[PS]Using Second-Order Adjoint State Methods in GPUS to Quantify Resolution on Full Waveform Inversions*

Sergio Abreo[1], Ana Ramirez[1], and Oscar Mauricio Reyes Torres[1]

Search and Discovery Article #42034 (2017)**
Posted March 13, 2017

*Adapted from poster presentation given at AAPG International Conference & Exhibition, Barcelona Spain, 2016

[1]Universidad Industrial de Santander, Santander, Colombia (abreosergio@gmail.com)

Abstract

The Hessian matrix in Full Waveform Inversion (FWI) allows quantifying resolution of the velocity model obtained. Although there are different ways to compute approximations of the Hessian matrix, such as BFGS, Newton, Gauss-Newton and Levenberg-Marquardt, our interest is to obtain the exact Hessian matrix. Particularly, we use the Second Order Adjoint State Method (SOASM) to obtain the Hessian matrix-vector products. This work makes use of Graphical Processing Units (GPUs) giving the inherent parallelism of the algorithm. In order to obtain the Hessian matrix-Vector products, it is necessary to perform four wave propagations using a finite differences scheme in time. In such scheme, the next layer is computed using information from the current layer and the previous layer. Furthermore, every spatial point of the next layer can be computed independently. In this work, this independence is exploited by an architecture with a high degree of parallelism such as the GPU. This work presents detailed interpretation and implementation of the SOASM theory to take advantage of GPU's architectures.

We use a section of the Marmousi velocity model in all the tests. Specifically, the size of the section is 5.25 km x 1.7 km (a grid of 210 x 68 points and spatial resolution of 25 m), which produces a Hessian matrix of 14280x14280 points. We compare the performance of two implementations: Intel Core I7, and Geforce GTX 860M. The CPU and GPU implementations compute a column of the Hessian matrix in 28.04 and 0.05956 seconds, respectively. It means a speedup factor of 470x by using the GPU. In our experiment, it is necessary to compute 14280 columns to obtain the complete Hessian matrix for one shot and one iteration. Assuming a linear performance of both implementations and extrapolating the measured times, we find a lower bound for both implementations using five shots per iteration. For the CPU implementation, the lower bound is 23.17 days whereas the GPU implementation has a lower bound of 1.18 hours.

Using second-order adjoint state method in GPUs to quantify resolution on Full Waveform Inversions

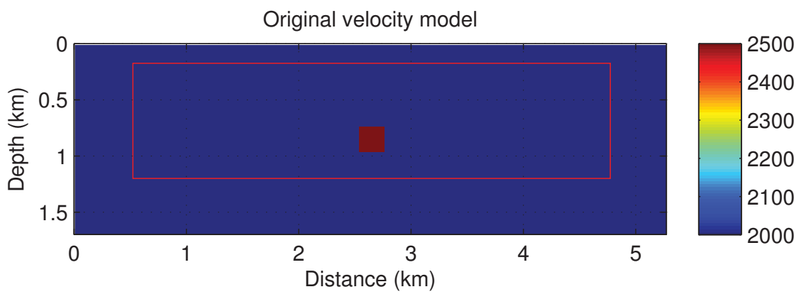
Sergio A. Abreo Ana B. Ramírez Oscar Reyes
Universidad Industrial de Santander, Bucaramanga, Colombia.

Introduction

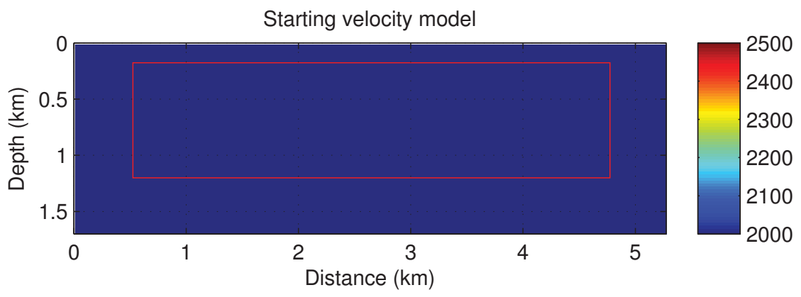
The Hessian matrix in Full Waveform Inversion (FWI) allows quantifying resolution of the estimated parameters. We study the computation of exact Hessian matrix, using the Second Order Adjoint State Method (SOASM) presented by Fichtner (2010) and Métivier et al. (2012). The SOASM method obtains the Hessian matrix-vector products separately, and therefore the method is highly parallelable. In this work, we propose a parallel implementation of the SOASM method using Graphical Processing Units (GPUs) to compute a full Hessian matrix of 7182 columns. We compare the resulting execution time of our implementation using GPUs, with the execution times required by a CPU implementation.

