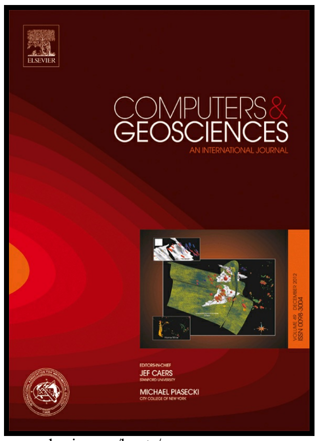

Time-domain seismic modeling in viscoelastic media for full waveform inversion on heterogeneous computing platforms with OpenCL

Gabriel Fabien-Ouellet, Erwan Gloaguen, Bernard Giroux

PII: S0098-3004(16)30781-6
DOI: http://dx.doi.org/10.1016/j.cageo.2016.12.004
Reference: CAGEO3882

To appear in: *Computers and Geosciences*

Time-domain seismic modeling in viscoelastic media for full waveform inversion on heterogeneous computing platforms with OpenCL

Gabriel Fabien-Ouellet[*], Erwan Gloaguen, Bernard Giroux

INRS-ETE, 490, Rue de la Couronne, Québec, QC, Canada G1K 9A9

[*]Email address: Gabriel.Fabien-Ouellet@ete.inrs.ca

Abstract

Full Waveform Inversion (FWI) aims at recovering the elastic parameters of the Earth by matching recordings of the ground motion with the direct solution of the wave equation. Modeling the wave propagation for realistic scenarios is computationally intensive, which limits the applicability of FWI. The current hardware evolution brings increasing parallel computing power that can speed up the computations in FWI. However, to take advantage of the diversity of parallel architectures presently available, new programming approaches are required. In this work, we explore the use of OpenCL to develop a portable code that can take advantage of the many parallel processor architectures now available. We present a program called SeisCL for 2D and 3D viscoelastic FWI in the time domain. The code computes the forward and adjoint wavefields using finite-difference and outputs the gradient of the misfit function given by the adjoint state method. To demonstrate the code portability on different architectures, the performance of SeisCL is tested on three different devices: Intel CPUs, NVidia GPUs and Intel Xeon PHI. Results show that the use of GPUs with OpenCL can speed up the computations by nearly two orders of magnitudes over a single threaded application on the CPU. Although OpenCL allows code portability, we show that some device-specific optimization is still required to get the best performance out of a specific architecture. Using OpenCL in conjunction with MPI allows the domain decomposition of large models on several devices located on different nodes of a cluster. For large enough models, the speedup of the domain decomposition varies quasi-linearly with the number of devices. Finally, we investigate two different approaches to compute the gradient by the adjoint state method and show the significant advantages of using OpenCL for FWI.

Keywords

OpenCL; GPU; Seismic; Viscoelasticity; Full waveform Inversion; Adjoint state method;

1. Introduction

In recent years, parallel computing has become ubiquitous due to a conjunction of both hardware and software availability. Manifestations are seen at all scales, from high-performance computing with the use of large clusters, to mobile devices such as smartphones that are built with multicore Central Processing Units (CPU) (Abdullah and Al-Hafidh, 2013). Graphics processing units (GPU) bring this trend to the next level, packing now up to several thousand cores in a single device. Scientific simulations have benefited from this technology through general-purpose processing on graphics processing units and, for certain applications, GPUs can speed up calculation over one or two orders of magnitude over its CPU counterpart. This has caused a surge in the use of GPUs in the scientific community (Nickolls and Dally, 2010, Owens, et al., 2008), with applications ranging from computational biology to large-scale astrophysics. Furthermore, GPUs are increasingly used in large clusters (Zhe, et al., 2004), and now several of the fastest supercomputers on earth integrate GPUs or accelerators (Dongarra, et al., 2015).

Nevertheless, GPUs are not fit for all kinds of scientific computations (Vuduc, et al., 2010). Potential gains from adopting GPUs should be studied carefully before implementation. In particular, the algorithm should follow the logic of the single-program multiple-data (SPMD) programming scheme, i.e. many elements are processed in parallel with the same instructions. In geophysics, and more precisely in the seismic community, GPU computing has been applied most successfully in modeling wave propagation with Finite-Difference Time-Domain (FDTD) schemes. Indeed, the finite-difference method is well suited to GPUs because the solution is obtained by independent computations on a regular grid of elements and follows closely the SPMD model (Micikevicius, 2009). For example, Michéa and Komatitsch (2010) show an acceleration by a factor between 20 to 60 between the single-core implementation of the FDTD elastic wave propagation and a single GPU implementation. Okamoto (2011) shows a 45 times speed-up with a single GPU implementation and presents a multi-GPU implementation that successfully parallelizes the calculation, although with a sub-linear scaling. Both Rubio, et al. (2014) and Weiss and Shragge (2013) present multi-GPU FDTD programs for anisotropic elastic wave propagation that shows the same unfavorable scaling behavior. Sharing computation through domain decomposition can be problematic mainly because the memory transfers between GPUs and between nodes are usually too slow compared to the computation on GPUs. GPU computing has also been applied successfully to the spectral element method (Komatitsch, et al., 2010), the discontinuous Galerkin method (Mu, et al., 2013) and reverse time migration (Abdelkhalek, et al., 2009), among others.

Nearly all of the seismic modeling codes written for GPUs have been implemented with the CUDA standard (Nvidia, 2007). CUDA allows easy programming on NVidia GPUs; however a CUDA program cannot run on devices other than NVidia GPUs. This can be problematic and is a leap of faith that NVidia devices are and will remain the most efficient devices for seismic modeling. Also, several clusters offer different types of GPU or, at least, a mix of GPU devices. Hence, the choice of CUDA limits the access to the full power of a cluster. On the other hand, OpenCL (Stone, et al., 2010) is an open programming standard capable of using most existing types of processors and is supported by the majority of manufacturers like Intel, AMD and Nvidia. On NVidia' GPUs, OpenCL performance is comparable to CUDA (Du, et al., 2012). Despite this advantage over CUDA, few published seismic modeling codes use OpenCL: Iturrarán-Viveros and Molero (2013) uses OpenCL in a 2.5D sonic seismic simulation, Kloc and Danek (2013) uses OpenCL for Monte-Carlo full waveform inversion and Molero and Iturrarán-Viveros (2013) perform 2D anisotropic seismic simulations with OpenCL.

Efficient seismic modeling is more and more needed because of the advent of full waveform inversion (FWI), see Virieux and Operto (2009) for an extensive review. FWI is the process of recovering the Earth (visco)-elastic parameters by directly comparing raw seismic records to the solution of the wave equation (Tarantola, 1984). Its main bottleneck is the numerical resolution of the wave equation that must be repeatedly computed for hundreds if not thousands of shot points for a typical survey. For large-scale multi-parameter waveform inversion, FDTD remains the most plausible solution for seismic modeling (Fichtner, 2011). In addition to forward seismic modeling, FWI requires the computation of the misfit gradient. It can be obtained by the adjoint state method (Plessix, 2006), which requires only an additional forward modeling of the residuals. Hence, it is based on the same modeling algorithm and the benefit of a faster FDTD code would be twofold.

In this study, we investigate the use of OpenCL for modeling wave propagation in the context of time domain FWI. The main objective is to present a scalable, multi-device portable code for the resolution of the 2D and 3D viscoelastic wave equation that can additionally compute the gradient of the objective function used in FWI by the adjoint state method. This paper does not go into specifics about the inversion process, as the gradient calculations calculated by our algorithm is general and can be used in any gradient-based optimization approach. The seismic modeling program, called SeisCL, is available under a GNU General Public License and is distributed over GitHub. The paper is organized in three parts. First, the equations for viscoelastic wave propagation, its finite-difference solution and the adjoint state method for the calculation of the misfit gradient are briefly discussed. In the second part, different algorithmic aspects of the program are presented in detail. The last section presents numerical results performed on clusters with nodes containing three types of processors: Intel CPUs, NVidia GPUs and Intel Xeon PHI. The numerical results show the validation of the code, the computational speedup over a single threaded implementation and the scaling over several nodes.

2. Theory

2.1 Finite difference viscoelastic wave propagation

FWI requires the solution of the heterogeneous wave equation. In this study, we consider the wave equation for an isotropic viscoelastic medium in two and three dimensions. We adopt the velocity-stress formulation in which the viscoelastic effects are modeled by L generalized standard linear solid (Liu, et al., 1976). The symbols used in this article and their meaning are summarized in table 1. The forward problem in 3D is a set of 9 + 6L simultaneous equations with their boundary conditions:

$$\partial_t v_i - \frac{1}{\rho}\partial_j \sigma_{ij} = f_{v_i}, \qquad \text{(1a)}$$

$$\partial_t \sigma_{ij} - \left[M\frac{(1+\tau_p)}{(1+\alpha\tau_p)} - 2\mu\frac{(1+\tau_s)}{(1+\alpha\tau_s)}\right]\partial_k v_k \delta_{ij} - \mu\frac{(1+\tau_s)}{(1+\alpha\tau_s)}\left(\partial_j v_i + \partial_i v_j\right) - r_{ijl}\delta_l = f_{\sigma_{ij}}, \qquad \text{(1b)}$$

$$\partial_t r_{ijl} + \frac{1}{\tau_{\sigma l}}\left[\left(M\frac{\tau_p}{(1+\alpha\tau_p)} - 2\mu\frac{\tau_s}{(1+\alpha\tau_s)}\right)\partial_k v_k \delta_{ij} + \mu\frac{\tau_s}{(1+\alpha\tau_s)}\left(\partial_j v_i + \partial_i v_j\right) + r_{ijl}\right] = 0, \qquad \text{(1c)}$$

$$v_i|_{t=0} = 0, \qquad \text{(1d)}$$

$$\sigma_{ij}\big|_{t=0} = 0, \qquad \text{(1e)}$$

$$r_{ij}\big|_{t=0} = 0, \qquad \text{(1f)}$$

$$n_j(\boldsymbol{s})\sigma_{ij} = 0, \qquad \text{(1g)}$$

where Einstein summation convention is used over spatial indices i, j, k and the Maxwell body indice l. Equation 1a comes from Newton's second law of motion. Equation 1b is the stress-strain relationship for the generalized standard linear solid model with L Maxwell bodies, which becomes the generalized Hooke's law when the attenuation is nil, i.e. when the attenuation levels τ_s and τ_p are set to zero. Equation 1c gives the variation of the so-called memory variables. Finally, the last four equations are the boundary conditions, respectively a quiescent past for velocities, stresses and memory variables and a free surface. Those equations are discussed in more details in several papers, see for example Carcione, et al. (1988), Robertsson, et al. (1994) and Blanch, et al. (1995).

The attenuation of seismic waves is often described by the quality factor, defined as the ratio between the real and imaginary parts of the seismic modulus (O'connell and Budiansky, 1978). In the case of a generalized standard linear solid, it is given by:

$$Q(\omega,\tau_{\sigma l},\tau)=\frac{1+\sum_{l=1}^{L}\frac{\omega^2\tau_{\sigma l}^2\tau}{1+\omega^2\tau_{\sigma l}^2}}{\sum_{l=1}^{L}\frac{\omega\tau_{\sigma l}\tau}{1+\omega^2\tau_{\sigma l}^2}}. \quad (2)$$

An arbitrary profile in frequency of the quality factor can be obtained by a least squares minimization over the relaxation times $\tau_{\sigma l}$ and the attenuation level τ. Usually, two or three Maxwell bodies are sufficient to obtain a relatively flat quality factor profile over the frequency band of a typical seismic source (Bohlen, 2002). The two variables involved have different influences on the frequency profile of the quality factor: $\tau_{\sigma l}$ controls the frequency peak location of the l^{th} Maxwell body, whereas τ controls the overall quality factor magnitude. In FWI, an attenuation profile in frequency is usually imposed on the whole domain (Askan, et al., 2007, Bai, et al., 2014, Malinowski, et al., 2011) and it is the magnitude of this profile that is sought. For this reason, $\tau_{\sigma l}$ is taken here as a spatially constant variable that is fixed before inversion, whereas τ is let to vary spatially and should be updated through inversion.

