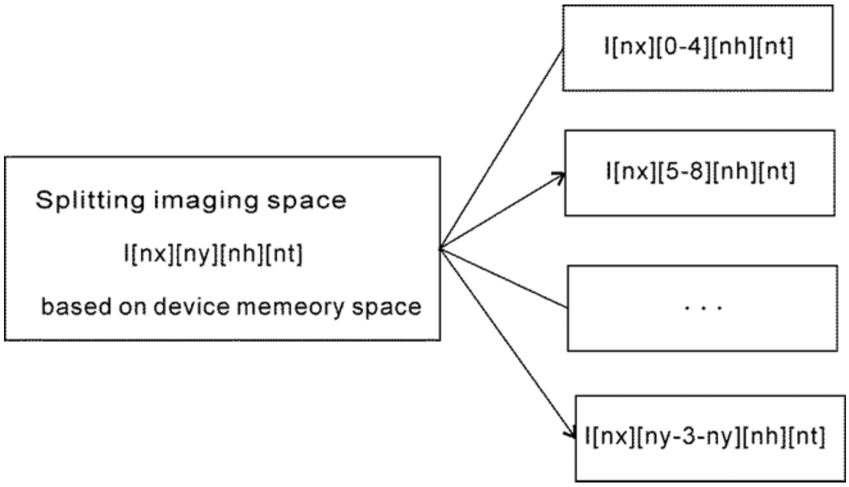

Fig. 9. Splitting the imaging space onto different GPU nodes.

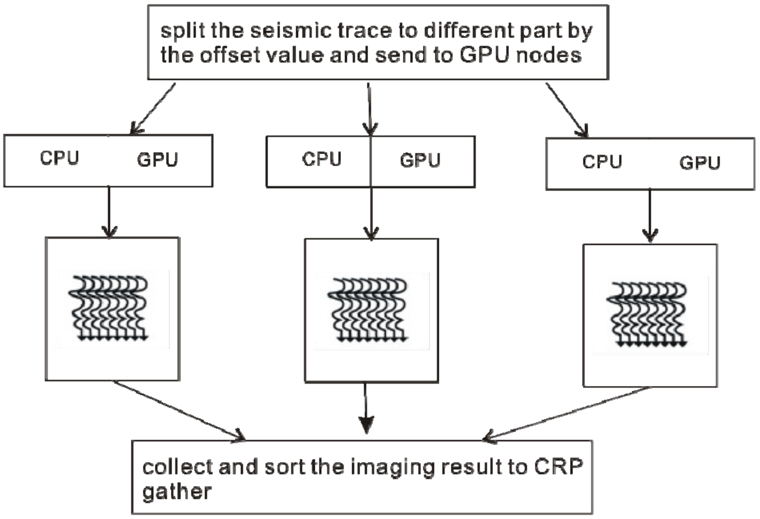
split the seismic trace to different part by the offset value and send to GPU nodes
CPU GPU
CPU GPU
CPU GPU
collect and sort the imaging result to CRP gather

Fig. 10. Flowchart of the proposed multi-GPU scheme.

In large industrial 3D surveys, the offsets of the input data often lack average distributions in a similar manner to joint processing for different surveys, so the following steps are recommend: (i) sort the input trace into the CMP offset order; (ii) split the data into several parts with the same trace numbers according to the number of GPU nodes and send each of them to the GPU nodes; (iii) compute the minimum and maximum offset for the data on each GPU node and set the output offset bins; (iv) migrate all of the imaging space, where several steps may be required depending on the GPU memory and offset number; (v) collect the migrated data from each GPU node and sort them into the CMP offset gather, before stacking the same offset for a CMP gather, thereby completing the migration process and the output is the CRP gather.

5. CONCLUSIONS

In this study, we proposed the practical implementation of PSTM on a GPU. We considered three main optimizations, *i.e.*, designing the configuration for execution to maximize the occupancy and make the device as busy as possible, using texture memory for velocity transparency and interpolation to increase the device bandwidth, and employing faster intrinsic functions for the device kernel. We tested the code with 20 GB of real 3D seismic data on an NVIDIA GTX 680 card, which obtained speedups greater than 45 times compared with the CPU. Furthermore, more time was saved when the imaging space was larger. To fully exploit GPUs, we recommend a scheme that splits the offset onto different GPU nodes, before collecting and sorting to obtain the CRP gather after completing the imaging process. This scheme can be used for data imaging with large volumes. Many possible optimizations can be considered, which requires a large amount of work. However, optimization is easier due to the development of appropriate hardware, and thus the PSTM efficiency can be improved greatly.

Acknowledgments. We would like to thank Dr. Michal Malinowski for his comments and suggestions, which improved this paper significantly, and we also thank the associate editor for his consideration. This study was supported by the NSFC (grant No. 41104083), Beijing Higher Education Young Elite Teacher Project, and Fundamental Research Funds for the Central Universities.

