# A GPU implementation of the reverse time migration algorithm.

Jhonatan Amado, UIS; William Salamanca, UIS; Flor Alba Vivas, Ecopetrol; and Ana Ramirez, UIS.

## Abstract

**This work presents a Graphic Processing Unit (GPU) implementation of a pre-stack two-way wave equation migration method known as reverse time migration (RTM). The GPU implementation of the RTM algorithm is based on the finite difference scheme in time and space, to compute the two-way wave propagation. The boundary conditions are based on the Convolutional Perfectly Matched Layer (CPML) strategy. Experimental results of the migrated image for the Marmousi model using the algorithm implementation are shown. In addition, results of the execution time of the RTM algorithm implementation in both architectures, a NVIDIA Fermi GPU and a INTEL CPU, show a speed-up of approximately 4x in the GPU implementation. Furthermore, results of disk transfer vs. RAM memory handling, and granularity show the benefits of the implementation on a GPU architecture.**

## Introduction

The seismic reflection is the most widely used method in oil prospecting. Traditionally, seismic reflection consists of three stages: design and data acquisition, seismic processing and imaging, and interpretation of the image of the subsurface. Different methods have been proposed for the seismic imaging stage. Basic and simple methods usually fail on finding migrated images of complex geological areas, because the reflectivity map has insufficient resolution or structures that cannot be clearly delineated, producing errors in the prospects. A pre-stack two-way wave equation migration known as Reverse Time Migration (RTM) has been proposed for imaging areas, where dips reach up to 90 degrees or areas containing turning waves, such as salt bodies [Pengliang *et al, 2004*].

The RTM uses the solution of the full wave equation for the accurate imaging of areas of abrupt changes of horizontal velocities, regardless of dips in such structure. However, RTM method is a computationally expensive process since it computes, for each point of the velocity model, two wave fields: source wave field and the receiver wave field, to then apply an image condition. The modeling of the source and receivers wave propagation uses the constant-density acoustic wave equation.

This process is done to each shot in the survey. The RTM migration time of a 3D shot in a 16 cores node is around 10 minutes for a shot of 5000 traces.

In modern marine acquisitions having hundreds of thousand shots, the full RTM migration of a survey is challenger in terms of time computing consuming and it can take weeks in a cluster of hundred nodes. Usually, the RTM migration of a 3D survey is done on a cluster of CPUs where each node migrates a shot, using parallel programming like MPI or OpenMP to distribute the work.

Alternatively, parallel programming of GPUs can be used in order to reduce execution time of each shot migration in a node where we have hybrid HPC with GPUs and CPUs.

### Numerical solution of acoustic wave equation

Since the RTM algorithm is based on the two-way wave propagation, it is necessary to first model the wave propagation equation, and then find a numerical solution of the model. The physical phenomena of the acoustic wave propagation through a constant density medium, can be described by the following differential equation

$$\frac{\partial^2 P}{\partial t^2} = C^2 \nabla^2 P \tag{1}$$

where, $P(x,z,t)$ is the pressure field, having $(x,z)$ as spatial coordinates, and $(t)$ as the time coordinate. The propagation speed of the wave in the given medium is given by $C(x,z)$. In equation (1), $\nabla^2$ is the Laplacian operator that represents the spatial differential.

The method used to find the numerical solution of the acoustic wave equation is the Finite Difference Time Domain (FDTD), in two dimensions. In particular, we use the second order approximation in time and the eighth order approximation of the Laplacian operator. Equation ( **1** ) can be discretized using the Taylor series in a homogeneous grid ($\partial x = \partial z = \partial h$) as follows

$$P_{(x,z)}^{n-1} - 2P_{(x,z)}^{n} + P_{(x,z)}^{n+1} = G^2 \nabla^2 P \tag{2}$$

$$G = \frac{C_{(x,z)} * \partial t}{\partial h} \tag{3}$$

where $G$ is the Courant parameter, which is required for the stability analysis of FDTD method.

## Dispersion and stability analysis

In the numerical solution obtained by using the FDTD approximation, it is necessary to know the accuracy and stability of the result. The conditions of distortion and stability for the solution given in equation ( **2** ) are

- Dispersion: errors related by the truncation of the Taylor series for differential operators and errors due to rounding the digital representation of operation.
- Stability: given by the consistency of the solution obtained by the numerical method when the errors are propagating.

