GEOPHYSICS, VOL. 79, NO. 2 (MARCH-APRIL 2014); P. S47–S61, 13 FIGS., 8 TABLES.
10.1190/GEO2013-0214.1

Closing the performance gap between an iterative frequency-domain solver and an explicit time-domain scheme for 3D migration on parallel architectures

H. Knibbe[1], W. A. Mulder[2], C. W. Oosterlee[3], and C. Vuik[1]

ABSTRACT

Three-dimensional reverse-time migration with the constant-density acoustic wave equation requires an efficient numerical scheme for the computation of wavefields. An explicit finite-difference scheme in the time domain is a common choice. However, it requires a significant amount of disk space for the imaging condition. The frequency-domain approach simplifies the correlation of the source and receiver wavefields, but requires the solution of a large sparse linear system of equations. For the latter, we use an iterative Krylov solver based on a shifted Laplace multigrid preconditioner with matrix-dependent prolongation. The question is whether migration in the frequency domain can compete with a time-domain implementation when both are performed on a parallel architecture. Both methods are naturally parallel over shots, but the frequency-domain method is also parallel over frequencies. If we have a sufficiently large number of compute nodes, we can compute the result for each frequency in parallel and the required time is dominated by the number of iterations for the highest frequency. As a parallel architecture, we consider a commodity hardware cluster that consists of multicore central processing units (CPUs), each of them connected to two graphics processing units (GPUs). Here, GPUs are used as accelerators and not as an independent compute node. The parallel implementation of the 3D migration in frequency domain is compared to a time-domain implementation. We optimize the throughput of the latter with dynamic load balancing, asynchronous I/O, and compression of snapshots. Because the frequency-domain solver uses matrix-dependent prolongation, the coarse-grid operators require more storage than available on GPUs for problems of realistic size. Due to data transfer, there is no significant speedup using GPU-accelerators. Therefore, we consider an implementation on CPUs only. Nevertheless, with the parallelization over shots and frequencies, this approach could compete with the time-domain implementation on multiple GPUs.

INTRODUCTION

The oil and gas industry makes use of computational intensive algorithms such as reverse-time migration (RTM) and full-waveform inversion to provide an image of the subsurface. The demand for better resolution increases the bandwidth of the seismic data and leads to larger computational problems. At the same time, high-performance computer architectures are developing quickly by having more and faster cores in the central processing units (CPUs) or graphics processing units (GPUs). The increase in the number of cores requires the development of scalable algorithms.

The finite-difference solution of the constant-density acoustic wave equation has become a common tool for RTM, usually discretized by high-order finite differences in space and second-order differencing in time. The discretization leads to a fully explicit method. Higher-order finite differences reduce problem size compared to low order because they require fewer grid points per wavelength (Alford et al., 1974) if the underlying model is sufficiently

Manuscript received by the Editor 9 June 2013; revised manuscript received 24 November 2013; published online 17 February 2014.
[1]Delft University of Technology, Delft Institute of Applied Mathematics, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft, The Netherlands. E-mail: hknibbe@gmail.com; c.vuik@tudelft.nl.
[2]Delft University of Technology, Department of Geoscience and Engineering, Faculty of Civil Engineering and Geosciences, Delft, The Netherlands and Shell Global Solutions International BV, Rijswijk, The Netherlands.
[3]Delft University of Technology, Delft Institute of Applied Mathematics, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft, The Netherlands and Centrum Wiskunde & Informatica, Amsterdam, The Netherlands. E-mail: c.w.oosterlee@cwi.nl.

smooth, which is usually the case in RTM. Explicit methods based on finite differences exhibit natural parallelism because the computation of one point in space for a given time step is independent of its neighboring points. They can be easily parallelized with OpenMP on shared-memory architectures and on GPUs (Micikevicius, 2009). We refer to the paper by Fu et al. (2012) for an overview.

Migration of seismic data is commonly carried out in the time domain. The classic RTM algorithms in the time domain are known to be computationally and I/O intensive (Ji et al., 2012; Liu et al., 2012) because the forward- and time-reversed wavefields have to be computed and stored. If the correlation between these fields is carried out during the time-reversed computation of the receiver data, only snapshots of the forward wavefield have to be stored.

There are two main strategies to reduce the overhead of storing snapshots. Reconstruction of the forward wavefield by marching backward in time using the last two wavefields is difficult, if not impossible, in the presence of absorbing boundary conditions. Clapp (2009) has proposed to circumvent this problem by only storing boundary values of the snapshots or by using random instead of absorbing boundaries. In the latter case, the energy of the wavefield entering the boundary is scattered and does not stack during the imaging condition. With checkpointing (Griewank and Walther, 2000; Symes, 2007), the forward wavefield is only stored intermittently at pairs of subsequent time steps. During the reverse-time computations and correlation, the missing time steps are recomputed by forward time stepping from stored snapshots over a relatively short time interval. These methods represent a trade-off between storage and computational time.

A second strategy is based on reducing the time needed to write the snapshots to disk, for instance, by asynchronous I/O (Cabezas et al., 2009) and wavefield compression. For the last, standard libraries with Fourier transformation or wavelet compression can be used (Ji et al., 2012).

Migration in the frequency domain is historically less mature because of the necessity to solve a sparse indefinite linear system of equations for each frequency, which arises from the discretization of the Helmholtz equation, whereas in the time domain, the discretization of the wave equation in space and time leads to an explicit time marching scheme. An important advantage of migration in the frequency domain is that the crosscorrelation needed for the imaging condition becomes a simple multiplication. As a result, no wavefields have to be stored. Parallelization over frequencies is natural. If a direct solver is used to compute the solution of the sparse matrix, typically a nested-dissection LU-decomposition is applied (George and Liu, 1981). When many shots need to be treated, the frequency-domain solver in two dimensions can be more efficient than a time-domain time-stepping methods (Marfurt and Shin, 1989; Mulder and Plessix, 2002) because the LU-decomposition can be reused for each shot as well as for each reverse-time computation. Also, lower frequencies can be treated on coarser meshes.

In 3D, however, frequency-domain migration is considered to be less efficient than its time-domain counterpart. One of the reasons is the inability to construct an efficient direct solver for problems of several millions of unknowns (Operto et al., 2007). Wang et al. (2010, 2011) have proposed a direct solver based on nested-dissection that compresses intermediate dense submatrices by hierarchical matrices.

An iterative solver is an obvious alternative; for instance, the one with a preconditioner that uses a multigrid method to solve the same wave equation but with very strong damping (Erlangga et al., 2006; Riyanti et al., 2006; Plessix, 2007; Knibbe et al., 2011). This method, however, needs a number of iterations that increases with frequency, causing the approach to be less efficient than a time-domain method. Note that the iterative method requires a call to the solver for each shot and each reverse-time computation, so the advantage of reusing a LU-decomposition is lost. This approach is parallelized by Riyanti et al. (2007). Knibbe et al. (2013) use GPUs to speed up the computations.

However, with the development of iterative methods on the one hand and hardware accelerators on the other hand, we have to reconsider the performance of migration in the frequency domain. As a parallel architecture, we consider a commodity hardware cluster that consists of multicore CPUs, each of them connected to two GPUs. In general, a GPU has a relatively small amount of memory compared to the CPU.

A GPU can be used in two different ways: as an independent compute node replacing the CPU or as an accelerator. In the first case, the algorithm is split into independent subproblems that are then transferred to the GPU and computed separately. To achieve the best performance, the data are kept on the GPU when possible. We have exploited this way of using a GPU for the Helmholtz equation earlier (Knibbe et al., 2011, 2013).

In the second case, the GPU is considered as an accelerator, which means that the problem is solved on the CPU while offloading the computational intensive parts of the algorithm to the GPU. Here, the data are transferred to and from the GPU for each new task. In this paper, we focus on the second approach.

The aim of this paper is to demonstrate that migration in the frequency domain, based on a Krylov solver preconditioned by a shifted-Laplace multigrid preconditioner on CPUs, can compete with RTM in the time domain on commodity parallel hardware, a multicore CPU connected to two GPUs.

We will make a comparison in terms of computational time, parallelization, and scalability aspects. We use a finite-difference discretization of the constant-density acoustic wave equation for computing the wavefields. Here, we solve the 3D wave equation in the frequency domain with the iterative Helmholtz solver described by Knibbe et al. (2013). This solver reduces the number of iterations by a complex-valued generalization of the matrix-dependent multigrid method. The price paid for improved convergence is that the implementation is no longer matrix-free. The matrix-dependent prolongation requires the storage of the coarse-grid operators. As a result, the use of a GPU as an independent computer node becomes less attractive for realistic problem sizes. Applying GPUs as accelerators involves substantial data transfer, requiring a significant amount of time and reducing the speedup as compared to parallel computations on CPUs. For that reason, we will only consider a CPU implementation for the Helmholtz equation. For the migration in the time domain, this problem disappears because of the explicit time stepping and we can exploit GPUs as accelerators. Complexity estimates show that both approaches scale in the same way with grid size, but that does not give an indication of the actual performance on a problem of realistic size. We therefore made a comparison of our actual implementations for the frequency and time domain.

