Memory efficient 3D reverse time migration

Chris Leader and Robert Clapp

ABSTRACT

Reverse time migration in three dimensions has two key bottlenecks - wavefield computation and IO limitations due to mass data transfer. Wave propagation and correlation in three dimensions imposes large computational constraints, both in terms of the number of floating point operations and the size of objects that need to be allocated. Furthermore, wavefields must be constantly written and read from disk since memory cannot possibly hold them all, this causes conventional RTM to become IO bound quickly. To address the computation requirements GPU propagation kernels are used to greatly reduce the computation time for source side modeling, achieving speeds in excess of 2.5 gigapoints calculated per second. Additionally, data handling requirements are vastly reduced by imposing pseudo-random boundaries on the velocity field, allowing time reversable source propagation. Achieving time reversable source propagation alleviates the requirement of checkpointing or boundary reinjection; this minimal data transfer results in GPU on-device operations becoming further accelerated.

INTRODUCTION

For advanced, accurate imaging Reverse Time Migration (RTM) is now the standard method. Despite the computational requirements relative to Kirchhoff or one-way wave-equation migration methods it has many characteristics that are desirable. In complex velocity fields the assumptions underlying one-way Wave Equation Migration (WEM) are insufficient, as only upward primary reflections are used, velocity approximations (FFD, PSPI etc) are made and only approximate phase and amplitude information is retained. Hence events with multiple reflections, overturning waves and steep dip information are not correctly positioned. Kirchhoff migration can image these steep dips, however the high frequency assumption results in sections of the data being incorrectly positioned and high angle artifacts in complex velocity structures. Conversely RTM images can resolve turning waves, horizontal waves and prismatic waves. In addition, no velocity, amplitude or phase approximations are necessary; though for practical purposes often such approximations are made.

For practical implementations, 3D RTM imposes large data handling problems. Generally, the source is fully propagated and saved in memory; the recorded data is then back propagated and correlated with the forward modeled shot at each imaging time step. Once geometries are 3D, it is not possible to hold this source wavefield fully in memory.

In addition to these large memory requirements, computational problems must also be addressed. As 3D surveys continue to become larger, with longer crossline and inline offsets and denser source sampling, then to fully benefit from this additional data the source modeling and receiver back propagation have to be performed over large velocity models. This problem is further compounded by the fact that finite difference wave propagation is less stable in as dimensionality increases, meaning smaller time steps have to be used (Dablain, 1986). Typical model dimensions can range from 500mb to tens of gigabytes, and the number of time steps needed to accurately model these data without dispersion is often around 10000. Using a spatial stencil order of 8 and temporal of 2, then over 1 giga point (Gpts) of calculations are needed per time step for both the source and the receiver.

In the following sections I will discuss how using GPUs can greatly accelerate finite difference time domain wave propagation, and how random velocity boundaries can improve the IO performance of RTM. Initially I will give a brief overview of GPUs. A standard measure for kernel performance is the number of model points calculated per second (Micikevicius, 2009), and this is the metric that will be used subsequently.

GENERAL PURPOSE GRAPHICS PROCESSING UNITS

Using a General Purpose Graphics Processing Unit (GPGPU) one can perform many independent parallel instructions simultaneously - far more than possible using a parallel CPU based system. These are known as SIMD (Single Instruction Multiple Data) devices, as they are capable or running one set of instructions many times over parallel threads. With the release of the programming language CUDA in 2006 Nvidia provided a way of harnessing the power of graphics processing units with limited knowledge of graphics processing.

The GPU can be thought of as a 2D structure, or grid. This grid is broken into blocks, each block in term consists of a group of threads, Figure 1. Each of these thread-blocks has its own shared memory, which can be unique per thread-block, and it's own set of registers, which are the same between all thread-blocks. Each thread in this hierarchy can execute a set of instructions (a kernel) concurrently, allowing for fine grain parallelism. The true potential of the GPGPU architechture lies in how memory latency can be hidden. The GPU partitions resources using registers and shared memory, these both have a latency of only a few cycles. Mass simultaneous execution effectively hides memory latency and context switching is (essentially) free. Parallel CPU execution, such as that possible when using OpenMP, MPI or POSIX, does not partition resources and features a memory latency at least an order of magnitude higher than the GPU equivalent, making this form of parallelisation vastly less efficient.

A schematic for how thread-block and global memories can interact is shown in Figure 2.