The errors produced by numerical dispersion can be observed in Figures 1 and 2. When the spatial component is discretized using 8th order, as it is shown in `

Figure 1**,** the wave front representation presents a clear improvement in comparison to the spatial discretization of 2nd order, as it is shown in

Figure 2.

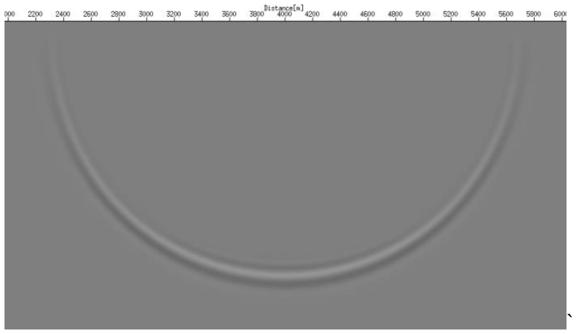


Figure 1: Wave front representation using a spatial discretization of 8th order.

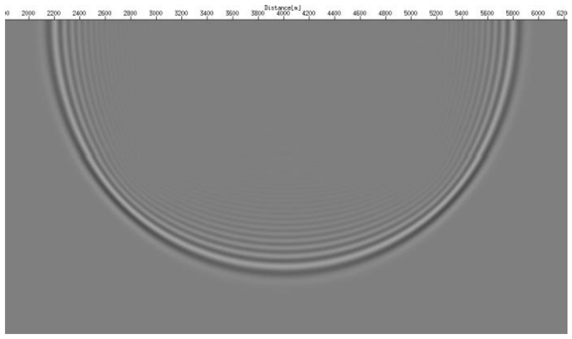


Figure 2: Wave front representation using a spatial discretization of 2nd order.

The stability condition for the finite difference scheme is given by the following expression [Bordind and Slawinski, 1998]

$$\frac{C * \partial t}{\partial h} \leq \sqrt{\frac{\varepsilon_1}{\varepsilon_2}} \qquad (4)$$

where $\varepsilon_1$ is the sum of the absolute values of the coefficients for the temporal operator, *i.e.,* $\varepsilon_1 = |1| + |-2| + |1| = 4$, and $\varepsilon_2$ represents the sum of the absolute values of the coefficients of the 8th order representation of the spatial operator, $\nabla^2 P$. The stability condition is therefore

$$\frac{C_{max} * \partial t}{\partial h} \leq \sqrt{\frac{4}{\sum coef\ Order\ 8}} \qquad (5)$$

## Stencil data arrangement on a GPU

Because GPUs do not have an operating system to manage their resources, and they are asynchronous devices, it is difficult to prevent failures in accessing memory. Therefore, a strict and detailed data management should be used. Computing the wave field at each grid point is given by the stencil data arrangement shown in Figure 3.

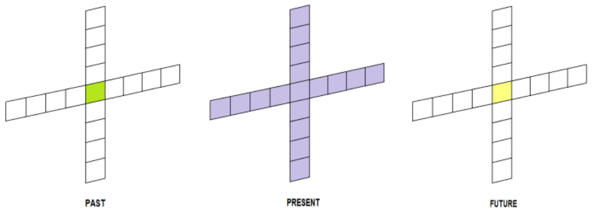


Figure 3: Stencil data arrangement. Each point $(\boldsymbol{x}, \boldsymbol{z})$ of future field (yellow) depends on a current field points (blue) in a given spatial order, and the point $(\boldsymbol{x}, \boldsymbol{z})$ of the past field (green).

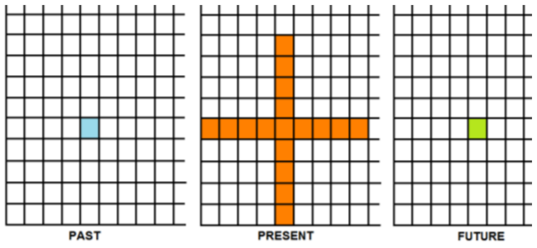


Figure 4: Stencil arrangement used to compute a green point in the future, without errors in the memory access.

To compute the green point shown in Figure 4, the stencil arrangement has access to the memory location of all the required points. However, if we want to compute the green point shown in Figure 5, access to data values represented by the red dots are not guaranteed. In the latter, wrong values of these points will cause a failure on the result of the future point of the field.