We will describe seismic modeling and migration in the time and in the frequency domain. The advantages of the frequency-domain solver are explained and demonstrated. We review the parallel

strategies for time and frequency domain and describe implementation details on multicores and on GPUs. Finally, we compare the parallel performance on two 3D examples.

CHOICE OF METHOD

The choice of the numerical scheme is motivated by complexity analysis. Consider a 3D problem of size $N = n^3$, with n_s shots, n_t time steps, n_f frequencies, and n_{it} iterations. The number of shots is usually $n_s \sim n^2$, the number of time steps $n_t \sim n$ (see equation 6), the number of frequencies is $n_f \sim n$ at most, and the number of iterations for the iterative frequency-domain method is $n_{\text{it}} \sim n_f$ (see equation 10).

The complexity of time-domain modeling is $n_s n_t O(n^3)$ (Mulder and Plessix, 2004a, 2004b). The direct solver in the frequency domain has a complexity of $O(n^7 + n_s n^3)$ for a single frequency when using a standard LU-decomposition. This can be reduced to $O(n^6 + n_s n^3)$ with nested dissection (George and Liu, 1981). Wang et al. (2010, 2011) suggest using a low-rank approximation of the dense matrices arising from the nested dissection. In that case, the complexity of the method lies between $O(n^3(\log n + n_s))$ and $O((n^4 + n_s n^3)\log n)$, depending on the problem. Riyanti et al. (2006, 2007) considered the complexity of the shifted Laplace solver preconditioned by a multigrid method and obtained $O(n_s n_f n_{\text{it}} n^3)$ for n_f frequencies.

Considering the parallel aspect over shots and frequencies, Table 1 captures the complexity of the algorithms mentioned above. Factors of $O(\log n)$ have been ignored. It is readily seen that the time domain is the most efficient method in the serial case. However, if the implementation is parallel over shots and frequencies, the time-domain and iterative frequency-domain methods appear to be the most attractive in terms of turnaround time, assuming sufficient resources for parallel computing are available and their cost is not included. This leaves questions about actual performance unanswered, because the constants in the complexity estimates are absent. To get an indication of the actual performance of the two algorithms, both were implemented and tested on a problem of realistic size.

MODELING

Modeling is a major component of migration and inversion algorithms. Traditionally, seismic data are modeled in the time domain because of the simplicity of implementation as well as the memory demands. However, modeling in the frequency domain offers such advantages as parallelization over frequencies and reuse of earlier results if an iterative solver is employed for computing the wavefields: for example, during least-squares migration or full-waveform inversion. We will compare modeling in the time domain to that in the frequency domain.

Modeling in the time domain

Modeling in the time domain requires the solution of the wave equation,

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} = f, \tag{1}$$

where $u(x, y, z, t)$ denotes the pressure wavefield, $c(x, y, z)$ is the velocity in the subsurface, and $f(x, y, z, t)$ describes the source, at positions (x, y, z) in a domain Ω. Discretization of the wave equation with second-order finite differences in space and time leads to an explicit time marching scheme of the form

$$\begin{aligned} u^{n+1}_{i,j,k} = {} & 2u^n_{i,j,k} - u^{n-1}_{i,j,k} + \Delta t^2 c^2_{i,j,k}\left(\frac{u^n_{i-1,j,k} - 2u^n_{i,j,k} + u^n_{i+1,j,k}}{\Delta x^2}\right. \\ & + \frac{u^n_{i,j-1,k} - 2u^n_{i,j,k} + u^n_{i,j+1,k}}{\Delta y^2} \\ & \left. + \frac{u^n_{i,j,k-1} - 2u^n_{i,j,k} + u^n_{i,j,k+1}}{\Delta z^2} + f^n_{i,j,k}\right), \end{aligned} \tag{2}$$

where the superscript $n + 1$ denotes a new time level that is computed using the solutions at the two previous time steps, n and $n - 1$. A higher-order discretization of the second derivative in space in one direction is obtained by

$$\left.\frac{\partial^{(2)} u}{\partial x^{(2)}}\right|_i \simeq -\frac{1}{\Delta x^2}\left(b^M_0 u_i + \sum_{k=1}^{M} b^M_k (u_{i-k} + u_{i+k})\right), \tag{3}$$

where $2M$ denotes the order of the spatial discretization and the coefficients are

$$b^M_0 = \sum_{m=1}^{M} \frac{2}{m^2}, \tag{4}$$

Table 1. Complexity of various methods, split between setup and application cost. The required amount of storage is given in column 4. Next, a factor that can be dealt with by trivial parallelization is listed. The last column shows the scaling of compute time when the trivial parallelization is applied, assuming n^3 grid points, $n_s \sim n^2$ shots, $n_t \sim n$ time steps, $n_f \sim n$ frequencies, and $n_{\text{it}} \sim n$ iterations.

Method	Setup	Apply	Storage	Parallel	Overall
Time domain	–	$n_s n_t n^3$	$n_s n^3$	n_s	$(0 + n^6)/n^2 \sim n^4$
LU, nested dissection	$n_f n^6$	$n_s n_f n^4$	$n_f n^4$	n_f	$(n^7 + n^7)/n \sim n^6$
LU, low rank	$n_f n^4$	$n_s n_f n^3$	$n_f n^3$	n_f	$(n^5 + n^6)/n \sim n^5$
Iterative	$n_f n_s n^3$	$n_s n_f n_{\text{it}} n^3$	$n_s n_f n^3$	$n_s n_f$	$(n^6 + n^7)/n^3 \sim n^4$

$$b_k^M = (-1)^k \sum_{m=k}^{M} \frac{2}{m^2} \frac{(m!)^2}{(m+k)!(m-k)!}, \quad k = 1, \ldots, M. \tag{5}$$

Dablain (1986) describes a higher-order discretization of the second derivative in time.

To ensure stability of the time marching scheme above, the time step has to satisfy the stability constraint

$$\Delta t \leq \mathrm{CFL} \frac{d}{c_{\max}}, \tag{6}$$

with the maximum velocity $c_{\max}$, the diameter

$$d = \frac{1}{\sqrt{1/\Delta x^2 + 1/\Delta y^2 + 1/\Delta z^2}}, \tag{7}$$

and the constant $\mathrm{CFL} = 2/\sqrt{a}$ with

$$a = \sum_{k=1}^{M/2} 4^k / \left(k^2 \binom{2k-1}{k-1} \right). \tag{8}$$

For details, see Fornberg (1988).

To simulate an infinite domain and avoid reflections from the boundaries, sponge absorbing boundary conditions have been implemented (Cerjan et al., 1985).

Modeling in the frequency domain

For wave propagation in the frequency domain, we consider the Helmholtz equation in a 3D heterogeneous medium,

$$-\frac{\partial^2 \phi}{\partial x^2} - \frac{\partial^2 \phi}{\partial y^2} - \frac{\partial^2 \phi}{\partial z^2} - k^2 \phi = g, \tag{9}$$

where $\phi = \phi(x, y, z, \omega)$ is the wave pressure field, $k = k(x, y, z, \omega)$ is the wavenumber, and $g = g(x, y, z, \omega)$ is the source term. The first- or second-order radiation boundary conditions (Engquist and Majda, 1977) are used to reduce the undesired reflections from the boundary. Equation 9 is solved with a Krylov method preconditioned by a shifted-Laplace preconditioner (Erlangga et al., 2006; Knibbe et al., 2011, 2013).

If the wavelet or signature of the source term g is given in the time domain, its frequency dependence is readily obtained by a fast Fourier transform (FFT). Given the seismic data, the Nyquist theorem dictates the frequency sampling and the maximum frequency. In practice, the maximum frequency in the data is lower and is defined by the wavelet. Given the range of frequencies defined by Nyquist's theorem and the data, the Helmholtz equation 9 is solved for each frequency and the wavefield is sampled at the receiver positions, producing a seismogram in the frequency domain. Finally, the wavelet and an inverse FFT are applied to obtain the synthetic seismogram in the time domain.

The discretization of equation 9 in space depends on the number of points per wavelength. The general rule of thumb is to discretize with at least 10 points per wavelength (Saleh, 2011). In that case, the error behaves as $(kh)^2$, which is inversely proportional to the square of the number of points per wavelength. To avoid the pollution effect, $kh = 0.625$ has been chosen constant, as described by Erlangga et al. (2006). Erlangga (2005) shows that the number of iterations of the Helmholtz solver does not depend on the problem size for a given frequency, but the number of iterations increases with frequency. Erlangga and Nabben (2008) and Erlangga and Herrmann (2008) present an improved version that leads to a significant cost reduction but still requires more iterations at higher frequencies. Therefore, the computational time for modeling in the frequency domain mainly depends on the highest frequency used and on the total number of frequencies. To reduce computational time, the computations for each frequency can be parallelized over several compute nodes. The question arises: How many compute nodes do we need to minimize the computational time?

It is obvious that for a parallel implementation over an unlimited number of compute resources, the computational time is at least equal to the time needed to solve the Helmholtz equation 9 for the highest frequency.

