A multigrid approach for time-domain FWI

Daniel Trad

ABSTRACT

Full Waveform inversion (FWI) has been implemented in many different ways. FWI is usually implemented in either time or frequency domains, and in some cases in a hybrid domain. The frequency-domain version (FD) is often used in academia because its formulation is simple to understand in terms of matrices. Another key advantage is that FD FWI permits the use of a multigrid approach consisting of bootstrapping from low to high frequencies where data bandwidth is gradually increased. A limitation of FD FWI, however, is that it becomes very expensive to calculate in 3 dimensions, because a large system of equations has to be solved at each frequency. The time-domain (TD) approach, however, seems to be more promising in terms of scalability. In this case, rather than solving one frequency of many shots at a time, all frequencies of one shot are solved at a time. However, this statement is an oversimplification, since parallelization and cycle skipping turn the algorithm into a very different dataflow. In practice, the time domain algorithm permits to solve many shots simultaneously as well, by using parallelization in clusters with Graphic Processing Units (GPUs), leading to a very efficient implementation with coarse-grained parallelism across nodes and fine-grained parallelism across GPUs. On the other hand, additional complications arise to use a multigrid approach which is, in turn, easily done in the frequency-domain. In this report, we will discuss a multigrid time-domain implementation of Full Waveform Inversion which will also help to understand and attenuate inverse crime effects when working with synthetic data.

INTRODUCTION

Full Waveform Inversion (FWI) is one of the most powerful but also difficult inversion algorithms to apply in real data processing (Schuster, 2017). It can bring up detailed velocity information about the subsurface by fitting modeled data to observations. Its power can be easily proven for synthetic data in ideal conditions but in practice, many difficulties need to be overcome to reach its goals. Some of these issues are related to the non-linearity of the inversion, usually manifested in the form of cycle-skipping. Other issues are related to the difficulties of replicating the complexities of wave propagation to predict realistic data with similar characteristics to acquired-processed data. Finally, its computational cost makes this technique one of the most expensive in seismic imaging. This report discusses an implementation/dataflow to help to address these issues. We will do so in the context of acoustic inversion only, with the understanding that elastic and anisotropic FWI are very important and also have to be addressed. This paper's scope, however, is how to make acoustic FWI work very efficiently.

At a larger picture, we should start by distinguishing between time and frequency domain implementations. Frequency domain versions consist of solving one frequency at a time for many shots using a large system of equations given by the Helmholtz wave equation. The number of equations to solve is the same as the number of cells in the velocity model. Therefore, for 3D surveys or even large 2D surveys, each frequency requires signif-

icant computing power which can be achieved by decomposing a large matrix into several computing nodes. Clusters with fast communication across nodes are required, which tend to be very expensive, and parallelization has to be implemented using specialized parallel libraries that can handle fine-grained parallelism efficiently. On the other hand, the frequency domain has some advantages. The non-linearity issue, that is the cycle-skipping, has a clean solution by bootstrapping from low to high frequencies (Pratt and Worthington, 1990). Also, the Hessian, which is the left-hand side of the Helmholtz system of equations, can be decomposed (for example using QR decomposition) to solve for many shots simultaneously. Finally, it can be argued that the implementation of one frequency at a time may have some advantages to replicate the frequency dependent data complexity like dispersive energy in the near-surface.

From the computational point of view, a time domain version can have very different properties. A simple inexpensive cluster, for example a Beowulf cluster, is sufficient, being the only requirement to have a large number of nodes. If the cluster has as many nodes as shots, each FWI iteration can be done in the same amount of time as it takes to forward model one shot four times. This is usually in terms of minutes, seconds if using GPUs. GPU nodes are inexpensive these days, and simple commodity hardware can be used in this case. This is different from specialized clusters where the communication has to be reduced by using low latency hardware, for example with Infiniband connection across nodes. Similarly, there is no scalability issue for 3D surveys, since Finite Difference 3D propagation is very fast with GPUs or many-core units (modern CPUs have 32-64 cores). Therefore, from the computational point of view, we can argue that time domain implementations have an easier time scaling to large surveys. Even parallel implementations are much simpler in the time domain than in the frequency domain and can be written without specialized libraries but simply by using a few commands of Memory Passing Interface (MPI) ("reduce" and "scatter" operations). On the other hand, the non-linearity issue and the replication of complex physics during the propagation bring some additional complications as follows.

When working in the frequency domain the data separation across frequency bands follows naturally by selecting frequency slices on the frequency-space plane domain. In the time domain, the separation between bands can not be done at abrupt boundaries so some kind of tapering is requiring. Simultaneously, the FWI modeling engine needs to adapt for different bandwidths by changing the source wavelet and the cell grid spacing. These factors introduce some discrepancy between predictions and data that make convergence harder and careful implementation is required to attenuate them. For real data, these differences would be much smaller in magnitude than other problems introduced by processing, like amplitude distortion caused by aggressive noise attenuation and differences between numerical and natural engines. Nonetheless, these factors would be large enough to delay or prevent convergence and their investigation brings us closer to understand how inverse crime effects play a key role in the inversion of synthetic data and can mislead conclusions if they are not eliminated.

In this report, we will work with synthetic data to make these discrepancies more evident. Notice that these issues are often artificially eliminated in synthetic tests when the data are generated by the same numerical engine used during the inversion. This is normally referred to as the inverse crime issue and exists both in the time and frequency do-

mains, but as we see below, the crime becomes more benign in the time domain because of multigriding.

The main goal of this report is to try a multigrid implementation in the time domain to understand the effects of small deviations from the inverse crime scenario. The approach is straightforward: starting from data generated on a dense grid with the given acquisition parameters, we will apply a shaping filter in different stages, each one with a higher dominant frequency. For each of these cases, we will apply FWI starting from the final model obtained in the previous stage after interpolating the grid to reduce the cell size. We will see that simple band-pass filtering is insufficient and a shaping filter is necessary to achieve coherence. In addition, we will use a simple time-shifting by cross-correlation, similar to as used in residual statics, to improve results significantly.

In addition to these simple tests, we will analyze why this process in the time domain can be very beneficial from the computation point of view. We will see that the time domain version can be implemented with coarse parallelization leading to efficient code that can be run in standard low-cost clusters as used in typical production jobs. On the other hand, the implementation in the frequency domain requires the use of libraries that can handle fine-grained parallelization which usually requires a higher-cost low communication latency cluster. Finally, we will discuss some simple examples in Compute Unified Device Architecture (CUDA) which can handle fine-grained parallelization.

FULL WAVEFORM INVERSION BASICS

FWI in the time domain vs the frequency domain

The theory for FWI is covered in numerous papers so I will mention just for completeness some concepts used in the rest of this report. FWI is a tool to refine velocities obtained by other methods, typically tomography, by adding into the modeling constraints the need to match not only phase but also amplitudes. The phase is an integral property, but amplitude is a derivative property. As a consequence, matching amplitudes is more demanding than phases alone, and forces the model to predict accurately high wavenumber components, therefore increasing velocity resolution. As usual in most inversion algorithms, this can be achieved by minimizing a cost function that penalizes poor performance for a velocity model that can not predict the data:

$$J = \|\mathbf{data} - \mathbf{L}(\mathbf{velocity})\| + \mathbf{constraints}_{\mathbf{model}} \tag{1}$$

Here $\mathbf{L}$ stands for some modelling engine, for example finite differences (FD) or Kirchhoff modelling. I don't specify what the constraints are as there are many possibilities well described in the literature and are not the focus of this work. What is important to note in this equation is that if the modeling engine cannot predict a waveform similar to the input data, this algorithm has no chance to converge. There are many reasons why this can happen. In fact, for real data is almost guaranteed it will not happen. That is why FWI is very different in theory and practice.