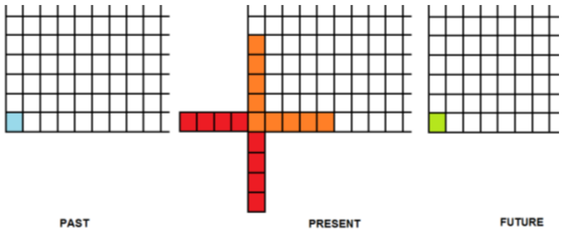


Figure 5: Stencil arrangement used to compute a green point in the wave field in the future, including errors in the memory access.

## Wave equation solution on a GPU

The main advantage of the implementation and execution of the RTM algorithm on a GPU, is the correct domain segmentation [Mckercher and Grossman, 2014]. Two structures in the CUDA APIs are used, *dimGrid* and *dimBlock*. These structures are used to tell the implementation the number of t*hreadsBlock* and Grids to be used by the GPU.

The sizes of velocity model change depending on the acquisition, and they are not commonly a multiple of the size of the Blocks. Therefore, the number of blocks is computed as

$$No.of\ Blocks = ceil\left(\frac{ModelSize - 1}{BlockSize} + 1\right) \quad (6)$$

where, *Modelsize* is the velocity model size, and *Blocksize* is the block size to be used by the GPU.

**CPML boundary conditions on a GPU**

In the numerical solution of the acoustic wave equation, the false reflections existing at the borders of the computational domain should be avoided. In this work, the Convolutional Perfectly Matched Layer (CPML) method proposed by [Pengliang *et al*., 2004] was implemented. The wave fronts near to the borders obtained using CPML are shown in Figure 6.

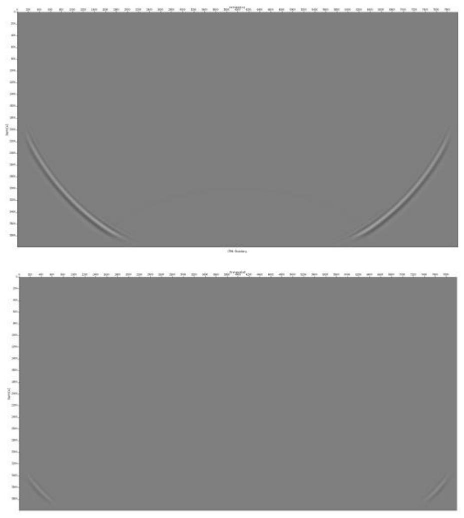


Figure 6: The wave front behavior in the borders on a constant velocity model (1.500 m/s) applying CPML.

**RTM 2D implementation on a GPU**

The RTM implementation on a GPU is divided into 4 subroutines. The first subroutine reads the seismic traces and the velocity field files. The second subroutine computes the stability parameters, initializes variables, allocates the required memory in the host and the device, and transfers data to the GPU to run the kernel. The third subroutine solves the source field or the receptors field, depending on a given flag. And the last subroutine computes the image condition.

The proposed GPU implementation requires the following software characteristics:

- Linux operating system
- GPU Nvidia compatible with CUDA
- Seismic Unix software
- Drivers and CUDA SDK 5.5 version or higher.

The data management of the RTM implementation contains 3 different areas: an area filled with zeros, a CPML and a model area. An example of the data management scheme is shown in Figure 7. The parameters used for the data management in Fig. 7 are Z=4, L=32, and Nx and Nz are the number of points in the velocity model.

Figure 7: Scheme used to compute the wave field. $N_x$ and $N_z$ are the number of points in the velocity model, Z=4 and L=32.

**Execution strategies**

Two strategies are proposed for the implementation of the RTM. The first strategy computes both the forward and backward fields and save them in hard drive. Afterwards, the image condition is computed by reading both fields from the hard drive memory. The output in this strategy are three files, two files having the source and receiver's fields, and one containing the map of reflectivity.

The second strategy is to avoid writing in hard drive. For this strategy, forward and backward fields are stored in RAM memory of the CPU. Then, the image condition is applied using both fields. The output of this strategy is the reflective map file only.

**Data flow dependency**

In order to make the RTM implementation compatible with Seismic Unix, the user can enter the parameters that are needed by the algorithm. Among those parameters, the user is required to enter:

- Velocity and seismic traces file.
- Velocity model size.
- Velocity model sampling parameter.
- Number of shots to be migrated.
- Frequency for source wavelet.
- Number of wavelength points.

A diagram that shows the data provided by the user and the parameters computed by the implementation, is shown in Figure 8.

