Computers & Geosciences 59 (2013) 17–23

Contents lists available at SciVerse ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

3D seismic reverse time migration on GPGPU

Guofeng Liu*, Yaning Liu, Li Ren, Xiaohong Meng

School of Geophysics and Information Technology, China University of Geosciences, Beijing 100083, China

ARTICLE INFO

Article history:
Received 25 October 2012
Received in revised form
21 May 2013
Accepted 22 May 2013
Available online 5 June 2013

Keywords:
Reverse time migration
CUDA
Random boundary condition
Shared memory

ABSTRACT

Reverse time migration (RTM) is a powerful seismic imaging method for the interpretation of steep-dips and subsalt regions; however, implementation of the RTM method is computationally expensive. In this paper, we present a fast and computationally inexpensive implementation of RTM using a NVIDIA general purpose graphic processing unit (GPGPU) powered with Compute Unified Device Architecture (CUDA). To accomplish this, we introduced a random velocity boundary in the source propagation kernel. By creating a random velocity layer at the left, right, and bottom boundaries, the wave fields that encounter the boundary regions are pseudo-randomized. Reflections off the random layers have minimal coherent correlation in the reverse direction. This process eliminates the need to write the wave fields to a disk, which is important when using a GPU because of the limited bandwidth of the PCI-E that is connected to the CPU and GPU. There are four GPU kernels in the code: shot, receiver, modeling, and imaging. The shot and receiver insertion kernels are simple and are computed using a GPU because the wave fields reside in GPU's memory. The modeling kernel is computed using Micikevicius's tiling method, which uses shared memory to improve bandwidth usage in 2D and 3D finite difference problems. In the imaging kernel, we also use this tiling method. A Tesla C2050 GPU with 4 GB memory and 480 stream processing units was used to test the code. The shot and receiver modeling kernel occupancy achieved 85%, and the imaging kernel occupancy was 100%. This means that the code achieved a good level of optimization. A salt model test verified the correct and effective implementation of the GPU RTM code.

1. Introduction

During the past few decades, three types of seismic prestack depth migration methods have been developed. The first method is Kirchhoff-based and uses ray tracing to approximate Green's function of wave propagation. This method can provide successful image results when the subsurface structure variations are mild (Thierry et al., 1999; Operto et al., 2000; Gray, 2005). However, if the subsurface structure contains extremely strong variations, such as salt bodies, then this method results in an inaccurate image of the subsurface. The second method is called one-way wave equation migration (Claerbout and Doherty, 1972; Gazdag, 1978) and uses full-waveform Green's functions for wave propagation. In theory, this method can obtain better images than Kirchhoff-based methods. The one-way wave equation migration algorithms decompose the seismic wave fields into upgoing and downgoing waves; however, these waves cannot generate reflections between layers. Additionally, turning waves are missing in these types of wave propagation simulations, which results in the poor imaging of steep-dip events around 90° (Zhang, 1993). The third method is called reverse time migration (RTM) and uses the two-way wave equation for wave propagation calculations. This method has the theoretical advantage of unlimited dip and better amplitude behavior. Currently, RTM is considered the best seismic prestack depth migration method (Huang et al., 2009).

RTM is not a new seismic prestack depth migration method. It was first introduced in the late 1970s (Hemon, 1978) and has been shown to have promising imaging capabilities (Baysal et al., 1983; Whitmore, 1983; McMechan, 1983; Loewenthal and Mufti, 1983; Bednar and Bednar, 2006). However, the intensive computational requirements of this method have limited its practical application, especially when a high-order scheme in both space and time is used (Chen, 2009) or an output angle gather is needed (Xu et al., 2011; Tang et al., 2013).

With recent advancements in computing capacity, a three-dimensional (3D) prestack RTM is now feasible (Yoon et al., 2003, 2004; Farmer et al., 2006; Nemeth et al., 2008; Micikevicius, 2008).

Driven by the insatiable market demand for real-time, high-definition 3D images, programmable NVIDIA graphic processing units (GPUs) have been developed for high-performance computing as co-processors for central processing units (CPUs). NVIDIA's GPUs contain a novel architecture known as Compute Unified Device Architecture (CUDA), which acts as a program module and compiler technology for general purpose GPU programming.

* Corresponding author. Tel./fax: +86 010 82321331.
E-mail addresses: liugf@cugb.edu.cn, liugfs@163.com (G. Liu).

0098-3004/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cageo.2013.05.009

The CUDA C language is an extension to the C programming language that makes it easier to allow hardware access to the massive parallel capabilities of modern GPUs without requiring the programmer to construct logical operations as graphical instructions. CUDA C programming involves running code simultaneously on two different platforms: a "host" system with one or more CPUs and a "device" system with one or more CUDA-enabled NVIDIA GPUs. The device code is always called a kernel, and is compiled and linked with a NVIDIA driver to run on a GPU (NVIDIA, 2011).