To solve numerically equation 1, we use a finite-difference time-domain approach similar to (Levander, 1988, Virieux, 1986). In time, derivatives are approximated by finite-difference of order 2 on a staggered grid, in which velocities are updated at integer time steps Δt and stresses and memory variables are updated at half-time steps in a classic leapfrog fashion. In space, the standard staggered grid is used. An elementary cell of the standard staggered grid is shown in Figure 1, summarizing the location of each seismic variable. The forward ${D_i}^+$ and backward ${D_i}^-$ differential operators of order $2N$ are given by:

$${D_i}^+f(i)=\frac{1}{\Delta x}\sum_{n=1}^{N}h_n[f(i+n)-f(i-n+1)], \quad (3a)$$

$${D_i}^-f(i)=\frac{1}{\Delta x}\sum_{n=1}^{N}h_n[f(i+n-1)-f(i-n)], \quad (3b)$$

where Δx is the spatial step and the h_n coefficients are obtained by Holberg's method (Holberg, 1987) which reduces dispersion compared to the Taylor coefficients. The choice of the forward or backward operator obeys the following simple rule: in the update equations (1a, 1b and 1c) of a variable "a", to estimate the derivative of a variable "b", the forward operator is used if variable "b" is located before variable "a" in the elementary cell (Figure 1) along the derivative direction. The backward operator is used otherwise. For example, the update formula for v_x is:

$$v_x^t = v_x^{t-1} + \frac{\Delta t}{\Delta x}\frac{1}{\rho}\left(D_x{}^{+}\sigma_{xx}^{t-1/2} + D_y{}^{-}\sigma_{xy}^{t-1/2} + D_z{}^{-}\sigma_{xz}^{t-1/2}\right). \quad (4)$$

The complete set of equations can be obtained with equations 1 and 3 and Figure 1. The reader is referred to the work of Bohlen (2002) for the complete list.

Finally, to emulate a semi-infinite half-space, artificial reflections caused by the edge of the model must be minimized. For this purpose, two types of absorbing boundaries are implemented: the convolutional perfectly matched layer (CPML) (Roden and Gedney, 2000) as formulated by Komatitsch and Martin (2007) for viscoelastic media and the dissipative layer of (Cerjan, et al., 1985). On the top of the model, a free surface condition is implemented by the imaging method of (Levander, 1988).

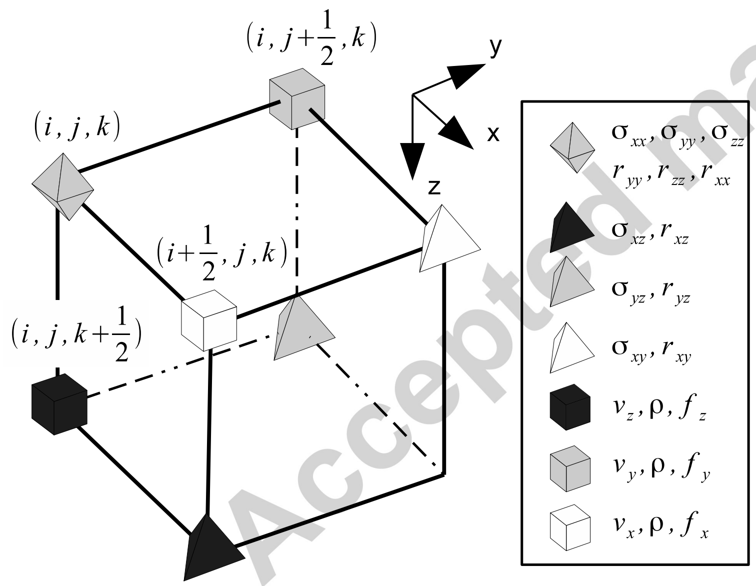

Figure 1 An elementary cell showing the node location for each seismic variable.

Table 1 Symbols used in this article

Symbol	Meaning
$v(\boldsymbol{x}, t)$	*Particle velocity*
$\sigma(\boldsymbol{x}, t)$	*Stress*
$f(\boldsymbol{x}, t)$	*Source term*
$r(\boldsymbol{x}, t)$	*Memory variable*

$\overleftarrow{\cdot}$	*Adjoint variable*
$\rho(\boldsymbol{x})$	*Density*
$M(\boldsymbol{x})$	*P-wave modulus*
$\mu(\boldsymbol{x})$	*Shear modulus*
$Q(\boldsymbol{x})$	*Quality factor*
$\tau_p(\boldsymbol{x})$	*P-wave attenuation level*
$\tau_s(\boldsymbol{x})$	*S-wave attenuation level*
$\tau_{\sigma l}$	*Stress relaxation time of the lth Maxwell body*
$\boldsymbol{d}$	*Recorded particle velocities*
T	*Recording time*
N_t	*Number of time steps*
J	*Cost function*

2.2 Full waveform inversion

The goal of full waveform inversion is to estimate the elastic parameters of the Earth based on a finite set of records of the ground motion $\boldsymbol{d_i}$, in the form of particle velocities or pressure. This is performed by the minimization of a cost function. For example, the conventional least-squares misfit function for particle velocity measurements is:

$$J(\rho, M, \mu, \tau_p, \tau_s) = \frac{1}{2}(\boldsymbol{S}(v_i) - \boldsymbol{d}_i)^T(\boldsymbol{S}(v_i) - \boldsymbol{d}_i)$$
$$+\tfrac{1}{2}\left(\boldsymbol{S}(\sigma_{ij}) - \boldsymbol{d}_{ij}\right)^T\left(\boldsymbol{S}(\sigma_{ij}) - \boldsymbol{d}_{ij}\right), \qquad (5)$$

where $\boldsymbol{S}(\cdot)$ is the restriction operator that samples the wavefield at the recorders' location in space and time. As 3D viscoelastic full waveform inversion may involve billions of model parameters, the cost function is usually minimized with a local gradient-based method. However, due to the sheer size of the problem, the computation of the gradient by finite difference is prohibitive. Lailly (1983) and Tarantola (1984) have shown that the misfit gradient can be obtained by the cross-correlation of the seismic wavefield with the residuals back propagated in time (see Fichtner, et al. (2006) for a more recent development). This method, called the adjoint state method, only requires one additional forward modeling. Based on the method of (Plessix, 2006), it can be shown (Fabien-Ouellet, et al., 2016) that the adjoint state equation for the viscoelastic wave equation of equation 1 is given by:

$$\partial_{t'}\overleftarrow{v}_i + \tfrac{1}{\rho}\partial_j\overleftarrow{\sigma}_{ij} = \tfrac{1}{\rho}\tfrac{\partial J}{\partial v_i}, \qquad (6a)$$

$$\partial_{t'}\overleftarrow{\sigma}_{ij} + \left[M\frac{(1+\tau_p)}{(1+\alpha\tau_p)} - 2\mu\frac{(1+\tau_s)}{(1+\alpha\tau_s)}\right]\partial_k\overleftarrow{v}_k\delta_{ij} + \mu\frac{(1+\tau_s)}{(1+\alpha\tau_s)}\left(\partial_j\overleftarrow{v}_i + \partial_i\overleftarrow{v}_j\right) + \overleftarrow{r}_{ijl}\delta_l =$$
$$\left[M\tfrac{(1+\tau_p)}{(1+\alpha\tau_p)} - 2\mu\tfrac{(1+\tau_s)}{(1+\alpha\tau_s)}\right]\tfrac{\partial J}{\partial\sigma_{kk}}\delta_{ij} + \mu\tfrac{(1+\tau_s)}{(1+\alpha\tau_s)}\left(1+\delta_{ij}\right)\tfrac{\partial J}{\partial\sigma_{ij}}, \qquad (6b)$$

$$\partial_{t'}\overleftarrow{r}_{ijl} + \frac{1}{\tau_{\sigma l}}\left[\left(M\frac{\tau_p}{(1+\alpha\tau_p)} - 2\mu\frac{\tau_s}{(1+\alpha\tau_s)}\right)\partial_k\overleftarrow{v}_k\delta_{ij} + \mu\frac{\tau_s}{(1+\alpha\tau_s)}\left(\partial_j\overleftarrow{v}_i + \partial_i\overleftarrow{v}_j\right) + \overleftarrow{r}_{ijl}\right] = 0, \qquad \text{(6c)}$$

$$\overleftarrow{v}_i|_{t'=0} = 0, \qquad \text{(6d)}$$

$$\overleftarrow{\sigma}_{ij}\Big|_{t'=0} = 0, \qquad \text{(6e)}$$

$$\overleftarrow{r}_{ij}\Big|_{t'=0} = 0, \qquad \text{(6f)}$$

$$n_j(\boldsymbol{s})\overleftarrow{\sigma}_{ij} = 0, \qquad \text{(6g)}$$

with $t' = T - t$. Comparing equations 1 and 6, we see that both sets of equations are nearly identical, the only difference being the sign of the spatial derivatives and the source terms (the terms involving the misfit function derivative). Hence, the adjoint solution for the viscoelastic wave equation can be computed with the same forward modeling code, with the source term taken as the data residuals reversed in time and with an opposite sign for the spatial derivatives. This allows using the same modeling code for the forward and adjoint problem, with only minor changes to store or recompute the forward and residual wavefields. Once both wavefields are computed, the gradient can be obtained by calculating their scalar product, noted here $\langle\cdot\,,\,\cdot\rangle$. The misfit gradient for density, the P-wave modulus, the P-wave attenuation level, the shear modulus and the S-wave attenuation level are given respectively by:

$$\frac{\partial J}{\partial \rho} = \langle\overleftarrow{v}_x\,,\partial_t v_x\rangle + \langle\overleftarrow{v}_y\,,\partial_t v_y\rangle + \langle\overleftarrow{v}_z\,,\partial_t v_z\rangle, \qquad \text{(7a)}$$

$$\frac{\partial J}{\partial M} = -c_M^1 P_1 + c_M^2 P_2\,, \qquad \text{(7b)}$$

$$\frac{\partial J}{\partial \tau_p} = -c_{\tau_p}^1 P_1 + c_{\tau_p}^2 P_2\,, \qquad \text{(7c)}$$

$$\frac{\partial J}{\partial \mu} = -c_\mu^1 P_3 + c_\mu^2 P_1 - c_\mu^3 P_4 + c_\mu^4 P_5 - c_\mu^5 P_2 + c_\mu^6 P_6\,, \qquad \text{(7d)}$$

$$\frac{\partial J}{\partial \tau_s} = -c_{\tau_s}^1 P_3 + c_{\tau_s}^2 P_1 - c_{\tau_s}^3 P_4 + c_{\tau_s}^4 P_5 - c_{\tau_s}^5 P_2 + c_{\tau_s}^6 P_6\,, \qquad \text{(7e)}$$

with

$$P_1 = \left\langle\overleftarrow{\sigma}'_{xx} + \overleftarrow{\sigma}'_{yy} + \overleftarrow{\sigma}'_{zz}\,,\partial_t\left(\sigma'_{xx} + \sigma'_{yy} + \sigma'_{zz}\right)\right\rangle,$$

$$P_2 = \left\langle\overleftarrow{R}_{xxl} + \overleftarrow{R}_{yyl} + \overleftarrow{R}_{zzl}, (1 + \tau_{\sigma l}\partial_t)\left(r_{xxl} + r_{yyl} + r_{zzl}\right)\right\rangle,$$