The user assigns the value of the wavelet source frequency $\mathrm{Fq}$, and the number of points per wavelength $S$. The maximum and minimum velocity values $\mathrm{Cmax}$, and $\mathrm{Cmin}$, respectively, are obtained from the input velocity model. Also, the sampling time $\mathrm{dt\_traces}$ and the location of the source and receivers $\mathrm{dg} = \mathrm{dh\_trin}$ are obtained from the header´s seismic traces. From $\mathrm{Cmin}$, $\mathrm{S}$ and $\mathrm{Fq}$, the algorithm computes the parameter $\mathrm{dh\_mod}$. This parameter is used to guarantee the stability condition, according to Eq. (4). If the value $\mathrm{dh\_mod}$ is not equal to $\mathrm{dh\_tr}$, the velocity model should be spatially resampled using the given value of $\mathrm{dh\_mod}$. Likewise, if the parameter $\mathrm{dt\_sta}$ is not equal or less than the parameter $\mathrm{dt\_traces}$, the velocity model must be resampled in time.

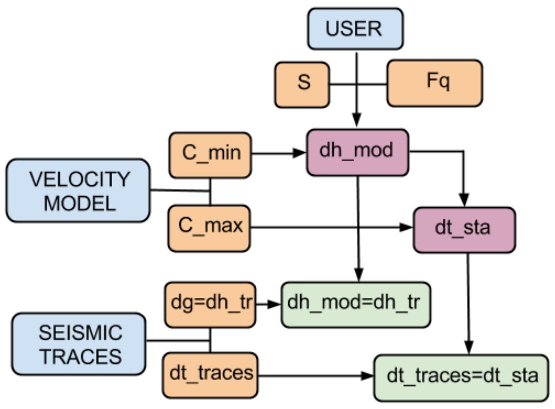


Figure 8: Data flow dependency.

## Numerical results of the RTM implementation

The two-dimensional model used to test the RTM algorithm implementation on a GPU is the Marmousi model. This complex model, shown in Figure 9, is traditionally used to test seismic data processing, and it was provided by the French Institute of Petroleum [Instituto Frances del Petroleo n.d.]. For all tests, the size of the Marmousi model was 1845x600. In addition, for all the tests, we used a Ricker source wavelet with frequency of 20 Hz, 10 points per wavelength, 5752 time steps, and a sampling time of 0.792ms.

Figure 10 shows the image obtained when 240 shots are migrated, and Figure 11 shows the migrated image when a Laplacian filter is used to eliminated the backpropagation effect.

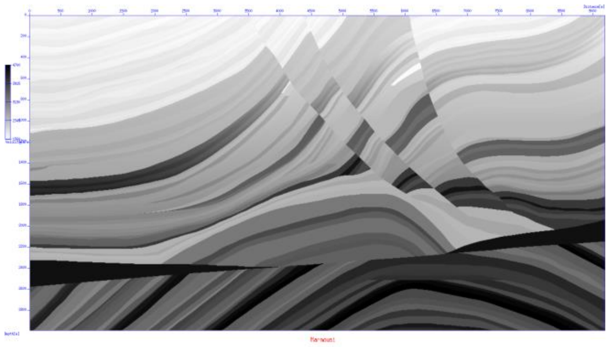


Figure 9: Marmousi exact velocity model

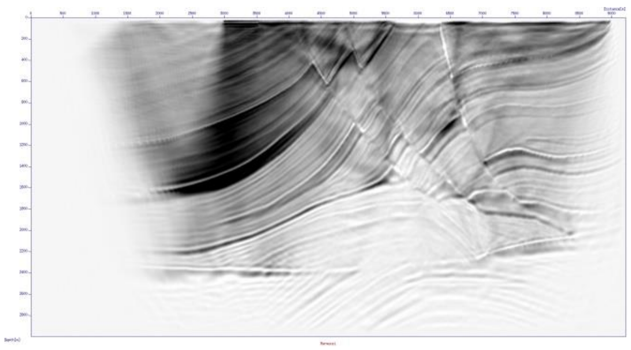


Figure 10: Migrated image in the GPU using 240 shots.