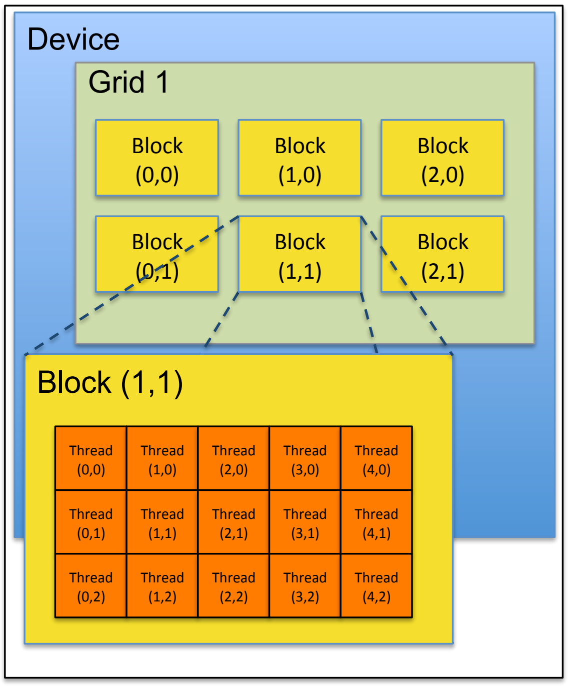

Figure 1: A schematic for the architechture of an Nvidia GPU [**NR**]

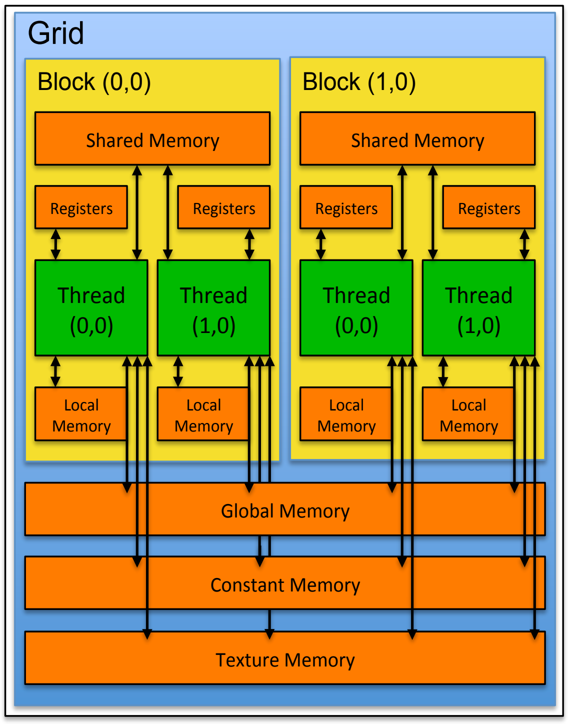

Figure 2: A schematic for the memory heirarchies within the GPU [**NR**]

3D WAVE PROPAGATION

The method used for wavefield forward and back propagation was time domain finite differencing of the 3D acoustic wave equation. This method approximates the temporal and spatial derivatives as a sum of Taylor series' about a symmetric sequence of points, such that this sum closely approximates the Taylor series of the desired derivative.

$$\frac{\partial^2 u(x,y,z)}{\partial t^2} = c(x,y,z)^2 \nabla^2 u(x,y,z) \tag{1}$$

The velocity model (Figure 3) used to test these propagation and imaging routines features velocities ranging between 1490 m/s to 4480 m/s; using a Ricker wavelet of central frequency 25 Hz then time steps of 0.5 ms and spatial sampling of 10m was used, adhering to the guidelines of Dablain (1986), extrapolated to 3D.

CPU Implementation

Solving Equation 1 numerically using a finite-difference time domain approach can be simply set up as a serial convolution repeated across the entire domain. Of course this is the most naive and least efficient way to do this. Over a very modest model size of 1 million points, for 8000 time steps this takes around 93 minutes to propagate. This serial methodology can be improved by blocking the domain into smaller 'pencil' shaped blocks about the fast axis; this will give a better cache hit rate, reduce calls to L3 memory and hide some latency. This can roughly half the computation time.

By parallelising over multiple cores a non-blocked speed up of around 3.5x is seen (over 8 cores, after this speed up saturates). Using the optimum blocking and parallelism discovered this computation time is reduced to around 16 minutes, giving a 5.5x - 6x speed up. Nonetheless, 3D model sizes are typically two or three orders of magnitude larger than this, meaning for a realistic model size this parallel, blocked CPU methodology will take several hours per shot. It should be stressed that this is the best speed up observed using a Fortran90 with OpenMP approach, by further vectorising loops and coding at assembly levels this can be further accelerated many times.

GPU Implementation

Using the identical serial kernel in CUDA this propagation takes 18 minutes. This is slower than the fastest CPU technique because GPU global memory latency is higher than on the CPU, to harness the power of the GPU shared memory and registers must be used.

