GPU-based explicit time evolution method

Jiarui Fang, Haohuan Fu and Guangwen Yang, Tsinghua University*
Wei Wu and Nanxun Dai, CNPC

Summary

Finite-difference (FD) methods have long been among the most popular solutions for RTM and FWI. Compared with FD methods, the Explicit Time Evolution (ETE) method is able to simulate the wave propagation in acoustic media with higher spatial and temporal accuracy, at the cost of a more complicated memory access pattern. Similar to FD, ETE performs a stencil operation on every grid point, except that the coefficients of the stencil change spatially with the velocity parameter of that position. While FD methods already have highly-efficient designs on GPU platforms, in ETE, the sharp velocity discontinuities can result in un-coalesced memory access patterns. Moreover, the increased number of involved off-axis points in the stencil and the increased number of different coefficients bring more pressure for the fast buffers and memory in the GPU. To solve these issues, in this paper, we decompose the complex stencil into a number of sub-components, so as to form a better memory access pattern for coefficients and to simplify the calculation of stencil operations. Finally, we combine the decomposition scheme with 2.5D spatial and 1D temporal blocking optimizations. With one K20 GPU card, we manage to achieve 5.5x speedup compared against 12 cores Intel E5- E5645 CPU.

Introduction

The finite-difference (FD) method is one of the most popular waveform modeling methods used in RTM and FWI. The spatial derivatives of FD are often approximated in higher order whereas the temporal derivatives are usually approximated in 2nd order. As a consequence, the FD method often suffers from low temporal accuracy. Compared to the FD method, FFT-based methods can achieve higher accuracy. Nevertheless, the computational cost of multi-dimensional Fourier transforms in each time step is more expensive than the FD method. Another drawback for such methods is that additional treatments are needed to suppress the wrap-around effect caused by the periodic source assumption. The Explicit Time Evolution (ETE) method (Liu et al., 2014) can achieve high spatial and temporal accuracy in acoustic media without using Fourier transforms. By adopting a FD-style operator in the discrete spatial-time domain to explicitly extrapolate wavefields in time, the ETE method achieves spectral-like accuracy by optimizing the spatial operator to fit an analytical counterpart in the wavenumber domain. The disadvantage of the ETE method is that coefficients of stencil points vary over the velocities of center points. In implementation, a coefficient matrix and an array of velocity parameters are required to be maintained in memory.

In recent years, GPU has become a popular parallel platform to accelerate FD methods for seismic computing. However, an efficient solution for ETE method has not been proposed on GPU. It is mainly because the sharp velocity discontinuities can result in un-coalesced memory access patterns. As a result, when calculating the stencil for a point, neighbor GPU threads need to perform un-coalesced accesses to the global memory to load the ETE coefficients, which bring significant performance damages. Similar challenges also exist in a variety of FD-like methods, such as (Liu, 2013; Wang et al., 2014; Tan et al., 2014).

To solve the above issues, in this paper, we proposed a highly-efficient method to implement the 8th-order 3D ETE stencil on GPU. We avoid the un-coalesced accesses to coefficients by decomposing the original stencil into 3 small stencils. Thus, most of coefficients are able to be stored in on-chip fast buffer on GPU. As a result, we conduct 3 GPU kernels to sweep through the spatial grid. A 2.5D spatial blocking and 1D temporal blocking strategies are also applied to further reduce the memory access time.

When processing ETE-based RTM or FWI jobs, our GPU-based scheme can improve the processing capability of each single node by 5.5 times when compared against 12 CPU cores. Using our proposed GPU-based ETE design, the practical RTM processing time for one shot on a single node can be reduced from 1.38 hours one shot to 15 minutes, making ETE a favorable design in many cases.