References

Amdahl, G. (1967), Validity of the single processor approach to achieving large-scale computing capabilities. **In:** *Proc. AFIPS '67 (Spring), 18-20 April 1967*, American Federation Information Processing Society, Vol. 30, 483-485, DOI: 10.1145/1465482.1465560.

Bevc, D. (1997), Imaging complex structures with semirecursive Kirchhoff migration, *Geophysics* **62**, 2, 577-588, DOI: 10.1190/1.1444167.

Capuzzo-Dolcetta, R., and M. Spera (2013), A performance comparison of different graphics processing units running direct N-body simulations, *Comput. Phys. Commun.* **184**, 11, 2528-2539, DOI: 10.1016/j.cpc.2013.07.005.

Chen, T., and D. Hale (1993), Network parallel 3-D phase-shift migration. **In:** *Expanded Abstracts of the 63rd SEG Annual Meeting*, Society of Exploration Geophysicists, Tulsa, USA, 177-180, DOI: 10.1190/1.1822430.

Dai, H. (2005), Parallel processing of prestack Kirchhoff time migration on a PC cluster, *Comput. Geosci.* **31**, 7, 891-899, DOI: 10.1016/j.cageo.2005.02.002.

He, C., C. Sun, M. Lu, and W. Zhao (2005), Prestack Kirchhoff time migration on high performance reconfigurable computing platform. **In:** *Expanded Abstracts of the 75th Annual Meeting of the Society of Exploration Geophysicists, 6-11 November, Houston, USA*, 1902-1905, DOI: 10.1190/1.2148076.

Hellman, K.J. (2000), Distributed memory prestack Kirchhoff time migration: Parallelization and scalability. **In:** *Expanded Abstracts of the 70th Annual Meeting of the Society of Exploration Geophysicists, 6-11 August 2000, Calgary, Canada*, 981-983, DOI: 10.1190/1.1816242.

Liu, G.F., H. Liu, B. Li, Q. Liu, and X.L. Tong (2009), Method of prestack time migration of seismic data of mountainous regions and its GPU implementation, *Chin. J. Geophys.* **52**, 6, 1381-1388, DOI: 10.1002/cjg2.1463.

Lu, F., J. Song, W. Lin, Y. Pang, K. Ren, and P. Shi (2013), Efficient utilization of launched threads on GPUs: The spherical harmonic transform as a case study, *Comput. Phys. Commun.* **184**, 11, 2494-2502, DOI: 10.1016/j.cpc.2013.06.019.

Lumley, D.E., J.F. Claerbout, and D. Bevc (1993), Anti-aliased Kirchhoff 3-D migration: a salt intrusion example. **In:** *Expanded Abstracts of the Annual SEG Summer Research Workshop on 3-D Seismology*, Society of Exploration Geophysicists, 115-123.

Morton, S.A., J.R. Davis, H.L. Duffey, G.L. Donathan, and S.N. Checkles (1999), Seismic processing on commodity supercomputers. **In:** *Expanded Abstracts of the 69th SEG Annual Meeting, 31 October – 5 November 1999, Houston, USA*, Society of Exploration Geophysicists, Tulsa, USA, 956-958, DOI: 10.1190/1.1821269.

NVIDIA (2012), CUDA compute unified device architecture programming guide (v 5.0), NVIDIA Co., Santa Clara, USA.

NVIDIA (2013), GPU computing developer homepage, NVIDIA Co., Santa Clara, USA, available from: http://developer.nvidia.com/object/gpucomputing.html.

Schleicher, K., and J. Copeland (1993), Parallel one-pass 3-D migration. **In:** *Expanded Abstracts of the 63rd SEG Annual Meeting*, Society of Exploration Geophysicists, Tulsa, USA, 174-176, DOI: 10.1190/1.1822429.

Shi, X., C. Li, S. Wang, and X. Wang (2011), Computing prestack Kirchhoff time migration on general purpose GPU, *Comput. Geosci.* **37**, 10, 1702-1710, DOI: 10.1016/j.cageo.2010.10.014.

Sun, C., and R.D. Martinez (2002), Amplitude preserving 3D prestack time migration for VTI media, *First Break* **19**, 618-624.

Westphal, E., S.P. Singh, C.-C. Huang, G. Gompper, and R.G. Winkler (2014), Multiparticle collision dynamics: GPU accelerated particle-based mesoscale hydrodynamic simulations, *Comput. Phys. Commun.* **185**, 2, 495-503, DOI: 10.1016/j.cpc.2013.10.004.

Received 1 August 2014
Received in revised form 22 April 2015
Accepted 2 June 2015