Micikevicius (2009) describes an efficient way of setting up the base wave propagation kernel that utilises latency hiding by copying stencil location values into shared memory, dependent on thread location. Performing this propagation in 3D is about 27 times faster than the naive GPU approach, with speeds of over 3 Gpts/s achievable for typical model dimensions, and a run time of 41s (for the previously used model).

Operations purely on GPU global memory are slower than on a CPU, with latencies of 400-600 cycles, rather than 100-200 on a CPU. Due to this, a typical school of thought minimises data manipulation on the GPU. This approach constitutes sending to host (CPU) the full 3D wavefield at each desired modeling time step, causing the CPU wavefield object to be 4D. This can then be transposed and windowed accordingly. This method gives a speed of 24x over the best, blocked and parallelised CPU equivalent. Having to copy data this way is a shortcoming of interacting Fortran90, C and CUDA.

Storing the entire 4D wavefield as an allocated object on the CPU is unwise - memory is quickly saturated. For example, a model size of 500x500x500 dictates that only 64 time steps could be performed (on a 4 Gb card).

Several improvements can be made to make the GPU propagation as fast as possible. A new windowing kernel can be used to only send 2D wavefield slices back to the host (corresponding to desired acquisition geometries), this only marginally slows down the modeling and makes the full implementation vastly more flexible. In fact since now less data has to copied back to the CPU a speed up of about 1.3x is seen.

The final speed up thus far when compared to most efficient, blocked and parallelised CPU routine is about 35x, and the current throughput is about 2.2 Gpoints/s. If the CPU modeling was fully optimised the observed speed up would be 15-20x. The evolution of computation speed (measured in million computed points per second) against propagation scheme can be observed in Figure 4.

The finite computational domain used when simulating wavefield propagation provides additional challenges. To accurately simulate real world physics, an infinite domain must be used, but of course this is not possible. If no boundary conditions are used then reflections from the computational domain are prevalent in the modeled data. There are many ways of reducing or hiding these domain reflections, such as zero value, zero order, absorbing, damping and Perfectly Matched Layer (PML) (Turkel and Yefet, 1998). For modeling an absorbtion scheme was used, which reduced throughput by about 10%.

This accelerated 3D wave propagation method will provide the engine for 3D RTM.