The ETE method

The acoustic wave equation in the time-wavenumber domain assuming a constant velocity is presented as follows:

$$\frac{d^2\hat{p}(\mathbf{k},t)}{dt^2} = -v^2|\mathbf{k}|^2\,\hat{p}(\mathbf{k},t)\,. \quad (1)$$

The analytical solution of equation (1) is as follows:

$$\hat{p}(\mathbf{k},t+\Delta t) = 2\cos\left(v\Delta t|\boldsymbol{k}|\right)\hat{p}(\mathbf{k},t) - \hat{p}(\mathbf{k},t-\Delta t) \quad (2)$$

For variable velocities, we consider velocity v as a "local constant" and convert equation (2) to the time-space domain:

$$p(\mathbf{x},t+\Delta t) = 2c(\mathbf{x}\,|\,v\Delta t) * p(\mathbf{x},t) - p(\mathbf{x},t-\Delta t) \quad (3)$$

Where $c(\mathbf{x}\,|\,v\Delta t)$ is a spatially variable 2D or 3D filter corresponding to $\cos(v\Delta t\,|\,\boldsymbol{k}\,|)$ in the wavenumber domain and * denotes a spatial convolution.

An explicit time evolution (ETE) operator for wavefield simulation based on equation (3) is formulated in a discrete space. For a selected stencil in a neighborhood of any given point in the space domain, we seek coefficients that minimize the difference between their discrete Fourier Transform and $\cos(v\Delta t\,|\,\boldsymbol{k}\,|)$ in the wavenumber domain. The accuracy of the time evolution of the wavefield will be solely determined by the fitting of the cosine function at each wavenumber. We seek c_j, the coefficients of each stencil point, in least square sense:

$$\operatorname*{MIN}_{c_j}\left|\sum_j c_j e^{i\Delta\mathbf{x}_j\Delta\mathbf{k}} - \cos\left(v(\mathbf{x})\Delta t\left|\boldsymbol{k}\right|\right)\right|^2 \tag{4}$$

where $\Delta\mathbf{x}_j = \mathbf{x}_j - \mathbf{x}$ is the distance vector from updating point to a stencil point. Once the coefficients c_j are determined, the ETE operator application to wavefield simulation is similar to FD schemes, as described in the following formula:

$$p(\mathbf{x}, t+\Delta t) = 2\sum_{j=1}^{N_e} c_j(\mathbf{x})\, p(\mathbf{x}+\Delta x_j, t) - p(\mathbf{x}, t-\Delta t). \tag{5}$$

As for the shape of ETE stencil, it has been proven that adding off-axial points can significantly reduce the misfit. A stencil including off-axial stencil points with minimal distances to the center appears to be optimal to reduce the misfit of cosine function. However these points increase the complexity of stencil, which bring challenge to implement ETE on GPU. According to ETE design, an 8th-order 3D ETE Stencil is optimum in real application, as is shown in Figure 1. The stencil operation involves 37 nearest neighboring points on a spatial grid. The ETE stencil sweeps through the entire 3D grid multiple times to update each grid point with calculations involving its nearest neighbors. The updating rule for one time sweep is indicated in equation (5).

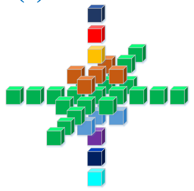

Figure 1. The shape of 8th-order

ETE stencil can sufficiently reduce the fitting error to a magnitude of 10^{-6}. In an 8th order ETE scheme, ETE operator contains 37 points while FD operator contains 25 in 3D. Numerical experiments indicate, however, that in order to achieve similar accuracy the FD scheme requires time steps about five times as fine as does ETE. Therefore, ETE operator is superior to the traditional FD scheme in terms of accuracy and overall performance (Dai et al., 2014).