Notice that we are free to choose different modeling algorithms. Typically in seismic exploration, the modeling is some version of the wave equation. The choice of the modeling engine is what distinguishes tomography (ray tracing) from FWI (finite difference or

Kirchhoff true amplitude) and least-squares migration (Born modelling). Most of the time, FWI is implemented with finite differences in the time domain or by solving the Helmholtz equation in the frequency domain. Let us assume density is constant for simplicity, and use the following wave equation in the time domain:

$$\frac{\partial^2 P_s}{\partial t^2} - v^2 \frac{\partial^2 P_s}{\partial \mathbf{x}^2} = source(t) \tag{2}$$

where I use P_s to denote the predicted wavefield on each point of the subsurface, and $v = v(\mathbf{x})$ is the velocity model, and $\mathbf{x} = (x, y, z)$, that is the space vector. Also, for simplicity I have kept density constant with $\mathbf{x}$. More complex versions of the wave equation can and should be used, including elasticity, anisotropy, density variations, and attenuation. Those refinements would certainly be critical for success in industrial applications but we will not include them here to keep the focus on the key concepts.

The frequency domain version can be obtained simply by transforming the time coordinate to the temporal frequency by the Fourier transform, which converts the second time derivative into a multiplication by ω^2.

$$\frac{\omega^2}{v^2} P_s(\omega, x) + \frac{\partial^2 P_s(\omega, x)}{\partial x^2} = source(\omega) \tag{3}$$

In addition to these two modeling equations, inversion algorithms use the adjoint equations that take the acquired data and create the receiver wavefield P_r.

$$\frac{\partial^2 P_r}{\partial t^2} - v^2 \frac{\partial^2 P_r}{\partial x^2} = 0 \tag{4}$$

$$P_r(x = x_r) = data(t) \tag{5}$$

for the time domain, and

$$\frac{\omega^2}{v^2} P_r(\omega, x) + \frac{\partial^2 P_r(\omega, x)}{\partial x^2} = 0 \tag{6}$$

$$P_r(\omega, x = x_r) = data(\omega, x) \tag{7}$$

for the frequency domain. These two wavefields, P_s and P_r are combined by using the corresponding imaging condition in time or frequency to obtain the reflectivity

$$Image = \int_{shots} \frac{\partial^2 P_s(t, x)}{\partial t^2} P_r(t, x) \tag{8}$$

$$Image = \int_{shots} \omega^2 P_s(\omega, x) P_r(\omega, x) \tag{9}$$

Following the adjoint-state principle, the reflectivity obtained by the imaging condition gives the direction for velocity updating in the first iteration of the inversion. For subsequent iterations, the data are replaced by the residuals to achieve further velocity updates.

$$\mathbf{v_{iter}} = \mathbf{v_{iter-1}} + \alpha \boldsymbol{\Delta} \mathbf{v}, \tag{10}$$

where α is the step size, a value usually calculated by a linear search to find the minimum of the cost function along the gradient. The gradient is the direction in the parameter space along which the residuals will decrease (if the step size is not too large):

$$\Delta v = scale \times Image(r_{iter-1}). \tag{11}$$

Here "scale" is a coefficient that converts reflectivity to velocity and

$$r_{iter-1} = d^{iter-1}_{predicted} - d_{acquired} \tag{12}$$

is the prediction error obtained with the current velocity.

What makes FWI a nonlinear problem is that the modeling operator depends on the unknown velocity. Therefore, in principle, each iteration requires changing the modeling operator. Furthermore, if the initial model is far from the correct solution, the updates could be biased by a poor prediction and create wrong updates that deviate the solution even further from the correct velocity. This problem can be addressed in principle by having a good initial model obtained for example by tomography, that is guaranteeing that the closest minimum of the cost function from the starting point is the global minimum, which is the best possible solution we can achieve given the data we have. However, this FWI nonlinearity becomes stronger and stronger as the frequency of the data increases. This has a mathematical foundation but it can be easily visualized by realizing data predicted with a too slow velocity from an earlier reflector can be mismatched with a true deeper reflector creating the illusion that the model is correct. This is the problem of cycle skipping (Virieux and Operto, 2009). The higher the frequency content of the data, the worse is the nonlinearity. As a consequence a bootstrapping technique is commonly used in the frequency domain, where data are inverted in frequency bands using the final velocity model of one band as the initial velocity model for the next one (Pratt and Worthington, 1990).

IMPLEMENTATION

The starting point for my FWI implementation is a code by Yang et al. (2015) available in the open-source software Madagascar (Fomel et al., 2012). I use Yang's optimization algorithm but apply several changes in the general dataflow and the finite-difference algorithm. The purposes of these changes are to obtain higher modeling accuracy as well as to achieve an object-oriented hybrid parallel implementation using MPI+OPENMPI. For other steps of the workflow that we will discuss below, I use some existing tools and implement new tools, all in Madagascar. The codes implemented for this report are listed in Fig 1, indicating the original codes that served as their inspiration (both from Madagascar and Seismic Unix).

Penliang Yang's implementation, written for teaching purposes, was the motivation for this report since it works surprisingly well given its limitations, which is a 2nd order finite differences scheme with Clayton-Enquist's boundary conditions. These limitations were a consequence of using simplicity for teaching as a guideline, which makes the author of this report very grateful. Open-source code like this is an invaluable resource for learning and further research.

One concept that has influenced much of the geophysical research during the last forty years is inversion by iterative application of forward/adjoint operators (Claerbout, 1992).

In this approach, many different processing-inversion techniques have a very similar layout consisting of an iterative application of forward and adjoint operators. When this concept is coupled with Object Oriented Programming (OOP), we have a recipe to implement many, if not most, of the algorithms used in seismic. This recipe is not only quite beautiful due to its generality and simplicity, but also it perfectly pairs with requirements for hybrid MPI+OPENMP implementations.

Parallel implementations are simpler and more efficient with coarse-grained parallelization. This term implies that communication across nodes is limited to a minimum. As a consequence communication cost does not negate the gains in speed achieved by parallelism. Earlier MPI implementations, 20 years ago, often used fine-grained parallelism due to the memory limitations on each node. Hardware evolution pushed node memory to larger numbers and this allows us to distribute algorithms in more efficient dataflows. The FWI dataflow used in this work is shown in Figure 2.

In this scheme, the data in the shot domain are distributed evenly across nodes. This distribution allows all data to be kept in random access memory (RAM) since the total memory is proportional to the number of nodes. Each node is responsible for performing all modelling/migration steps, which are the main computational burden of the algorithm. The master performs all the optimization algorithm by collecting information from the nodes. This requires a couple of "reduce" and "scatter" MPI operations between master and nodes on each iteration. This is a much smaller burden than spreading the model across nodes which requires heavy communication on each time step. This approach reduces communication overhead by one or two orders of magnitude and eliminates the need for overlapping on model windows, which also would require overlapping in data distribution increasing therefore the size of the survey.

The only drawback of this data decomposition approach is that the master and nodes require enough memory to store a complete velocity model in RAM, in addition to enough memory to contain part of the data (size of data/number of nodes). In modern days, with nodes having hundreds of Gigabytes of RAM this is feasible. However, it does explain why earlier implementations of FWI and Least Squares Migration did instead use model decomposition with a much bigger overhead on communication.

Now let us see where the OOP concept enters this picture. OOP allows us to decompose the algorithm into logical units (classes). Each class has its methods and information that makes sense for the part of the problem that the class is designed for. These classes can be maintained, upgraded and reused in different problems, independently of each other. The encapsulation properties of OOP becomes essential for code maintenance. In the implementation of FWI I used the following classes:

- Driver class (master and slaves): contains the dataflow, including the MPI methods to read and distribute the data across nodes and save the output.
- Parameter class (master and slaves): contains the user interface and all parameters required in different stages of the algorithm. Also, it contains all validations methods to assure parameters are consistent.

