Reverse Time Migration on a GPU Cluster Using the Seismic Data Parallelism Strategy

Ivan Obregon [1], William Salamanca [1], Ana B. Ramirez [1].
[1] Universidad Industrial de Santander, Colombia.

Abstract

The seismic migration is one of the seismic processing stages employed by the oil and gas industry to generate subsurface images. This process is responsible for relocating the recorded seismic events to its correct spatial position and collapse the diffractions to their scattering points. Reverse-Time Migration (RTM) is one of the most common methods because it generates subsurface images with high quality in scenarios with complex structures. However, the method implies a high computational cost because it uses the solution of the wave equation to find the source wavefield, increasing the runtime. In this work, we propose a Reverse-Time Migration implementation using a GPU cluster, by taking advantage of the independence of the seismic data, more specifically the shots gathers acquired in the field. The proposed strategy consists in split the data to be processed in different GPUs using Message Passing Interface (MPI). Then, each GPU independently applies the RTM method to its corresponding portion of the total data. The final migrated image is generated by adding the results obtained from each GPU. The advantage of this strategy is its scalability, because the performance can be improved by adding more hardware (GPUs). We tested the method by using the synthetic Marmousi II model, obtaining a speed up factor of 5.61 when 3 nodes are used in comparison to the implementation in a single GPU.

Introduction

In the oil and gas industry, the seismic exploration is a fundamental pillar in their functioning, due to the high demand for crude oil to satisfy the energy, transport or other financial markets. The seismic exploration of oil and gas is composed of some stages as: acquisition, inversion, seismic migration, analysis and interpretation. The seismic migration is responsible for generating the final subsurface image that will be used for its posterior analysis and interpretation.

Currently, different seismic migration techniques are available: One-Way Wave Equation Migration (OWWEM) and the Reverse-Time Migration (RTM). OWWEM is commonly used due to its low computational cost and acceptable degree of accuracy. On the other hand, RTM uses the two-way wave equation, which provides a better accuracy in the migrated final images for areas of complex geology, but it has a high computational cost (Araya-Polo et al. (2008)).

RTM was introduced by (Baysal et al. (1983)) and it is based in the exploding reflector model to recover the amplitudes at zero time which denote the location and strength of the reflectors. Because it uses the two-way wave equation to model the wave propagation, then its computational cost in terms of memory and execution time is high.

Some authors are investigating new techniques to reduce these computing times. One of them is based on using Graphics Processing Units (GPUs) because it is a parallel programming paradigm that allows execute many task at the same time (Foltinek et al. (2009)) (Panetta et al. (2009)) (Amado et al. (2015)). Typically, these GPUs are programmed in CUDA-C language and are allocated in clusters, where the GPUs have a communication protocol, allowing the implementation in multiple GPUs.

In this paper, we present a RTM technique implementation that uses a GPU cluster, taking advantage of the independence of the seismic data. We use MPI-CUDA and CUDA-C implementation to reduce compute times approximately to the number of GPUs used.

Methodology

RTM Theory

The RTM algorithm has three principal steps: forward propagation, backward propagation and the imaging condition. These processes are applied for each shot obtained during the acquisition, and they generate a portion of the final image. A sum of partial images is performed to obtain the final subsurface image.

In the forward propagation stage, the propagation of a point source through a known medium is computed obtaining the source wavefield (*SF*). In this work, the source wavelet propagation was modeled using the acoustic wave equation in an isotropic medium with variable density, which is defined by

$$\frac{1}{\rho(x,z)}\frac{\partial^2 P(x,z,t)}{\partial t^2} = c(x,z)^2 \cdot \left(\frac{\partial^2 P(x,z,t)}{\partial x^2} + \frac{\partial^2 P(x,z,t)}{\partial z^2}\right) + src(x,t), \quad (1)$$

where *P(x,z,t)* is the pressure field, $src(x,t)$ is the point source, *t* is the time variable, *x* is the spatial inline variable, *z* is the spatial depth variable and *c(x,z)* is the velocity model. We used finite difference in time domain (FDTD), with 8^{th} spatial order and 2^{nd} time order, to approximate the solution of the acoustic wave equation.