Because the problem size is the same for each frequency and the iterative method has a fixed number of matrix-vector and vector-vector operations, it is easy to see that the time per iteration is the same for each frequency. In principle, the lower frequencies can be calculated on coarser meshes (Mulder and Plessix, 2004b). However, using a Krylov solver preconditioned with a shifted-Laplace multigrid method, the number of iterations is already quite low. Therefore, the additional complexity due to interpolation between different grid sizes does not pay off. The number of iterations per frequency f_i, $i \in \mathbb{N}$, can be expressed as

$$n_i \approx \gamma f_i, \tag{10}$$

where $\gamma = n_N / f_N$, f_N denotes the maximum frequency, and n_N the number of iterations for f_N. The frequencies are given by $f_i = i\Delta f$ where Δf is the frequency sampling interval. The total number of iterations for computing all frequencies is given by

$$\sum_{i=1}^{N} n_i \approx \gamma \sum_{i=1}^{N} i\Delta f = \gamma \Delta f \frac{N(N+1)}{2} = n_N \frac{N+1}{2}. \tag{11}$$

In other words, the least-computational time can be achieved by using a number of compute nodes equal to half the number of frequencies. As an example, let us consider a problem with maximum frequency $f_{\max} \simeq 30$ Hz and $\Delta f \simeq 1/6$ Hz. Then, 180 frequencies need to be computed, which would require 90 compute nodes. Here, we assume that the problem size corresponding to the maximum frequency fits into the memory of a single compute node.

We can adopt a different point of view by fixing the number of compute nodes and then determining the minimum workload of each node in terms of the number of iterations. Let us denote by M the number of available compute nodes. Then, using equation 11, the minimum time per node is equal to

$$T_{\min} = \frac{t_N (N+1)}{2M}, \tag{12}$$

where t_N is the compute time needed for the highest frequency.

MIGRATION

Migration algorithms produce an image of the subsurface given seismic data measured at the surface. In particular, prestack depth migration produces the depth locations of reflectors by mapping seismic data from the time domain to the depth domain, assuming a sufficiently accurate velocity model is available. The classic imaging principle (Hagedoorn, 1954; Claerbout, 1971) is based on the correlation of the forward-propagated wavefield from a source and a backward-propagated wavefield from the receivers. To get an approximation of the reflector amplitudes, the correlation is divided by the square of the forward wavefield (Baysal et al., 1983; Whitemore, 1983). For true-amplitude or amplitude-preserving migration, there are publications based on the formulation of migration as an inverse problem in the least-squares sense (Lailly, 1983; Tarantola, 1984; Beylkin, 1985; Beylkin and Burridge, 1990; Plessix and Mulder, 2004). For our purpose of comparing migration in the time and frequency domain, we focus on the classical imaging condition (Claerbout, 1985)

$$I(\mathbf{x}) = \sum_{\text{shots}} \sum_{t} W_s(\mathbf{x}, t) W_r(\mathbf{x}, t), \tag{13}$$

in time domain or

$$I(\mathbf{x}) = \sum_{\text{shots}} \sum_{\omega} W_s^*(\mathbf{x}, \omega) W_r(\mathbf{x}, \omega), \tag{14}$$

in the frequency domain. Here, I denotes the image, W_s is the wavefield propagated from the source and W_r from the receivers, respectively; t denotes time, and ω denotes the frequency. The star indicates the complex conjugate.

Born approximation

The seismic data that need to be migrated should not contain multiple reflections from the interfaces if imaging artifacts are to be avoided. Often, the Born approximation is used for modeling without multiples. Migration can be viewed as one step of an iterative procedure that attempts to minimize the difference between observed and modeled data subject to the Born approximation of the constant-density acoustic wave equation (Lailly, 1983; Tarantola, 1984; Beylkin and Burridge, 1990; Mulder and Plessix, 2004a).

The wave equation 1 in matrix form is

$$Au = f, \tag{15}$$

with wave operator $A = m\partial_{tt} - \Delta$ and model parameter $m = 1/c^2$. The last can be split into $m = m_0 + m_1$, where m_0 ideally does not produce reflections in the bandwidth of the seismic data. The wavefield can be split accordingly into $u = u_0 + u_1$ into a reference and a scattering wavefield, respectively. The reference wavefield u_0 describes the propagation of a wave in a smooth medium without any hard interfaces. The scattering wavefield u_1 represents a wavefield in a medium which is the difference between the actual and reference medium. Wave propagation in the reference medium is then described by $A_0 u_0 = f$ with $A_0 = m_0\partial_{tt} - \Delta$ in the time domain. What remains is $A_0 u_0 + A_1 u_0 + A_1 u_1 = 0$ with $A_1 = A - A_0 = m_1\partial_{tt}$. In the Born approximation, the term $A_1 u_1$ is removed, leading to the system of equations

$$A_0 u_0 = f, \tag{16}$$

$$A_0 u_1 = -A_1 u_0. \tag{17}$$

Its counterpart in frequency domain is

$$A = -k^2 - \Delta, \quad A_0 = -k_0^2 - \Delta, \quad A_1 = A - A_0 = -k_1^2, \tag{18}$$

where k_0 is the wavenumber in the reference and k_1 in the scattering medium, respectively.

With this, migration becomes the linear inverse problem of finding a scattering model $m_1(x, y, z)$ that minimizes the difference between the recorded and modeled wavefields u_1 in a least-squares sense. This assumes that the recorded data were processed in such a way that only primary reflections are preserved, because wavefield u_1 will not contain the direct wave and multiple reflections. A few iterations with a preconditioned Krylov subspace method will suffice to solve the linearized inverse problem and just one iteration may already produce a useful result (Mulder and Plessix, 2004a; Plessix and Mulder, 2004).

To illustrate the difference between modeling with the wave equation and its Born approximation, we consider the simple velocity model shown in Figure 1. The background velocity (white) is 1500 m/s and the horizontal layer shown with gray color has a velocity of 2500 m/s.

We model the seismogram in the time domain; see Figure 2 (left). The seismogram obtained using the Born approximation is presented in Figure 2 (right). Time-weighting was applied to boost the amplitude of the reflections. The first event in both figures represents the reflection from the shallow interface and the second from the deeper. The third event in Figure 2 (left) is the interbed multiple that does not appear with the Born approximation.

Migration in the time domain

We briefly summarize the algorithm for RTM in the time domain. The method consists of three major parts: forward propagation of

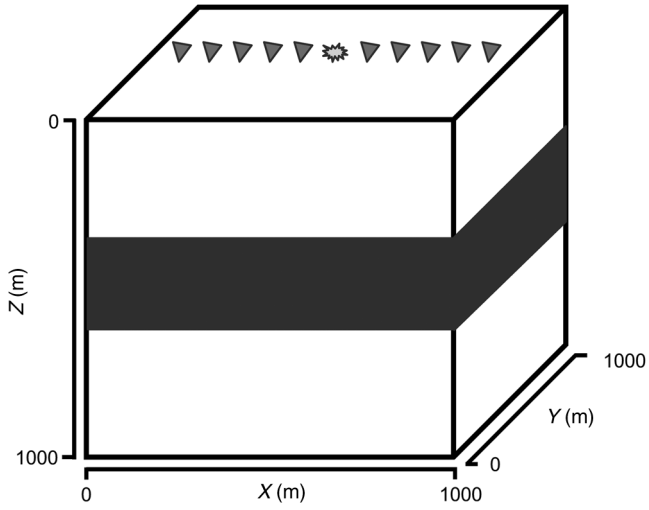

Figure 1. Model with a single high-velocity layer of 2500 m/s (gray) in a homogeneous background of 1500 m/s (white). The star represents the source and the triangles the receivers.

the wavefield from a source, backward propagation from the receivers while injecting the observed seismic data, and the imaging condition.

During forward propagating in time, the snapshots of the wavefield are stored at every imaging time step, so that they can be reused later for imaging, as sketched in Figure 3. After that, the wavefield is propagated backward in time using the seismic data as sources at the receiver locations. The image is built by correlation, taking the product of the forward and backward wavefields stored at the same imaging time step and summing the result over time. The imaging time step is often the same as the sampling interval of the seismic data, which is usually larger than the time step used for modeling.

To save time and storage space, the imaging condition can be incorporated in the backward propagation. During the forward wave propagation, the wavefields are written to disk because for problems of realistic size, random-access memory is usually too small. While backward propagating every imaging time step, the forward wavefields are read from disk and correlated with the computed backward wavefield. Even if this reduces the amount of I/O, disk access can take a significant amount of time compared to the computations.

Symes (2007) uses an optimized check-pointing method that only saves the wavefields at predefined checkpoints in time and recomputes the wavefields at other instances from these checkpoints. The amount of recomputation is reduced by choosing optimal checkpoints. However, the recomputation ratio may be very high if the number of checkpoints is not large enough. If the number of checkpoints is too large, the disk space demand and I/O will be high.