GPU optimizations for ETE

It is noticeable that there is significant data reuse of points in current grid $p(t)$ across their spatial neighbors. With spatial blocking, a chunk of data in 3D gird can be loaded into the fast buffers of GPU architectures to be reused for computations. However, since the fast buffers available per multi-processor are not sufficient to store the entire 3D subdomain of the problem, as is shown in Figure 2, a 2.5D blocking scheme (Nguyen, 2010) is taken to increase the data locality. 2.5D spatial blocking blocks in two dimensions and streams through the third dimension. When implemented on GPU, points along the third dimension are assigned to one thread to perform stencil operations. Registers and shared memory, from which the latency of data fetching is two orders of magnitude lower than that of global memory, serve as fast buffers on GPU. These fast buffers can only be shared with threads in same block. Only a few layers of planes in current grid are required to be cached in fast buffer for calculating one result plane of next gird. We stream through the third dimension to perform stencil operations plane by plane.

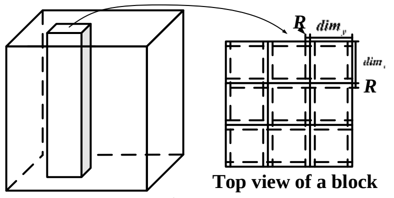

Figure 2. A 2.5D blocking of 3D grid streaming through axis *z*. $dim_x \times dim_y$ is size of *xy*-plane in one block. *R* is size of boundary.

Different from FD stencil, the coefficients of each stencil point are not a constant value. The coefficients for different velocities are calculated before spatial sweeps of ETE stencil using the least-square method. A 2D coefficient matrix and a 3D coefficient index array have to be stored in the GPU global memory before the sweeps. For the coefficient matrix (c in equation (5)), the number of rows is N_e (the number of spatial points involved in the stencil), and the number of columns is the number of different discretized velocities that we use to calculate the ETE coefficients in the model. The size of the index array is the same as the grid size, which is $nx \times ny \times nz$. When conducting the stencil operation, a thread first accesses the coefficient index array according to its coordinate in the 3D grid. The index determines which column to get the coefficients for the N_e =37 neighboring points in the 8th order 3D ETE stencil.

The real geological structures usually come with sharp velocity discontinuities. Such velocity distribution leads to discontinuous accesses to the coefficient matrix, which results in un-coalesced memory accesses of neighboring threads on GPU. GPU devices provide a very high off-chip memory bandwidth (up to 208 GB/sec), but this bandwidth is only achievable with coalesced access. In another word, data from the global memory is transferred to the GPU device in contiguous blocks, and high bandwidth can only be achieved when requests by concurrent 32 adjacent threads fall within such contiguous blocks. When non-continuous memory locations are accessed by threads, the achieved bandwidth is much lower than the peak, leading to stalling and wasted compute cycles.

To resolve the above issue, we propose to decompose the ETE stencil into a number of sub-components, each of which can perform independent scaling and sum calculation to update the grid points with fewer rows of matrix coefficients. By using such a decomposition scheme, we can then fit the required coefficients into the shared memory, and improve the memory access efficiency.

As for the 8th order ETE stencil, a decomposition scheme is carefully designed. As shown in Figure. 3, sub-component (a) is a 2D stencil on the *xz*-plane. We can sweep the spatial grid along the *y* axis to perform a 2D stencil operation. Sub-components (b) and (c) are with same shape and are symmetric to the center point in original stencil. When sweeping along *z* axis, only 2 points on upper and lower *xy*-plane are required for stencil (b) and (c). The corresponding rows of coefficient matrix can fit into the shared memory. In stencil (a), the size of all involved coefficients may be slightly beyond the shared memory capacity. We store most of the coefficients in the shared memory, and store the rest of them in the read-only cache. An implementation of numerically equivalent ETE-method is illustrated in Table 1.

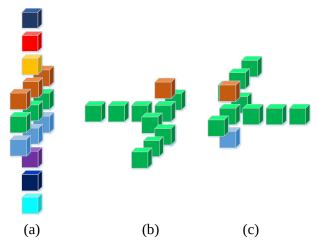

Figure 3. Decomposition of 8th order-ETE stencil. The points in same layer are with the same color.