$$P_3 = \left\langle\overleftarrow{\sigma}'_{xy}\,,\partial_t\sigma'_{xy}\right\rangle + \left\langle\overleftarrow{\sigma}'_{xz}\,,\partial_t\sigma'_{xz}\right\rangle + \left\langle\overleftarrow{\sigma}'_{yz}\,,\partial_t\sigma'_{yz}\right\rangle,$$

$$P_4 = \left\langle \overleftarrow{\sigma}'_{xx}\,, \partial_t\left((N_d-1)\sigma'_{xx} - \sigma'_{yy} - \sigma'_{zz}\right)\right\rangle$$
$$+\left\langle \overleftarrow{\sigma}'_{yy}\,, \partial_t\left((N_d-1)\sigma'_{yy} - \sigma'_{xx} - \sigma'_{zz}\right)\right\rangle$$
$$+\left\langle \overleftarrow{\sigma}'_{zz}\,, \partial_t\left((N_d-1)\sigma'_{zz} - \sigma'_{yy} - \sigma'_{xx}\right)\right\rangle,$$

$$P_5 = \langle \overleftarrow{R}_{xyl}\,, (1+\tau_{\sigma l}\partial_t) r_{xyl}\rangle$$
$$+\langle \overleftarrow{R}_{xzl}\,, (1+\tau_{\sigma l}\partial_t) r_{xzl}\rangle$$
$$+\langle \overleftarrow{R}_{yzl}\,, (1+\tau_{\sigma l}\partial_t) r_{yzl}\rangle,$$

$$P_6 = \left\langle \overleftarrow{R}_{xxl}\,, (1+\tau_{\sigma l}\partial_t)\left((N_d-1)r_{xxl} - r_{yyl} - r_{zzl}\right)\right\rangle$$
$$+\left\langle \overleftarrow{R}_{yyl}\,, (1+\tau_{\sigma l}\partial_t)\left((N_d-1)r_{yyl} - r_{xxl} - r_{zzl}\right)\right\rangle$$
$$+\left\langle \overleftarrow{R}_{zzl}\,, (1+\tau_{\sigma l}\partial_t)\left((N_d-1)r_{zzl} - r_{yyl} - r_{xxl}\right)\right\rangle \qquad (7f)$$

where $\overleftarrow{R}_{ijl} = \int_0^T \overleftarrow{r}_{ijl}dt$, $\sigma'_{ij} = \sigma_{ij} - \sum_l R_{ijl}$ and N_d is the number of dimensions (2 or 3). Coefficients c are given in the appendix. The misfit gradients for the P-wave modulus M and the P-wave attenuation level τ_p have the same structure and differ only by the coefficients that weight the scalar products. The same relationship exists between μ and τ_s.

In the time domain, the scalar product takes the form:

$$\langle a(t)\,, b(t)\rangle = \int_T a(t)b(t)\,dt, \qquad (8)$$

which is the zero-lag cross-correlation in time of the real-valued functions $a(\boldsymbol{x})$ and $b(\boldsymbol{x})$. When discretized in time, it is the sum of the product of each sample. Using Parseval's formula, the last equation can also be expressed in the frequency domain:

$$\langle a(t)\,, b(t)\rangle = \frac{1}{2\pi}\int_\omega A^*(\omega)B(\omega)\,d\omega, \qquad (9)$$

where $A(\omega)$ and $B(\omega)$ are the Fourier transform of the functions $a(t)$ and $b(t)$ and * indicates complex conjugation. The formulation in frequency can be used to perform frequency domain FWI (Pratt and Worthington, 1990) with a time-domain forward modeling code as done by Nihei and Li (2007). The frequency components of the seismic variables can be obtained with the discrete Fourier transform:

$$A(m\Delta f) = \Delta t \sum_{n=0}^{N_t-1} a(n\Delta t)\exp\left[-\frac{i2\pi mn}{N_t}\right], \qquad (10)$$

where a is the discrete function in time, A is the discrete function in the Fourier domain, Δt is the time interval, Δf is the frequency interval and m is the frequency label. The calculation of a frequency component with the discrete Fourier transform involves the sum of all the time samples weighted by a time varying function given by the complex exponential. In the FDTD scheme, the running sum can be updated at each time step for all or a selected number of frequencies (Furse, 2000). Because FDTD must be oversampled to remain stable (CFL condition), the discrete Fourier transform can be performed at a higher time interval to mitigate its computational cost, e.g. several time steps can be skipped in equation 10, up to the Nyquist frequency of the highest selected frequency. Also, to save memory and reduce computing time, only a handful of frequencies can be used during the inversion (Sirgue and Pratt, 2004).

Once the gradient is computed, different algorithms can be used to solve the inversion system, from the steepest descent to the full Newton method (Virieux and Operto, 2009). This issue is not the focus of this study. However, all of these local methods need at least the computation of the forward model and the misfit gradient, both of which are the main computational bottlenecks. Hence, a faster forward/adjoint program should benefit all of the local approaches of FWI.

2.3 Background on heterogeneous computing

Heterogeneous computing platforms have become the norm in the high-performance computing industry. Clusters generally include different kinds of processors (Dongarra, et al., 2015): the most common being CPUs, GPUs and Many Integrated Core (MIC), also known as accelerators. Those devices may possess different architecture and usually codes written for one type of device is not compatible with others. Writing a specific code for each type of processor can be tedious and non-productive. One solution is given by OpenCL (Stone, et al., 2010), an open standard cross-platform for parallel programming. OpenCL allows the same code to use one or a combination of processors available on a local machine. This portability is the main strength of OpenCL, especially with the actual trend of massively parallel processors. For the moment, it cannot be used for parallelization on a cluster, but can be used in conjunction with MPI.

Even though OpenCL allows the same code to be compatible with different devices, the programmer always has to make a choice with the initial design because code optimization can be very different for CPUs, GPUs or MICs architectures. The program presented in this study was written for the GPU architecture, which is arguably the most efficient type of processor available today for finite-difference algorithms. For a good summary of the concepts of GPU computing applied to seismic finite-difference, see (Michéa and Komatitsch, 2010). Essential elements to understand the rest of the article are presented in this section, using the OpenCL nomenclature.

A GPU is a device designed to accelerate the creation and manipulation of images, or large matrices, intended primarily for output to a display. It is separated from the CPU (host) and usually does not directly share memory. The set of instructions that can be accomplished on a GPU is different than on the CPU, and classical programming languages cannot be used. A popular application programming interface for GPUs is CUDA (Nvidia, 2007). However, CUDA is a closed standard owned by NVIDIA that can only be used with Nvidia GPUs. It is the main reason why OpenCL was favored over CUDA in this work.

In order to code efficiently for GPUs, it is important to understand their architecture. The smallest unit of computation is a work item (a thread in CUDA) and is executed by the processing elements (CUDA cores in the NVidia nomenclature). A single device can contain thousands of processing elements that execute the same control flow (instructions) in parallel on different data in the single instruction, multiple thread fashion. The processing elements are part of groups that are called compute units (thread blocks in CUDA). In NVidia devices, the compute units contain 32 consecutive processing elements. In OpenCL, the programmer sends the work items, organized into work groups, to be computed by the processing elements of a compute unit, located in a given device.

Several levels of memory exist in a GPU. This is schematized in Figure 2, in the context of a GPU cluster. First, each processing element has its own register (private memory), a limited in size but very fast memory. Second, inside each compute unit, threads share a low-latency memory, called the local memory. This memory is small, usually in the order of several kilobytes. The main memory, called global memory, is shared between all processing elements and is the place where the memory needed for the different kernels is located. Usually, this memory is not cached and is very slow compared to the local or private memory.

One of the most important aspects of GPU programming is the access to the global memory. Depending on the memory access pattern, read/write operations can be performed in a serial or a parallel fashion by the compute units. Parallel (coalesced) memory access is possible when a number of consecutive work items inside a work group performing the same instructions are accessing consecutive memory addresses. For most NVidia devices, consecutive work items, or what is called a warp, can read 32 floats in a single instruction when memory access is coalesced. With finite-difference codes, the number of instructions performed between the read/write cycles in global memory is fairly low, which means that kernels are bandwidth limited. The memory access pattern is then one of the main areas that should be targeted for optimization.

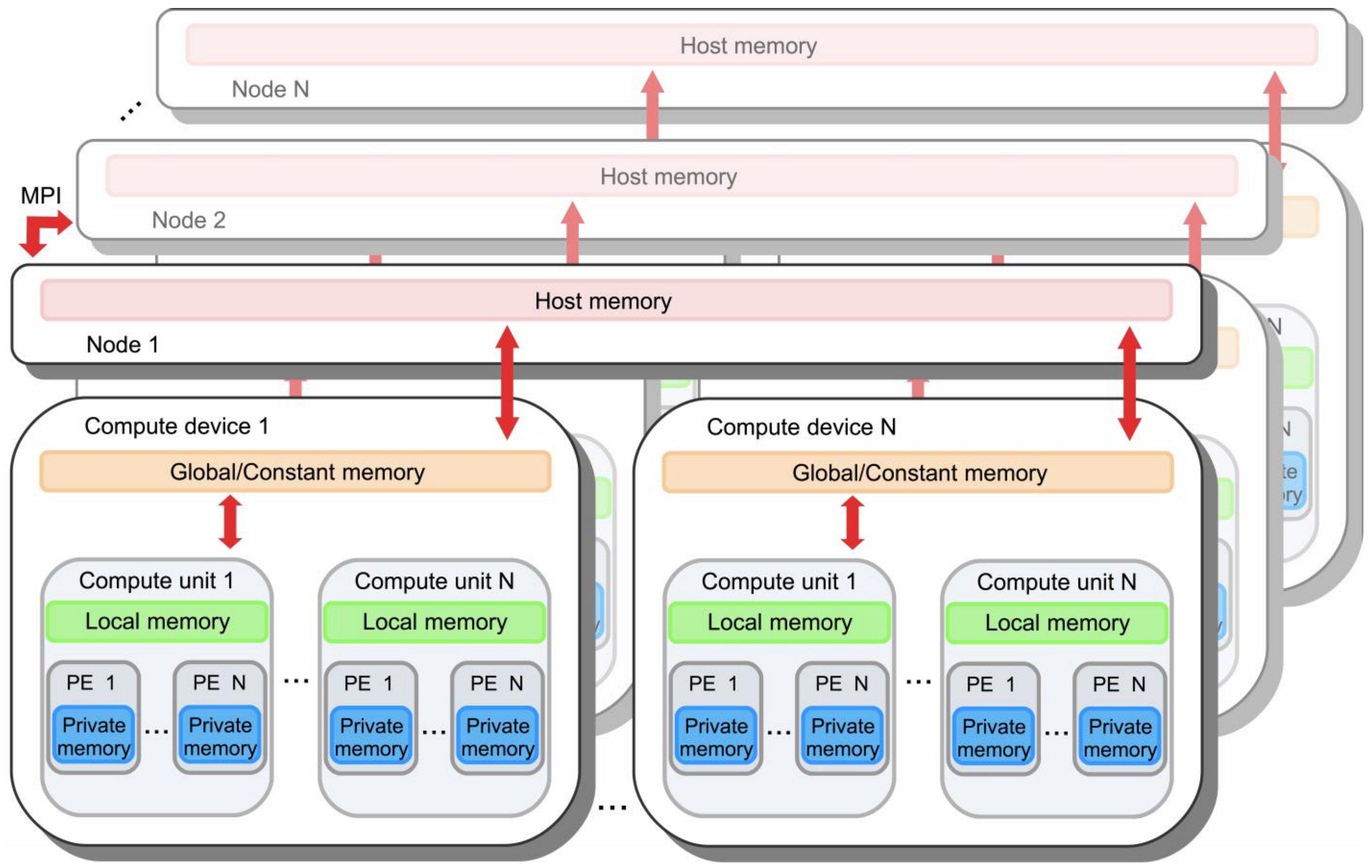

Figure 2 OpenCL memory diagram used in conjunction with MPI in the context of a cluster, inspired by (Howes and Munshi, 2014).

