Implementation of Kirchhoff prestack depth migration on GPU

Davide Teixeira, Alex Yeh and Sampath Gajawada, TGS*

Summary

The massively parallel nature of Graphics Processing Units has made them an attractive platform for some computationally intensive algorithms. This article presents a method to run 3D Kirchhoff prestack depth migration on GPU-based clusters. Compared to a CPU only version of the same algorithm, the new approach delivers a significantly greater efficiency. An actual production run with field data reveals the extent of the improvements.

Introduction

3D Kirchhoff prestack depth migration (KPSDM) is one of the most commonly used tools for processing seismic data.
CPU-based KPSDM has been well used in production on CPU-clusters. Some algorithms like Reverse Time Migration (RTM), run preferentially on Graphics Processing Units (GPU) based clusters (Sun and Suh, 2011).

Investing in GPU-based clusters means more software needs to be efficiently converted in order to maximize hardware utilization and also decrease seismic processing costs. KPSDM seems to be a prime candidate for such a conversion.

To date, most of the articles about porting Kirchhoff migration algorithms to GPU are in time domain (Panetta *et al.*, 2009, Shi *et al.*, 2009, Brouwer *et al.*, 2011). This article first details the different phases of our KPSDM algorithm and the challenges we encountered. We then go through the different methods used to convert KPSDM to run efficiently in a production environment on GPU-based clusters. Finally, we show an example of such a run and the performance comparison with the CPU multi-threaded version of the same algorithm.

KPSDM Algorithm

The Kirchhoff migration can be characterized as the summation of the reflection amplitudes along the diffraction travel time curves to obtain the output images (Garabito *et al.*, 2011). There are many different implementations of KPSDM but they usually have the following major steps in common:

- Computing the travel time tables using ray-tracing and a velocity model
- Preprocessing the input seismic traces
- Migrating these data traces using the Kirchhoff algorithm
- Post-processing and outputting the results

In our algorithm, the travel time tables for the input shot and receiver locations are pre-computed using ray tracing. Then to migrate one input trace, the sequence of operations is:

- Get the travel time tables for the source and receiver locations
- Preprocess the input seismic trace
- Using the travel times, add a sample of the input trace data to the correct location in the output volume

After all input traces have been migrated, the output volume is then post-processed and saved on the disk.
Each of these steps has its own set of challenges.

Challenges

Compared to Kirchhoff time migration, KPSDM involves more Input / Output (I/O) from the system. Porting the ray-tracing algorithm to GPU could not be done in an efficient manner because of the nature of the algorithm and because of memory limitations on the GPU. We could still compute the travel time tables directly on the GPU clusters using only the CPUs but that would make the GPU resources idle when some other program could use them. For this reason, the travel time data are first computed on CPU clusters and then transferred through the network to the GPU clusters.

The input seismic data also need to be transferred to the GPU clusters to be migrated. The migration itself is computational so it is done using the GPU. Finally, the post-processing and writing out the results are mostly I/O bound so there is no advantage to port that part of the code to the GPU.

The large part of I/O in KPSDM means we need to find a way to reduce its impact in the overall computation time.

Reducing The I/O Footprint In KPSDM

One of the first ideas to increase I/O bandwidth is to upgrade the hardware. This option can become prohibitively expensive both in time and money. So without any hardware change, we devised a procedure to greatly reduce the time it takes to transfer data. This procedure takes advantage of the fact that most Ethernet connections are full-duplex which means that they support simultaneous transmission of data in two directions. Instead of sending data sequentially from one location to several cluster nodes, we have each cluster node receive, transmit and process data at the same time as pictured in Figure 1.

DOI http://dx.doi.org/10.1190/segam2013-0979.1

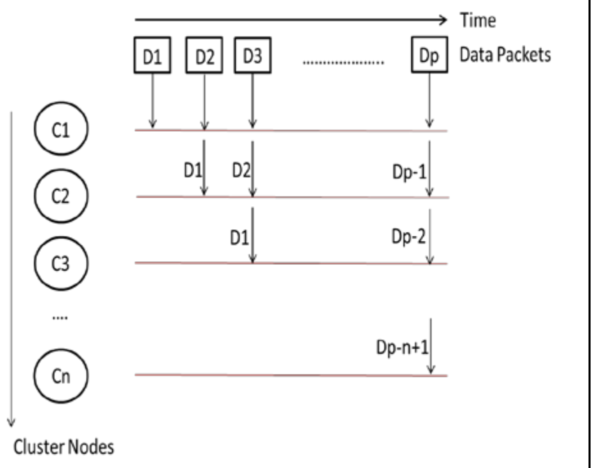

Figure 1: Simultaneous transfer of data across cluster nodes

Let's compare this new approach to the one where data are consecutively copied from one node to many. Let *T* be the time needed to transfer one data packet, *p* the number of packets and *n* the number of cluster nodes. The total time it takes to transfer all data to all nodes for the one-to-many method is:

$$\text{Total transfer time} = T * p * n$$

Whereas for the new method, it is:

$$\text{Total transfer time} = T * (p + n - 1)$$

As the number of cluster nodes increases, the new method becomes significantly faster than the one-to-many approach.