Synthetic model

The synthetic velocity model used to obtain the observed data (d_{obs}) is presented in Figure 1.a, with 211 points horizontally, 68 points in depth and a grid space in both dimensions of 25 m ($\Delta x = \Delta z = \Delta h = 25$ m). The background velocity is 2000 m/s and a diffracting area has a velocity of 2500 m/s. The diffracting element is a square with an area of 225×225 m^2 centered at the position $x = 2650$ m and $z = 850$ m.



(a) Original.



(b) Initial.

Figure : 1. Velocity models.

The modeled data ($d_{syn} = R(\tilde{u})$) is obtained from applying the first step of the SOASM's algorithm over the velocity model presented in Figure 1.b. The starting velocity model has the same dimensions, grid space and background velocity as the original velocity model, but it lacks the diffracting area.

The observed and modeled data were produced using one source located at ($x = 2650$ m, $z = 125$ m). The receivers were placed every 25 m, at the same depth (125 m), starting from the position 525 m up to the position 4775 m, having a total of 171 receivers. Each receiver records 3.5 s at 250 Sa/s ($\Delta t = 4 \times 10^{-3}$ s). The source is a Ricker wavelet with a central frequency of 3 Hz.

The non-natural boundaries in both velocity models (left, right and down of the red square) are modeled using Convolutional Perfectly Matched Layer (CPML), proposed by Pasalic et al. (2010), and explained in detail in Abreo et al. (2015). The CPML parameters of this implementation are $CPML_{area} = 20$ (grid points), $Lx = 500$ m, $R = 10^{-15}$ and $V_{max} = 4622$ m/s; where V_{max} is obtained from the Courant's stability criterion.

References

Abreo, S., A. Ramirez, O. Reyes, D. Abreo, and H. Gonzalez, 2015, A practical implementation of acoustic full waveform inversion on graphical processing units: CT&F - Ciencia, Tecnología y Futuro, 5–19.

Fichtner, A., 2010, Full seismic waveform modelling and inversion: Springer Science & Business Media.

Métivier, L., R. Brossier, S. Operto, J. Virieux, et al., 2012, Second-order adjoint state methods for full waveform inversion: Presented at the EAGE 2012-74th European Association of Geoscientists and Engineers Conference and Exhibition.

Pasalic, D., R. McGarry, et al., 2010, Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations: Presented at the 2010 SEG Annual Meeting, Society of Exploration Geophysicists.

SOASM's algorithm

Require: Compute $L(u, m) = f$. ▷ $f = src(x, z)$ = Normal source.

1: $\frac{1}{v^2(x,z)}\frac{\partial^2 \tilde{u}}{\partial t^2} = \frac{\partial^2 \tilde{u}}{\partial x^2} + \frac{\partial^2 \tilde{u}}{\partial z^2} + src(x, z)$ ▷ $\tilde{u}(x, z, 0) = 0$; $\frac{\partial \tilde{u}}{\partial t}(x, z, 0) = 0$

Require: Compute $\frac{\partial(L(u,m)-f)}{\partial m}$

2: $\frac{-2}{v^3(x,z)}\frac{\partial^2 \tilde{u}}{\partial t^2}$

Require: Compute $L(\lambda, m)^\dagger = -R^\dagger(R(\tilde{u}) - d_{obs})$ ▷ $-R^\dagger$ =flip up to down the array.

3: $\frac{1}{v^2(x,z)}\frac{\partial^2 \tilde{\lambda}}{\partial t^2} = \frac{\partial^2 \tilde{\lambda}}{\partial x^2} + \frac{\partial^2 \tilde{\lambda}}{\partial z^2} + flipud(d_{syn} - d_{obs})$ ▷ $\tilde{\lambda}(x, z, T) = 0$; $\frac{\partial \tilde{\lambda}}{\partial t}(x, z, T) = 0$

Require: Compute $\frac{\partial(L(\lambda,m)^\dagger + R^\dagger(R(\tilde{u})-d_{obs}))}{\partial m}$

4: $\frac{-2}{v^3(x,z)}\frac{\partial^2 \tilde{\lambda}}{\partial t^2}$

Require: Compute $\Phi_w = -\sum_{j=1}^{k} w_j \frac{\partial(L(u,m)-f)}{\partial m}$ ▷ w_j = Perturbation mask.