In practice, a program based on OpenCL is organized as follows, regardless of the type of processor used. First, instructions are given to the host to detect the available devices (GPUs, CPUs or accelerators) and connect them in a single computing context. Once the context is established, memory buffers used to transfer data between the host and the devices must be created. Then, the kernels are loaded and compiled for each device. This compilation is performed at runtime. The kernels are pieces of code written in C that contain the instruction to be computed on the devices. After that, the main part of the program can be executed, in which several kernels and memory transfers occur, managed on the host side by a queuing system. Finally, buffers must be released before the end of the program. Several OpenCL instances can be synchronized with the help of MPI, as shown in Figure 2.

3. Program structure

This section describes the implementation of the finite-difference algorithm for viscoelastic modeling and the calculation of the adjoint wavefield in an OpenCL/MPI environment. The program contains many kernels, and its simplified structure is shown in Algorithm 1.

This algorithm presents a typical gradient calculation over several seismic shots, on a parallel cluster where each node contains several devices. Its main features are discussed in the following sections.

Algorithm 1 Pseudo-code for the parallel computation of the gradient with the adjoint state method.

```
Initialize MPI
Initialize OpenCL
Initialize model grid
1.  for all groups in MPI do
2.    for all shots in group do
3.      for all nodes in group do
4.        for all devices in node do
5.          Initialize seismic grid $(v_i, \sigma_{ij}, r_{ij}, \overleftarrow{v}_i, \overleftarrow{\sigma}_{ij}, \overleftarrow{r}_{ij})$
6.          Execute time stepping on shot
7.          Compute residuals
8.          Execute time stepping on residuals
9.          Compute gradient
10.       end for
11.     end for
12.   end for
13. end for
```

3.1 Node and device parallelism

In order to take advantage of large clusters, we use the MPI interface to parallelize computations between the nodes of a cluster. A popular approach to parallelizing finite-difference seismic modeling is domain decomposition (Mattson, et al., 2004). It consists of dividing the model grid into subdomains that can reside on different machines. At each time step, each machine updates its own velocity and stress sub-grids. As the finite-difference update of a variable at a given grid point requires the values of other variables at neighboring grid points (see equations 3 and 4), values defined at grid points on the domain boundary must be transferred between adjacent domains at each time steps. This is depicted in Figure 3.

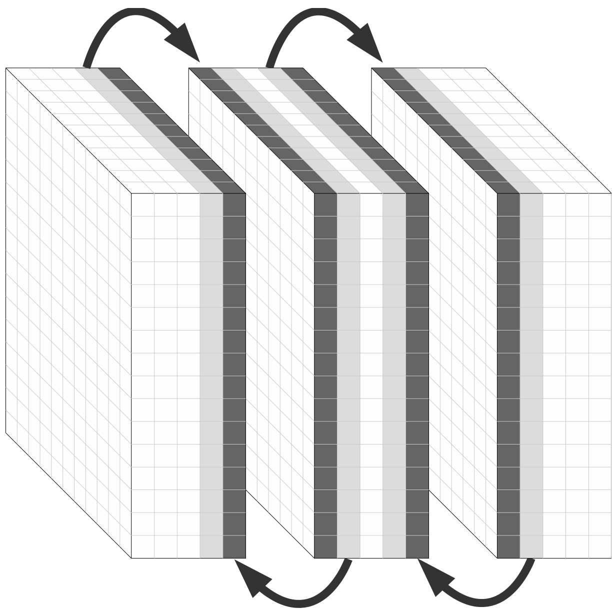

Figure 3 Domain decomposition for three devices for a finite-difference order of 2. Light gray cells are updated inside the device and transferred to the dark gray cells of the adjacent device.

Fast interconnects are needed for this memory transfer that occurs at each time step, otherwise the scaling behavior can become unfavorable. For example, Bohlen (2002) observes super-linear scaling for up to 350 nodes on a cluster with 450 Mb/s interconnects, but only linear scaling with up to 12 nodes on a cluster with 100 Mb/s interconnects. When using GPUs, not only transfers are needed between nodes, but also between the devices and the host. This dramatically worsens performance. For example, Okamoto (2011) observes a scalability between N and $N^{2/3}$. For this reason, we chose to implement two different parallelism schemes in addition to the inherent OpenCL parallelization: domain decomposition and shot parallelization.

Nodes of a cluster are first divided into different groups: within each group, we perform domain decomposition and each group is assigned a subset of shots. Shot parallelism best corresponds to a task-parallel decomposition, and is illustrated in Algorithm 1 by the loop on all the groups of nodes that starts at line 1, and by the loop on all shots assigned to the groups at line 2. Parallelizing shots does not require communication between nodes and should show a linear scaling. Let's mention that a typical seismic survey involves hundreds if not thousands of shot points. This should be at least on par with the number of available nodes on large clusters. On the other hand, domain decomposition is required to enable computations for models exceeding the memory capacity of a single device. For this level of parallelism, MPI manages communications between nodes and the OpenCL host thread manages the local devices. The communications managed by MPI and OpenCL are illustrated respectively by the loop on all nodes belonging to the same group starting at line 3 of Algorithm 1 and by the loop on all devices found on the node starting at line 4.

To further mitigate the communication time required in domain decomposition, the seismic updates are divided between grid points on the domain boundary that needs to be transferred and interior grid points that are only needed locally. This is described in Algorithm 2. The grid points on the boundary are first updated, which allows overlapping the communication and the computation of the remaining grid points, i.e. lines 3 and 4 and lines 7 and 8 of Algorithm 2 are performed simultaneously for devices supporting overlapped communications. This is allowed in OpenCL by having two different queues for each device: one for buffer communication and the other for kernel calls.

Algorithm 2 Pseudo code showing the overlapping computation and memory transfer for domain decomposition.

```
1.  while $t < N_t$
2.      Call kernel_updatev on domain boundary
3.      Transfer $\mathrm{v_i}$ in boundary of devices, nodes
4.      Call kernel_updatev on domain interior
5.      Store $S(\mathrm{v_i})$ in seismo at $t$
6.      Call kernel_updates on domain boundary
7.      Transfer $\sigma_{\mathrm{ij}}$ in boundary of devices, nodes
8.      Call kernel_updates on domain interior
9.      Increment $t$
10. end while
```

3.2 GPU kernels

The main elements of the kernels used to update stresses and velocities are shown in Algorithm 3. For better readability, the algorithm is simplified and does not include viscoelastic computations or CPML corrections. Note that the "for" loops in this pseudo-code are implicitly computed by OpenCL. The most important features of this algorithm are steps 3 and 4, where seismic variables needed in the computation of the spatial derivatives are loaded from the global memory to the local memory. As the computation of the spatial gradient of adjacent grid elements repeatedly uses the same grid points, this saves numerous reads from global memory. To be effective, those reads must be coalesced. This is achieved by setting the local working size in the z dimension, which is the fast dimension of the arrays, to a multiple of 32 for NVidias' GPUs. Hence, seismic variables are updated in blocks of 32 in the z dimension. In the x and y dimensions, the size of the local working size does not impact coalesced memory reading. They are set equal to a magnitude that allows all the seismic variables needed in the update to fit in the local memory. This is illustrated in Figure 4.

Algorithm 3 Pseudo code for the seismic update kernels showing how local memory is used.

```
1. for all local_domains in global_domain do
2.     for all grid point in local_domain do
3.         Load $v_i$ ($\sigma_{ij}$) from global to local memory
4.         Compute $\partial_i v_i$ ($\partial_i \sigma_{ij}$) from local memory
5.         Update $\sigma_{ij}$ ($v_i$) in global memory
6.     end for
7. end for
```

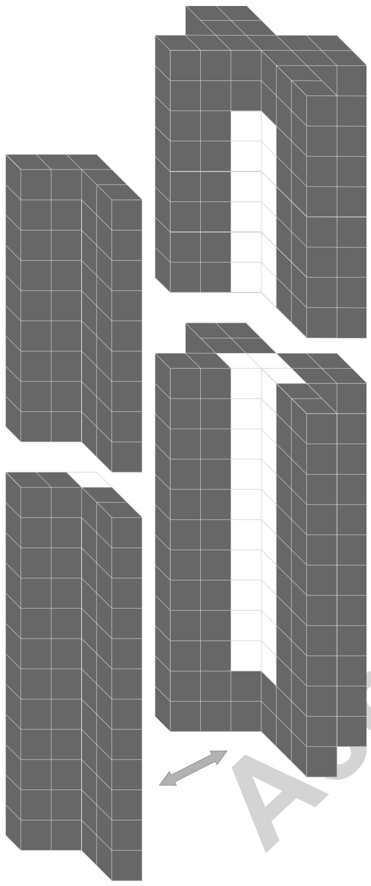

Figure 4 Exploded view of the local memory containing a seismic variable during update (equations 1a and 1b), for the 2nd order scheme. White cells are cells being updated and gray cells are loaded into local memory only to update white cells.

3.3 Misfit gradient computation

The cross-correlation of the direct and residual fields requires both fields to be computed at the same time step (see equation 8). This is challenging because forward computations are performed from time zero, whereas adjoint computations are performed from final time. Several strategies can be employed to achieve this task (see (Dussaud, et al., 2008, Nguyen and McMechan, 2015) for comparisons between methods).

1. When propagating the direct field, the whole grid for the particle velocities and stresses at each time step or a subset of the time steps can be saved into memory. When the residual field is propagated from final time, the direct field is read from memory for all grid points and the scalar product is evaluated iteratively, time step per time step.
2. In the so-called the backpropagation scheme (Clapp, 2008, Yang, et al., 2014), only the outside boundary of the grid that is not in the absorbing layer is saved at each time step. The direct field is recovered during the residual propagation by propagating back in time the direct field from the final state, injecting the saved wavefield on the outside boundary at each time step. As both the forward and adjoint wavefields are available at the same time step, the scalar products can be computed directly with equation 8.
3. A selected number of frequencies of the direct and residual field can be stored. This is performed by applying the discrete Fourier transform incrementally at each time step (equations 9 and 10), as done by (Sirgue, et al., 2008). The scalar product is evaluated at the end of the adjoint modeling in the frequency domain with equation 9. An alternative way of computing the chosen frequencies (Furse, 2000) seems to be advantageous over the discrete Fourier transform, but has not been tested in this study.
4. In the optimal checkpointing method proposed by (Griewank, 1992, Griewank and Walther, 2000), and applied by (Symes, 2007), the whole forward wavefield is stored for a limited number of time steps or checkpoints. To perform the scalar product, the forward wavefield is recomputed for each time step during the backpropagation of the residuals from the nearest checkpoint. For a fixed number of checkpoints, an optimal distribution that minimizes the number of forward wavefield that has to be recomputed can be determined. For this optimal distribution, the number of checkpoints and the number of recomputed time steps evolve logarithmically with the number of total time steps. Further improvements of the method have been proposed by (Anderson, et al., 2012) and by (Komatitsch, et al., 2016) in the viscoelastic case.