To reduce the I/O footprint, we also compress the travel time tables and buffer them as needed. Input seismic data are also buffered. This way we can overlap data and travel times reads with the migration kernel which is executed on GPU.

GPU Implementation

Compared to current CPU, GPU have a lot more cores. Unfortunately, to leverage this parallelism, an existing CPU-only code needs to be modified. These changes are not straightforward. They require more knowledge about the architecture of GPU hardware than when implementing a CPU-only software. Some of the main requirements for good performance are memory alignment and access patterns ("CUDA Toolkit Documentation"). So in our implementation, we make sure all the memory allocations and accesses are aligned to a warp (i.e. a group of 32 threads). The entire output volume can be quite large so it is stored in the GPU global memory. In this volume, the data are stored in Z first then X, Y and offset order to promote data locality. We also store the input seismic traces in global memory. We tell the GPU to favor the use of the L1 cache to improve memory access speed. Since execution on a GPU is asynchronous with CPU execution, we overlap it with other operations as described in Figure 2.

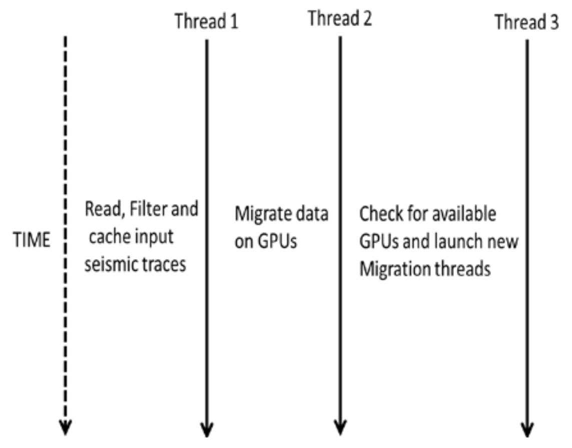
Thread 1
Thread 2
Thread 3
TIME
Read, Filter and cache input seismic traces
Migrate data on GPUs
Check for available GPUs and launch new Migration threads

Figure 2: Overlap of GPU and CPU operations

There can be more than one migration thread depending on the number of GPU available on a particular cluster node. The program dynamically detects unused GPU and seizes them to increase the number of input seismic traces being migrated at a given time. This way we also improve the load balancing between jobs when more than one is run on a single node. The GPU kernel migrates one trace at a time. Inside this kernel, each thread is responsible for migrating a subset of a trace in the output 4D volume.

Let now see how this implementation performed when migrating a real 3D dataset.

Example

The following data were used to run the GPU-based KPSDM on a single node using only one GPU card. It was then used to run the CPU-based version on a CPU cluster using 10 nodes with 4 cores each. The input seismic data consisted of 883,385 traces. The aperture was set at 10 Km. Volume dimensions for the migration output are:

- 26 offsets
- Nx = 4,511, Ny = 8, Nz = 1,201
- dx = 25 m, dy = 30 m, dz = 10 m.

Figure 3 shows the input velocity model used for both runs.

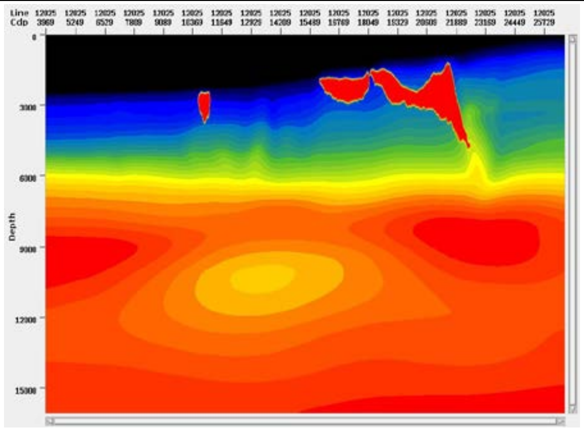

Figure 3: Velocity model

The stack results of the GPU (see Figure 4) and of the CPU programs are almost identical.

