GPU implementation of minimal dispersion recursive operators for reverse time migration

Allon Bartana, Dan Kosloff, Brandon Warnell, Chris Connor, Jeff Codd and David Kessler, SeismicCity Inc.*
Paulius Micikevicius, Ty Mckercher, Peng Wang and Paul Holzhauer, Nvidia Corporation

Summary

The implementation of new recursive operators for computation of numerical derivatives results in minimal dispersion in Reverse Time Migration Prestack Depth Migration (i.e. RTM PSDM). Compared to the more commonly used finite difference operators, the presented new method enables imaging of higher frequencies in RTM PSDM. Since RTM PSDM is the modern method of choice for imaging many exploration targets, computer optimization is an integral part of code development. We present here details of the GPU implementation for newly developed spatial derivative operators which enable routine use in industrial settings.

Introduction

RTM is now the algorithm of choice for imaging of seismic data used in exploration of potential hydrocarbons reservoirs. Commonly, the high computational cost of commercial RTM leads to use of Finite difference (i.e. FD) operators for computation of spatial derivatives. However, application of FD operators may result in numerical dispersion that appears as coherent noise on depth migrated seismic data. In geological settings that are difficult to illuminate and image such as those close to and beneath salt bodies, coherent noise can be interpreted as primaries and can cause incorrect analysis of the seismic data. Since so many exploration targets rely on RTM PSDM accuracy, it is very important to develop new generation of RTM PSDM algorithms that will produce minimal dispersion PSDM seismic data and therefore improve confidence in interpreted results.

Numerical approximation of spatial derivatives for RTM

Here we present a new spatial recursive operator that is used for application of spatial derivatives in RTM algorithms in an effort to produce more accurate and minimal dispersive RTM PSDM data. The application of these operators requires the solution of tri-diagonal linear equation systems which can be carried out efficiently. The derivative operators are designed by a Remez exchange procedure (Kosloff et. al. 2008). The importance of using minimal dispersive numerical operators is that it enables us to use higher frequencies in commercial RTM PSDM while maintaining reasonable computational cost. This is compared to FD based RTM PSDM that requires increased computational resources for migrating higher frequencies.

Recursive derivative operators

We illustrate here the recursive derivative operator for approximating the second spatial derivative. The approximation of other higher order spatial derivatives proceeds in an analogous manner.

Given a function $f(x)$, we denote its sampled values by

$$f[j] = f(x = jdx)$$

Derivative approximation can be written as

$$\frac{\partial^2 f}{\partial x^2}[j] = \frac{a_0 + a_1\Delta_1 + a_2\Delta_2 + \cdots + a_N\Delta_N}{1 + b_1\Delta_1 + b_2\Delta_2 + \cdots + b_M\Delta_M} f[j] \quad (1)$$

Where $\Delta_k f[j] = f[j+k] + f[j-k]$.
We will consider operators for which $M \le N$.
In this case it can be recast in an equivalent form

$$\frac{\partial^2 f}{\partial x^2}[j] = \left(c_0 + \cdots + c_{N-M}\Delta_{N-M} + \frac{d_0}{1+\beta_0\Delta_1} + \cdots + \frac{d_{M-1}}{1+\beta_{M-1}\Delta_1}\right) f[j] \quad (2)$$

Each of the rational terms in (2) comprises a tri-diagonal equation system. In the present implementation, only one rational term is used.

GPU implementation

In general, computer implementation of RTM algorithms involves code optimization to take advantage of the provided hardware resources. Since most of the compute time in RTM code is in the spatial derivatives, the main focus in the computer implementation is in the spatial derivatives operators. In a typical application of commercial RTM, migrating a single shot gather aimed at imaging a potential sub-salt target in a deep water environment, numerical spatial derivatives are calculated approximately 30,000 times. This operation is repeated for surveys that contain tens of thousands of recorded shot gathers. Therefore, the hardware implementation of these numerical operators plays a crucial factor in any commercial implementation of RTM PSDM. Due to the vast amount of computational resources needed, hardware accelerators such as GPUs provide a very practical solution and are successfully used as the compute engine of RTM PSDM algorithms.

The numerical approximation of the spatial derivatives of the seismic wavefield is characterized by a small number of arithmetic operations applied to a large number of data points. This operation is applied at every point in the seismic cube, represented on a Cartesian grid. A typical size of the grid is several Gigabytes (i.e. GB). The main challenge is to efficiently supply the processing unit with access to data. The rate of data transfer between the different types of memory is the most critical part of the implementation.
To understand the GPU implementation with recursive operators, and the challenges it presents, we first review the main elements of the GPU implementation using FD approximation of the derivatives (Micikevičius, 2009 and 2011).