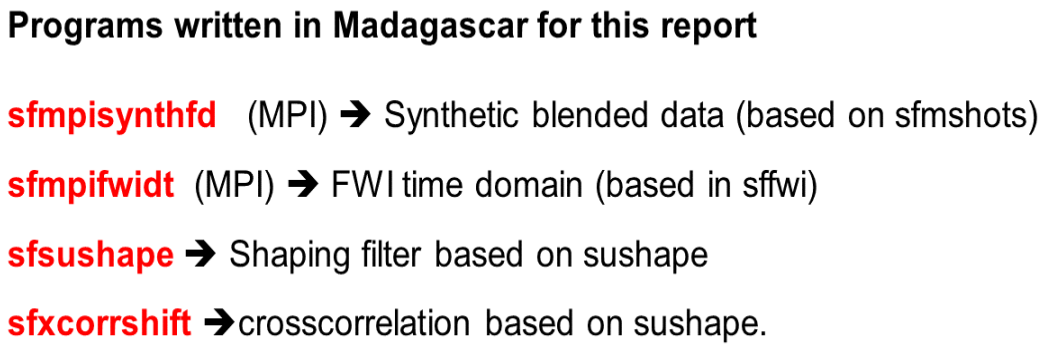

FIG. 1. List of programs written for this report in C++ using the Madagascar framework. On the right I indicate the codes that served as starting point for each module ("sf*" are Madagascar codes, "su*" are Seismic Unix codes.

- Optimization Solver (master only): contains an iterative algorithm that calls the forward and adjoint operators implemented in the migration class. Upgrades in optimization algorithms, even changes in cost functions, are localized in this class.
- Migration operator (slaves only): contains the geophysical part of the problem, for example, a forward finite different modelling and reverse time migration algorithm. Different methods can be used without changing the rest of the implementation (for example, RTM, Kirchhoff, PSPI, etc).

The operator can be parallelized on its own to take advantage of shared memory and also can be optimized for GPUs using CUDA. Therefore, we can speed up each iteration by upgrading nodes with more cores or GPUs (faster modeling for each shot), or by modelling more shots simultaneously (adding more nodes). This implementation has the property of almost perfect scalability, which is a key goal for high-performance computing by parallelization. Once the number of shots is as least as large as the number of nodes, the computation time will increase linearly with the number of shots. The breaking point in this scalability is when RAM is not large enough to hold the model. This can happen in other problems, like 3D least-squares migration with angle gathers in offset and azimuth (5D models). For FWI, the model is usually 3D and nodes can easily contain the whole model (typically nodes have 128 Gb of memory or more, a typical 3D model will only be a few gigabytes).

INVERSE CRIMES

A common issue when working with synthetic data is to achieve over-optimistic results as a consequence of the so called "inverse crime". This is the consequence of using the same engine for both data synthetics and data predictions (Schuster, 2017). The inverse crime is extremely effective at eliminating difficulties that will normally appear with real data. I will illustrate this statement by starting on an inverse crime scenario and moving gradually away from it by changing the examples.

The FWI implementation used in the following tests uses a finite difference engine with 4th or 8th orders in space and an absorbing boundary condition (ABC) in the form

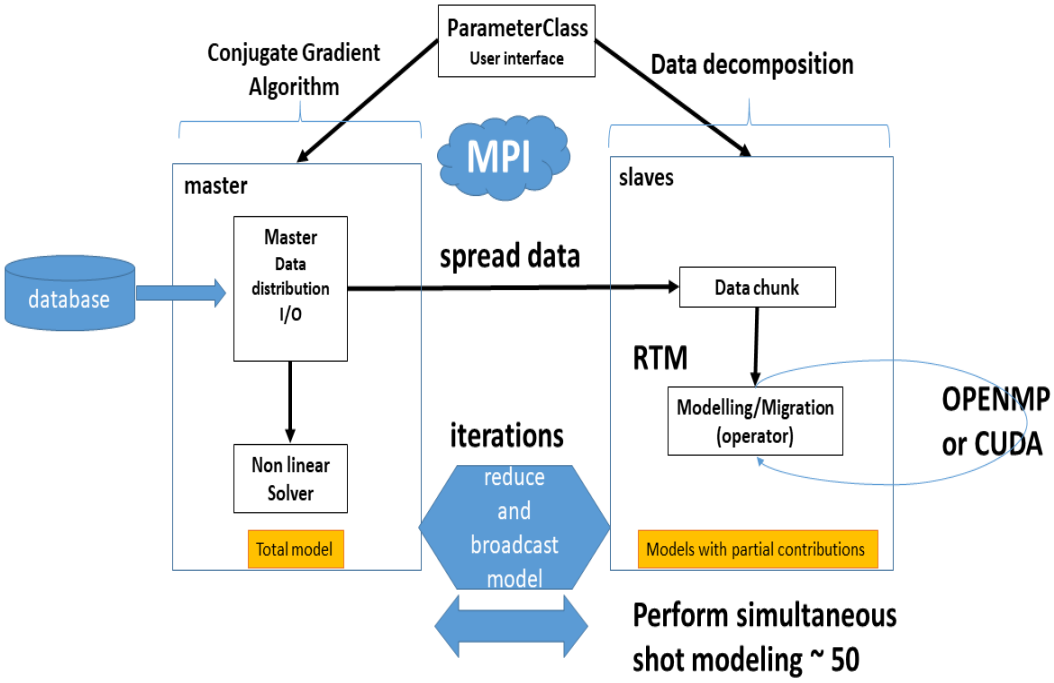

FIG. 2. Hybrid MPI-OPENMPI (or CUDA) parallel implementation.

of a sponge of 30 cells wide on each side of the model (except the top, to allow surface multiples for reasons to be discussed later).

As a first step I will illustrate the inverse crime scenario situation. In Figure 3 we see a test where the synthetic data were created by using a coarse modeling grid of 32×32 meters and a low frequency wavelet with a dominant frequency of 4Hz. I use 40 synthetic shots as input, one of which is shown in Fig 3a. A decimated version of the Marmousi velocity model was used to generate these shots. Smoothing was applied to address some potential aliasing before decimation but essentially these shots were generated in the "true" velocity model (Figure 3b), where the word "true" is in the context of synthetic experiments. Figure 3c shows the starting model and 3d shows the final velocity model obtained in only 20 iterations.This could look like a successful result but it is not and let us discuss why.

One of the hot-topics on FWI research these days is high-resolution FWI. If a high frequency velocity model can be achieved by FWI, this model can be used to estimate the reflectivity directly without migration. There are many advantages of this approach, in particular the possibility to bypass many or most processing steps, but its drawback is the computational costs which can be much higher than the traditional approach of FWI followed by migration or LSMIG.