With the use of absorbing boundary conditions to simulate infinite domains, the recomputation of the forward wavefield may require some special techniques. One possibility is to store only the boundary values of the wavefield. However, in 3D, the boundaries can have a substantial width and this may not be efficient. Another possibility is to use the random boundary technique (Clapp, 2009; Shen and Clapp, 2011), which leads to random scattering of the wavefield at the boundary. The idea is that the forward and backward wavefields have different sets of random numbers and the artifacts due to scattering do not stack in the final image. In this case, the propagation effort is doubled because the forward wavefield is first computed, after which the backward propagation and the time-reversed forward wavefield are calculated simultaneously. This method has been implemented on a GPU by Liu et al. (2012).

If the noise due to random scatterers is to be avoided, alternative techniques to reduce the effect of I/O for the RTM algorithm can be applied. One of them is to hide the writing and reading times of the snapshots by using asynchronous I/O. However, this is only effective when the reading of the wavefields from disk is faster than the computations needed for one imaging time step. One way to achieve this is by compressing the wavefield before storing it to disk on a GPU or on a CPU (e.g., Cabezas et al., 2009).

The Fourier transform can offer compression. The periodicity of trigonometric functions requires special care at the boundaries. An alternative is to use functions that are compact in space and time, such as wavelets. The wavelet transform has been extensively documented (e.g., Louis et al., 1997; Mallat, 2008). Ji et al. (2012) use wavelets for compression of the wavefields. In that way, disk I/O is reduced and the GPU, CPU, and disk I/O are balanced well. The idea is to decompose the snapshot by means of the wavelet transform into "average" and "detailed" parts. The average part contains the dominant features of the data and the detailed part contains small-scale features. We are interested in keeping the average part as it is and focusing on the details. Before compressing the snapshots, we can choose the amount of detail we would like to keep. For the detailed part, the mean and deviation are calculated. We introduce a parameter λ that is multiplied by the deviation. This product defines the threshold for compression. Table 2 describes the effect of values of λ on the compression ratio, which is defined

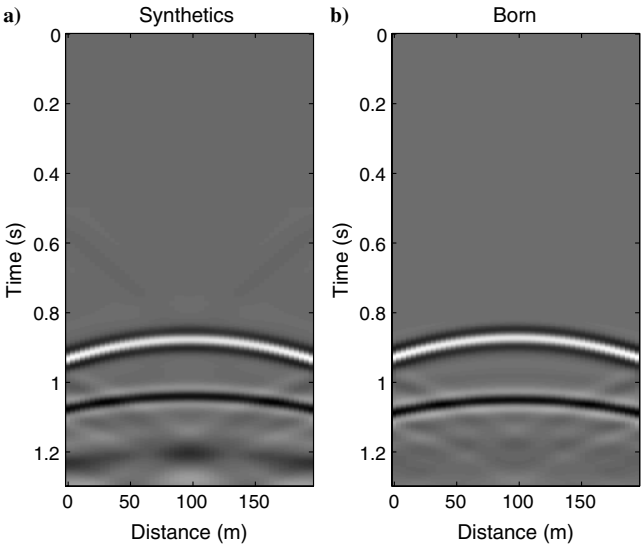

Figure 2. (a): Synthetic seismic produced with regular modeling in the time domain. The direct arrival has been removed. (b): With Born approximation in the time domain, showing that the interbed multiple around 1.2 s has disappeared.

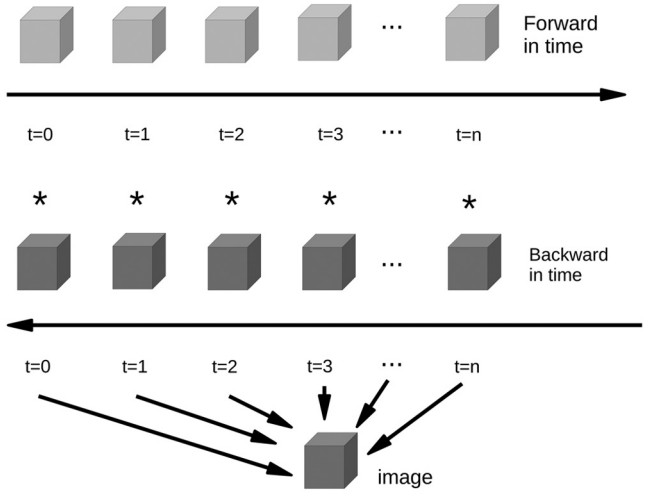

Figure 3. Migration in the time domain. The forward-propagated wavefield from the source is stored at each imaging time step (light gray cubes). Then, the wavefield is propagated backward in time while injecting seismic data at the receiver locations and the snapshots are stored (dark gray cubes). The imaging condition involves the summation of the product of the forward and backward wavefields.

as the size of the original data divided by the size of the compressed data, as well as the compression time. It is clear that the compression ratio depends on the parameter λ, which means that the more data we remove, the better the compression. The second column in the table shows the L_1-norm of the absolute differences between the original and the compressed wavefields. The third column represents the relative L_2-norm, which is the usual L_2-norm of the difference scaled by the L_2-norm of the original wavefield. We observe that the more the wavefields are compressed, the more the L_1- and L_2-norms increase monotonically, due to the loss of information during compression.

Which is the optimal compression parameter? There is a trade-off between small L_1- and L_2-errors and a large compression ratio. The table shows that the required compression time hardly changes with various choices of λ. Figure 4 depicts the compression ratio as a function of parameter λ. It starts with an exponential increase and becomes linear for $\lambda > 1$. Therefore, we have selected $\lambda = 1$ as our compression parameter, because it provides an acceptable balance between compression errors and required compression time.

Migration in the frequency domain

Migration in the frequency domain requires the selection of a set of frequencies that avoids spatial aliasing (Mulder and Plessix, 2004b). The seismic data and the source signature are transformed into frequency domain by an FFT. For each frequency, the Helmholtz equation is solved iteratively. The imaging condition in the frequency domain consists of a simple multiplication of the wavefields at each frequency, followed by a summation over the selected frequencies. Figure 5 illustrates the procedure. The forward and backward propagation are computed in parallel and there is no need to store the wavefields on disk. Basically, for each frequency, the forward and backward fields are computed one after the other and are then multiplied with each other. Only two wavefields are kept in memory, whereas in time domain, all the consecutive wavefields for the forward propagation need to be stored.

Table 2. Compression for wavefield snapshots for a problem of size 512^3 and about 100 MB of storage. The L_1-norm measures the difference between the original and compressed wavefield. The relative L_2-norm is the usual L_2-norm of the differences, scaled by the L_2-norm of the original snapshot. The compression ratio is defined as the size of the original data divided by the size of the compressed data.

λ	L_1-norm	Relative L_2-norm	Compr. ratio	Compr. time (s)
0.1	$1.08e-1$	$1.98e-3$	3.88	16.16
0.5	$1.14e-1$	$2.11e-3$	4.16	15.46
1	$1.52e-1$	$2.87e-3$	4.32	15.27
1.5	$2.21e-1$	$4.27e-3$	4.42	15.07
2	$3.07e-1$	$5.95e-3$	4.51	14.92
2.5	$4.06e-1$	$7.86e-3$	4.59	14.77
3	$5.22e-1$	$1.01e-2$	4.67	14.58
3.5	$6.47e-1$	$1.25e-2$	4.75	14.53
4	$7.96e-1$	$1.53e-2$	4.83	14.37
4.5	$9.44e-1$	$1.82e-2$	4.91	14.27
5	$1.09e-0$	$2.09e-2$	4.99	14.28

IMPLEMENTATION DETAILS

As mentioned before, we consider the GPU as an accelerator in our implementation strategy and we will use the terms "CPU" and "computer node" when referring to a multicore CPU machine.

Presently, a common hardware configuration is a CPU connected to two GPUs that contain less memory than the CPU. We have identified the parts of the algorithms that can be accelerated on a GPU and implemented them in CUDA 5.0. As already explained, the parallelization process for migration in the time domain is different from that for the frequency domain because explicit time stepping is used in the time domain, whereas the frequency domain requires solving a linear system of equations. We summarize the levels of parallelism in Table 3.

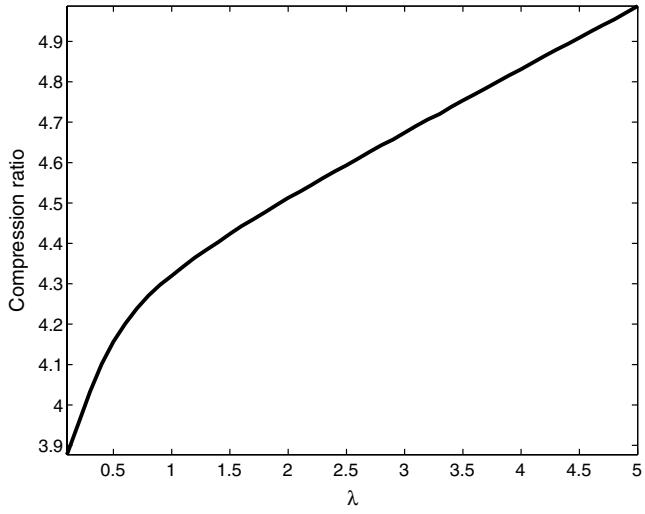

Figure 4. Compression ratio as a function of the parameter λ.