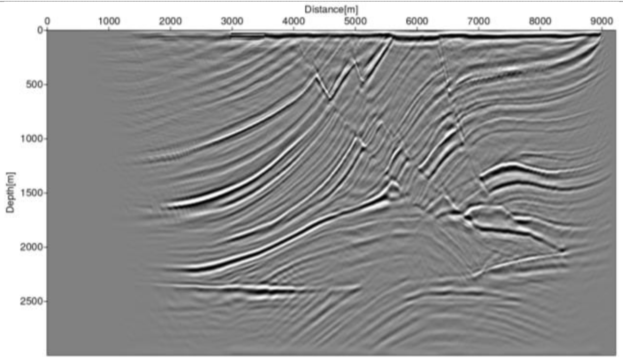
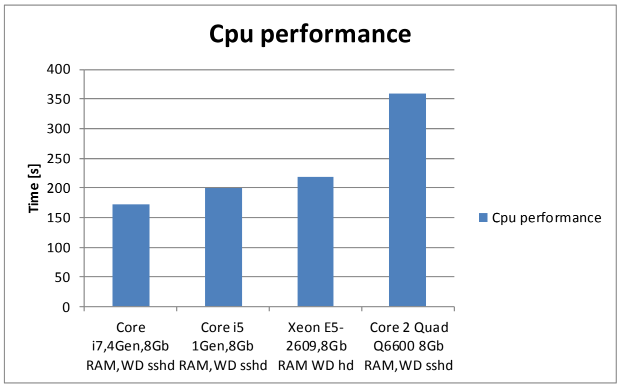


Figure 11: Migrated image with a Laplacian filter.



shows the execution time used to migrate one shot in four different CPU architectures. All the four devices have the same number of cores and same clock operating frequency. The RAM and hard disk memory size are the same in all devices. The difference among the tested devices is the type of hard drive. The resulting execution times are expected, except for the INTEL Xeon E5-2609. Even though the INTEL Xeon E5-2609 device has larger L3 cache memory than the other architectures, the hard drive attached to this device was slower, giving a very low performance in the execution time.

Figure 13 shows the comparison of the RTM execution time on the CPU devices with the best performance in previous test, and the RTM execution time on GPU. The GPU used in all tests was Nvidia GeForce GTX-580. The strategy tested in this experiment is the first strategy previously described, where the forward and backward fields are stored in hard drive. In this test, the GPU implementation gives approximately a 2.3x speedup when compared with the CPUs devices.

The third experiment is to test the two different strategies for the RTM implementation. The first strategy requires the transfer the forward and backward fields to the hard drive. In the second strategy, only CPU RAM memory is used to handle the forward and backward fields. In this test, both strategies were implemented on the GPU, and the execution times were measured. Figure 14 shows the execution times required depending on the type of strategy, and the type of CPU-RAM used.

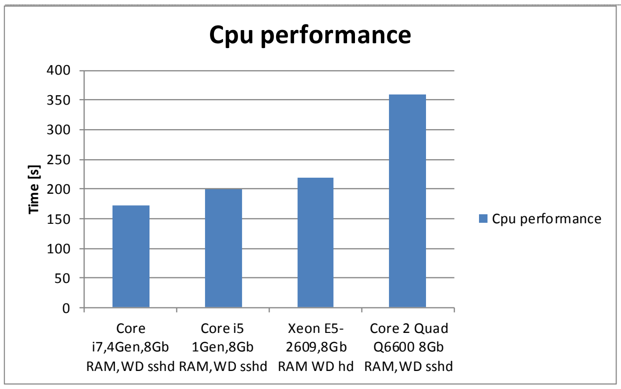


Figure 12: Execution time of the RTM algorithm in different types of CPU architecture; keeping constant the same RAM size and hard disk type unless otherwise stated.

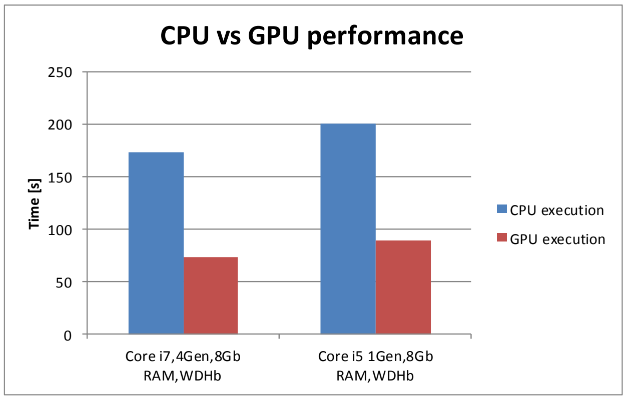


Figure 13: Execution time of the RTM implementation on two different CPU devices compared to the execution time of the implementation on the Nvidia GeForce GTX-580 GPU.

The last test consists on evaluating the granularity impact of the GPU implementation. Four different sizes of threads blocks were tested. Figure 15 shows the RTM execution time as a function of the size of the threads block.