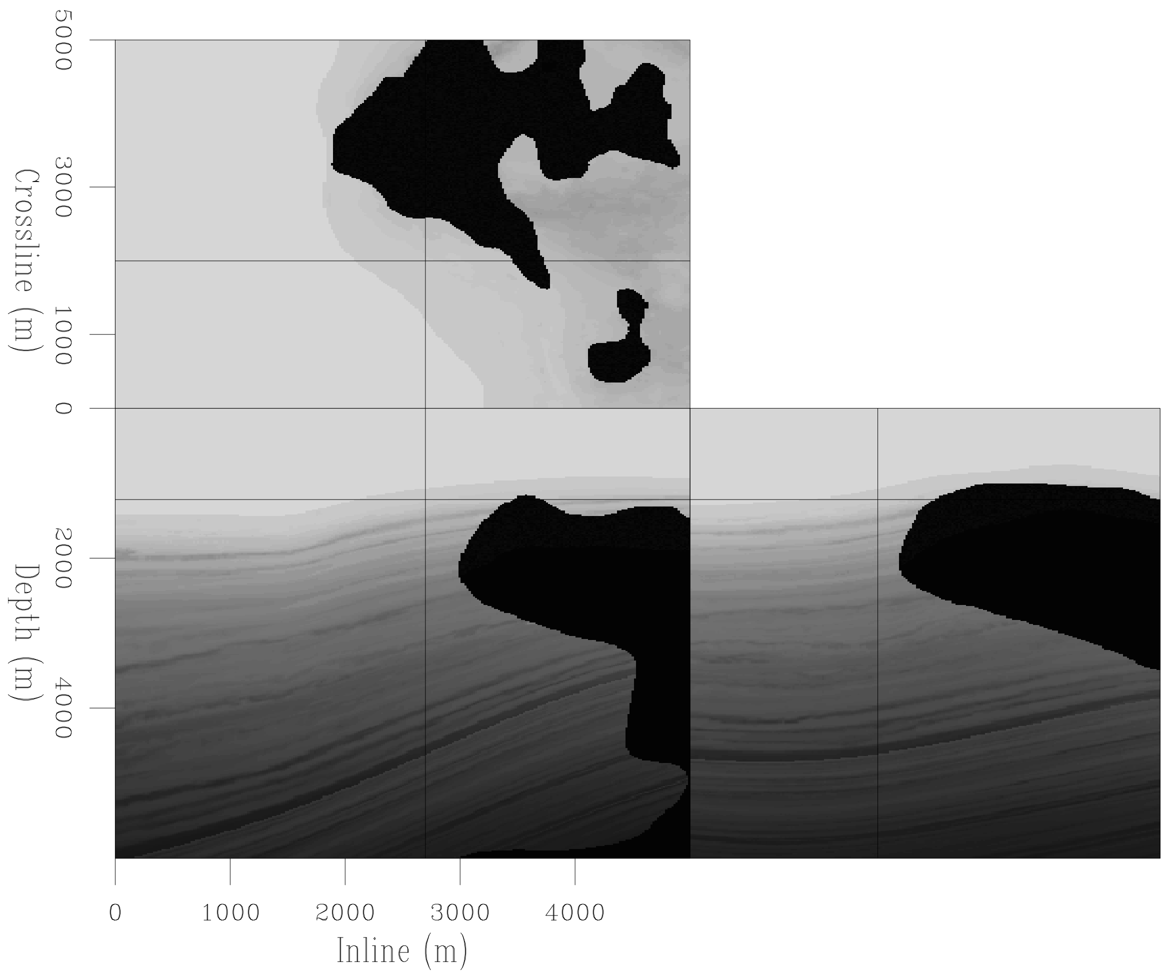

Figure 3: A 3D cubeplot of the velocity model used for 3D imaging [**ER**]

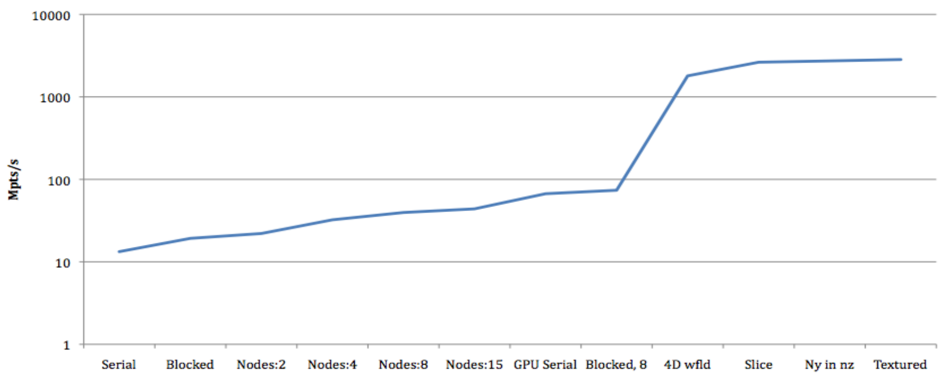

Figure 4: Graphing the speed up between various CPU and GPU propagation schemes [**NR**]

REVERSE TIME MIGRATION

Reverse time migration is a well known method, whereby reflection events are imaged in a kinematically accurate manner. Baysal and Sherwood (1983) and Claerbout (1985) pioneered the method, whereby an image is constructed by back propagating the receiver wavefield (recorded data) and forward propagating the source wavefield (modeled data, using assumed wavelet) and then cross correlating these at each imaging step. Mathematically this reduces to the imaging condition shown in Equation 2. It can be shown that this system is simply the adjoint of linearised modeling (Biondi, 2010).

$$I(z,x,y) = \sum_{i}^{nshots} \sum_{t}^{n} P_s(t,z,x,y;\mathbf{s}_i) P_g(t,z,x,y;\mathbf{s}_i) \tag{2}$$

$I(x,y,z)$ is the image at point (x,y,z), $P_s(t,z,x,y;\mathbf{s_i})$ is the source wavefield for shot $\mathbf{s_i}$ and $P_g(t,z,x,y;\mathbf{s_i})$ is the corresponding recorded wavefield. This can be extended to a prestack imaging condition where subsurface offsets are included. This is necessary velocity estimation and many inversion methods. Equation 2 simply extracts the zero subsurface offset image.

$$I(z,x,y,x_h,y_h) = \sum_{i}^{nshots} \sum_{t}^{n} P_s(t,z,x+x_h,y+y_h;\mathbf{s}_i) P_g(t,z,x-x_h,y-y_h;\mathbf{s}_i) \tag{3}$$

In Equation 3, x_h and y_h represent the subsurface half offsets. For each set of subsurface offsets the two wavefields are laterally shifted and the imaging condition reapplied, hence a 5D prestack image can be constructed (Biondi, 2006).