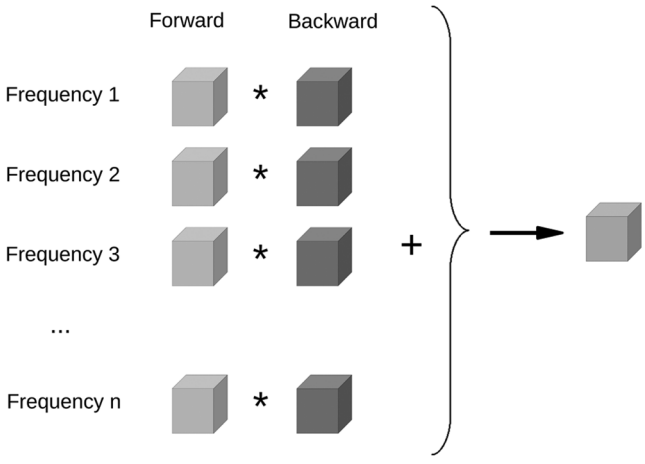

Figure 5. Migration in the frequency domain requires multiplication of forward and backward wavefields, followed by summation.

The highest level of parallelization for time-domain migration is over the shots. Each shot is treated independently. We assume that the migration volume for one shot is computed on one computer node connected to one or more GPUs.

The next step is to split the problem into subdomains that will fit on a GPU. We use a domain decomposition approach or grid partitioning. The idea is that, at each time step, the subdomains are treated independently of each other. Once the computation is completed, the subdomains are copied back to the CPU, after which the next time step can be started. The third level of parallelization is to perform computations and data transfer simultaneously, to save time and achieve optimal load balancing. The compression algorithms and simultaneous computations of the forward and backward propagation are also part of the third level. The fourth level of parallelism for time-domain computations is data parallelism (e.g., Micikevicius, 2009).

For migration in the frequency domain as well as in the time domain, the highest level of parallelization is over the shots. The next level of parallelism involves the frequencies. For each frequency, a linear system of equations needs to be solved. As mentioned before, the matrix size and memory requirements are the same for each frequency, but the lower frequencies require less computing time than the higher ones (Erlangga et al., 2006). Here, we assume that one shot in the time domain and one shot for one frequency in the frequency domain fits in one compute node connected to one or more GPUs, respectively. The third level of parallelism includes matrix decomposition, where the matrix for the linear system of equations is decomposed into subsets of rows that fit on a single GPU (e.g., Knibbe et al., 2013). With this approach, we can deal with problems that are larger than can be handled by a single GPU. So far, the simultaneous use of two GPUs to accelerate offloaded matrix-vector multiplications (MVMs) of a large sparse-matrix have not produced any performance improvements compared to a multicore CPU due to the data transfer. Therefore, we use only CPUs for the frequency-domain approach. Note that, with an increasing number of GPUs connected to the same CPU (faster PCI buses, etc.), this situation may change. The last level of parallelism for migration in frequency domain is parallelization of MVMs and vector-vector operations.

Domain decomposition approach

The time-domain implementation on multi-GPUs is done by domain decomposition. The problem is divided into subdomains that fit in the limited memory of a GPU. This approach can also be applied if a large problem needs to be computed on a single GPU.

For simplicity of implementation and communication between GPUs, the domain is split only in the z-direction, as Figure 6 shows. The overlapping areas are attached to both subdomains. The size of a subdomain is determined by the available memory divided by the number of discretization points in the x-and y-directions, multiplied by the byte-size of a floating-point number. Each subdomain should fit entirely in GPU memory.

After partitioning the problem, tasks are set up, where each task represents a subdomain. These tasks are added to a queue and handled by pthreads (Barney, 2010) that are distributed among the GPUs.

Once a subdomain has been processed, the interior domain (i.e., the domain without the overlapping parts), is copied back to the CPU. Then, the next time step can be performed.

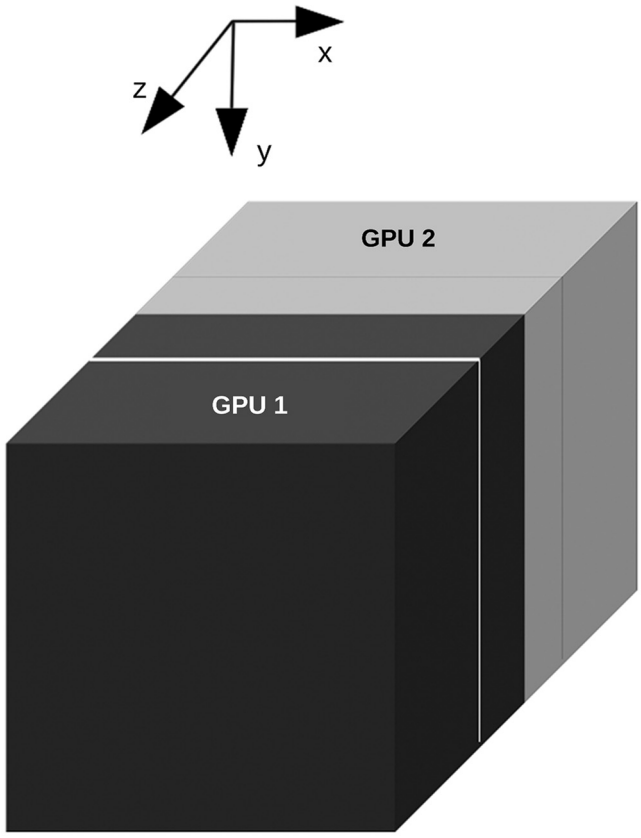

Figure 6. An example of domain decomposition into two subdomains in the z-direction. The overlapping area has half the size of the discretization stencil.

Table 3. Levels of parallelism for migration in time and frequency domain.

	Time domain	Frequency domain
Level 1	Parallelization over shots	Parallelization over shots
Level 2	Domain decomposition	Parallelization over frequencies
Level 3	Overlap for computations with memory transfer, load balancing	Matrix decomposition
Level 4	Data parallelism (grid points)	Linear algebra parallelization (MVM, vector operations)

Implicit load balancing

A common approach for parallelization across multiple CPU nodes in the cluster is the so-called *server-to-client approach* (e.g., Stevens, 1998). The server is aware of the number of nodes and the amount of work involved. It equally distributes the workload among the nodes. This approach is efficient on clusters with homogeneous nodes as all CPU nodes have the same characteristics.

In this paper, we propose a *client-to-server approach* where clients request tasks to the server. GPU clusters are either heterogeneous or they have to be shared simultaneously among the users. For example, the cluster at our disposition, Little Green Machine (LGM, 2012), has the same hardware (with the exception of one node). Similarly, within one computer node, the GPUs have the same specifications, but one GPU can already be used by a user while the other one remains available. To address the issue of load balancing, we created a task system. When doing migration in time domain, an "MPI-task" defines the work to be done for one shot. For each time step during forward modeling, one "GPU-task" is created for each subdomain of the domain decomposition algorithm, as Figure 7 shows. The philosophy behind a task system is the same over multiple compute nodes as well as over multiple GPUs within one node: process all the tasks as fast as possible until they are all processed. In this approach, the work is spread dynamically according to the speed of the computing nodes and GPUs. Depending on the level of parallelism, the implementation of the tasks systems differs.

MPI-tasks

The server or "master node" creates one task per shot. Each task is added to a queue. When a client requests a task, a given task is moved from the queue to the active list. It can happen that a node will crash due to a hardware failure. In that case, the task will remain on the active list until all the other tasks have finished. Once that happens, any unfinished tasks will be moved back to the queue, so that another computer node can take over the uncompleted work.

Toward the end of the migration, the queue is empty while the lists of active tasks is not. Given that there is no way of telling whether a task has crashes or just takes long time, the former is assumed. The task that is being processed for the longest period of time is submitted again but to a different node. At this point, this particular task may end up being processed by two nodes. As soon as the server receives the result of one of these tasks, the other task is killed. When all tasks have been processed, the master node saves the migration image to disk and stops.

GPU-tasks

GPU hardware failures are less frequent than the compute node ones; therefore, there is no need to have two different queues for GPU-tasks. As an example, let us consider time domain modeling on a CPU node with two GPUs. A GPU-task defines a subdomain and consists of the following workflow: transfer the subdomain from CPU to GPU, propagate wavefield, and finally, copy the data back to CPU.

When the program starts, first the data have to be transferred to both GPUs. We expect that the two GPUs start transferring data and then will compete for the PCI-bus. Eventually, one GPU will be slightly faster with either the computations or with the data transfer. Then, the two GPUs will work asynchronously, meaning that while one GPU is computing, the other is transferring data. This leads to dynamic load balancing, self-regulated by the system.