In Figure 4 we see my first attempt to increase the data bandwidth by increasing the dominant frequency for the source from 4Hz to 8Hz. One shot gather is shown in Figure 4a and the initial model apperas in Figure 4c). To address finite difference constraints, I have reduced the cell size from 32 meters to 16 meters. Comparing with the previous

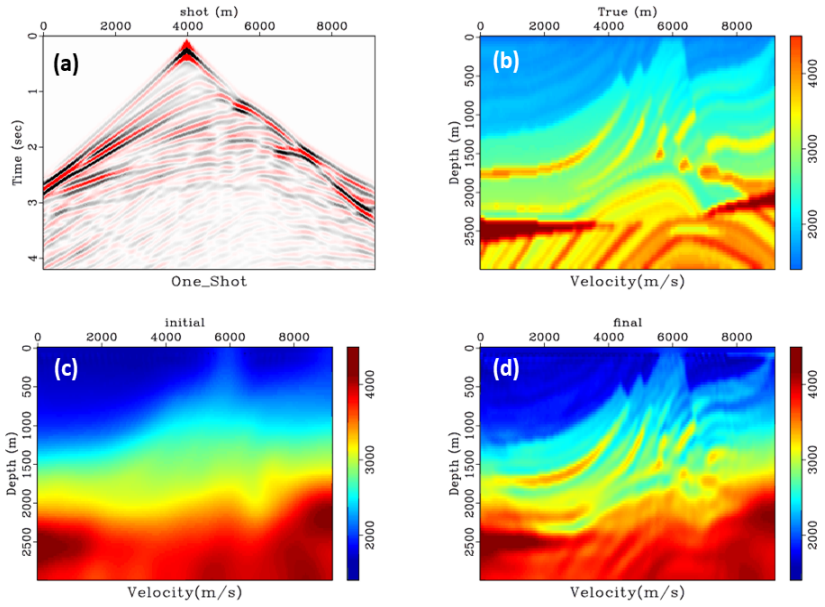

FIG. 3. Results using a coarse grid of 32x32m and a low frequency data set generated on the same coarse grid with a 4Hz dominant frequency Ricker wavelet.

test, we notice that the results after 20 iterations have degraded significantly on the shallow part (Figure 4d). Notice that we are still in an inverse crime scenario but we are facing the cycle-skipping (non-linearity) problem. As mentioned before, frequency domain implementations use a multigrid approach, solving in bandwidths with increasing frequencies and updating the initial velocity model on each band. These tests are performed in the time domain, so we need to implement a time domain multigrid approach.

Let us replicate this in the time domain by solving the FWI problem with increasing frequency data. In the time domain, however, there is one more factor to take into account: cell sizes have to decrease with increasing frequency because of finite difference constraints. On the other hand, making cells as coarse as possible for each band allows one to achieve better convergence (because there are less unknowns in the inversion) and (much) faster computation (we need less number of operations to produce the wavefields). Figure 5 shows this approach starting at a coarse grid with $32m$ and a dominant frequency of $4Hz$ (Figure 5a) and obtaining the model in Figure 5b after 20 iterations. Then we interpolate the obtained velocity to $16 \times 16m$ and use this model as the starting velocity for inverting input data with a dominant frequency of $8Hz$ (Figure 5c). Finally we interpolate this model to $8 \times 8m$ cells and use this as starting velocity to invert data with a higher dominant frequency to $16Hz$ (Figure 5d). The increasing in resolution is noticeable as we go from 4 to 8 and to 16 Hz dominant frequencies, and 32 to 16 and to 8 meters cell size.

However, although these results may look successful, they have a serious flaw: they rely on the inverse crime since the data for the inversion of each frequency band on each stage have been generated with the frequency and cell size required for that particular band (Figure 6). To move away from the inverse crime scenario in this synthetic experiment, we need to work with only one data set generated with the full frequency band, in the supporting cell grid size and reduce the frequencies as needed on each stage. This can be

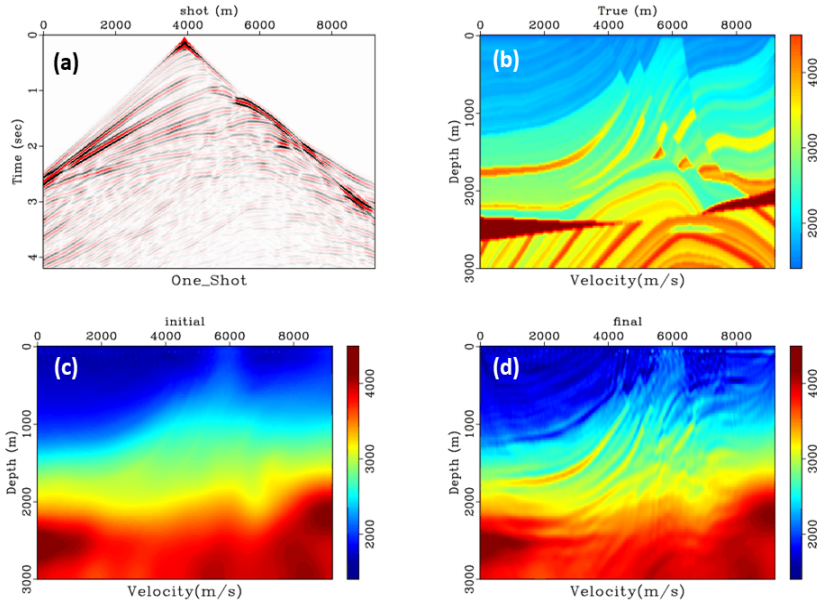

FIG. 4. First attempt to increase resolution by doubling the dominant frequency of the source to 8Hz and generating the data in a finer velocity grid of 16x16m.

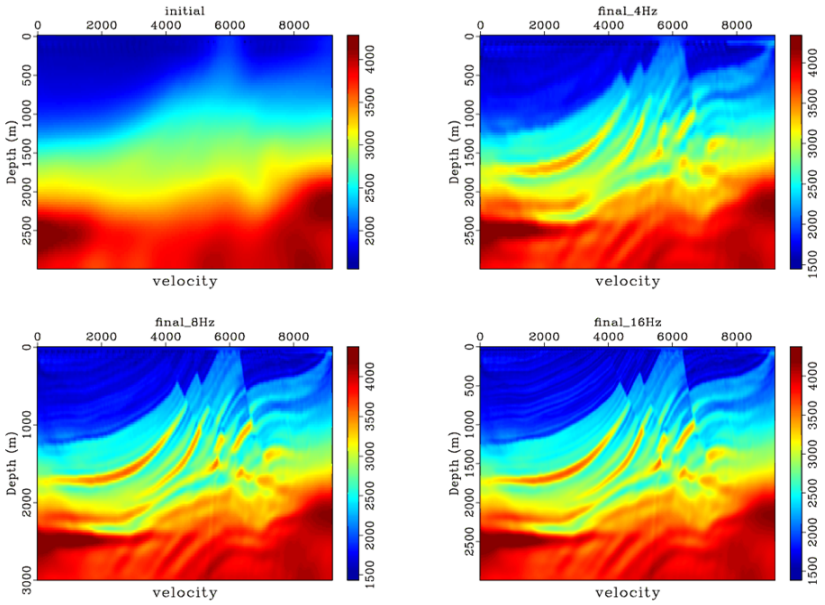

FIG. 5. Multigrid results (3 stages) in the inverse crime scenario. We transition from 4Hz dominant frequency to 8Hz and then to 16Hz by using 32m cell size, then 16m cell size and finally 8 m cell size.