In 3D the computational intensity of RTM can become a hinderance relative to simpler migration operators. This is because the modeled shot, $P_s(\mathbf{s_i})$, has to run forward in time, and the recorded data, $P_g(\mathbf{s_i})$, is reversed in time. Since domain boundary damping makes the shot modeling non-reversable then practically we model and save the entirity of $P_s(\mathbf{s_i})$, and hold this in memory as we reverse through $P_g(\mathbf{s_i})$, applying the imaging condition at each time step. Of course, for large wavefields this is unpractical. One partial solution is referred in literature as checkpointing (Symes (2007); Dussaud and Cherrett (2008)). Here we effectively block the propagation and periodically reinject the source model. For example, let some intermediate position in the propagation be t_{check}, initially $P_s(t_{max} - t_{check}; \mathbf{s_i})$ is read, the receiver incrementally back propagated to t_{check} and the correlations applied. This can then be done t_{max}/t_{check} times (Clapp, 2008a) and can be parallelised. The memory problem is thus solved, however this process is fundamentally IO bound.

Clapp (2008b) proposed a different scheme - to use pseudo-random boundaries optimised to scatter these wavefields incoherently (Figure 5). The RTM imaging

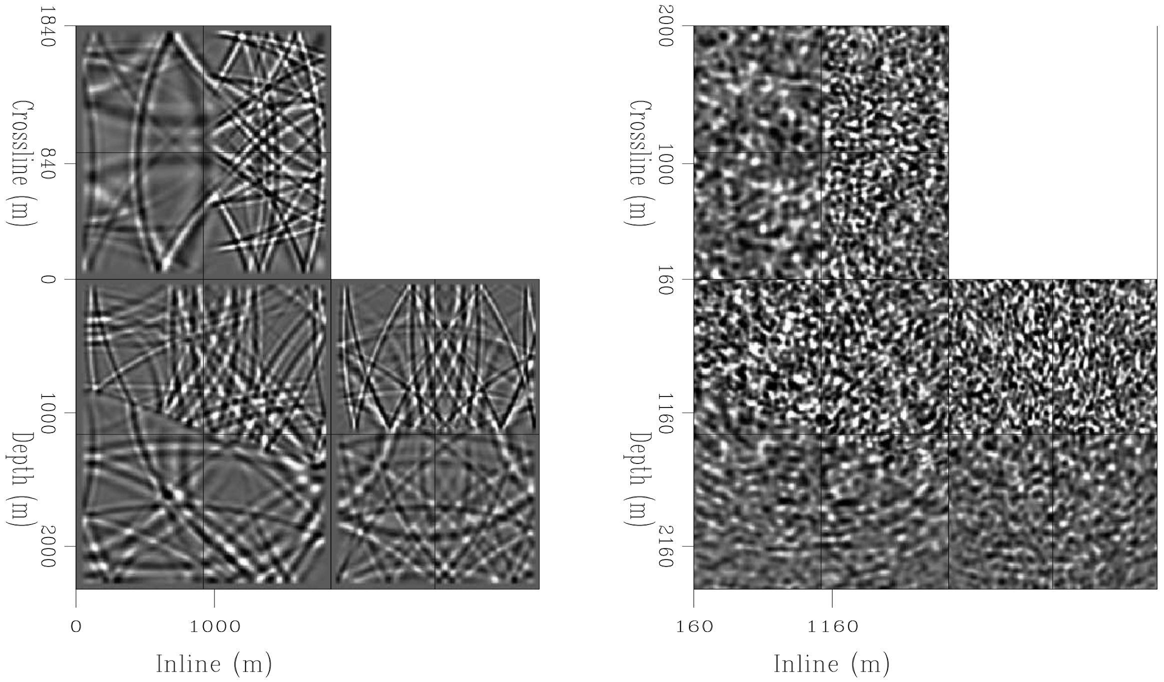

Figure 5: A slice of the 3D wavefield after long propagation times, without and with random boundaries [**CR**]

principle operates on the fact that incoherent data are either correlated out in imaging, or stacked out subsequently. Pseudo-random boundaries create a lot of noise in the propagated shot, however this will not correlate coherently with the receiver wavefield. Furthermore this propagation is now entirely time reversible, and by saving the velocity field the shot can be reverse propagated to its original state within machine precision (Figure 6). The benefits of this are twofold - computationally the source wavefield is time reversible, and so only two source wavefields are needed for any given time and no checkpointing is necessary; geophysically this algorithm provides accurate imaging, and any residual boundary scattered noise is well below the noise threshold. Thus a manner in which 3D RTM is both computationally and memory efficient can be constructed.

As seen in Figure 5, the low frequency component of the wavefield is not well scattered. For RTM this is not a significant problem, since low frequency artifacts are often prevalent and are removed by a low cut filter (or domain decomposition) before imaging (Taweesintananon (2011) provides an overview of these methods). However techniques such as waveform inversion methods rely on low frequency information. Shen and Clapp (2011) show how random boundaries can be further optimised, such that low and high frequencies become incoherent after scattering.