The first option is usually impractical, as it requires a huge amount of memory even for problems of modest size. In 3D, it requires on the order of $O(N_t N^3)$ elements to be stored, which becomes quickly intractable. Let's mention that the use of compression and subsampling can be used to mitigate these high memory requirements (Boehm, et al., 2016, Sun and Fu, 2013). The backpropagation scheme requires far less memory, on the order $O(N_t N^2)$ in 3D, but doubles the computation task for the direct field. Also, it is not applicable in the viscoelastic case. Indeed, in order to back-propagate the wavefield, the time must be reversed

$t \rightarrow -t$ and, doing so, the memory variable differential equation (equation 1c) becomes unstable. Hence, when dealing with viscoelasticity, the frequency scheme and the optimal checkpointing scheme are the only viable options. The memory requirement of the frequency scheme grows with the number of computed frequencies on the order of $O\left(N_f N^3\right)$. However, as is common in FWI, only a selected number of frequencies can be used (Virieux and Operto, 2009). The optimal checkpointing method requires $O(N_c N^3)$ where N_c is the number of checkpoints. Because of the logarithmic relationship between the number of time steps, the number of checkpoints and the number of additional computations, the required memory should stay tractable. For example, for 10 000 time steps, with only 30 buffers, the computing cost of this option is 3.4 times that of the forward modeling. In this work, we implemented the backpropagation scheme for elastic propagation and the frequency scheme using the discrete Fourier transform for both elastic and viscoelastic propagation. The implementation of the optimal checkpointing scheme or the hybrid backpropagation/checkpointing scheme of (Yang, et al., 2016) is left for future work.

The gradient computation involving the backpropagation of the direct field is illustrated in Algorithm 4. At each time step of the direct field propagation, the wavefield value at grid points on the outer edge of the model is stored. Because of the limited memory capacity of GPUs, this memory is transferred to the host. As mentioned before, this communication can be overlapped with other computations with the use of a second queue for communication. After obtaining the residuals, the residual wavefield is propagated forward in time using the same kernel as the direct wavefield. The back-propagation of the direct wavefield is calculated using the same kernel, the only difference being the sign of the time step $\Delta t \rightarrow -\Delta t$. Also, at each time step, the stored wavefield on the model edges is injected back. With this scheme, both the residual and the direct fields are available at each time step and can be multiplied to perform on the fly the scalar products needed to compute the gradient.

Algorithm 4 Pseudo code for the backpropagation scheme.

```
1.  while $t < N_t$
2.      Call kernel_updatev
3.      Store $v_i$ in boundary of model
4.      Call kernel_updates
5.      Store $\sigma_{ij}$ in boundary of model
6.      Increment $t$
7.  end while
8.  Calculate residuals
9.  while $t < N_t$
10.     Call kernel_updatev on $v_i, \overleftarrow{v}_i$
11.     Inject $v_i$ in boundary of model
12.     Call kernel_updates on $\sigma_{ij}, \overleftarrow{\sigma}_{ij}$
```

```
13.     Inject $\sigma_{ij}$ in boundary of model
14.     Compute gradient
15.     Increment $t$
16. end while
```

The frequency scheme is illustrated in Algorithm 5. It first involves computing the direct wavefield and its discrete Fourier transform on the fly at each time step, for each desired frequency (equation 10). Afterward, the residual wavefield is obtained in exactly the same fashion. At the end, the scalar product needed for the gradients can be computed with the selected frequencies.

Algorithm 5 Pseudo code for the frequency scheme.

```
1.  while $t < N_t$
2.      Call kernel_updatev for $v_i$
3.      Call kernel_updates for $\sigma_{ij}$
4.      Call DFT for $v_i$, $\sigma_{ij}$ for freqs
5.      Increment $t$
6.  end while
7.  Compute residuals
8.  while $t < N_t$
9.      Call kernel_updatev for $\overleftarrow{v}_i$
10.     Call kernel_updates for $\overleftarrow{\sigma}_{ij}$
11.     Call DFT for $\overleftarrow{v}_i, \overleftarrow{\sigma}_{ij}$ for freqs
12.     Increment $t$
13. end while
14. Compute gradients
```

4. Results and discussion

This section shows several numerical results obtained with SeisCL. The following tests were chosen to verify the performance of OpenCL in the context of FWI on heterogeneous clusters containing three different types of processors: Intel CPUs, Intel Xeon PHI (MIC) and NVidia GPUs.

4.1 Modeling validation

In order to test the accuracy of our forward/adjoint modeling algorithm, two synthetic cases are presented. First, the finite-difference solution of the viscoelastic wave equation is compared to the analytic solution. The analytic solution for the viscoelastic wave propagation of a point source derived by Pilant (2012) is used here in the form given by Gosselin-Cliche and Giroux (2014) for a quality factor profile corresponding to a single Maxwell body. The source is a Ricker wavelet with a center frequency of 40 Hz, oriented in the z direction. The viscoelastic model is homogeneous with V_p=3500 m/s, V_s=2000 m/s, ρ=2000 m/s with a single Maxwell body. We tested 4 attenuation levels $\tau_p = \tau_s = \{0, 0.01, 0.2, 0.4\}$, i.e. $Q = \{\infty, 200, 10, 5\}$ at the center frequency of 40 Hz. Using a finite-difference stencil of order 4, a 6 m (8.33 points per wavelength) spatial discretization is used to avoid numerical dispersion with a 0.5 ms time step for numerical stability. Figure 5 shows the comparison between the analytic solution and the solution obtained with SeisCL. The traces represent the particle velocities in the z direction for an offset of 132 m in the z direction. For the elastic case (τ=0), the analytical solution is perfectly recovered by SeisCL. Using higher attenuation levels does, however, introduce some errors in the solution. This error increases with τ and for an attenuation level of 0.4, the discrepancy becomes obvious for the offset used herein. It is, however, the expected drawback of using an explicit time domain solution and similar time-domain algorithms show the same behavior, see (Gosselin-Cliche and Giroux, 2014). Also, for reasonable attenuation levels, the errors appear negligible and will not impact FWI results much. Accuracy could become an issue for very high attenuating media and long propagation distances.

The second test aims at validating the misfit gradient output of SeisCL. For this test, a synthetic 2D cross-well tomographic survey is simulated, where a model perturbation between two wells is to be imaged. The well separation is 250 m and the source and receiver spacing are respectively 60 m and 12 m (Figure 5). Circular perturbations of a 60 m radius for the five viscoelastic parameters were considered at five different locations. The same homogeneous model as the first experiment is used with τ=0.2 and with perturbations of 5 % of the constant value. Because significant crosstalk can exist between parameters, especially between the velocities and the viscous parameters (Kamei and Pratt, 2013), we computed the gradient for one type of perturbation at a time. For example, the P-wave velocity gradient is computed with constant models for all other parameters other than V_p. This eliminates any crosstalk between parameters and allows a better appraisal of the match between the gradient update and the given perturbations. Note that because the goal of the experiment is to test the validity of the approach, geological plausibility was not considered. As no analytical solution exists for the gradient, the adjoint state gradient was compared to the gradient computed by finite-difference. The finite-difference solution was obtained by perturbing each parameter of the grid sequentially by 2%, for all the grid position between the two wells. The adjoint state gradient was computed with the frequency scheme using all frequencies of the discrete Fourier transform between 0 and 125 Hz.

The results of this second experiment are shown in Figure 6. In this figure, each column represents a different perturbed parameter. The first row shows the perturbation, the second the steepest descent direction (minus the misfit gradient) obtained by finite-difference and the third the steepest descent direction given by the adjoint state model. Note that the gradients were normalized in this figure. As can be visually appraised, an excellent agreement is obtained between both methods, for all parameters. Although the inversion has not been performed here, it should converge to the right solution in the five different cases, the update correction being already in the right direction. This is expected considering the small value of the perturbation used in this experiment; the inverse problem is more or less linear in such circumstances. The good agreement between the finite-difference and the adjoint state gradients shows that the latter could be used in any gradient-based inversion approach. However, the adjoint approach is orders of magnitude faster than the finite-difference approach: the first grows proportionally to the number of frequencies (see next section) while the second grows linearly with the number of parameters. For this particular experiment, the adjoint approach required minutes to complete whereas the finite-difference approach required days.

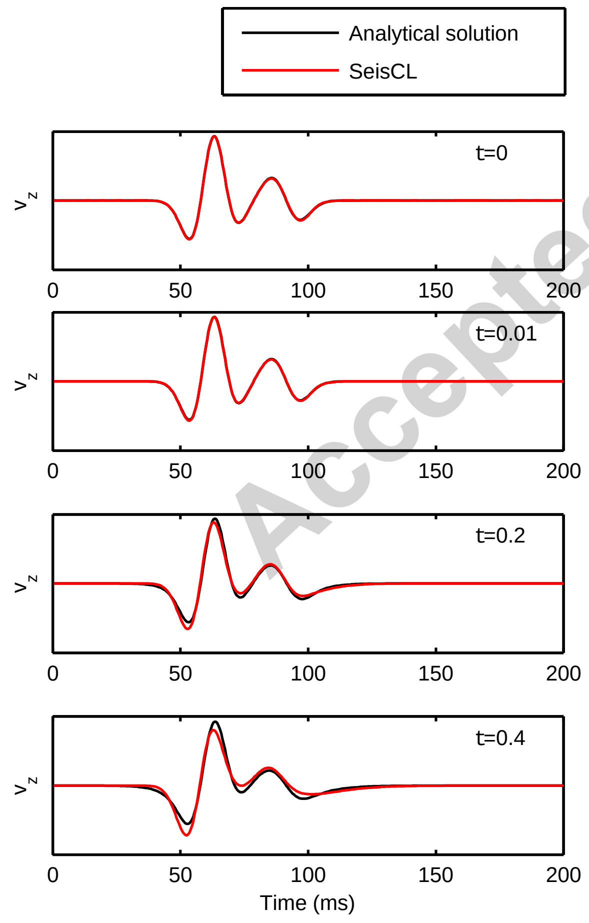

Figure 5 Comparison between the analytical solution and SeisCL results for different attenuation levels, from the elastic case (τ=0) to strong viscoelasticity (τ=0.4).