Similarly, for the backward propagation, the same

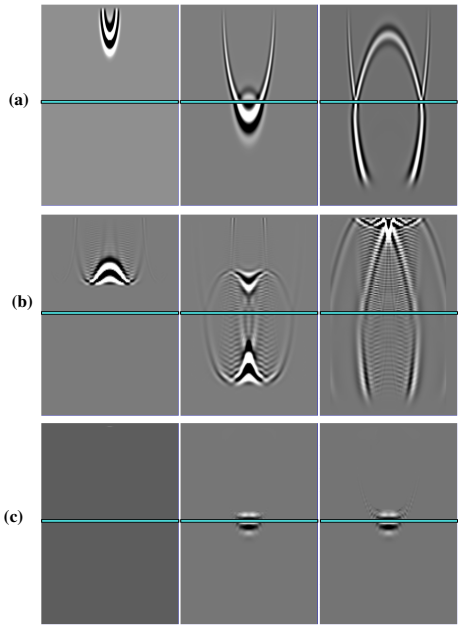

Figure 1: RTM algorithm. (a) Forward Propagation, (b) Backward propagation and (c) Imaging condition for times $t = 1$ (left column), 2.1 (middle column) and 3 (right column) seconds.

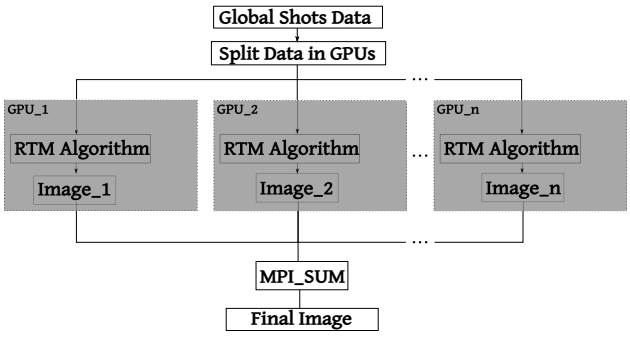

Figure 2: Behaivor of the RTM algorithm. (a) Forward Propagation, (b) Backward propagation and (c) imaging condition for times 1, 2.1 and 3 seconds.

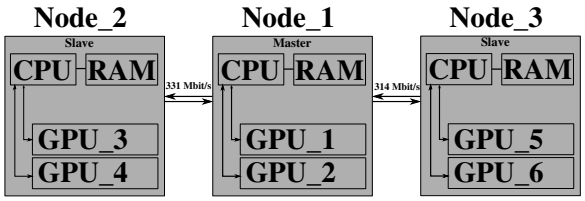

Figure 3: GPU Cluster of the implementation.

procedure is repeated in backwards time, i.e., start from the traces obtained at the surface level, named observed data, as sources and perform the backpropagation in time. The backward propagation is called the receivers wavefield (*RF*). When both fields are computed, an imaging condition is performed which is given by the cross-correlation operation described by

$$\sum_t SF(x,z,t) \cdot RF(x,z,t) \tag{2}$$

The Figure 1 shows the behavior of the three steps of the algorithm, for times $t = 1$ s (left), $t = 2.1$ s (center) and $t = 3$ s (right) for (a) the source wavefield and (b) the receivers wavefield. The imaging condition is applied to generate the subsurface image in (c) for a two-layer model.

The RTM implementation strategy was coded in a cluster GPU using C and CUDA-C languages with libraries of MPI (Message Passing Interface) standard. The following section describes the implementation strategy.

Seismic Data Parallelism strategy

The strategy was implemented taking into account the independence of the shots, such that we divide the global data in equal parts, generating and processing each shot in different GPUs. We use standard MPI to communicate and control the process between the GPUs. Figure 2 shows the diagram of the proposed strategy.

In order to control the distribution of the shots for each GPU, we used the standard MPI. MPI is a library developed to divide the CPU's resources into ranks, these ranks can execute task in parallel to make process more efficiently or create communication between devices (Gropp et al. (1999)).

For the implementation, we used a GPU cluster with 3 nodes, where each node has a CPU and RAM with 2 GPUs. We took the Node 1 as master (see Figure 3), which is in charge of distributing the information and the tasks to the other nodes and to itself. The master node controls the others nodes using the MPI ranks; therefore, exist a master rank. This master rank loads the models and the shots to distributes to the others ranks, i.e. all ranks working have a copy of the files necessary to work. Each node contains 2 ranks (same as a number of GPUs) except the master node which has 3 ranks.