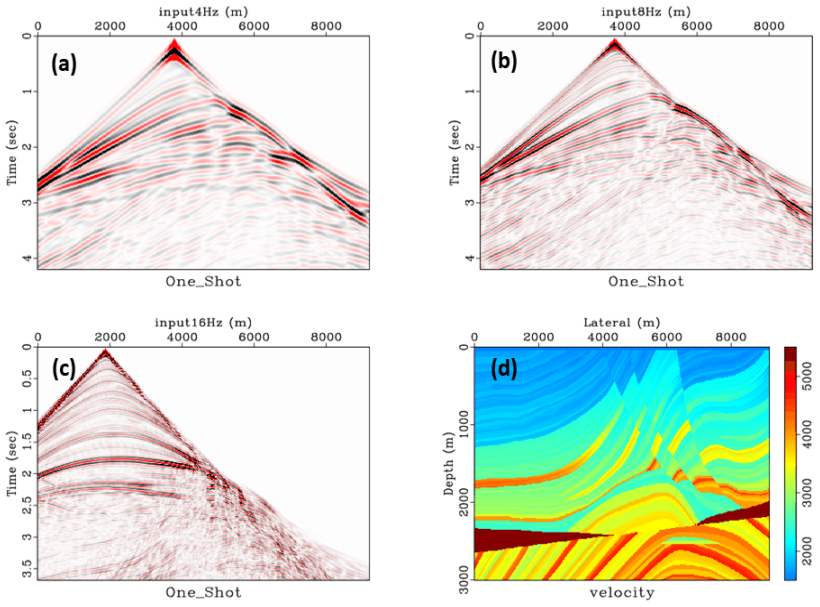

FIG. 6. Input data used to obtain the results in Figure 5. Each stage used data generated in a grid with the same cell size and dominant frequency as the corresponding inversion.

done, for example, by using a bandpass filter at each stage. As a first attempt, we can apply a simple band pass filtering (Butterworth) at each band. However, we can see in Figure 7 that this approach fails, since results fall apart compared to the models shown in Figure 5c. This failure teaches us how deeply rooted in the inverse crime our previous results were. The bandpass filtering distorts the source wavelet enough to prevent a perfect match between data and predictions. Besides, finite difference artifacts present on the synthetic data are different from artifacts generated during the inversion since they are dependent on cell size and thickness of boundary conditions. In other words, we moved away from the inverse crime scenario for the first time in this report, the results fall apart completely. This illustrates that the inverse crime has a tremendous effect on synthetic experiments and we should be careful to prevent it or at least attenuate its effects.

A workflow without inverse crime (almost)

The workflow presented in the previous section could work but we need to be careful on how to separate the data in bands to preserve a waveform matching with the data predicted by finite differences. Among several options, a relatively straightforward method is a shaping filter. This filter can be estimated by least-squares fitting between the wavelet in the data after filtering and the wavelet injected during finite differences. We can solve this by least-squares minimization of the following cost function:

$$J = \|\mathbf{wavelet_{FD}} - (\mathbf{wavelet_{data}}) * \mathbf{filter}\| \tag{13}$$

Here we are seeking to minimize the energy of the differences between the wavelet we use for forward modeling during the inversion ($wavelet_{FD}$) and the filtered wavelet we used to generate the synthetic data (or our best wavelet estimation for real data). Notice that if

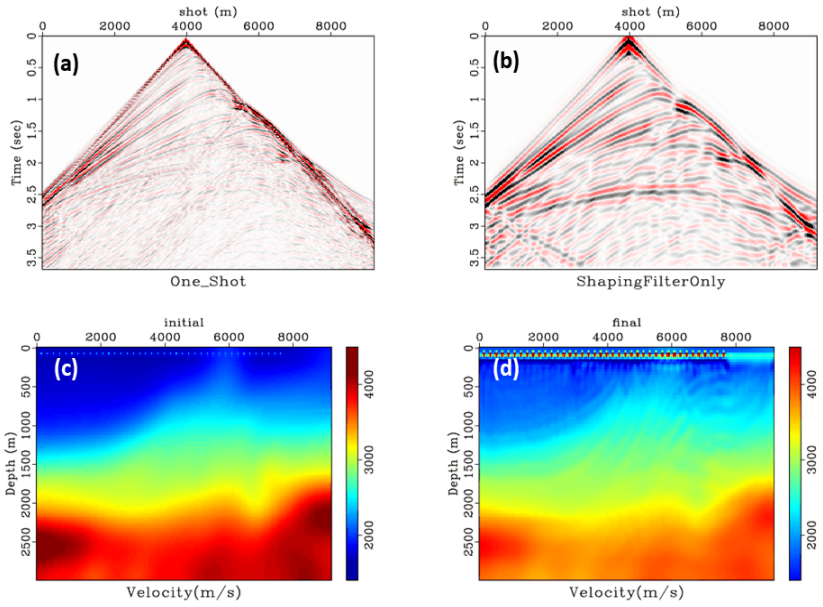

FIG. 7. Low frequency inversion of high frequency data after band pass filter. Inversion fails because a simple band pass filter changes the waveforms enough to destroy data-modeling matching.

we do this with the data instead of the wavelet, the filter would be too powerful and would take information away from the inversion. Instead, by estimating the filter from the source wavelets (one estimated and one chosen) we don't take away from the inversion the sensitivity required to estimate velocities. In addition, doing the matching with the predicted data from the true model we would be falling again into the inverse crime scenario. With these choices, a solution to equation (13) is the well known shaping filter:

$$filter = \frac{wavelet_{input} \times wavelet_{FD}}{wavelet_{input} \times wavelet_{input}} \tag{14}$$

In Figure 8a we see a seismic trace chosen randomly from the input dataset. This trace was obtained by finite differences with a Ricker wavelet with a dominant frequency of $16Hz$ (roughly with frequencies between 1-25Hz). To the right, in Figure 8b we see the corresponding seismic trace predicted by the finite-difference algorithm at a band with $4Hz$ dominant frequency and $32m$ cell size. A simple Butterworth bandpass filter applied to the input produces distortions in the phase (Figure 8c) that are sufficient to hinder convergence. Instead, the shaping filter result in Figure 8d has a better matching. Some differences with the prediction can still be observed and will be discussed later.

A similar plot is shown in Figure 9 but for all the traces in one of the 40 input shots. Notice that there are still differences between filtered high-frequency data and predicted low-frequency data. (Figures 9d and 9b respectively). This is caused by the different performance of the absorbing boundary conditions for different grid cell size (the number of cells on the ABC are kept constant, which make it ticker for coarse cells). A different choice could be made, but we want to have differences between data and predictions to depart from the inverse crime.

By applying this filter to the input data, we can match the frequency content and phase

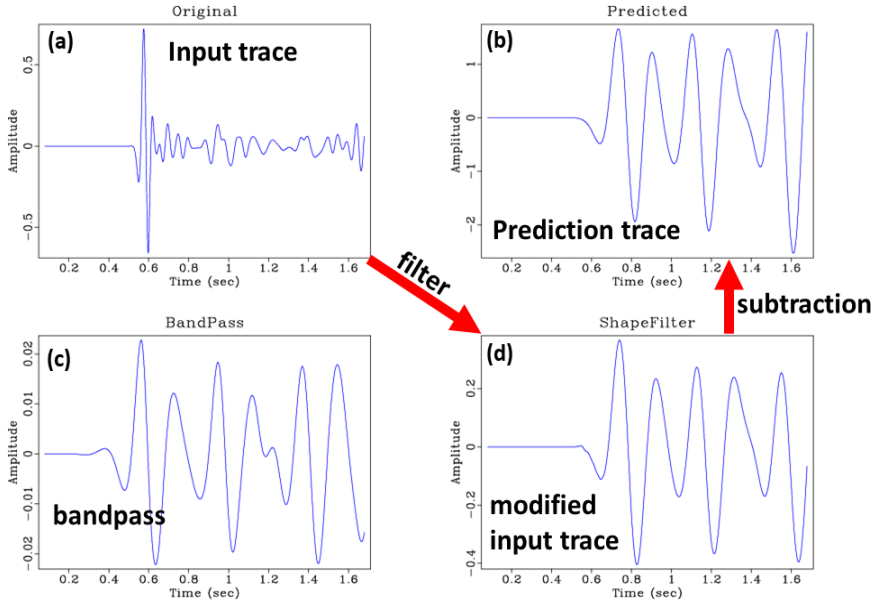

FIG. 8. Trace comparisons for original data (a) after bandpass (c) and shaping filter (d) with FD prediction in (b)