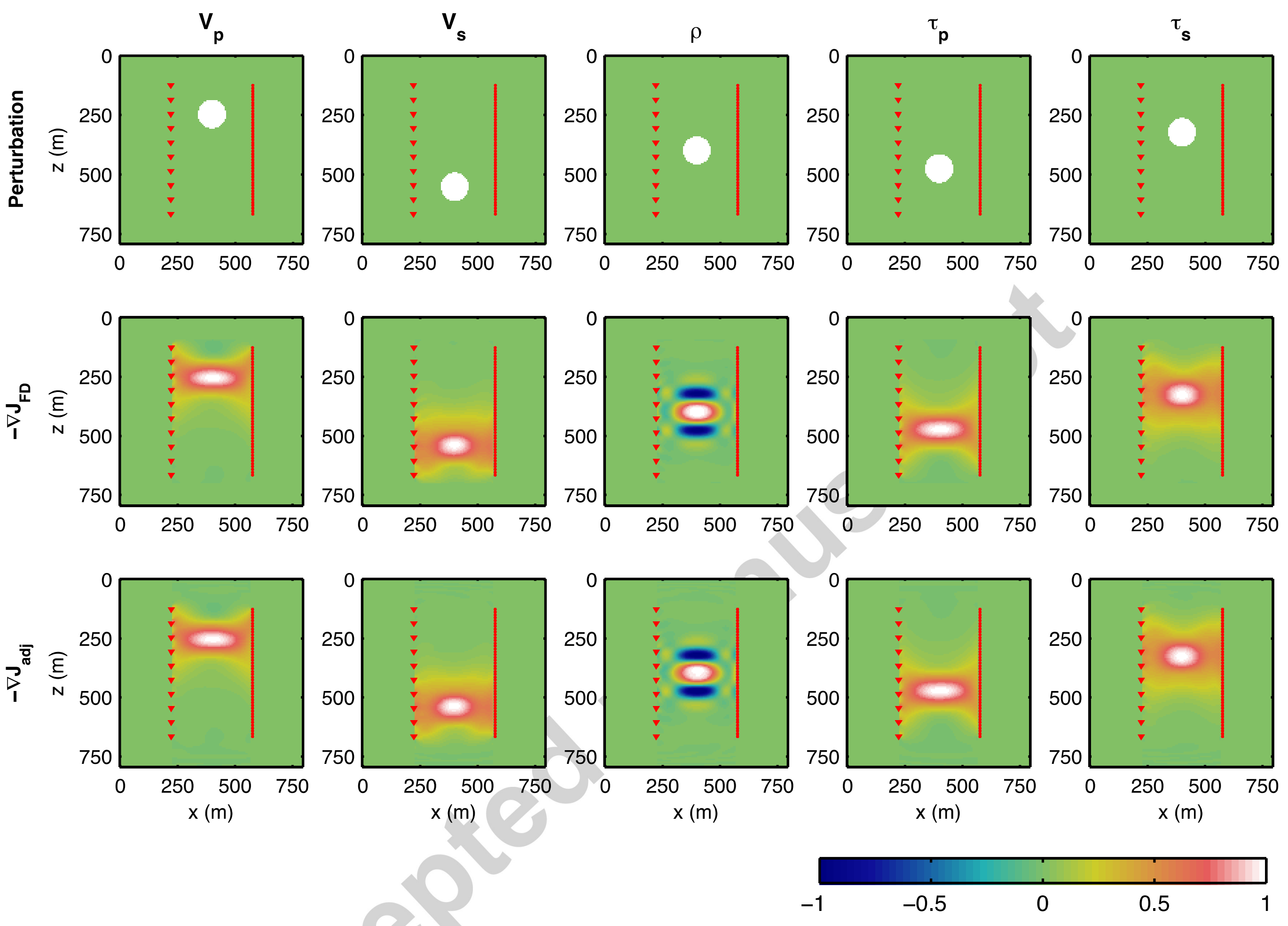

Figure 6 A cross-well experiment to test the validity of the misfit gradient. The red triangles represent the sources position and the red dots the receiver positions. Each column represents a different parameter. The first row shows the location of the perturbation, the second row represents the opposite of the misfit gradient obtained by finite-difference and the third row represents the opposite of the misfit gradient obtained by the adjoint state method.

4.2 Performance comparison

The effort required to program with the OpenCL standard would be vain without a significant gain in the computing performance. In the following, several tests are presented to measure the performance of SeisCL. As a measure, one can compute the speedup, defined here as:

$$\text{Speedup} = \frac{T_{baseline}}{T_{SeisCL}}. \quad (11)$$

Different baselines are used depending on the test. In order to show the OpenCL compatibility of different devices, all tests are performed on three types of processors: Intel CPUs, Intel Xeon PHI (MIC) and NVidia GPUs. Unless stated otherwise, the CPU device consists of 2 Intel Xeon E5-2680 v2 processors with 10 cores each at a frequency of 2.8 GHz and with 25 MB of cache. The GPU is an NVidia Tesla K40 with 2880 cores and 12 GB of memory, and the MIC is an Intel Xeon Phi 5110P.

4.2.1- Speedup using SeisCL over a single threaded CPU implementation

As a baseline, *SOFI2D* and *SOFI3D,* the 2D and 3D implementations of Bohlen (2002) are used with a single core. This baseline can be compared to SeisCL as both codes use the same algorithm. It is also representative of the speed that can be achieved for a FDTD code written in C, arguably one of the fastest high level languages for the CPU. In Figure 7, the speedup is measured as a function of the model size for the 3D and 2D cases, where the model size is a cube and a square respectively with edges of N grid points. The speed-up varies significantly with the model size. The highest speedups are attained with the GPU, which ranges between 50 to more than 80 in 3D and between 30 and 75 in 2D. Significant speedups are also obtained with CPUs, as high as 35 times faster. This is higher than the number of cores (20) available. We make the hypothesis that this is caused by a better cache usage of the OpenCL implementation, i.e. usage of local memory increases significantly the cache hits during computation compared to the C implementation of Bohlen (2002). The 2D implementation seems less impacted by this phenomenon and speedups are in a more normal range, between 11 and 25. We also noted that the time stepping computation can be very slow in the first several hundred time steps for the C implementation. This is the source of the strong variations in speedups observed in Figure 7. Finally, the Xeon Phi speedups are disappointing compared to their theoretical computing capacity. However, SeisCL has been optimized for GPUs, not for the Xeon Phi. Even if we have not tested it, it is possible that with small modifications of the code, improved performance could be attained. This shows, however, the limits of device portability with OpenCL: code optimization is paramount to achieve high performances and this optimization can be quite different for different devices.

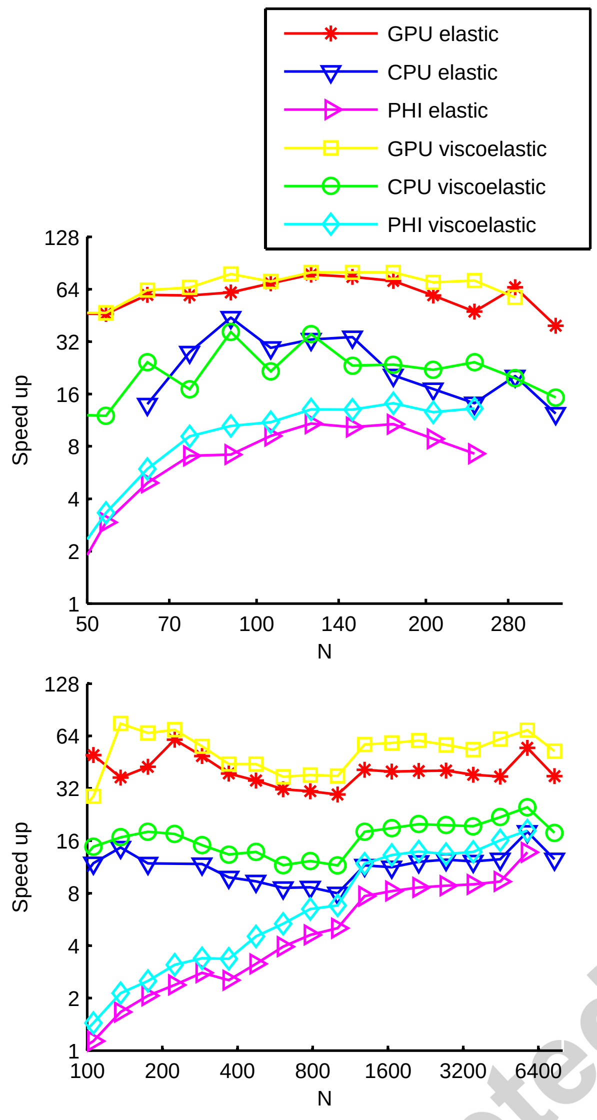

Figure 7 Speedup of SeisCL over a single threaded CPU implementation for different model sizes in 3D (top) and 2D (bottom), for different processor types.

4.2.2- Performance of the gradient calculation

The next test aims at assessing the performance of the two different gradient schemes. For this experiment, the baseline is the time required to perform one forward modeling run, without the gradient calculations. The computing times are measured for the backpropagation scheme and the frequency scheme, for model sizes of 100x100x100 and 1000x1000 grid nodes in 3D and 2D respectively. The results are shown in Figure 8. For the frequency scheme, the computing time increases linearly with the number of frequencies. The cost rises faster in 3D than in 2D, which can be explained by the higher number of variables needed to be transformed in 3D. Surprisingly, the computation time for the Xeon PHI seems to increase much slower than for the CPU or the GPU. It is to be noticed that for testing purposes, the discrete Fourier transform was computed at every time step. However, significant savings could be achieved if it was computed near the Nyquist frequency. Nevertheless, this test shows that the cost of computing the discrete Fourier transform during time stepping is not trivial but remains tractable. Finally, the backpropagation scheme has a cost that is roughly 3 times the cost of a single forward modeling for all devices. Hence, in the elastic case, the backpropagation scheme outperforms the frequency scheme no matter the number of frequencies. It also has the added benefit of containing all frequencies.

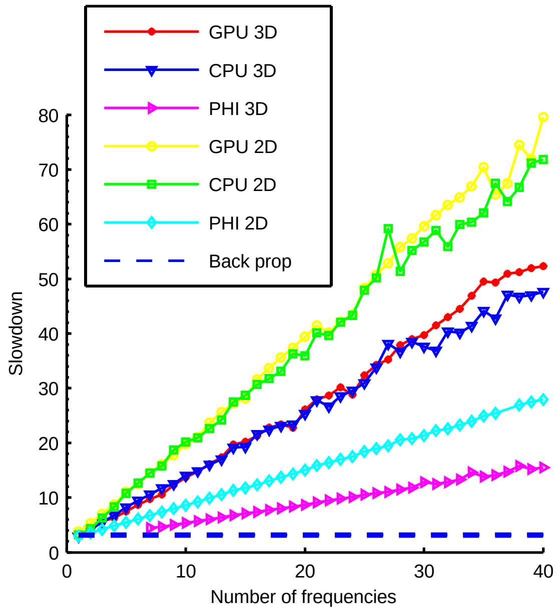

Figure 8 Ratio of the computing time between the forward modeling and the adjoint modeling in the frequency scheme for an increasing number of frequencies. The dashed line denotes the back-propagation scheme for all devices.

4.2.3- Measure of the cost of using higher order finite-difference stencils on different devices

The baseline for this test is the computation time of the 2nd order stencil for each device. The slowdown is used here as a measure, i.e. the inverse of the speedup. The same spatial and temporal step lengths were used for each order. As can be seen in Figure 9, for all three types of device, the slowdown is quite low and does not exceed 1.5 for the highest order of 12 considered here, except for the GPU in 3D where it exceeds 3 for an order of 12. Note that up to the 8th order, the GPU performance is comparable to the other device types. The higher cost for the GPU in 3D for orders 10 and 12 is caused by the limited amount of local memory. Indeed, for those orders, the amount of local memory required to compute the derivative of a single variable exceeds the device capacity. In those circumstances, SeisCL turns off the usage of local memory and uses global memory directly. The abrupt slowdown is manifest of the importance of using local memory. The reason why higher order stencils do not affect significantly the computing time of SeisCL is that it is bandwidth limited: access to the memory takes more time than the actual computations. As memory access is locally shared, the higher number of reads required for higher finite-difference order does not increase significantly. The impact on computation at each time step is thus marginal. In most cases, the advantages of using higher orders outweigh the computational costs, because it allows reducing the grid size. For example, using a 6th order over a 2nd order stencil allows reducing the grid point per wavelength from around 22 to 3, i.e. it reduces the number of grid elements by a factor of 400 in 3D. However, in some situations, for instance in the presence of a free surface, topography or strong material discontinuities, higher order stencils introduce inaccuracies (Bohlen and Saenger, 2006, Kristek, et al., 2002, Robertsson, 1996). Hence, the choice of the order should be evaluated on a case-by-case basis.

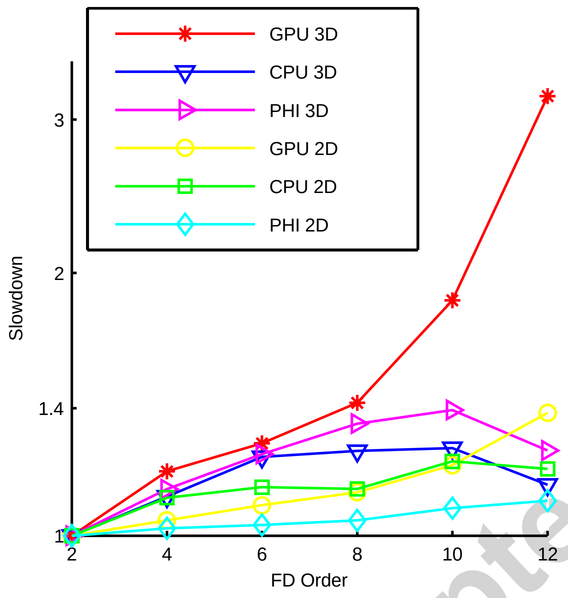

Figure 9 Slowdown of the computation using higher finite-difference order compared to the 2nd order for different devices.