5: $\Phi_w = -\sum_{j=1}^{N_x \cdot N_z} w_j(-\frac{2}{v^3(x,z)}\frac{\partial^2 \tilde{u}}{\partial t^2})$

Require: Compute $L(\tilde{\alpha}, m) = \Phi_w$ ▷ $\tilde{\alpha}(x, z, 0) = 0$; $\frac{\partial \tilde{\alpha}}{\partial t}(x, z, 0) = 0$

6: $\frac{1}{v^2(x,z)}\frac{\partial^2 \tilde{\alpha}}{\partial t^2} = \frac{\partial^2 \tilde{\alpha}}{\partial x^2} + \frac{\partial^2 \tilde{\alpha}}{\partial z^2} - \sum_{j=1}^{N_x \cdot N_z} w_j(-\frac{2}{v^3(x,z)}\frac{\partial^2 \tilde{u}}{\partial t^2})$

Require: Compute $\frac{\partial(L(\tilde{\alpha},m)-\Phi_w)}{\partial m}$

7: $\frac{-2}{v^3(x,z)}\frac{\partial^2 \tilde{\alpha}}{\partial t^2} + \sum_{j=1}^{N_x \cdot N_z} w_j(\frac{6}{v^4(x,z)}\frac{\partial^2 \tilde{u}}{\partial t^2})$

Require: Compute $L(\tilde{\mu}, m) = -\sum_{j=1}^{N_x \cdot N_z} w_j(-\frac{2}{v^3(x,z)}\frac{\partial^2 \tilde{\lambda}}{\partial t^2}) - R^\dagger R\tilde{\alpha}$ ▷ $R\tilde{\alpha}$ = Seismic traces of step 6.

8: $\frac{1}{v^2(x,z)}\frac{\partial^2 \tilde{\mu}}{\partial t^2} = \frac{\partial^2 \tilde{\mu}}{\partial x^2} + \frac{\partial^2 \tilde{\mu}}{\partial z^2} - \sum_{j=1}^{N_x \cdot N_z} w_j(-\frac{2}{v^3(x,z)}\frac{\partial^2 \tilde{\lambda}}{\partial t^2}) - R^\dagger R\tilde{\alpha}$ ▷ Final conditions $\tilde{\mu}(x, z, T) = 0$; $\frac{\partial \tilde{\mu}}{\partial t}(x, z, T) = 0$.

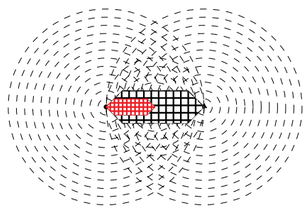
Require: Compute $\frac{\partial^2(L(u,m)-f)}{\partial m^2}$

9: $\frac{6}{v^4(x,z)}\frac{\partial^2 \tilde{u}}{\partial t^2}$

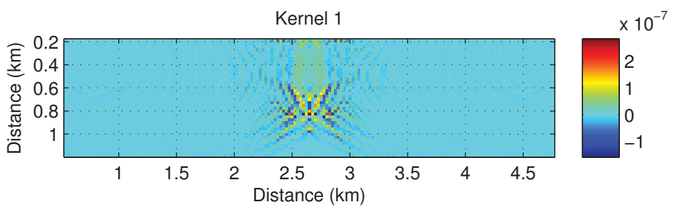
Require: $H(m)w_i = < \tilde{\mu}, \frac{\partial(L(u,m)-f)}{\partial m} > + < \tilde{\lambda}, \frac{\partial(L(\tilde{\alpha},m)-\Phi_w)}{\partial m} > + \sum_{j=1}^{N_x \cdot N_z} w_j < \tilde{\lambda}, \frac{\partial^2(L(u,m)-f)}{\partial m^2} >$ ▷ $< \cdots, \cdots >= \int_0^T$ inner-product$(\cdots, \cdots)dt$.

10: $H(m)w_i =< Step\ 8, Step\ 2 > + < Step\ 3, Step\ 7 > + \sum_{j=1}^{N_x \cdot N_z} w_j < Step\ 3, Step\ 9 >$

Results

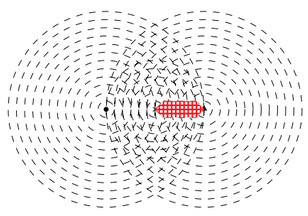


(a) Theoretical. Source ●, perturbation ■ and receiver ▲

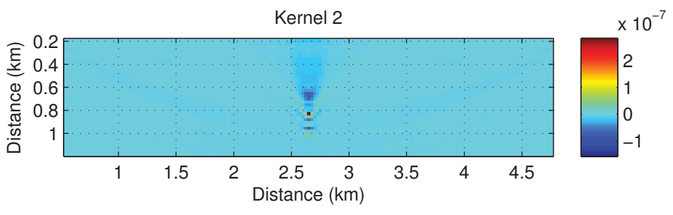


(b) Computed.