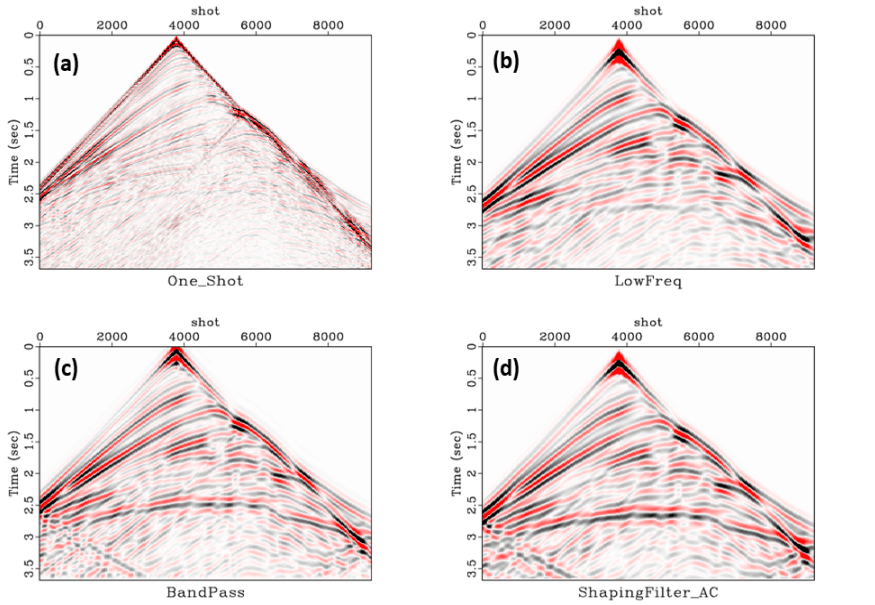

FIG. 9. Comparison of different inputs to the workflow: a) input shot, b) predicted shot at the first stage (4Hz dominant frequency, 32x32m), c) bandpass filtered version of a), shaped filtered version of a)

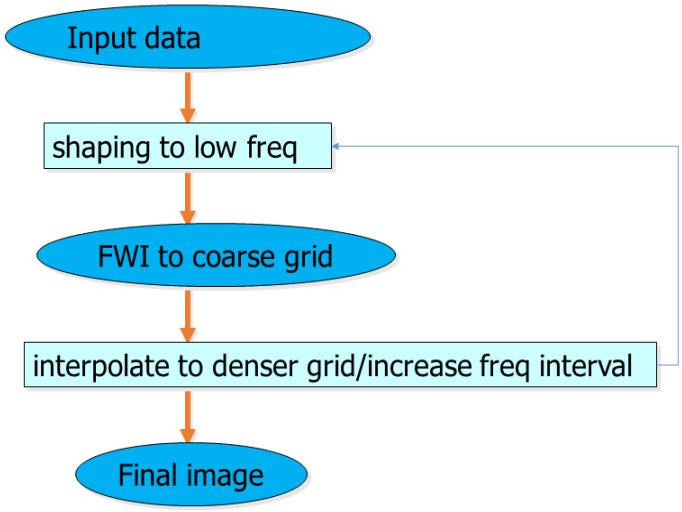

FIG. 10. Dataflow (first version) for the time domain multigrid approach.

for the predictions at each band. The first version of this dataflow is illustrated in Figure 10. This allows us to obtain similar results (almost) as good as those obtained via inverse crime. Figure 11 shows the input data, filtered data, initial model and final model after 20 iterations. These results look very similar to the results shown in Figure 3 but this time there was (almost) no crime!

There are still some possible improvements to this filter. First, some minor benefits can be obtained by including in the shaping filter an overall scalar. This can be done shot by shot or in smaller data subsets. Figure 12 shows the equivalent of Figure 11 with this minor change. We see some better amplitude balancing in the shallow part.

Going a bit further, other corrections can be used to address minor overall phase distortions. An additional phase correction can be applied by calculating the cross-correlation between the predictions and the filtered data and using the time of the cross-correlation peak to shift the traces vertically. Figure 13 shows the new dataflow, which has now the additional cross-correlation shifts after the shaping filter. In Figure 14 we can see the maximum of these cross-correlations for one shot. Notice the time scale is exaggerated because time shifts are very small.

Figure 15 shows now the results with the new dataflow. There are significant benefits, in particular at the shallow. However, we should note that by using the predictions in the final grid to calculate the cross-correlations we are getting back into using information we shouldn't have. One way around this is to add the cross-correlation shifts in a final stage after the velocity model is close enough to the true solution.

Finally, Figure 16 shows the final multigrid results as we move from band to band. Comparing 16 with Figure 4 we see significant progress on achieving HighRes FWI.