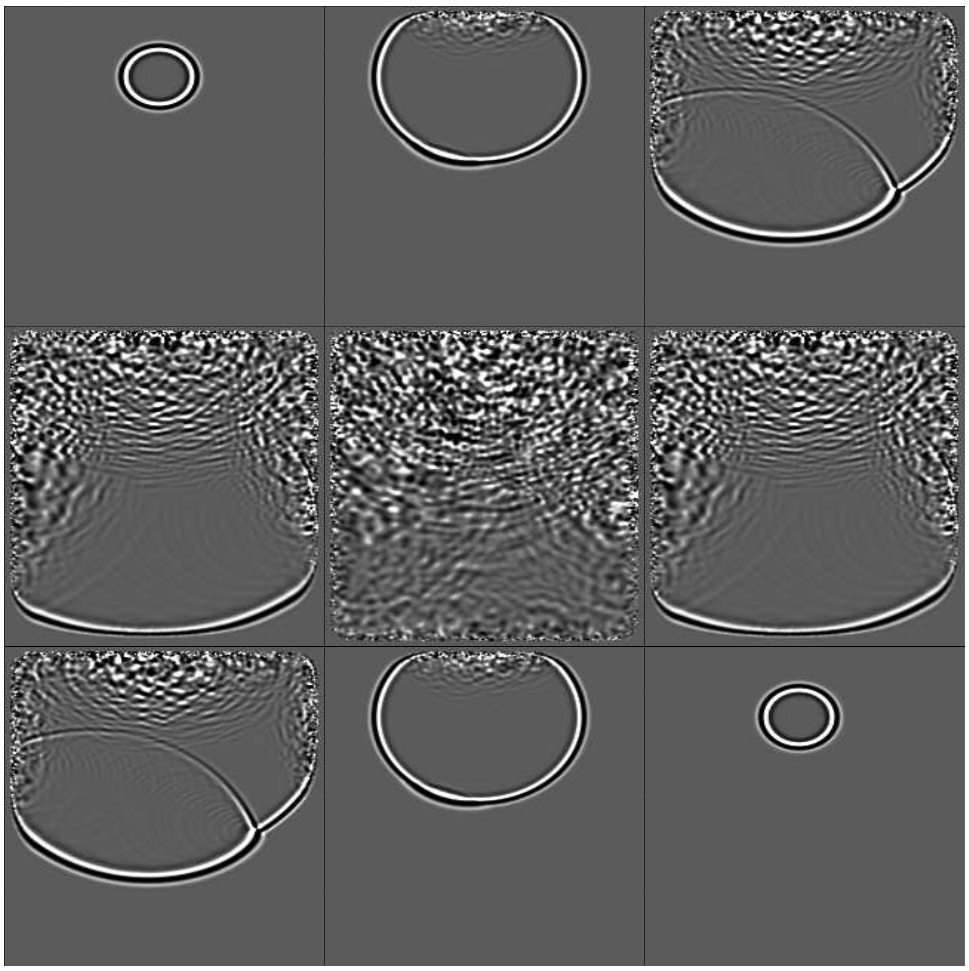

Figure 6: An example of the time reversable wave propagation [**CR**]

GPU implementation

Setting up this scheme optimally requires some thought. A random boundary of user input depth is imposed on this velocity section, which is then bound as a texture on the GPU. The source is then fully propagated and the final two 3D wavefields saved; now we have all the array inputs for our migration. This runs from t_{max} to t_0, and at each time step source and receiver wavefields are injected and propagated, at each imaging time step these fields are multiplied and the image stored and updated on the GPU. Damping boundaries are used for the receiver propagation, only one image object is used, and all correlations are stacked into this, then after propagation is completed the stacked image is sent back to the CPU. Under this scheme six 3D objects need to be stored - two wavefield slices for both the receiver and the source fields, the velocity model and the image. The fact that no device to host transfers occur during the time loop makes this kernel very efficient.

The extension to subsurface offsets adds data handling and numerical complexity. The aforementioned scheme produces a poststack image, since each image is stacked into one object on the GPU. When applying the imaging condition in Equation 3 the wavefields must be shifted and correlated mutliple times, for each time step, to produce the subsurface offset gathers. For practical execution we now need to transfer each imaging time step back to the host. Furthermore this multiplication is inefficient on the GPU, since the threads may need re-indexing after shifting, and a global memory operation is used. Future work is underway to see if these shifted correlations are better performed on the CPU. Other work at SEP on how compressive sensing can be used to reduce this new bottleneck is currently being performed (Clapp, 2011).