The principal benefit of this strategy is their scalability, i.e., if we want to improve the performance, we can add more hardware, in this case, more GPUs or nodes to the cluster.

Example

The model used to test the proposed GPU parallel implementation was the Marmousi II, having a size of 12.5 km wide and 4.4 km deep as is shown in Figure 4. The model was discretized using a grid of *25x25* m such that the the number of grid points is given by $nx = 501 \times nz = 176$. We used 70 shots with 421 receivers per shot.

The CPML zone was implemented according to (Pasalic and McGarry (2010)). The parallel implementation was executed on a cluster GPU having 6 NVIDIA Tesla K40, such that 4 GPUs migrated 48 shots and 2 GPUs 22 shots to complete 70 shots.

In the proposed strategy, we process the shots in three different nodes, generating a three portion of the final image. These images are shown in the Figure 5.

Figure 6 shows the final migrated image obtained when 70 shots are used and Figure 7 shows the migrated image

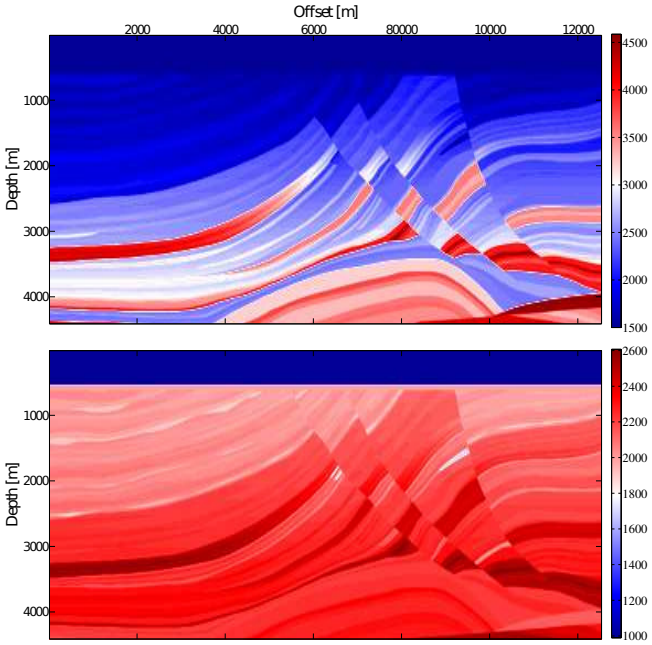

Figure 4: Velocity in $[m/s]$ (top) and density in $[kg/m^3]$ (bottom) model for the Marmousi II model.

when a Laplacian filter is applied to the final image such that the backpropagation effect is eliminated.

Results

We evaluate the performance of the cluster implementation by measuring the time spent by a single GPU, and using such value as reference value (t_{ref}) in the speedup computation given in Equation 3. Also, we executed the proposed strategy for 2 (1 node), 4 (2 nodes) and 6 (3 nodes) GPUs ($t_{strategy}$) in Equation 3, to calculate the speed up factor and the total memory used in the GPU and the CPU. The speed up factor is defined by

$$SpeedUp = \frac{t_{ref}}{t_{strategy}}. \tag{3}$$

For the GPU memory (in MiB) required by the proposed strategy is

$$GPU_{mem} = [18 \cdot (nx \cdot nz) + 2 \cdot (nx + nz) + nt + 2 \cdot (nx \cdot nt \cdot spgpu)] \cdot \frac{4}{2^{20}}. \tag{4}$$

For the CPU memory in the master rank (in MiB) is given by

$$CPU_{memMR} = [5 \cdot (nx \cdot nz) + 2 \cdot ns + nt + (nx \cdot nt) + (nx \cdot nt \cdot ns)] \cdot \frac{4}{2^{20}}. \tag{5}$$