4.2.4- Tests on heterogeneous clusters

To evaluate the scalability of our code over large clusters, a strong scaling test was performed. Here, strong scaling refers to the variation of the computational time for a model of fixed sized for an increasing number of processors. The following results were obtained for a grid size of 96x96x9000 elements and an increasing number of devices for the domain decomposition. This test was performed on two different clusters: *Helios* of Laval University, Canada and *Guillimin* from McGill University, Canada. The *Helios* nodes contain 8 NVidia K80 GPUs (16 devices). This cluster was used to test strong scaling for GPUs on a single node of a cluster, which does not involve MPI. Two types of nodes were used on *Guillimin*: nodes containing two Intel Xeon X5650 with 6 cores each at

2.66 GHz and 12 MB of cache and nodes containing 2 NVidia K20 GPUs in addition to the same two Xeon CPUs. This cluster was used to test strong scaling across several nodes, which requires MPI communication.

Results are shown in Figure 10. The best scaling behavior is shown by the nodes on *Guillimin* with two GPUs, which is very nearly linear over the tested number of devices (blue triangles on Figure 10). Surprisingly, the scaling is slightly worse for many devices located on the same node (*Helios* nodes, red stars in Figure 10). We interpret this result as being caused by the increasing burden on the processor when a higher number of GPUs must be scheduled on the same nodes: at some point, the CPU becomes too slow to keep all GPUs busy. Compared to *Guillimin* nodes using CPUs, *Guillimin* nodes using GPUs also scale better. Still, the CPU scaling remains quite favorable and is higher than $N^{4/5}$. Those results are better than the results reported by Okamoto (2011), Rubio, et al. (2014), Weiss and Shragge (2013). We explain this favorable behavior by the separate computation of grid elements inside and outside of the communication zone in our code.

The strong scaling tests show that for large models that fit only on multiple nodes and devices, SeisCL can efficiently parallelize the computation domains with a minimal performance cost. Still, parallelization over shots should be favored when models fit in the memory of a single device because no fast interconnects are needed in this situation, and because SeisCL is somewhat more efficient when memory usage attains a certain level, as shown in Figure 7. In short, having both types of parallelization allows a greater flexibility over the type of cluster that can be used with SeisCL.

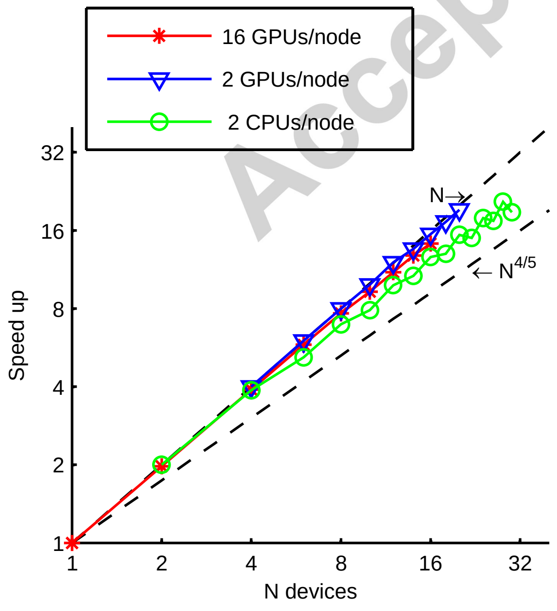

Figure 10 Strong scaling tests for a grid size of 96x96x1000. Red corresponds to results from *Helios*. Green and blue to *Guillimin*.

5. Conclusion

In this article, we presented a program called SeisCL for viscoelastic FWI on heterogeneous clusters. The algorithm solves the viscoelastic wave equation by the Finite-Difference Time-Domain approach and uses the adjoint state method to output the gradient of the misfit function. Two approaches were implemented for the gradient computation by the adjoint method: the backpropagation approach and the frequency approach. The backpropagation approach was shown to be the most efficient in the elastic case, having roughly the cost of 3 forward computations. It is, however, not applicable when viscoelasticity is introduced. The frequency approach has an acceptable cost when a small number of frequencies is selected, but becomes quite prohibitive when all frequencies are needed. Future work should focus on the implementation of the optimal checkpointing strategy, which is applicable to both elastic and viscoelastic FWI and strikes a balance between computational costs and memory usage.

It was shown that using OpenCL speeds up the computations compared to a single-threaded implementation and allows the usage of different processor architectures. To highlight the code portability, three types of processors were tested: Intel CPUs, Nvidia GPUs and Intel Xeon PHI. The best performances were achieved with the GPUs: a speedup of nearly two orders of magnitude over the single-threaded code was attained. On the other hand, code optimization was shown to be suboptimal on the Xeon PHI, which shows that some efforts must still be spent on device-specific optimization. For the GPU, memory usage was the main area of code optimization. In particular, the use of OpenCL local memory is paramount and coalesced access to global memory must be embedded in the algorithm.

When using domain decomposition across devices and nodes of a cluster, overlapping communications and computations allowed hiding the cost of memory transfers. Domain decomposition parallelization was shown to be nearly linear on clusters with fast interconnects using different kinds of processors. Hence, SeisCL can be used to compute the misfit gradient efficiently for large 3D models on a cluster. Furthermore, the task-parallel scheme of distributing shots allows flexibility when the speed of interconnects between the nodes limits the computational gain. Together, both parallelization schemes allow a more efficient usage of large cluster resources.

In summary, the very good performance of SeisCL on heterogeneous clusters containing different processor architectures, particularly GPUs, is very promising to speed up full waveform inversion. Presently, the most efficient devices for SeisCL are GPUs, but this can change in the future. The open nature and the flexibility of OpenCL will most probably allow SeisCL to use new hardware developments. SeisCL is distributed with an open license over Github.

Apppendix A

This section lists the misfit gradient coefficients. First, some constants are defined:

$$\alpha = \sum_{l=1}^{L} \frac{\omega_0^2 \tau_{\sigma l}^2}{1+\omega_0^2 \tau_{\sigma l}^2}, \tag{A-1}$$

$$b_1 = \left(N_d M \frac{(1+\tau_p)}{(1+\alpha\tau_p)} - 2(N_d - 1)\mu \frac{(1+\tau_s)}{(1+\alpha\tau_s)} \right)^{-2}, \tag{A-2}$$

$$b_2 = \left(N_d M \frac{\tau_p}{(1+\alpha\tau_p)} - 2(N_d - 1)\mu \frac{\tau_s}{(1+\alpha\tau_s)} \right)^{-2}. \tag{A-3}$$

The misfit gradient coefficients are given by:

$$c_M^1 = \frac{(1+\tau_p)}{(1+\alpha\tau_p)} b_1 \tag{A-4}$$

$$c_M^2 = \frac{\tau_p}{(1+\alpha\tau_p)}\ b_2, \tag{A-5}$$

$$c_\mu^1 = \frac{1}{\mu^2} \frac{(1+\alpha\tau_s)}{(1+\tau_s)}, \tag{A-6}$$

$$c_\mu^2 = \frac{(N_d+1)}{3} \frac{(1+\tau_s)}{(1+\alpha\tau_s)} b_1 \tag{A-7}$$

$$c_\mu^3 = \frac{1}{2N_d} \frac{1}{\mu^2} \frac{(1+\alpha\tau_s)}{(1+\tau_s)}, \tag{A-8}$$

$$c_\mu^4 = \frac{1}{\mu^2} \frac{(1+\alpha\tau_s)}{\tau_s}, \tag{A-9}$$

$$c_\mu^5 = \frac{(N_d+1)}{3} \frac{\tau_s}{(1+\alpha\tau_s)} b_2, \tag{A-10}$$

$$c_\mu^6 = \frac{1}{2N_d} \frac{1}{\mu^2} \frac{(1+\alpha\tau_s)}{\tau_s}, \tag{A-11}$$

$$c_{\tau_p}^1 = (1-\alpha) \frac{M}{(1+\alpha\tau_p)^2} b_1, \tag{A-12}$$

$$c_{\tau_p}^2 = \frac{M}{(1+\alpha\tau_p)^2} b_2, \tag{A-13}$$

$$c_{\tau_s}^1 = \frac{1}{\mu} \frac{(1-\alpha)}{(1+\tau_s)^2}, \tag{A-14}$$

$$c_{\tau_s}^2 = \frac{(N_d+1)}{3}(1-\alpha)\frac{\mu}{(1+\alpha\tau_s)^2}b_1 \qquad \text{(A-15)}$$

$$c_{\tau_s}^3 = \frac{(1-\alpha)}{2N_d\mu(1+\tau_s)^2}\,, \qquad \text{(A-16)}$$

$$c_{\tau_s}^4 = \frac{1}{\mu\tau_s^2}, \qquad \text{(A-17)}$$

$$c_{\tau_s}^5 = \frac{(N_d+1)}{3}\frac{\mu}{(1+\alpha\tau_s)^2}b_2, \qquad \text{(A-18)}$$

$$c_{\tau_s}^6 = \frac{1}{2N_d\mu\tau_s^2}, \qquad \text{(A-19)}$$

Acknowledgements

This work was funded by a Vanier Canada Graduate Scholarship and supported by the Canada Chair in geological and geophysical data assimilation for stochastic geological modeling.

References

Abdelkhalek, R., Calandra, H., Coulaud, O., Roman, J. and Latu, G., 2009, Fast seismic modeling and Reverse Time Migration on a GPU cluster, p. 36-43, 10.1109/hpcsim.2009.5192786.
Abdullah, D. and Al-Hafidh, M. M., 2013, Developing Parallel Application on Multi-core Mobile Phone: Editorial Preface, v. 4, no. 11.
Anderson, J. E., Tan, L. and Wang, D., 2012, Time-reversal checkpointing methods for RTM and FWI: Geophysics, v. 77, no. 4, p. S93-S103, 10.1190/geo2011-0114.1.
Askan, A., Akcelik, V., Bielak, J. and Ghattas, O., 2007, Full Waveform Inversion for Seismic Velocity and Anelastic Losses in Heterogeneous Structures: Bulletin of the Seismological Society of America, v. 97, no. 6, p. 1990-2008, 10.1785/0120070079.
Bai, J., Yingst, D., Bloor, R. and Leveille, J., 2014, Viscoacoustic waveform inversion of velocity structures in the time domain: Geophysics, v. 79, no. 3, p. R103-R119, 10.1190/geo2013-0030.1.
Blanch, J. O., Robertsson, J. O. A. and Symes, W. W., 1995, Modeling of a constantQ: Methodology and algorithm for an efficient and optimally inexpensive viscoelastic technique: Geophysics, v. 60, no. 1, p. 176-184, 10.1190/1.1443744.
Boehm, C., Hanzich, M., de la Puente, J. and Fichtner, A., 2016, Wavefield compression for adjoint methods in full-waveform inversion: Geophysics, v. 81, no. 6, p. R385-R397, 10.1190/geo2015-0653.1.
Bohlen, T., 2002, Parallel 3-D viscoelastic finite difference seismic modelling: Computers & Geosciences, v. 28, no. 8, p. 887-899, 10.1016/s0098-3004(02)00006-7.
Bohlen, T. and Saenger, E. H., 2006, Accuracy of heterogeneous staggered-grid finite-difference modeling of Rayleigh waves: Geophysics, v. 71, no. 4, p. T109-T115, 10.1190/1.2213051.
Carcione, J. M., Kosloff, D. and Kosloff, R., 1988, Viscoacoustic wave propagation simulation in the earth: Geophysics, v. 53, no. 6, p. 769-777, 10.1190/1.1442512.
Cerjan, C., Kosloff, D., Kosloff, R. and Reshef, M., 1985, A nonreflecting boundary condition for discrete acoustic and elastic wave equations: Geophysics, v. 50, no. 4, p. 705-708, 10.1190/1.1441945.
Clapp, R. G., 2008, Reverse time migration: saving the boundaries, Technical Report SEP-136, Stanford Exploration Project, p. 137.
Dongarra, J. J., Meuer, H. W. and Strohmaier, E., 2015, Top500 supercomputer sites, http://www.top500.org/ (Accessed on 01/12/2015)

Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G. and Dongarra, J., 2012, From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming: Parallel Computing, v. 38, no. 8, p. 391-407, 10.1016/j.parco.2011.10.002.
Dussaud, E., Symes, W. W., Williamson, P., Lemaistre, L., Singer, P., Denel, B. and Cherrett, A., 2008, Computational strategies for reverse- time migration, p. 2267-2271, 10.1190/1.3059336.
Fabien-Ouellet, G., Gloaguen, E. and Giroux, B., The Adjoint State Method for the Viscoelastic Wave Equation in the Velocity-stress Formulation, in Proceedings 78th EAGE Conference and Exhibition 2016, May 2016, 10.3997/2214-4609.201600826.
Fichtner, A., 2011, Full Seismic Waveform Modelling and Inversion.
Fichtner, A., Bunge, H. P. and Igel, H., 2006, The adjoint method in seismology - I. Theory: Physics of the Earth and Planetary Interiors, v. 157, no. 1-2, p. 86-104, 10.1016/j.pepi.2006.03.016.
Furse, C. M., 2000, Faster than Fourier: ultra-efficient time-to-frequency-domain conversions for FDTD simulations: IEEE Antennas and Propagation Magazine, v. 42, no. 6, p. 24-34, 10.1109/74.894179.
Gosselin-Cliche, B. and Giroux, B., 2014, 3D frequency-domain finite-difference viscoelastic-wave modeling using weighted average 27-point operators with optimal coefficients: Geophysics, v. 79, no. 3, p. T169-T188, 10.1190/geo2013-0368.1.
Griewank, A., 1992, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation: Optimization Methods and Software, v. 1, no. 1, p. 35-54, 10.1080/10556789208805505.
Griewank, A. and Walther, A., 2000, Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation: ACM Transactions on Mathematical Software, v. 26, no. 1, p. 19-45, 10.1145/347837.347846.
Holberg, O., 1987, Computational Aspects of the Choice of Operator and Sampling Interval for Numerical Differentiation in Large-Scale Simulation of Wave Phenomena*: Geophysical Prospecting, v. 35, no. 6, p. 629-655, 10.1111/j.1365-2478.1987.tb00841.x.
Howes, L. and Munshi, A., 2014, The OpenCL Specification: Khronos OpenCL.
Iturrarán-Viveros, U. and Molero, M., 2013, Simulation of sonic waves along a borehole in a heterogeneous formation: Accelerating 2.5-D finite differences using [Py]OpenCL: Computers & Geosciences, v. 56, p. 161-169, 10.1016/j.cageo.2013.03.014.
Kamei, R. and Pratt, R. G., 2013, Inversion strategies for visco-acoustic waveform inversion: Geophysical Journal International, v. 194, no. 2, p. 859-884, 10.1093/gji/ggt109.
Kloc, M. and Danek, T., 2013, The Multi GPU Accelerated Waveform Inversion in Distributed OpenCL Environment, Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering, Springer, p. 237-244.
Komatitsch, D., Erlebacher, G., Göddeke, D. and Michéa, D., 2010, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster: Journal of Computational Physics, v. 229, no. 20, p. 7692-7714, 10.1016/j.jcp.2010.06.024.
Komatitsch, D. and Martin, R., 2007, An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation: Geophysics, v. 72, no. 5, p. SM155-SM167, 10.1190/1.2757586.
Komatitsch, D., Xie, Z., Bozdağ, E., Sales de Andrade, E., Peter, D., Liu, Q. and Tromp, J., 2016, Anelastic sensitivity kernels with parsimonious storage for adjoint tomography and full waveform inversion: Geophysical Journal International, v. 206, no. 3, p. 1467-1478, 10.1093/gji/ggw224.
Kristek, J., Moczo, P. and Archuleta, R. J., 2002, Efficient Methods to Simulate Planar Free Surface in the 3D 4th-Order Staggered-Grid Finite-Difference Schemes: Studia Geophysica et Geodaetica, v. 46, no. 2, p. 355-381, 10.1023/a:1019866422821.
Lailly, P., The seismic inverse problem as a sequence of before stack migrations, in Proceedings Conference on inverse scattering: theory and application, 1983, Society for Industrial and Applied Mathematics, Philadelphia, PA, p. 206-220.
Levander, A. R., 1988, Fourth- order finite- differenceP-SVseismograms: Geophysics, v. 53, no. 11, p. 1425-1436, 10.1190/1.1442422.
Liu, H.-P., Anderson, D. L. and Kanamori, H., 1976, Velocity dispersion due to anelasticity; implications for seismology and mantle composition: Geophysical Journal International, v. 47, no. 1, p. 41-58.
Malinowski, M., Operto, S. and Ribodetti, A., 2011, High-resolution seismic attenuation imaging from wide-aperture onshore data by visco-acoustic frequency-domain full-waveform inversion: Geophysical Journal International, v. 186, no. 3, p. 1179-1204, 10.1111/j.1365-246X.2011.05098.x.
Mattson, T. G., Sanders, B. A. and Massingill, B. L., 2004, Patterns for parallel programming, Pearson Education.
Michéa, D. and Komatitsch, D., 2010, Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards: Geophysical Journal International, p. no-no, 10.1111/j.1365-246X.2010.04616.x.
Micikevicius, P., 2009, 3D finite difference computation on GPUs using CUDA, p. 79-84, 10.1145/1513895.1513905.
Molero, M. and Iturrarán-Viveros, U., 2013, Accelerating numerical modeling of wave propagation through 2-D anisotropic materials using OpenCL: Ultrasonics, v. 53, no. 3, p. 815-822.
Mu, D., Chen, P. and Wang, L., 2013, Accelerating the discontinuous Galerkin method for seismic wave propagation simulations using the graphic processing unit (GPU)—single-GPU implementation: Computers & Geosciences, v. 51, p. 282-292.
Nguyen, B. D. and McMechan, G. A., 2015, Five ways to avoid storing source wavefield snapshots in 2D elastic prestack reverse time migration: Geophysics, v. 80, no. 1, p. S1-S18, 10.1190/geo2014-0014.1.
Nickolls, J. and Dally, W. J., 2010, The GPU Computing Era: IEEE Micro, v. 30, no. 2, p. 56-69, 10.1109/mm.2010.41.

Nihei, K. T. and Li, X., 2007, Frequency response modelling of seismic waves using finite difference time domain with phase sensitive detection (TD-PSD): Geophysical Journal International, v. 169, no. 3, p. 1069-1078, 10.1111/j.1365-246X.2006.03262.x.

Nvidia, C., 2007, Compute unified device architecture programming guide.

O'connell, R. and Budiansky, B., 1978, Measures of dissipation in viscoelastic media: Geophysical Research Letters, v. 5, no. 1, p. 5-8.

Okamoto, 2011, Accelerating large-scale simulation of seismic wave propagation by multi-GPUs and three-dimensional domain decomposition: Earth, Planets and Space, v. 62, no. 12, p. 939-942, 10.5047/eps.2010.11.009.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. and Phillips, J. C., 2008, GPU Computing: Proceedings of the IEEE, v. 96, no. 5, p. 879-899, 10.1109/jproc.2008.917757.

Pilant, W. L., 2012, Elastic waves in the earth, Volume 11, Elsevier.

Plessix, R. E., 2006, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications: Geophysical Journal International, v. 167, no. 2, p. 495-503, 10.1111/j.1365-246X.2006.02978.x.

Pratt, R. G. and Worthington, M., 1990, Inverse theory applied to multi-source cross-hole tomography. Part 1: acoustic wave-equation method: Geophysical Prospecting, v. 38, no. 3, p. 287-310, 10.1111/j.1365-2478.1990.tb01846.x

Robertsson, J. O. A., 1996, A numerical free- surface condition for elastic/viscoelastic finite- difference modeling in the presence of topography: Geophysics, v. 61, no. 6, p. 1921-1934, 10.1190/1.1444107.

Robertsson, J. O. A., Blanch, J. O. and Symes, W. W., 1994, Viscoelastic finite- difference modeling: Geophysics, v. 59, no. 9, p. 1444-1456, 10.1190/1.1443701.

Roden, J. A. and Gedney, S. D., 2000, Convolution PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media: Microwave and Optical Technology Letters, v. 27, no. 5, p. 334-339, 10.1002/1098-2760(20001205)27:5<334::aid-mop14>3.0.co;2-a.

Rubio, F., Hanzich, M., Farrés, A., de la Puente, J. and María Cela, J., 2014, Finite-difference staggered grids in GPUs for anisotropic elastic wave propagation simulation: Computers & Geosciences, v. 70, p. 181-189, 10.1016/j.cageo.2014.06.003.

Sirgue, L., Etgen, J. and Albertin, U., 3D frequency domain waveform inversion using time domain finite difference methods, in Proceedings 70th EAGE Conference and Exhibition incorporating SPE EUROPEC 2008, 2008.

Sirgue, L. and Pratt, R. G., 2004, Efficient waveform inversion and imaging: A strategy for selecting temporal frequencies: Geophysics, v. 69, no. 1, p. 231-248, 10.1190/1.1649391.

Stone, J. E., Gohara, D. and Shi, G., 2010, OpenCL: A parallel programming standard for heterogeneous computing systems: Computing in science & engineering, v. 12, no. 1-3, p. 66-73.

Sun, W. and Fu, L.-Y., 2013, Two effective approaches to reduce data storage in reverse time migration: Computers & Geosciences, v. 56, p. 69-75, 10.1016/j.cageo.2013.03.013.

Symes, W. W., 2007, Reverse time migration with optimal checkpointing: Geophysics, v. 72, no. 5, p. SM213-SM221, 10.1190/1.2742686.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, v. 49, no. 8, p. 1259-1266, 10.1190/1.1441754.

Virieux, J., 1986, P-SVwave propagation in heterogeneous media: Velocity- stress finite- difference method: Geophysics, v. 51, no. 4, p. 889-901, 10.1190/1.1442147.

Virieux, J. and Operto, S., 2009, An overview of full-waveform inversion in exploration geophysics: Geophysics, v. 74, no. 6, p. WCC1-WCC26, 10.1190/1.3238367.

Vuduc, R., Chandramowlishwaran, A., Choi, J., Guney, M. and Shringarpure, A., On the limits of GPU acceleration, in Proceedings Proceedings of the 2nd USENIX conference on Hot topics in parallelism, 2010, USENIX Association, p. 13-13.

Weiss, R. M. and Shragge, J., 2013, Solving 3D anisotropic elastic wave equations on parallel GPU devices: Geophysics, v. 78, no. 2, p. F7-F15, 10.1190/geo2012-0063.1.

Yang, P., Brossier, R., Metivier, L. and Virieux, J., 2016, Checkpointing-assisted reverse-forward simulation: An optimal recomputation method for FWI and RTM, p. 1089-1093, 10.1190/segam2016-13685603.1.

Yang, P., Gao, J. and Wang, B., 2014, RTM using effective boundary saving: A staggered grid GPU implementation: Computers & Geosciences, v. 68, p. 64-72, 10.1016/j.cageo.2014.04.004.

Zhe, F., Feng, Q., Kaufman, A. and Yoakum-Stover, S., 2004, GPU Cluster for High Performance Computing, p. 47-47, 10.1109/sc.2004.26.

Highlights

- An open source software for viscoelastic full waveform inversion is presented.
- This software is based on OpenCL and can run on CPUs, GPUs and accelerators.
- On large clusters, MPI is used and a nearly linear scaling is achieved.
- Using GPUs, we obtain a speed-up of up to 80x over a single threaded CPU code.