RESULTS

Using the latest SEAM velocity model both 2D and 3D random boundary migration routines were experimented with. A simple RTM result using a 2.5D dipping reflector is shown in Figure 7; this was created using a single shot at 20m depth and then applying a very low cut filter. Here very little background noise can be seen from the random boundaries, and with more shots this will reduce further. Few of the typical RTM artifacts are seen because of the simplistic nature the ray paths take in this model - all multiples and direct arrivals are muted and there is no multi-pathing or overturning of waves.

In the vastly more complex SEAM model multiple shots are required to suppress artifacts, especially since there are now many more conventional RTM artifacts due to the basic imaging condition used, such as those from correlating wavefields travelling in the same direction. Currently more work must be undertaken to ensure the velocity model is fully block-aligned in the GPU and the determine the best approach to begin to suppress these new imaging artifacts; a basic one shot RTM over the SEAM model can be seen in Figure 8. Some reflectors are noticeable and continuous, however there are several erroneous events and a lot of shallow noise.

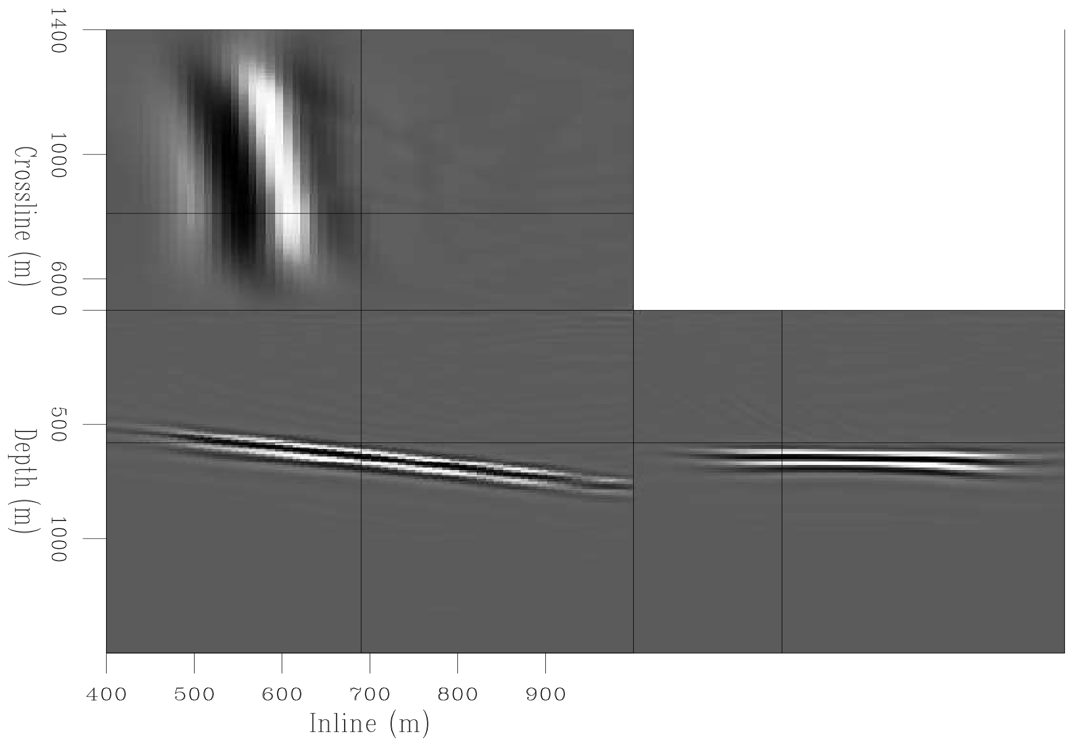

Figure 7: RTM result using one shot over a single dipping reflector [**CR**]

CONCLUSIONS

The goal of this investigation was to try and reduce the RTM bottlenecks of 3D wavefield computation and memory transfers. The former of these was vastly reduced by using a series of GPU oriented kernels to perform wave propagation in a massively parallel sense, a total speed up of 30x was seen. The memory transfer data handling problem within RTM was diminished by using increasingly random boundaries on the velocity model to produce time-reversable wavefield modeling, alleviating the method from the intensely IO bound technique of checkpointing. A simple 2.5D single reflector model showed an encouraging RTM result, which demonstrates that random boundaries in 3D do indeed provide results comparable to checkpointing; work is underway to obtain detailed images of more complex velocity structures.