Input is high frequency, shaping filter applied

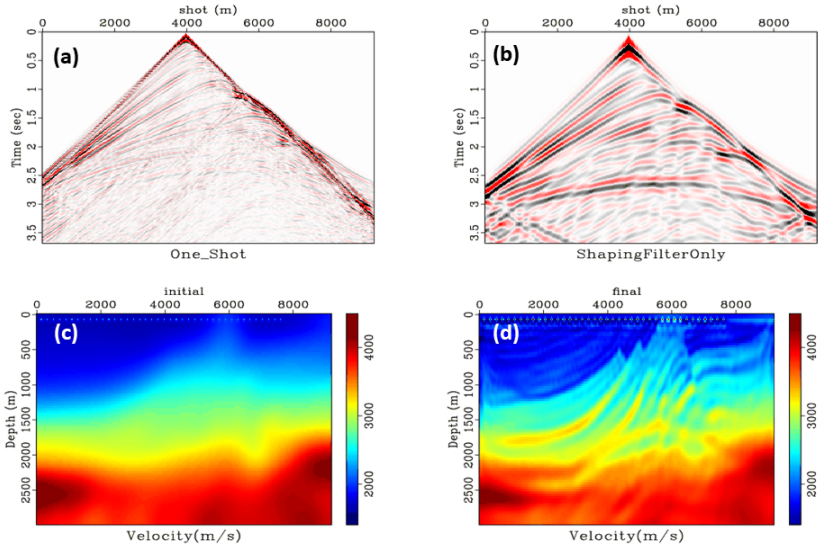

FIG. 11. Results with shaping filter

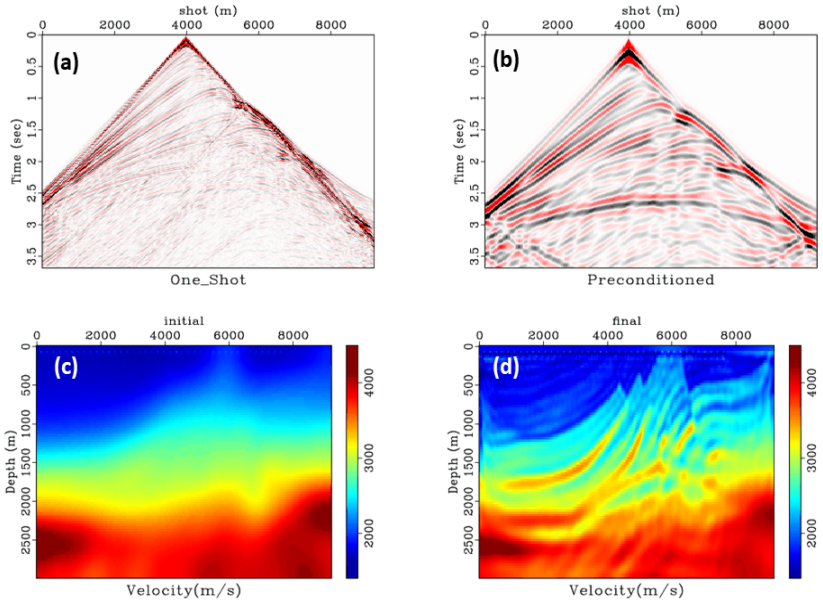

FIG. 12. Results with shaping filter and global scalar

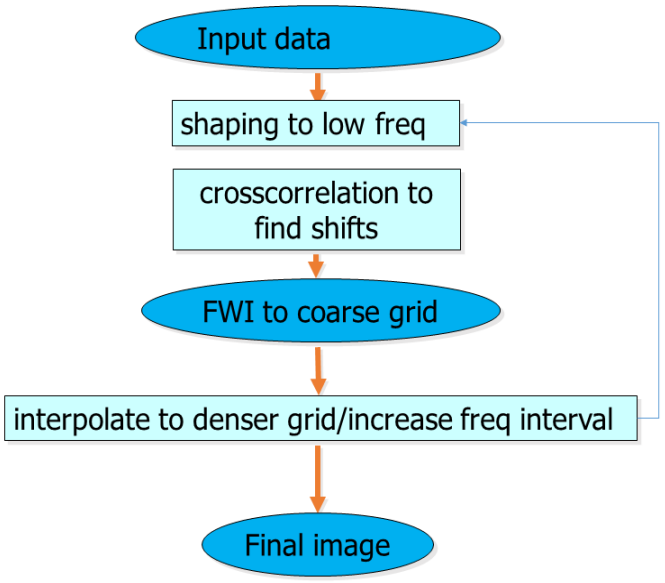

FIG. 13. Dataflow (final version) for preprocessing.

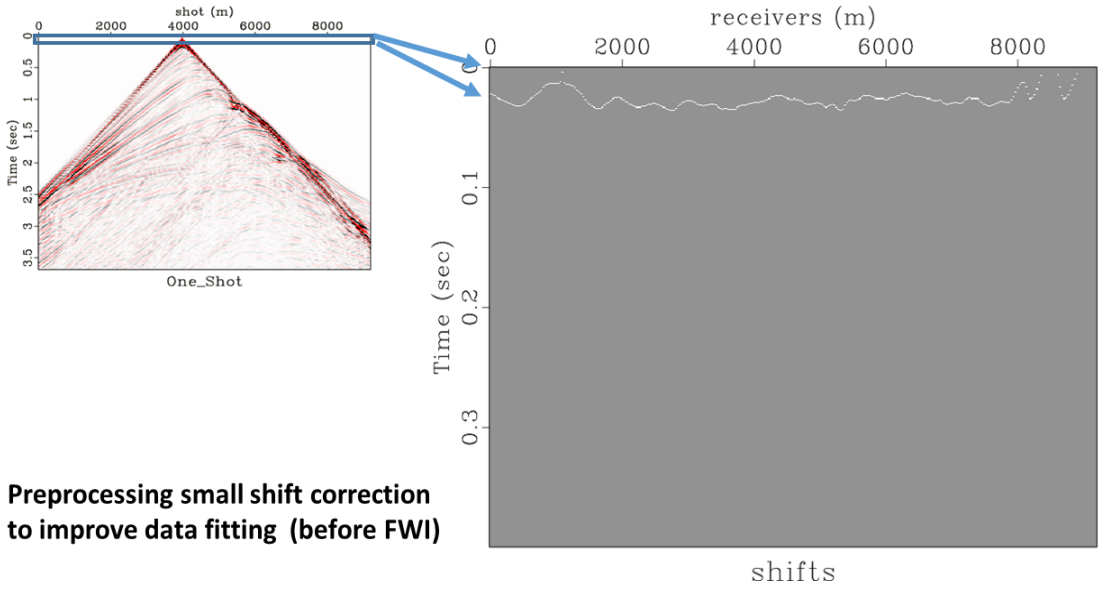

FIG. 14. Time locations for the peaks of the cross correlation function (filtered data vs predicted data) for one shot.

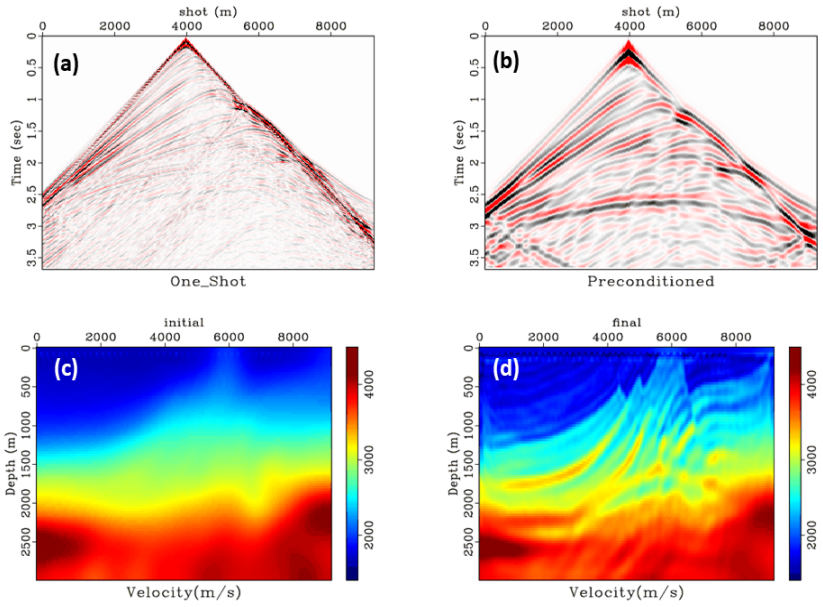

FIG. 15. Results with shaping filter, global scalar and cross-correlation

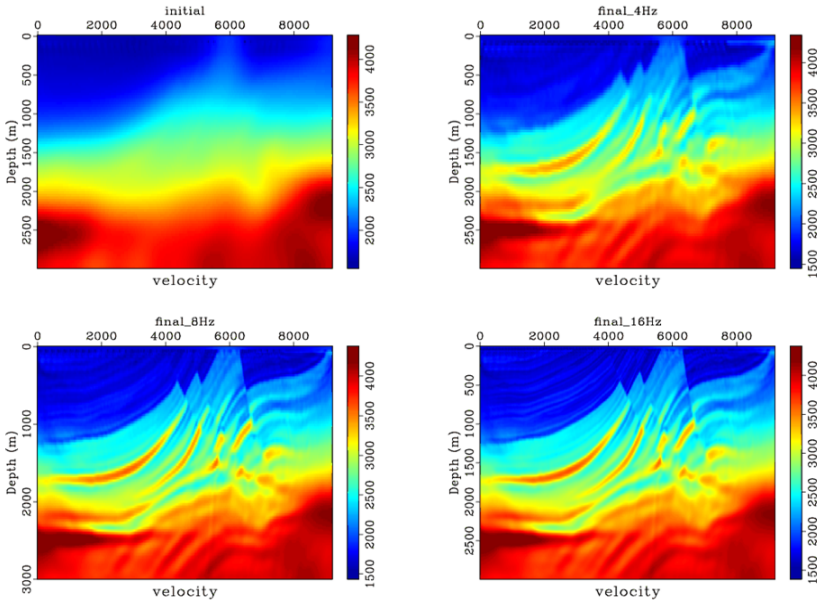

FIG. 16. Final multigrid results with the shaping filter approach. As before we have achieved inversion of high frequency data (0-25Hz) by going through 3 stages.