We performed profiling with the CUDA profiler, as illustrated in Figure 8. Interestingly enough, the two GPUs are acting asynchronously almost from the start of the program execution. The column on the left in Figure 8 shows that we have used two GPUs GeForce GTX 460, each of which has to perform a memory copy from host to device (*MemCpy(HtoD)*), a memory copy from device to host (*MemCpy(DtoH)*), and computations of the wave propagation kernel (*Compute*). On the right side of the figure, the horizontal axis denotes time, bars represent different tasks and the length of a bar shows the duration of a task. It is easy to see that, at the beginning, both GPUs simultaneously start to transfer data from the host, because the dashed bars are on top of each other. Then, GPU[1] starts computing (dark gray bar) and the GPU[0] is idle. Afterward, GPU [1] transfers the results to the CPU, and GPU[0] is computing at the same time. We have performed several profiling tests and every time, we obtained the same outcome, with the GPUs running in asynchronous mode already from the start. This provides an optimal load balancing.

Table 4 presents elapsed times (in s) for the modeling in time domain for a finite-difference discretization with several discretization orders. For problems of larger size, domain decomposition is applied and the task system with load balancing is used. The column "slice" denotes the maximum number of (x, y)-slices in the z-direction that will fit in the memory of one GPU (1 GB). If a slice is smaller than the number of points in z-direction, then domain decomposition has to be applied, because the problem does not fit in GPU memory. It is clearly seen that a problem of size 200^3 is sufficiently small to fit into 1 GByte of memory, but larger problems have to be split. Also, the last column shows that the size of the slice does not give rise to significant changes in time with increasing discretization order. A higher discretization order requires more floating point operations per discretization point and causes an increase in the compute time on the CPU. However, the GPU time hardly changes with increasing order. The reason for this behavior is the

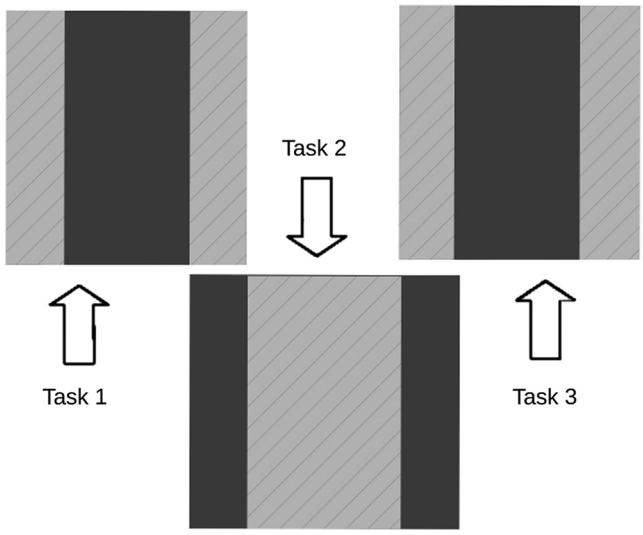

Figure 7. An example of task distribution into three tasks. Note that the overlapping areas have to be assigned to all neighbors.

load-balancing strategy. As the profiling suggests, the transfer and computational time overlap in time asynchronously. On the one hand, the wave propagation kernel takes less time than the data transfer. On the other hand, the transfer time does not change significantly for higher-order discretizations. Therefore, the increase of computational and transfer time for higher-order discretizations is hidden and the overall GPU-time stays the same.

We propose the following workflow for migration in time domain, combining the techniques mentioned above. For forward propagation, first, a main thread is created on a CPU. Its role is to launch other threads, create tasks, and be responsible for the GPU-CPU and CPU-GPU transfer. Two child threads are created, one for each GPU, that perform the actual time-stepping computations on the GPUs. When one imaging time step is finished, the

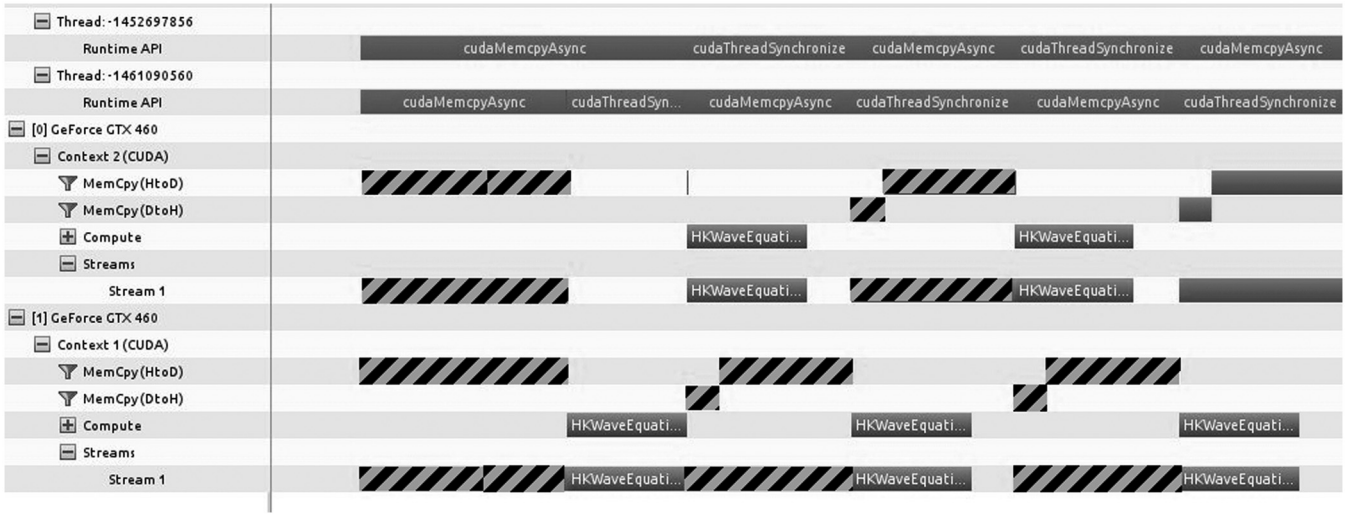

Figure 8. Profiling for seismic modeling in the time domain using a 16th order discretization on two GPUs. The column on the left shows the tasks per GPU, such as data transfer from host to device *MemCpy(HtoD)*, from device to host *MemCpy(DtoH)* and wave propagation *Compute*. The length of a bar on the left represents the duration of a task. Dashed bars show the data transfer and dark gray bars represent the computational kernel. The dark gray bars are only overlapping in time with the dashed bars, illustrating that both GPUs are operating asynchronously.

Table 4. Elapsed time comparisons (in s) for fourth-, sixth-, eighth-, and 16th-order discretizations for a 3D problem of size n^3. The speedup is computed as the ratio of CPU time to GPU time. The slice column denotes the number of (x,y)-slices in z-direction that will fit in the memory of one GPU (1 GB).

Order	n	CPU	1 GPU	Speedup	2 GPUs	Speedup	slice
4	200	6	3	1.93	3	1.95	1565
	400	49	25	1.94	24	2.02	391
	800	398	202	1.97	137	2.9	97
	1200	1342	673	1.99	442	3.03	43
6	200	8	3	2.33	3	2.32	1565
	400	61	26	2.34	25	2.48	391
	800	489	207	2.36	138	3.56	97
	1200	1709	684	2.50	468	3.65	43
8	200	9	3	2.74	3	2.77	1565
	400	73	26	2.75	26	2.83	391
	800	591	209	2.82	137	4.31	97
	1200	2002	685	2.92	473	4.23	43
16	200	15	3	4.34	3	4.36	1565
	400	125	29	4.30	27	4.58	391
	800	1008	221	4.55	142	7.09	97
	1200	3475	671	5.18	433	8.02	41

wavefield is copied from the GPU to the CPU. The main thread launches several child threads to keep every CPU-core busy. The role of those processes is to compress the wavefield on the fly, using the wavelet transform described above, and write the results to disk. For the backward propagation, the workflow is similar, except that the wavefields are read from memory and decompressed on the fly. Moreover, during the backward propagation the imaging condition is applied. Computations on the GPUs as well as the compression and disk I/O are all done in parallel. For the proposed workflow, I/O (including compression and decompression) in the time domain takes about 5% of the overall computational time.

RESULTS

In this section, we present some results for migration in the time and frequency domain and make comparisons in terms of performance.

Wedge

The first example is a wedge model that consists of two dipping layers, depicted in Figure 9. The main purpose is to validate the migration results in time and frequency domain. The problem is defined on a cube of size $[0, 1000]^3$ m^3. For our experiments, we consider a series of uniform grids with increasing size n^3. The grid size satisfies the condition $h\max_{x,y,z} k(x, y, z) = 0.625$, where the grid spacing is $h = \frac{1000}{n-1}$ and the wavenumber $k(x, y, z)$ is given by $k(x, y, z) = 2\pi f/c(x, y, z)$ with velocity $c(x, y, z)$. The problem is discretized with fourth-order finite differences in space and second-order in time for migration in time domain and second-order finite differences in space for migration in frequency domain. The source is a Ricker wavelet with a peak frequency of 15 Hz located at (500, 500, 10). The receivers are placed at a horizontal plane on a regular grid of 50×50 m^2 at a depth of 20 m. The sampling interval for the seismic data is 4 ms and the maximum simulation time is 2 s. The imaging time step is 4 ms. The experiment was carried out on the Little Green Machine LGM (2012).