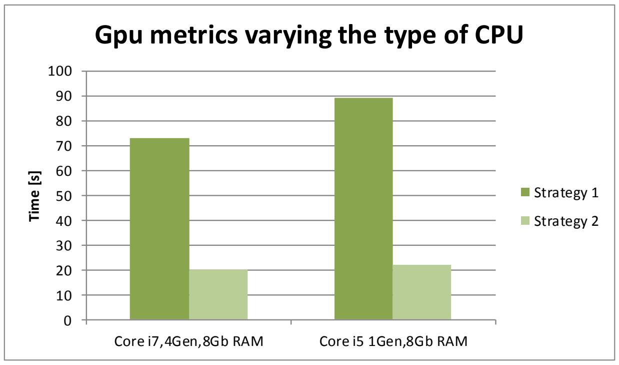


Figure 14: GPU execution time for the two different strategies used to implement the RTM algorithm. *Strategy 1* uses the hard drive device to store the fields, and *strategy 2* uses the CPU-RAM to store the fields.

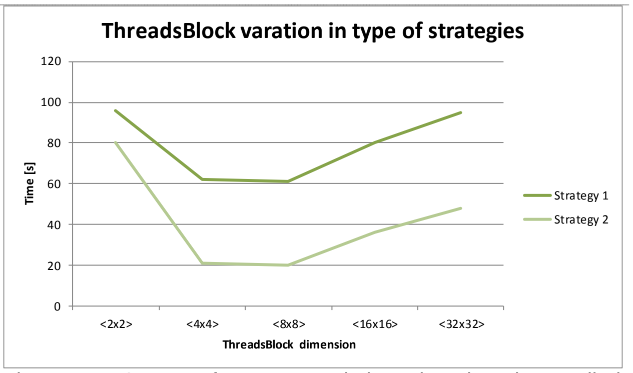


Figure 15: GPU performance variation changing the applied granularity.

## Conclusions

The RTM algorithm is of great interest for the oil and gas industry because it allows to reconstruct images of highly complex areas. However, due to the high computational cost of this algorithm, and its inherent parallelism, a GPU implementation of the RTM algorithm can be proposed.

When the GPU implementation uses the strategy of saving the forward and backward fields on a hard drive, the main bottleneck is the speed of hard drive rather than the type of processor. SSD disks are faster than hybrid SSHD or HD disks.

The execution time of the algorithm on one GPU is always faster compared to the execution time on one CPU core, even when the fields are transferred from and to the hard disk. Since the bottleneck is storage, the strategy of saving the fields in the CPU-RAM will be better than storing the fields in the hard drive.

The speedup achieved by the implementation on a GPU when the CPU-RAM is used is approximately 4 times. On the other hand, the speedup achieved by the implementation on a GPU is only 2.3 times when the fields are stored on disk.

Unfortunately since the field sizes exceed the amount of memory inside the GPU, storing the fields inside the memory hierarchy of the GPU is impossible.

## Acknowledgements

This work was supported by the Colombian Oil Company ECOPETROL and COLCIENCIAS, as a part of the research project grant No. 0266-2013

## References

Billete, Frederic, Sverre Brandsberg, and Dahl. "The 2004 BP velocity benchmark." 2005.

Boarding, Phillip R. "Finite Difference Modeling - Nearly Optimal Sponge Boundary Conditions." 2004.

Bording, Lines P, and R L Slawinski. "A Recipe For Stability Analysis Of Finite Difference Wave Equation Computations." 1998.

Fornberg, Bengt. "Generation of Finite Difference Formulas on Arbitrarily Spaced Grids. Mathematics of Computation." 1988.

*Instituto Frances del Petroleo.* n.d. http://www.ifp.fr/IFP/en/aa.htm (accessed Abrii 20, 2014).

Mckercher, Cheng, and Grossman. *Professional CUDA C Programming.* 2014.

NVIDIA. "White Paper Nvidia Next Generation and Cuda Compute." *Cuda Compute Architecture: Fermi Generation. Pages 1-22.* n.d. http://www.nvidia.com (accessed Enero 2014).

Pengliang, Yang, Gao Jinghuai, and Wang Baoli. "RTM using effective boundary saving: A staggered grid GPU implementation." 2004.

Vivas, Flor Alba, and Jesus David Cataño. *Análisis de la estabilidad, dispersión numérica y costo computacional de la migracion RTM de la ecuación de onda acustica en dos dimensiones.* 2011.