The 1D FD approximation of the second spatial derivative reads:

$$\frac{\partial^2 f}{\partial x^2}[k] \cong \sum_{j=0}^{N/2} a_j(f[k+j]+f[k-j]) \quad (3)$$

Where $f[k]$ is the function value at grid point $x = kdx$. N is the approximation order with the FD coefficients a_j.

In a typical efficient 3D implementation, the operations in Equation 3 are carried out in the 3 spatial directions. For evaluating the spatial derivative at a point (i_x, i_y, i_z) in the calculated volume, we need to use the data values at $N/2$ points in each direction around the target point.

To optimize the data access, it is arranged in depth slices. In each depth slice the X direction is the axis in which the data is contiguous. The Y direction is the second axis and the depth (Z) direction is the slowest axis. A GPU thread block of size $DIMX \cdot DIMY$ computes the spatial derivatives of $DIMX \cdot DIMY \cdot N_z$ grid points, where N_z is the number of points in the volume along the Z direction (see Figures 1 and 2). Each thread is computing N_z points by looping on N_z. At each depth level the derivatives of $DIMX \cdot DIMY$ points are computed by the thread block.

GPU block shared memory is used for $DIMX \cdot DIMY$ slices for optimizing the computation of the derivatives in the X and Y directions. Local memory is used for computing the vertical derivative.

The main computational effort in applying the recursive operators is in a term of the form $\frac{d_0}{1+\beta_0\Delta_1} f[j]$ in Equation 2. This is solved as a tri-diagonal system of equations.

The solution involves two recursive loops:

```
aux[0] = f[0];
for( $i_x$=1; $i_x$<$N_x$; ++$i_x$ ) {
    aux[$i_x$] = f[$i_x$] - G[$i_x$] * aux[$i_x$-1];
}

d2f[$N_x$-1] = aux[$N_x$-1] * H[$N_x$-1];
for($i_x$=$N_x$-2; $i_x$>=0; --$i_x$ ) {
    d2f[$i_x$] = (aux[$i_x$] - d2f[$i_x$+1]) * H[$i_x$];
}
```

$f[i_x]$ is the data value at a grid point i_x. $H[i_x]$ and $G[i_x]$ are coefficients designed for an accurate approximation of the second derivative of $f[i_x]$. The function $d2f[i_x]$ becomes a part of the resulting second derivative approximation at point i_x after the second loop.

In both forward and the backward loops the result at each point depends on the result at the previous point, which makes the loops recursive.

In 3D the same type of operation is applied in each coordinate direction. The forward and backward loops are applied in each direction independently. In the GPU implementation we cannot share data between threads in the same way as the FD implementation. Using FD, the loop over the slow axis (Z) is very efficient in terms of accessing data between global memory and local and shared memories.

In the recursive operators we can use the same idea as in FD to fetch the data from global to local memory when applied to the slow (Z) axis and to the second (Y) axis. However, special consideration is taken to access the data in global memory for computing the derivative along the first (X) axis.

In order to improve the computational performance, the computation is split into two parts. First the forward loop is computed. The intermediate result *aux* is stored. Second, the backward loop is performed.

The main components for the first stage are:
1. Each thread block works on one depth slice of size $N_x \cdot DIMY$ where N_x is the total number of samples in the X direction. The computation along the X axis is performed in chunks of size SMX, where SMX is the number of samples in the X direction that can fit to shared memory (see Figure 3).
2. A shared memory array of dimensions $DIMY \cdot SMX$ is allocated.

DOI http://dx.doi.org/10.1190/segam2015-5754164.1

3. Data of size $DIMY \cdot SMX$ is read from global memory and stored in shared memory. The read is performed by looping along the Y direction and reading the data that is contiguous along the X axis.
4. SMX samples of the recursive forward loop are computed using the data from shared memory.
5. The temporary result *aux* is stored in global memory as an auxiliary array in a transposed order in which Y is the fast axis and X is the slower.
6. Steps 3 to 5 are repeated N_x/SMX times.

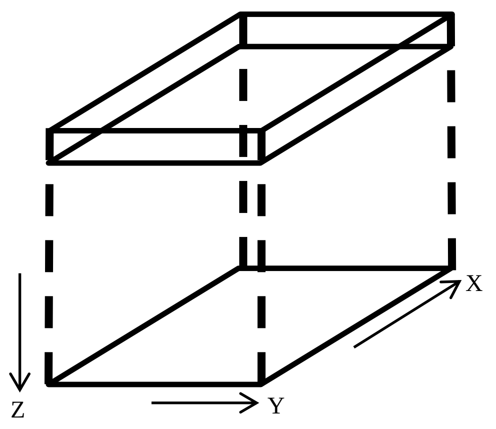

Figure 1: The 3D volume is arranged in depth slices. The data is contiguous in the X direction. Y is the second direction and Z is the slow axis.