COMPUTATION TIMES AND EXTENSIONS

It is not easy to use actual computing times to evaluate performance since running times depend on hardware as much as on software. However, just as an indication of how times change as we go to finer scales, I show in Table 1 the running times in a 10 node cluster, each node with 4 cores running with hyper-threading (8 threads). The program is a hybrid OPEMPI-OPENMP, implemented in such a way that there is freedom on how to combine them. In this particular test, I set each node to run as two slaves, and each slave to execute the FD algorithm with 4 threads. We can see how as we go to higher frequency bands the running times expand by several factors: the number of cells multiplies by 4 (2 in each direction), and the number of time steps has to be increased as a consequence of FD limitations demanding a smaller time step. Assuming doubling the dominant frequency we should roughly expect an increase of 8 in the computation time from band to band. On the other hand, we are doing a very simple implementation of the wave equation, order 4 along space and 2 along time. To account for example for variable density, we would need a staggered grid with 3 times more operations. Elastic propagation would require roughly around 5 times more operations. A 3D inversion would require around 2 to 3 orders of magnitude more computations since it would scale with the number of cells along y as well as x.

Another factor as we move to larger surveys and 3D would be the memory requirements. For 2D the RAM requirements are minimum so we can put as many shots as needed on each node. A 3D survey would limit how many shots we could run on each node.

Table 1. Computation times

model size	cell size	time steps	nshots	iterations	time
96 x 288	32, 32	2800	40	20	196 secs
188 x 576	16, 16	2800	40	20	576 secs
376 x 1151	8, 8	4600	40	20	4481 secs

As mentioned earlier, the cluster in which I am running these tests is very low cost because it does not have a very fast intra-node communication. This does not affect much the time domain performance, since communication across nodes is kept to a minimum. A frequency domain version would require a more costly cluster to run efficiently.

CUDA implementation

An improvement on the parallelization described above is to implement the finite difference code on each node to run in GPUs using CUDA. With the rise of machine learning as a powerful computational tool, GPUs have gone through a rapid increase in power over the last decade. Machine learning is only competitive with traditional techniques by the use of massively parallel architectures and in particular GPUs. These devices are designed to launch thousands of threads simultaneously, each one very lightweight (see Figure 17a). Of course, GPUs also can be and are used to do traditional algorithmic computations. Using these thousands of threads that GPUs provide, however, requires to move the memory contents from the host (where the CPUs have access) to the global or shared memories (that GPUs can access) (Figure 17b). This communication is time consuming and can eas-

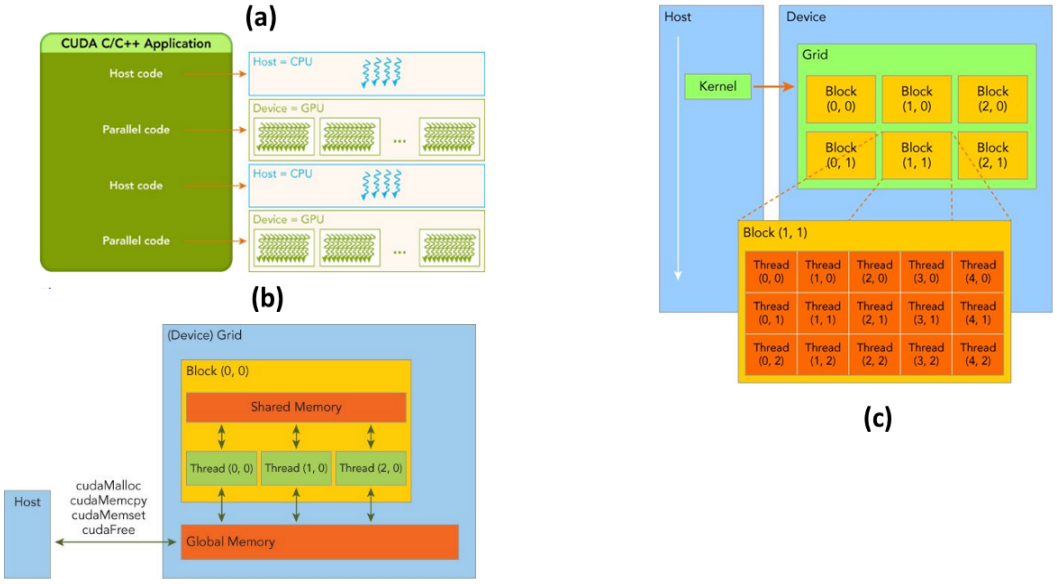

FIG. 17. GPU computational model, a) multithreading, b) shared vs global memory, c) grid of blocks) (Cheng et al., 2014) .

ily eliminate the efficiency gains achieved by the massive multithreading. In addition, to achieve maximum efficiency on the calculations, a shared memory (common across threads in each block) should be used. Therefore, efficient computation is a balancing act of communication across different threads which needs to be synchronized across shared memory, global memory and main RAM (Figure 17c). However, when all these pieces are properly put together, efficiency gains of one or two orders of magnitude are common. For example, simulating 10 shots on 12 threads of an hexacore machine (using hyperthreading and OPENMP) took 18 seconds (1.8 seconds per shot). The same operation on a GPU 2070 took 3.04 secs (0.3 seconds per shot). These times are for second-order in space. When comparing 4th and 8th order in space, differences become even more dramatic (two orders of magnitude) since CPU times increase by 10 and 50 respectively, but GPU times remain the same (0.3 seconds per shot).

DISCUSSION

One can argue that for the results shown in this report, we are still relying somewhat on the inverse crime scenario since data has been generated with the same engine (finite differences) as the input data. Although that is true, there are several factors that reduce this problem. When operating in the time domain approach, with different cell sizes and different source wavelets, many details change during the wavefield simulation. As mentioned above, the ABC are different (different thickness). The source is different. Multiples are different. As a consequence, the finite difference artifacts are very different and they do not help the inversion as they do when keeping the grid and frequencies unchanged. Certainly, we are still facing some bias comparing to what real data look like, but it is a very large step in the right direction and a must for synthetic testing.

CONCLUSIONS

In this report, we have seen an FWI implementation and dataflow that permits one to obtain High-Resolution FWI. This approach mimics the multigrid approach commonly used in Frequency Domain FWI. Because of differences between FD in the time domain and the solution of the Helmholtz equation, the multigrid approach requires some preprocessing to accommodate the waveform of the data to the waveforms predicted on each band. As a very interesting effect of this multigrid approach, we partially cure the inverse crime biased introduced when working with synthetic data.

ACKNOWLEDGMENTS

I thank Sam Gray for many fruitful discussions, Penliang Yang for his enormous contribution with Madagascar examples, and CREWES sponsors for contributing to this seismic research. I also gratefully acknowledge support from NSERC (Natural Science and Engineering Research Council of Canada) through the grants CRDPJ 461179-13, CRDPJ 543578-19 and NSERC Discovery Grant.

REFERENCES

Cheng, J., Grossman, M., and McKercher, T., 2014, Professional CUDA C Programming: Wrox Press Ltd., GBR, 1st edn.

Claerbout, J., 1992, Earth sounding analysis, Processing versus inversion: Blackwell Scientific Publications, Inc.

Fomel, S., Sava, P., Vlad, I., Liu, Y., Jennings, J., Browaeys, J., Bashk ardin, V., Godwin, J., Song, X., and Hennenfent, G., 2012, Madagascar software package and reproducible research.

Pratt, R. G., and Worthington, M., 1990, Inverse theory applied to multi-source cross-hole tomography. part 1: Acoustic wave-equation method 1: Geophysical prospecting, **38**, No. 3, 287–310.

Schuster, G. T., 2017, Seismic inversion: Society of Exploration Geophysicists.

Virieux, J., and Operto, S., 2009, An overview of full-waveform inversion in exploration geophysics: Geophysics, **74**, No. 6, WCC1–WCC26.

Yang, P., Gao, J., and Wang, B., 2015, A graphics processing unit implementation of time-domain full-waveform inversion: Geophysics, **80**, No. 3, F31–F39.