Table 5 lists the timings for migration in the time domain and Table 6 for the frequency domain. The first column contains the size of the problem, excluding an additional 40 points at each absorbing boundary. The second and third columns show the elapsed time for the forward and backward propagation, respectively. For the experiment in the frequency domain, the timings are given for the highest frequency of 30 Hz, because the preconditioned Helmholtz solver requires the longest computational time at this frequency. From Tables 5 and 6, it is clear that migration in the frequency domain is more than two times faster than the migration in time domain. Here, we assume that we have enough compute nodes for the calculation of all frequencies in parallel.

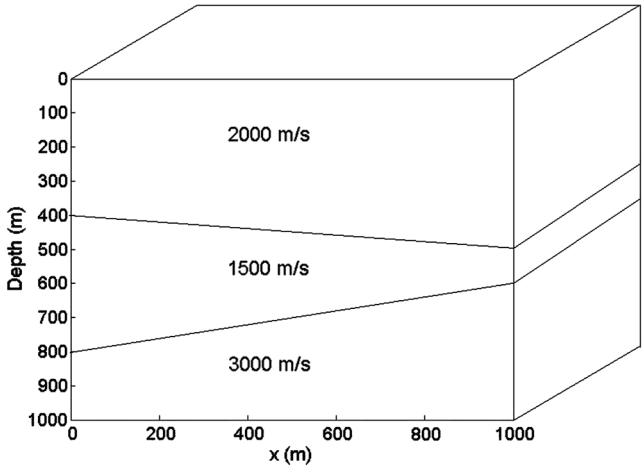

Figure 9. Velocity profile for the wedge model.

Overthrust model

The SEG/EAGE overthrust model has been introduced in Aminzadeh et al. (1997). It represents an acoustic constant-density medium with complex, layered structures and faults. We choose a subset of the large initial model, containing the fault features shown in Figure 10, and rescale it to fit on a single computer node. The volume has a size of $1000 \times 1000 \times 620$ m^3. The problem is discretized on a grid with $301 \times 301 \times 187$ points and a spacing of 3.33 m in each coordinate direction. As described earlier, one criterion for choosing the grid spacing is the number of points per wavelength needed to accurately model the maximum frequency. Another criterion is the available memory size of the computational node. In addition, we add 40 points for each absorbing boundary in the time-domain scheme to avoid boundary reflections. The discretization for migration in time domain is fourth-order in space and second-order in time and for migration in frequency domain is second-order in space. A Ricker wavelet with a peak frequency of 15 Hz is chosen for the source and the maximum frequency in this experiment is 30 Hz. Note that, by reducing the maximum frequency, we can increase the grid spacing. For instance, by choosing a maximum frequency of 8 Hz, the grid spacing can be chosen as 25 m in each direction. The line of sources is located at a depth

Table 5. Performance of the migration for one source in the time domain in the wedge problem. The problem size is n^3, not counting the extra points at each absorbing boundary.

n^3	Timings forward (s)	Timings backward (s)	Migration ime (s)
201	837	846	1683
251	1653	1663	3317
301	2998	2990	5988
901	6649	7015	13,666

Table 6. Performance of migration for one source in the frequency domain for the wedge problem at highest frequency 30 Hz.

n^3	Timings forward (s)	Timings backward (s)	Migration time (s)
201	375	365	740
251	732	718	1450
301	1119	1145	2264

of 10 m and is equally spaced with an interval of 18.367 m in the x-direction. The receivers are equally distributed in the two horizontal directions with the same spacing as the sources, at the depth of 20 m. The sampling interval for the modeled seismic data is 4 ms. The maximum simulation time is 0.5 s. For migration in the time domain, an imaging time-step of 0.0005 s was chosen, whereas in the frequency domain we chose a frequency interval of 2 Hz.

Images produced by RTM in the time domain and in the frequency domain are shown in Figures 11 and 12, respectively.

The timings for migration in the time domain are given in Table 7 and for migration in the frequency domain in Table 8, respectively. The first and second column show the elapsed time for the forward and backward propagation, correspondingly. For the experiment in the frequency domain, the timings are given for a highest frequency of 30 Hz, because the preconditioned Helmholtz solver requires the longest computational time for this frequency. The timings show that migration in the frequency domain is about four times faster than in time domain. Here, we again assume that we have enough computational nodes for the calculation of all frequencies in parallel.

DISCUSSION

For a single source, migration in the frequency domain would be more time consuming on one computational node because all the frequencies are computed sequentially. However, if enough computational nodes are available, then migration in the frequency domain can compete with migration in the time domain as shown in Table 1. Our experiments in the previous section confirm that.

One might wonder what the actual timings would be in the time and in the frequency domain for a given number of computational nodes for a given problem. The wall-clock time for the time domain can be estimated as a function of n_s sources on n_c computational nodes

$$t^{td} = T^{td}_{\max} \max\left(1, \frac{n_s}{n_c}\right), \tag{19}$$

where $T^{td}_{\max}$ is the computational time for one source on one computational node. For the frequency domain, it also depends on n_f frequencies

$$t^{fd} = T^{fd}_{\max} \max\left(1, \frac{n_s n_f}{2n_c}\right), \tag{20}$$

where $T^{fd}_{\max}$ is the computational time needed for the maximum frequency on one computational node. Combining the number of sources and computational nodes in a new variable n_c/n_s, the time function for the time and frequency domain is shown in Figure 13,

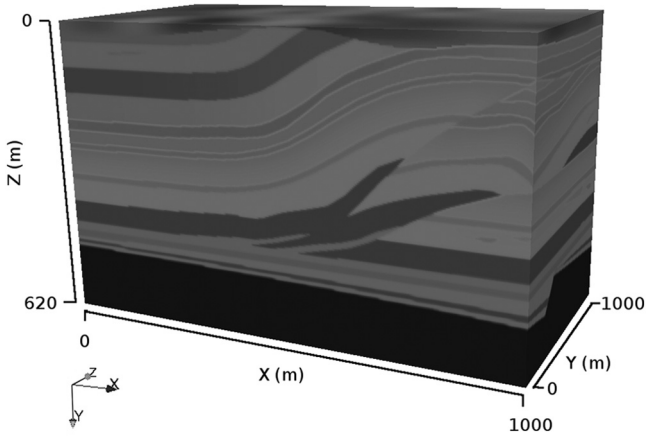

Figure 10. Overthrust velocity model.

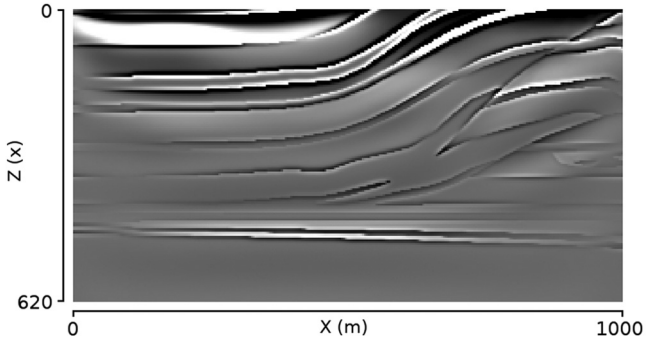

Figure 11. Migration in the time domain for a subset of the SEG/EAGE Overthrust problem.

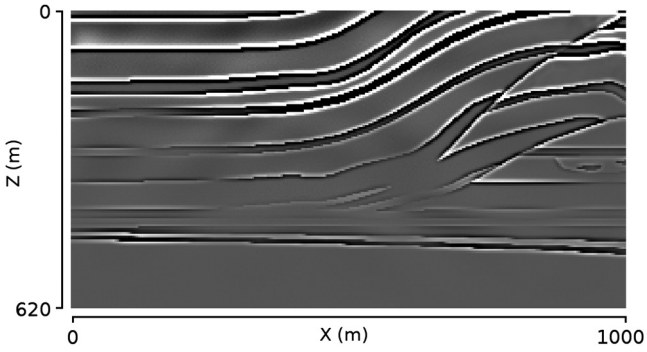

Figure 12. Migration in the frequency domain for the Overthrust problem.

Table 7. Performance of migration of one source in the time domain for the Overthrust problem.

Timings forward (s)	Timings backward (s)	Migration time (s)
1156	1168	2324

Table 8. Performance of migration of one source in the frequency domain for the Overthrust problem at the highest frequency of 30 Hz.