Table 1. A stencil decomposition scheme for ETE method

Kernel 1

Load $c(\mathbf{x})$ of stencil (a) into shared memory and read-only cache

$$p(\mathbf{x},t+\Delta t)=2\sum_{j\in stencil\,(a)} c_j(\mathbf{x})\,p(\mathbf{x}+\Delta\mathbf{x}_j,t)-p(\mathbf{x},t-\Delta t)$$

Kernel 2

load $c(\mathbf{x})$ of stencil (b) into shared memory

$$p(\mathbf{x},t+\Delta t)+=2\sum_{j\in stencil\,(b)} c_j(\mathbf{x})\,p(\mathbf{x}+\Delta\mathbf{x}_j,t)$$

Kernel 3

load $c(\mathbf{x})$ of stencil (c) into shared memory

$$p(\mathbf{x},t+\Delta t)+=2\sum_{j\in stencil\,(c)} c_j(\mathbf{x})\,p(\mathbf{x}+\Delta\mathbf{x}_j,t)$$

3.5D spatial/temporal blocking implementation on GPU

Since sweeps of 3D grid will be executed for multiple time steps, we can further combine the 2.5D spatial blocking scheme with an additional 1D temporal blocking scheme by executing 2 time steps of blocked data so that intermediate data can reside in shared memory and registers. A 3.5D scheme is applied to stencil (b) and stencil (c). Details of the 3.5D implementation are illustrated in Figure 4. We rearrange the temporal calculation order of *xy*-plane so that the 3.5D ETE-stencil can work in pipeline. Each calculation step, a *xy*-plane is loaded into fast buffer from the grid of time step *t* and a *xy*-plane is written to the grid of time step $t+2\Delta t$. The grid points of time step $t+\Delta t$ are stored in registers and shared memory on GPU. The *xy*-planes of 3D grid are written to global memory every 2 time steps.

This scheme saves all global memory transfers to and from gird $p(t+\Delta t)$, leaving one load and one store (eviction) for two points updates. In addition, the coefficients of the 3.5D scheme can be reused by 2 time steps in ETE. However, an extra piece of shared memory is required in 3.5D scheme, which reduces the available shared memory for coefficient matrix.

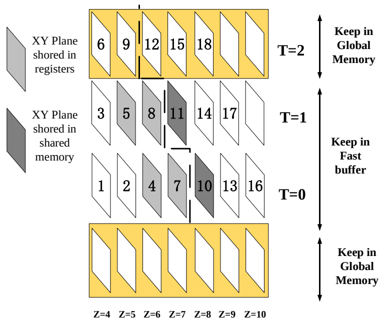

Figure 4. A snapshot of the 3.5D blocking scheme. At the current calculation step, *xy*-plane with z = 6 at time step T = 2 is calculated. (Numbers labeled in *xy*-planes illustrate the order of execution. The Z at bottom indicates the z coordinate of *xy*-plane. The T on the right demonstrates the time step of propagation.)

Experiment results

We implemented ETE method on multi-core CPUs and GPU in a 3D domain. Our GPU platform is a Tesla K20 card with driver version 6.5. Our CPU platform is a dual-socket 12-core Intel Xeon E5-E5645 CPU. For our 3D problem model, the dimension size on the *z* axis (*nz*) is set to be 912. The dimension sizes on the *x* and *y* axes are kept as 320. Our velocity model is a complete random distribution of 902 different velocity values. The performance of different optimization schemes is illustrated in Figure 5. 12-core CPUs refers to a CPU implementation with 12 cores sharing a common 48GB memory. We carefully block the 3D domain for cache locality and apply SIMD for achieving the best vectorized performance. The performance on 12-core CPUs is 12.97GFlops. K20-naïve refers to our straightforward implementation on GPU, with coefficients read directly from the global memory. K20 (decomposition) refers to our improved design that decomposes the stencil into 3 ones, most of whose coefficients are loaded in shared memory initially. K20 (decomposition +3.5D) refers to our final design that uses the 3.5D spatial and temporal blocking scheme, and coefficients are accessed from shared memory. We can obtain 72.07 GFlops in this scheme.