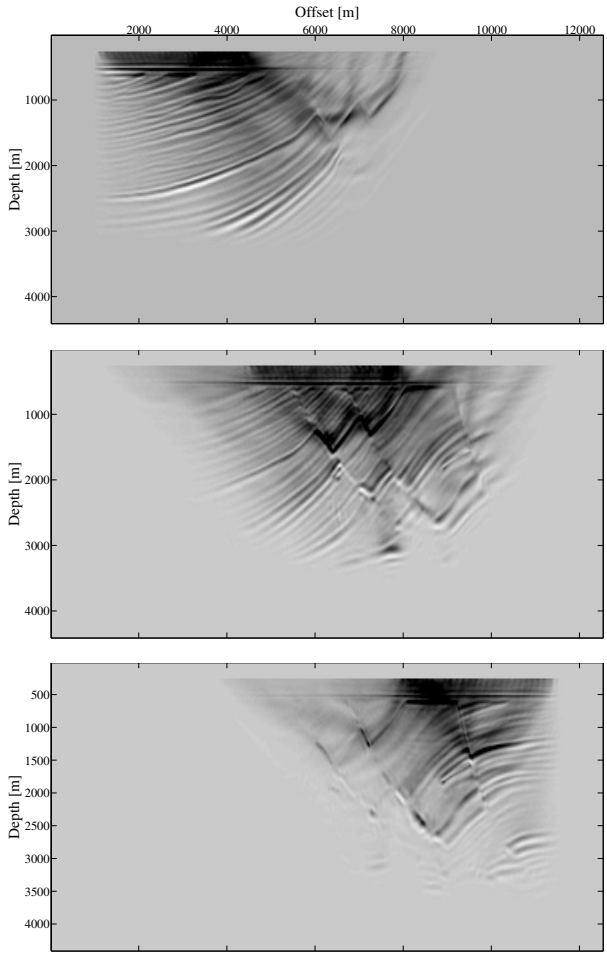

Figure 5: Image migrated of Node 1 (top), Node 2 (middle) and Node 3 (bottom).

And the CPU memory for the other ranks (in MiB) is obtained by

$$CPU_{memOR} = [5 \cdot (nx \cdot nz) + 2 \cdot ns + nt + (nx \cdot nt) + (nx \cdot nt \cdot spgpu)] \cdot \frac{4}{2^{20}}, \tag{6}$$

where *nx* and *nz* is the dimension of the model in points, *nt* is the number of time steps of the propagation, *ns* is the number of shots and *spgpu* is the shots per GPU to process.

Figure 8 shows the speed up of the strategy, observing a positive value and greater than one when the number of GPUs increases. Ideally, the speeds up and number of GPUs might be the same but in practice we can see for 6 GPU its speed up is 5.61, due of between nodes exist a physical connection that causes a delay when the final image is created.

The memory used in the test was measured per node and per rank. Table 1 show the values of the amount of memory

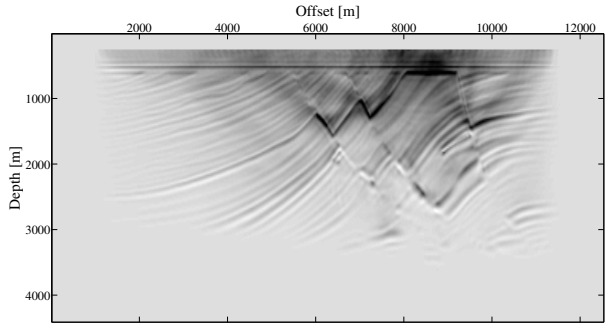

Figure 6: Migrated image in 6 GPUs using 70 shots.

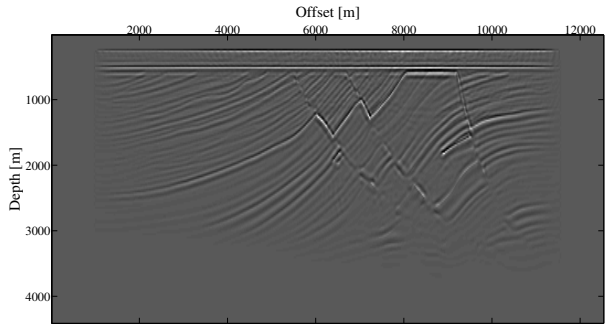

Figure 7: Migrated image with a Laplacian filter.