GPGPU has been used successfully in Kirchhoff-based imaging (Shi et al., 2011) and one-way wave equation imaging (Liu et al., 2012), both of which yield a great improvement in efficiency from GPU usage. For reverse time migration, choosing the most suitable hardware to deal with the intensive computation is also an issue (Clapp et al., 2010). In this paper, we present a solution for using RTM with GPGPU.

2. Algorithm overview

We begin with the acoustic wave equation,

$$\frac{\partial^2 u}{\partial t^2} = v^2\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right) \tag{1}$$

where u is the pressure and v is the velocity of the subsurface material. The central component of the RTM algorithm that solves Eq. (1) is a finite difference modeling kernel, that uses second-order and higher-order finite difference algorithms to approximate the time and space derivatives. The following pseudo-code shows how the algorithm propagates forward for nt steps with an interval of dt. The mesh has nx, ny, nz cells in the x, y, and z directions, indexed by ix, iy, and iz, with intervals of dx, dy, and dz. The algorithm uses a second-order approximation of the time and space derivatives.

2-order Modeling kernel

```
Input: velocity model v
       wavefields(it), s(it-1)
       dt
Output: wavefields(it+1)

for (it=3;it<nt;it++){
    for(ix=2;ix<nx;ix++){
      for(iy=2;iy<ny;iy++){
        for(iz=2;iz<nz;iz++){
          s(iz,iy,ix,it)=2*s(iz,iy,ix,it-1)-s(iz,iy,ix,it-2)+
          v(iz,iy,ix)*v(iz,iy,ix)*dt*dt*s(iz,iy,ix,it)*8-
          (s(iz-1,iy,ix,it)+s(iz+1,iy,ix,it))/dz/dz-
          (s(iz,iy-1,ix,it)+s(iz,iy+1,ix,it))/dy/dy-
          (s(iz,iy,ix-1,it)+s(iz,iy,ix+1,it))/dx/dx;
      } }
} }
```

In the RTM process, the kernel models two wave fields: the forward source wave field and the receiver wave field. The forward source wave field $s(z,y,x,t)$ is modeled using a wavelet as the source. The source wave propagates from $t=0$ to $t=t_{\max}$. The reflection wave field is the backward time propagation of the receiver wave field $g(z,y,x,t)$, taken during the same shot from $t=t_{\max}$ to $t=0$. The reflections where the energy propagates from the source and the receiver are located at the same position at the same time. The final image is the summation of correlations between the source and receiver wave fields at every time step and for every shot record in the seismic data.

$$\text{image}(z,y,x)=\sum_{\text{shots}}\sum_{t=0}^{t=t_{\max}} s(z,y,x,t)^{*}g(z,y,x,t) \tag{2}$$

The following pseudo-code displays how RTM is implemented by the modeling kernel.

Forward propagation	Backward propagation and imaging
Input: velocity model Shot location Output: forward wavefields	Input: velocity model Receivers' traces and locations Source wavefields Output: image(i)

for(it=0;t<itmax;it++)
{
get s(it+1)with algorithm (1) iz,iy,ix loop;
add source wavelet
store s(it+1) to the disk
}
	for(it=itmax;it>0;it–) {
	get g(it-1)with algorithm (1) iz, iy,ix loop;
	add receivers' wavefields imagine
	read s(it-1) from disk
	for(i=0;i<nz*ny*nx;i++){
	image(I)=image(I)+s(I,it-1)*g(I, it-1);
	}

Two factors slow the RTM calculations. The first is the intensive computation necessary in the modeling module. In order to obtain a stable and non-dispersive solution, we must use small time steps and grid intervals, and higher-order approximations of the space derivatives. These requirements make the modeling module the most computationally demanding task of the RTM. The second bottleneck is the memory transformation from the CPU to the disk. It is impractical to store the four-dimensional volumes $s(z,y,x,t)$ and $g(z,y,x,t)$ in the CPU memory because these volumes are often several terabytes in size. To solve this problem, the source wave fields must be stored to the disk. These issues can be fixed or bypassed using the GPU method that will be introduced in this paper. Because the GPU only has a memory size of four or less gigabytes, we must first determine how to deal with the large size of the source wave field data.

3. The CUDA implementation of reverse time migration

3.1. Random boundary conditions for source wave field propagation

In the RTM process introduced above, because the wave fields must be correlated at equivalent time positions, the source wave fields must propagate from $t=0$ to $t=t_{\max}$ while the receiver wave fields must propagate from $t=t_{\max}$ to $t=0$. One propagation must be stored to the disk and read back to memory from the disk during the imaging step.

Symes (2007) and Dussaud et al. (2008) discussed checkpoint methods for handling the different propagation directions. Dussaud et al. (2008) and Clapp (2009) suggested an alternate approach for propagating the source wave fields to the maximum recording time and reversing the propagation to maintain consistency with the receiver propagation direction. The use of damping schemes around the boundary makes it necessary to inject energy as undamped, forward propagating wave fields into