In the second stage of the computation, the backward loop is implemented in chunks of the same size $DIMY \cdot SMX$. For each chunk, SMX samples of the backward loop are computed by reading the data from the auxiliary (transposed) array that is stored in global memory. The partial result of the backward loop for each chunk is stored in shared memory. This partial result of the second derivative can then be used efficiently to complete the computation in the original order (X axis is fast in global memory).

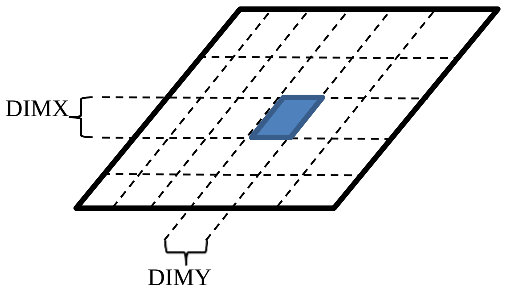

Figure 2: A depth slice is divided to GPU thread blocks of size $\boldsymbol{DIMX \cdot DIMY}$

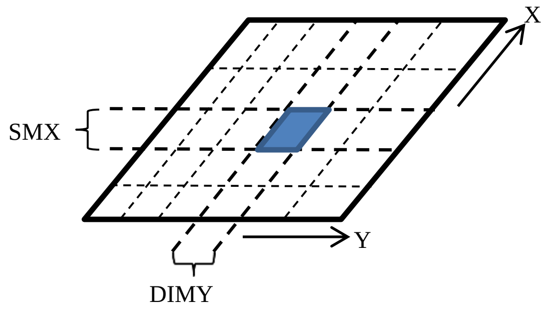

Figure 3: At each depth level a thread block of size DIMY calculates the recursive loops in chunks of size $\boldsymbol{SMX \cdot DIMY}$

FD operators can be implemented on hardware accelerators in a relative straight forward manner as described above, but can result with less than optimally imaged seismic data. This raises the challenge implementing accurate spatial derivatives such as recursive operators for hardware accelerators as the core kernel of a RTM PSDM algorithm. A solution to this challenge uses all the resources provided in a compute node, including both CPU and GPU memory and fast data exchange between the two, as well as a balanced split of computation between CPU cores and GPU cores. This implementation results with a CPU/GPU RTM code that is 30% slower than an equivalent high order FD based algorithm, but will achieve the goal of providing higher frequency RTM PSDM migrated data with minimal dispersion.

Example

For illustrating the RTM PSDM recursive operators, we constructed a salt environment anisotropic TTI model showing potential exploration targets around and beneath a salt body. The TTI model is shown in figures 4-6. The model is 10 miles long and 30,000 ft. deep. The salt body ranges from 5,000 ft. to 20,000 ft. The subsalt targets are depths greater than 20,000 ft.

For generating the input dataset, we created a set of 800 simulated shots over the target salt body. Shot point spacing for the simulation is 123.03 ft., Receiver spacing 41.01 ft. and the cable length is 10Km. Twelve second recording time was used in the simulation.

The sedimentary velocity field (shown in figure 4) of the model ranges between water velocity and 13,000 ft/s. The salt velocity is 14,471 ft/s. The anisotropy delta field (shown in figure 5) ranges between 0% - 10% and the anisotropic epsilon field ranges between 0% -16%. The anisotropic dip field (show in figure 6) was calculated based on the sedimentary section dips and ranges from -60 degrees to +60 degrees. The common shot TTI PSDM results of the simulated dataset using the recursive operators RTM is shown in figure 7. The model layers are displayed in figure 7 to illustrate the accuracy of the PSDM algorithm.

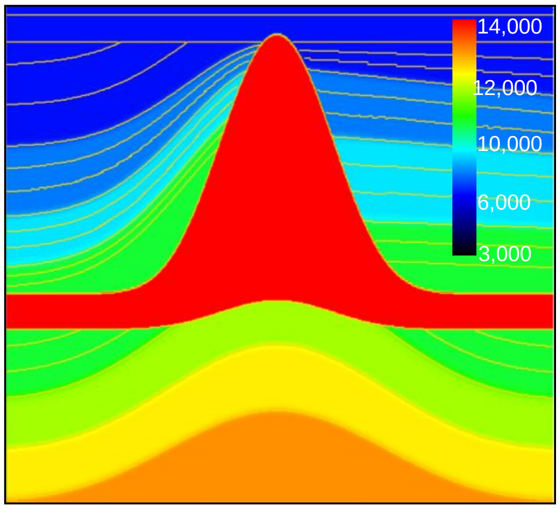

Figure 4: Salt related velocity model. Sedimentary layers are truncated against the salt dome.