Table 2. Global memory access for one stencil operation in 3.5D scheme

Global memory access type	*Load coefficients (float) (coefficient matrix size/block size/2)**	*Load grid points (float)*	*Write grid points (float)*	*Load velocity parameter (integer)*
Data required	16.29*4B	3*4B	3/2*4B	2*2B

As for the K20 (3.5D) scheme, a stencil operation consists of 75 floating-point operations. And data access times from global memory for a stencil are illustrated in Table 2. The theoretical ratio of floating point rate (flop/second) versus the memory bandwidth (bytes/second) for our 3.5D implementation is 2.29. Because the peak bandwidth of K20 is 208 GBps for all memory access coalesced, the theoretical peak computing performance is 90.82 GFlops. Our 3.5D scheme achieves 72.07Gflops, which is 79.35% of theoretical peak performance on GPU. In other words, our optimizations significantly eliminate the performance damage from memory un-coalesced access.

In addition, we compare the performance of 8th-order ETE and 8th-order FD stencil. The time interval of 8th-order FD is 5 times smaller than that of ETE so at to achieve similar accuracy (Dai et al., 2014). We simulate the wave propagation process with $\Delta t = 0.2$ ms for FD and with $\Delta t = 1$ ms for ETE. The time performance for 1s simulation is illustrated in Table 3. Our ETE-based propagation achieves 15% performance improvement. A section of the GPU-based ETE RTM image of a 3D land seismic data is shown in Figure 7. The velocity model with sharp discontinuity is presented on left side.

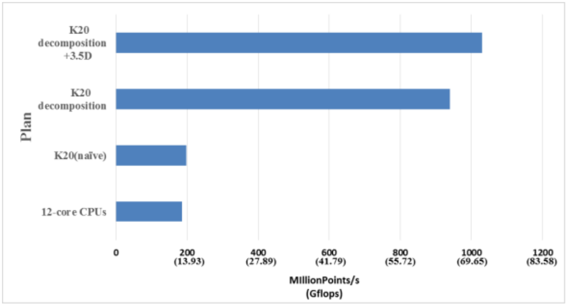

Figure 5. Performance of different optimization schemes

Table 3. Time performance for ETE and FD in 1second

	8-order FD	*8-order ETE*
Times(s)	105.35	90.6756

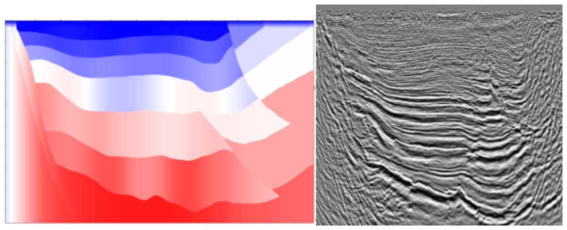

Figure 6. A section of 3D RTM image using GPU-based ETE

Conclusions

In this paper, we present our work on a parallel GPU-based ETE solution. When performing ETE stencil, coefficients of each point in stencil have be loaded from global memory. The sharp discontinuity of the coefficient matrix access pattern brings significant challenge for GPU implement-tation. We replace original 8th-order 3D complex ETE stencil by 3 more simple ones with better properties. Most part of the coefficient matrix can be stored in shared memory in these 3 simple stencils. A 3.5D scheme, which combines the spatial and temporal blocking, is applied to GPU-based ETE. We achieve approximate 5.5x speedup on one Tesla K20 card, compared with a 12-core CPU node. Our GPU-based ETE is able to effectively exploit the theoretical floating point efficiency of hardware and enable ETE method to become a strong competitor to FD method on GPU platform. Our design also brings inspiration to a board of FD-like method with variable coefficients.

Acknowledgments

The authors would like to thank BGP and CNPC for the supporting the project, for providing devices to perform the experiments and for permission to present this paper.