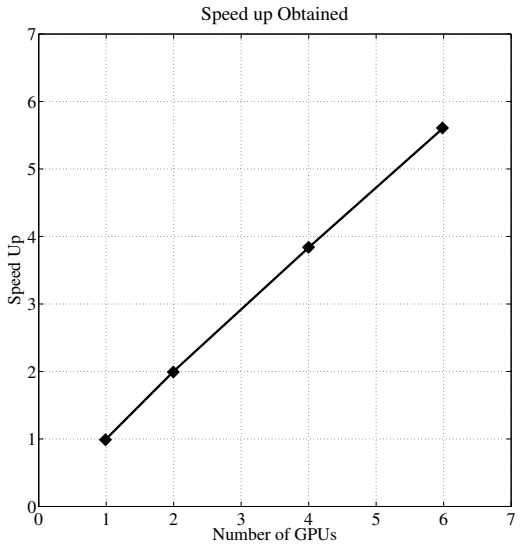

Figure 8: Speed up obtained for 1, 2, 4 and 6 GPUs.

Table 1: Memory used for one GPU.

Node	rank	GPU Memory [MiB]	CPU Memory [MiB]
1	1	3390.81	88.5
	2	0	0

Table 2: Memory used for the proposed strategy

Node	rank	GPU Memory [MiB]	CPU Memory [MiB]
1	1	-	473.3
	2	2616	89.8
	3	2616	89.8
2	4	2616	89.8
	5	2616	89.8
3	6	2602	82.9
	7	2602	82.9

used in the reference strategy and the table 2 shows the memory used in the proposed strategy.

The GPU memory used in the proposed strategy vary depends the number of shots, because no all ranks will be work with the same number of shots. Moreover, the rank 1 does not use a GPU memory because it only loads the data and distributes the shots to the other ranks. The CPU memory varies for the same reason of the GPUs memory except for the rank 1 that needs more memory to load the total data.

Conclusions

The computational cost of the proposed strategy shows that there is a significantly positive acceleration compared to the implementation in a GPU, decreasing the time 5.6 times. Also, we can observe will be exist a max of GPUs for this strategy due of exist an external communication between nodes for exchange of data. About the memory, exist a reduction of the GPU memory but increasing the total CPU memory.

The scalability that contributes to the process using the proposed strategy allows increase the performance of the method adding more hardware but taking into account that will be exist a maxima of GPUs to work.

Acknowledgements

This work is supported by Colombian Oil Company ECOPETROL and COLCIENCIAS as a part of the research project grant 0266 of 2013. The authors gratefully acknowledge the support of CPS research group of the Industrial University of Santander.

References

Amado, J., W. Salamanca, F. A. Vivas, and A. Ramirez, 2015, A gpu implementation of the reverse time migration algorithm: 14th International Congress of the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 3-6 August 2015, Brazilian Geophysical Society, 1016–1021.

Araya-Polo, M., F. Rubio, M. Hanzich, R. de la Cruz, J. M. Cela, and D. P. Scarpazza, 2008, High-performance seismic acoustic imaging by reverse-time migration on the cell/be architecture: ISCA2008.

Baysal, E., D. D. Kosloff, and J. W. Sherwood, 1983, Reverse time migration: Geophysics, **48**, 1514–1524.

Foltinek, D., D. Eaton, J. Mahovsky, P. Moghaddam, R.

McGarry, et al., 2009, Industrial-scale reverse time migration on gpu hardware: Presented at the 2009 SEG Annual Meeting, Society of Exploration Geophysicists.

Gropp, W., E. Lusk, and R. Thakur, 1999, Using mpi-2: Advanced features of the message-passing interface: MIT press.

Panetta, J., T. Teixeira, P. R. de Souza Filho, C. A. da Cunha Finho, D. Sotelo, F. M. R. da Motta, S. S. Pinheiro, I. P. Junior, A. L. R. Rosa, L. R. Monnerat, et al., 2009, Accelerating kirchhoff migration by cpu and gpu cooperation: Computer Architecture and High Performance Computing, 2009. SBAC-PAD'09. 21st International Symposium on, IEEE, 26–32.

Pasalic, D., and R. McGarry, 2010, Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations, *in* SEG Technical Program Expanded Abstracts 2010: Society of Exploration Geophysicists, 2925–2929.