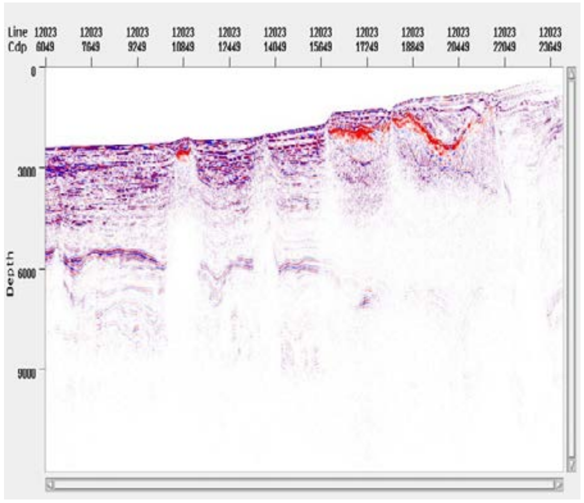

Figure 4: Stack result of GPU-based 3D KPSDM

The run time for both programs is illustrated in Figure 5.

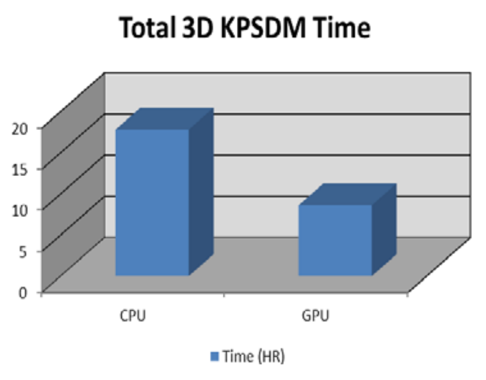

Figure 5: 3D KPSDM run time comparison between 10 CPU-cluster nodes with 4 cores each and a single node with a single GPU card

For this example, the GPU-based 3D KPSDM was about twice as fast as the CPU-based version. In other words, one single GPU node was equivalent to 20 CPU nodes with 4 cores each.

Conclusions

Our implementation of 3D KPSDM on a GPU-based cluster significantly outperforms the same algorithm run on a CPU-based cluster. It maximizes the use of I/O and computation resources for efficiency. This implementation is now being used in a production environment. The advent of co-processors like GPU has greatly improved the computation throughput. On the other hand, it has also highlighted the limiting factor of I/O and memory bandwidth where further progress is needed.

Acknowledgements

The authors would like to thank Sang Suh and Jing Xie for their input. We also thank Bin Wang, Laurie Geiger, Simon Baldock and Chuck Mason for reviewing and proof-reading this paper. Finally, we wish to thank TGS management for the permission to publish this paper.

http://dx.doi.org/10.1190/segam2013-0979.1

EDITED REFERENCES

Note: This reference list is a copy-edited version of the reference list submitted by the author. Reference lists for the 2013 SEG Technical Program Expanded Abstracts have been copy edited 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

Brouwer, W., V. Natoli and M. Lamont, 2011, A novel GPGPU approach to Kirchhoff time migration: 81st Annual International Meeting, SEG, Expanded Abstracts, 3465–3469.

CUDA Toolkit Documentation: http://docs.nvidia.com, accessed 3 April 2013.

Garabito, G. and P. Stoffa, 2011, Slowness-driven Kirchhoff prestack depth migration: Application in synthetic OBS data: 81st Annual International Meeting, SEG, Expanded Abstracts, 3351–3355

Panetta, J., T. Teixeira, P. R. P. de Souza Filho, C. A. da Cunha Filho, D. Sotelo, F. M. Roxo da Motta, S. S. Pinheiro, I. Pedrosa Junior, A. L. Romanelli Rosa, L. R. Monnerat, L. T. Carneiro, and C.H.B. de Albrecht, 2009, Accelerating Kirchhoff migration by CPU and GPU cooperation: Proceedings of the 21st International Symposium on Computer Architecture and High Performance Computing, 26–32.

Shi, X., C. Li, X. Wang, and K. Li, 2009, A practical approach of curved ray prestack Kirchhoff time migration on GPGPU, *in* Y. Dou, R. Gruber, and J. Joller, eds., Advanced parallel processing technologies: Springer, 165–176.

Sun, X., and S. Suh, 2011, Maximizing throughput for high-performance TTI-RTM: From CPU-RTM to GPU-RTM: 81st Annual International Meeting, SEG, Expanded Abstracts, 3179–3183.