Figure : 2. First kernel. $< Step\ 8, Step\ 2 >$

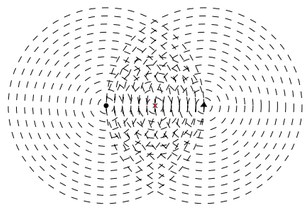


(a) Theoretical. Source ●, perturbation ■ and receiver ▲

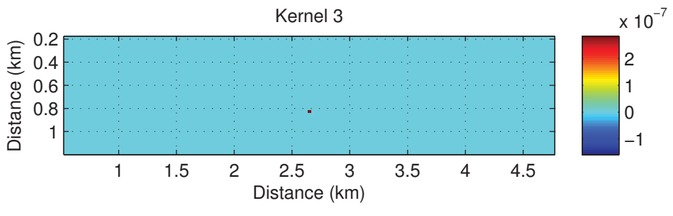


(b) Computed.

Figure : 3. Second kernel. $< Step\ 3, Step\ 7 >$

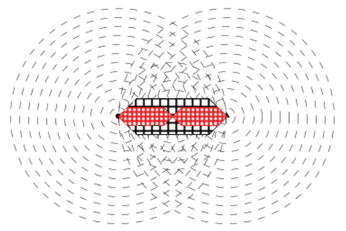


(a) Theoretical. Source ●, perturbation ■ and receiver ▲

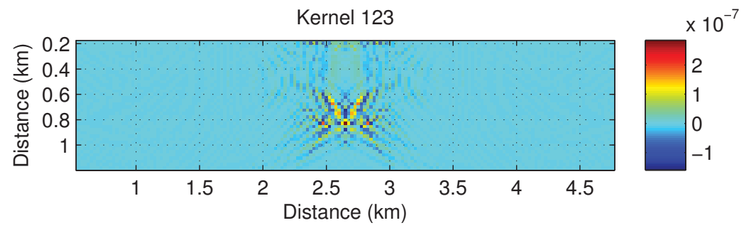


(b) Computed.

Figure : 4. Third kernel. $\sum_{j=1}^{N_x \cdot N_z} w_j < Step\ 3, Step\ 9 >$



(a) Theoretical. Source ●, perturbation ■ and receiver ▲



(b) Computed.

Figure : 5. Column of the Hessian matrix. $H(m)w_i$

Hardware and Results

The technical specifications of the CPU, and the GPU architectures are given in Tables 1, and 2.

CPU architecture	Details
Processor	Intel(R) Xeon(R) CPU E5-2609 v2
Frequency	2.5 GHz
CPU cores per socket	4
Sockets	2
L3 Cache size	10240 kB
RAM speed	1333 MHz
RAM size	256 GiB

Table : 1. Intel(R) Xeon(R) CPU E5-2609 v2.

Item	Tesla K40c
Stream Multiprocessors.(SMs)	15
Cuda cores per SMs	192
Number of Cuda Cores	2880
Blocks per SM	2
Threads per Block (Max.)	1024
Registers per Block (Max.)	65536
Global memory	12GB
L2 cache size	1572864 bytes
Shared memory per block	49152 bytes
Clk frequency	745MHz

Table : 2. Nvidia Tesla K40c architecture.

We obtain a speedup factor of 79x when our implementation is compared with a serial Ansi-C CPU implementation. Table 3 shows the execution time required by each architecture.

Architecture	Language	Performance (s)
Intel(R) Xeon(R) CPU E5-2609 v2	ANSI-C	14,78
Tesla K40c	CUDA-C	0.187155

Table : 3. Execution times of SOASM.

Conclusions

We conclude that current high-performance computing technologies make feasible to obtain the exact Hessian matrix using an FDTD implementation of the SOASM. Computing Hessian matrices are of interest in the Geophysics community to know the resolution of estimated velocity models, and it may be used for uncertainty quantification on parameters estimated from Full Waveform Inversions

Acknowledgements

This work is supported by Colombian Oil Company ECOPETROL and COLCIENCIAS as a part of the research project grant 0266 of 2013, and agrement 004 of 2014. The authors gratefully acknowledge the support of CPS research group of Industrial University of Santander and Colombian Petroleum Institute, ICP.

Contact us: abreosergio@gmail.com