ACKOWLEDGMENTS

I would like to thank Bob Clapp for his patience and assistance on CUDA and C++ programming, and for providing an object oriented framework which inspired how this procedure was set up. I would also like to thank Xukai Shen and Ali Almomin for their CUDA debugging and receiver gather interpretation expertise.

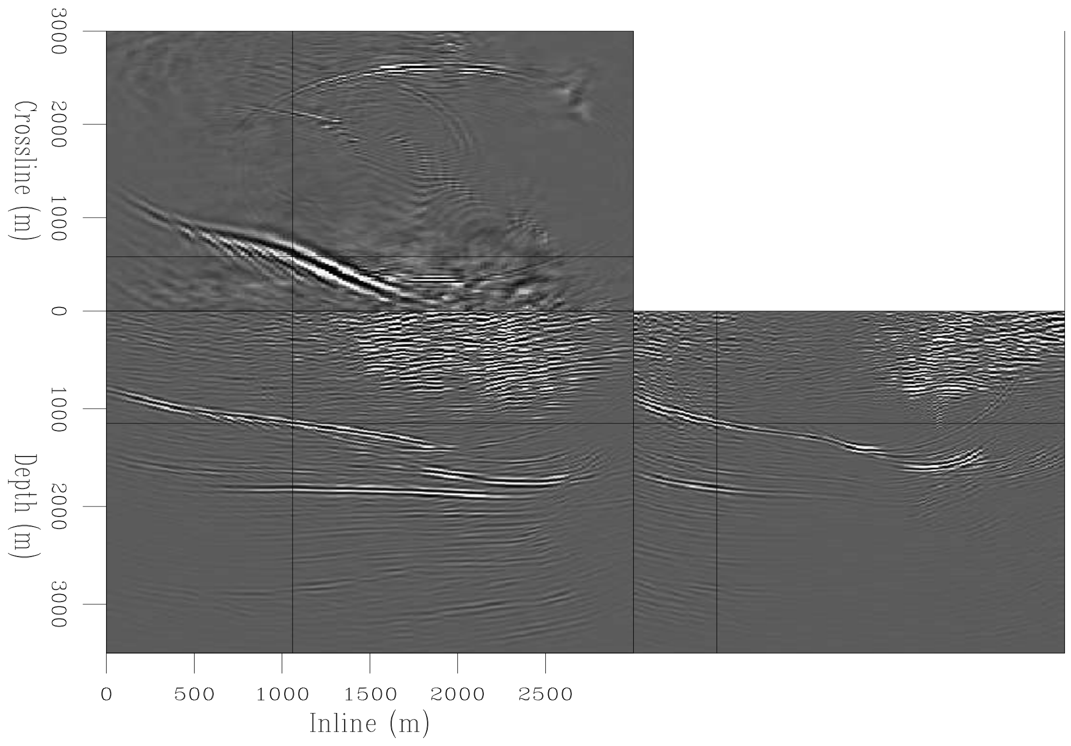

Figure 8: RTM result using one shot over a windowed section of the SEAM model [**CR**]

REFERENCES

Baysal, E, K. D. and J. Sherwood, 1983, Reverse time migration: Geophysics, **45**, 1514–1524.

Biondi, B., 2006, 3d seismic imaging: Society of Exploration Geophysicists.

———, 2010, 3d seismic imaging, additional notes: Migration as adjoint of linearized modeling: Society of Exploration Geophysicists.

Claerbout, J., 1985, Basic earth imaging: Society of Exploration Geophysicists.

Clapp, R., 2008a, Reverse time migration: Saving the boundaries: SEP-136, **136**.

———, 2008b, Reverse time migration with random boundaries: SEP-138, **138**.

Clapp, R. G., 2011, Imaging using compressive sensing: SEP-Report, **143**, 149–158.

Dablain, M. A., 1986, The application of high-order differencing to the scalar wave equation: Geophysics, **51**, 54–66.

Dussaud, E, S. W. L. L. S. P. D. B. and A. Cherrett, 2008, Computational strategies for reverse time migration: SEG Expanded Abstracts, **78**, 2874–2877.

Micikevicius, P., 2009, 3d finite difference computation on gpus using cuda: GPGPU, **2**.

Shen, X. and R. G. Clapp, 2011, Random boundary condition for low-frequency wave propagation: SEP-Report, **143**, 85–92.

Symes, W., 2007, Reverse time migration with optimal check pointing: Geophysics, **72**, 213–221.

Taweesintananon, K., 2011, Reverse-time migration using wavefield decomposition: SEP-Report, **143**, 171–190.

Turkel, E. and A. Yefet, 1998, Absorbing pml boundary layers for wave-like equations: Applied numerical mathematics, **27**.

OPEN DAILY
HALLIBURTON