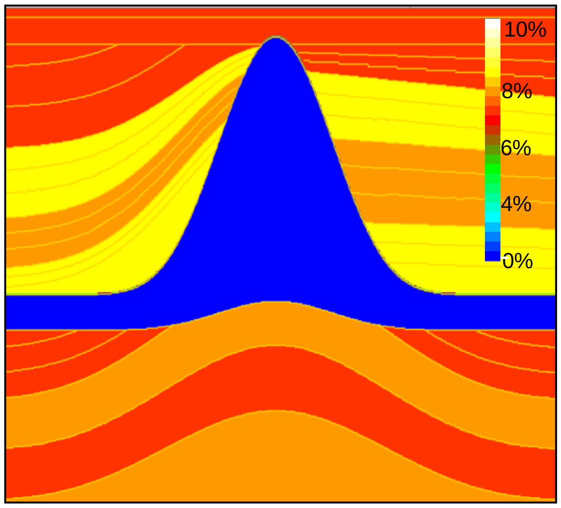

Figure 5: Anisotropic delta model. The anisotropic epsilon field is 1.6 greater than the delta field.

Conclusions

RTM PSDM is widely used today to assist in the imaging of both exploration and production projects. In most cases, however, the full frequency range is not input to RTM due to (a) the cost implication and (b) the risk of producing numerical dispersion. In this study we tackle these two limitations by (a) introducing a new derivative operator that can be used for migrating higher frequencies with minimal numerical dispersion, and (b) implementing these operators on a GPU architecture so it can be used cost effectively in a production environment. Our observation is that by combining a more accurate numerical scheme and a better computer optimization, the resulting RTM PSDM can be more reliable and the additional computational cost can be offset by use of advanced accelerators such as GPUs.

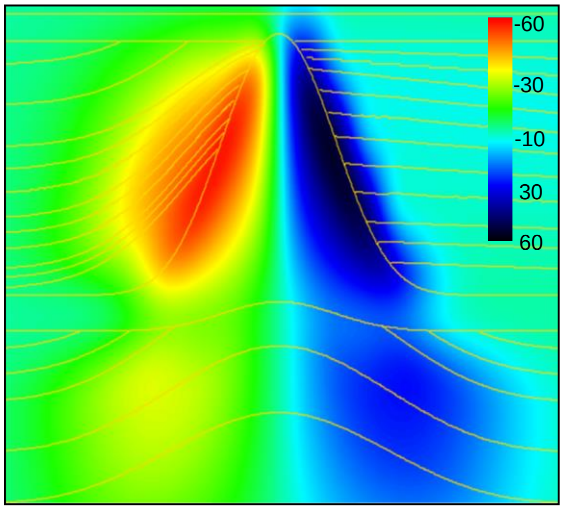

Figure 6: Dip model. The dip model was constructed by calculation of the normal to the layers geometry.

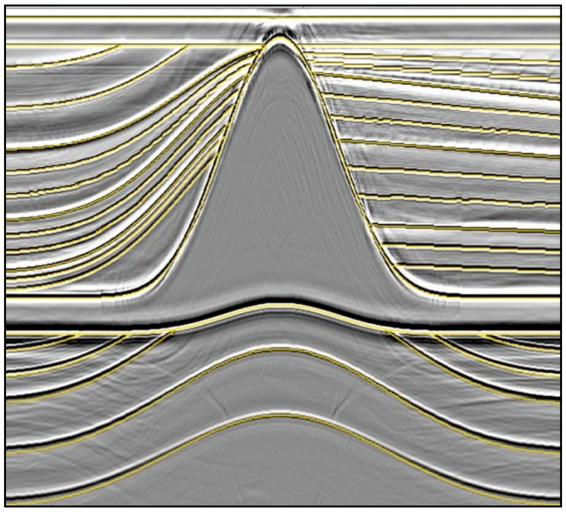

Figure 7: TTI RTM PSDM section produced using the minimal dispersion recursive spatial operators.

DOI http://dx.doi.org/10.1190/segam2015-5754164.1

EDITED REFERENCES

Note: This reference list is a copyedited version of the reference list submitted by the author. Reference lists for the 2015 SEG Technical Program Expanded Abstracts have been copyedited 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

Kosloff, D., R. Pestana, and H. Tal-Ezer, 2008, Numerical solution of the constant density acoustic wave equation by implicit spatial derivative operators: 78th Annual International Meeting, SEG, Expanded Abstracts, 2057–2061.

Micikevicius, P., 2009, 3D finite difference computation on GPUs using CUDA: Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU), 79–84, doi:10.1145/1513895.1513905.

Micikevicius, P., 2011, Stencil computation on GPU for seismic migration isotropic, VTI, and TTI RTM kernels: SEG Workshop on High Performance Computing in the Geosciences, http://www.seg.org/documents/4670773/4673682/Stencil+Computation+on+GPU+for+Seismic+Migration.