Timings forward (s)	Timings backward (s)	Migration time (s)
276	294	570

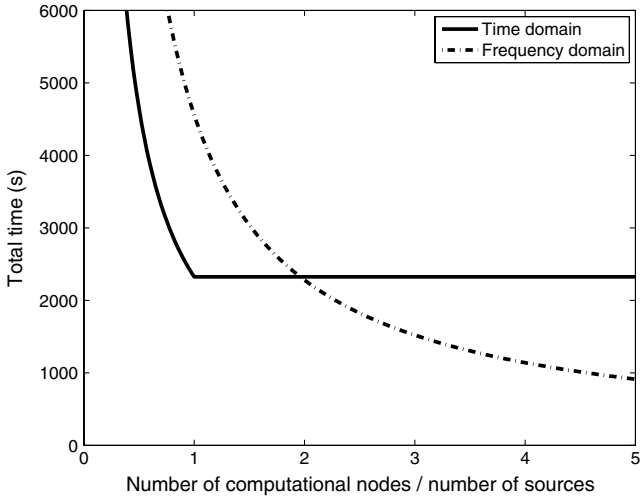

Figure 13. Performance of the time-domain scheme (dashed line) and the frequency-domain solver (solid line) as a function of the number of compute nodes divided by the number of sources.

given the experimental results from the Overthrust model. If the number of sources is smaller than twice the number of compute nodes, then the time domain is faster on our hardware. Otherwise, the frequency-domain approach outperforms the time-domain method.

Moreover, from the experiments described in the previous sections, one can obtain the relation between memory usage and problem size. For the frequency domain, we find

$$\text{Memory(GB)} = 2 \cdot 10^{-7} \cdot N, \tag{21}$$

where N is the total number of grid points including absorbing boundary conditions. For the time domain, we obtain

$$\text{Memory(GB)} = 5.5 \cdot 10^{-8} \cdot N. \tag{22}$$

On the one hand, the frequency-domain method uses more than twice as much memory as the time-domain scheme. On the other hand, the migration in the frequency domain does not make use of disk for storing snapshots.

Obviously, there is a trade-off between the computational time, the amount of memory, and disk usage when considering performance for migration in the time and frequency domain. At this point, the memory needed for the solver is the main bottleneck for solving larger problems.

CONCLUSIONS

We have considered migration in the frequency domain based on a Krylov solver preconditioned by a shifted-Laplace multigrid method. Its implementation has been compared to the implementation of the RTM in the time domain in terms of performance and parallelization. The hardware configuration is a multicore CPU connected to two GPUs that contain less memory than the CPU. The implementation in frequency domain is using parallel techniques on multicore CPU and the implementation in time domain is accelerated using GPUs. The parallelization strategy uses domain decomposition and dynamic load balancing.

The experiments show that migration in the frequency domain on a multicore CPU is faster than RTM in the time domain accelerated by GPUs, given enough compute nodes to calculate all frequencies in parallel. This observation is based on our own implementation of both approaches, optimization details, and the hardware we had access to. Despite such uncertainties, the methods can obviously compete. We expect to have similar results on different hardware because the GPU-CPU performance ratio does not change dramatically.

ACKNOWLEDGMENTS

The authors thank Delft University of Technology for granting access to the Little Green Machine, partly funded by the `Nederlandse Organisatie voor Wetenschappelijk Onderzoek' (NWO, the Netherlands Organisation for Scientific Research) under project number 612.071.305.

APPENDIX A

SOLVER

For modeling in the frequency domain, we consider the Helmholtz equation for wave propagation in a 3D heterogeneous medium,

$$-\frac{\partial^2\phi}{\partial x^2} - \frac{\partial^2\phi}{\partial y^2} - \frac{\partial^2\phi}{\partial z^2} - k^2\phi = g, \tag{A-1}$$

where $\phi = \phi(x, y, z, \omega)$ is the wave pressure field, $k = k(x, y, z, \omega)$ is the wavenumber, and $g = g(x, y, z, \omega)$ is the source term. The corresponding differential operator has the following form

$$\mathcal{A} = -\Delta - k^2,$$

where Δ denotes the Laplace operator. The discretized Helmholtz equation

$$A\phi = g, \quad A \in \mathbb{C}^{N\times N}, \quad \phi, g \in \mathbb{C}^N \tag{A-2}$$

is solved by the Bi-CGSTAB method (der Vorst, 1992; Saad, 2003). The advantage of this method is that it is easily parallelizable. However, even if the Bi-CGSTAB method converges for small wavenumbers k, the convergence is too slow and it strongly depends on the grid size, see Erlangga (2005). The original system of linear equations A-1 can be replaced by an equivalent preconditioned system

$$AM^{-1}u = g, \qquad M^{-1}u = \phi, \tag{A-3}$$

where the inverse of M is easy to compute. The matrix AM^{-1} is well-conditioned, so that the convergence of Bi-CGSTAB (and any other Krylov method) is improved.

As the preconditioner for Bi-CGSTAB we consider the shifted Laplace preconditioner introduced by Erlangga et al. (2003, 2004, 2006), which is based on the following operator

$$\mathcal{M}_{(\beta_1,\beta_2)} = -\Delta - (\beta_1 - i\beta_2)k^2, \qquad \beta_1, \beta_2 \in \mathbb{R}, \tag{A-4}$$

with the same boundary conditions as $\mathcal{A}$. The system A-2 is then preconditioned by

$$M_{(\beta_1,\beta_2)} = -L - (\beta_1 - i\beta_2)k^2 I, \qquad \beta_1, \beta_2 \in \mathbb{R}, \qquad \text{(A-5)}$$

where L is the discretized Laplace operator, I is the identity matrix, and β_1, β_2 can be chosen optimally. Depending on β_1 and β_2, the spectral properties of the matrix AM^{-1} change. In Erlangga et al. (2006), the Fourier analysis shows that A-1 with $\beta_1 = 1$ and $0.4 \leq \beta_2 \leq 1$ gives rise to favorable properties that considerably improve convergence of Krylov methods (e.g., Bi-CGSTAB), see also van Gijzen et al. (2007).

If $A\phi = g$ in A-2 is solved using the standard multigrid method, then conditions on the smoother and the coarse-grid correction must be met. For the smoother, the conditions are:

- k^2 must be smaller than the smallest eigenvalue of the Laplacian;
- The coarsest level must be fine enough to allow for the representation of the smooth solution components.

Furthermore, the standard multigrid method may not converge in case k^2 is close to an eigenvalue of M. This issue can be resolved by using subspace correction techniques (Elman et al., 2001).

Because of the above reasons, we choose a complex-valued generalization of the matrix-dependent multigrid method by de Zeeuw (1990) as a preconditioner in A-3. It provides an h-independent convergence factor in the preconditioner, as shown in Erlangga et al. (2006).

In the coarse-grid correction phase, the Galerkin method is used to get coarse-grid matrices

$$M_H = RM_hP, \qquad \text{(A-6)}$$

where M_H and M_h are matrices on the coarse and fine grids, respectively, P is prolongation, and R is restriction. The prolongation P is based on the 3D matrix-dependent prolongation, described in Zhebel (2006) for real-valued matrices. Because the matrix M_h is a complex symmetric matrix, the prolongation is adapted for this case. This prolongation is also valid at the boundaries.

The restriction R is chosen as full-weighting restriction and not as adjoint of the prolongation. It provides a robust convergence for several complex-valued Helmholtz problems (Erlangga et al., 2006).

As mentioned before, classical iterative methods in general do not converge for the Helmholtz equation, but we can apply them as smoothers for the multigrid method. We consider a parallel version of the Gauss-Seidel method as the smoother, the so-called multicolored Gauss-Seidel smoother. In our 3D case, we use eight colors, where the neighbors of a grid point do not have the same color.

APPENDIX B

LITTLE GREEN MACHINE

Little Green Machine configuration consists of the following nodes interconnected by 40 Gbps Infiniband:

- 1 head node
  - 2 Intel hexacore X5650
  - 24 GB memory
  - 24 TB disk (RAID)
- 1 large RAM node
  - 2 Intel quadcore E5620
  - 96 GB memory
  - 8 TB disk
  - 2 NVIDIA C2070
- 1 Tesla node
  - 2 Intel quadcore E5620
  - 24 GB memory
  - 8 TB disk
  - 2 NVIDIA GTX480
- 1 test node
  - 2 Intel quadcore E5620
  - 24 GB memory
  - 2 TB disk
  - 1 NVIDIA C2050
  - 1 NVIDIA GTX480
- 20 LGM general computing nodes
  - 2 Intel quadcore E5620
  - 24 GB memory
  - 2 TB disk
  - 2 NVIDIA GTX480

REFERENCES

Alford, R., K. Kelly, and B. Boore, 1974, Accuracy of finite-difference modeling of the acoustic wave equation: Geophysics, **39**, 834–842, doi: 10.1190/1.1440470.

Aminzadeh, F., J. Brac, and T. Kunz, 1997, 3-D salt and overthrust models: SEG.

Barney, B., 2010, POSIX threads programming: Lawrence Livermore National Laboratory: https://computing.llnl.gov/tutorials/pthreads.

Baysal, D. E., D. Kosloff, and J. W. C. Sherwood, 1983, Reverse time migration: Geophysics, **48**, 1514–1524, doi: 10.1190/1.1441434.

Beylkin, G., 1985, Imaging of discontinuities in the inverse scattering problem by inversion of a causal generalized Radon transform: Journal of Mathematical Physics, **26**, 99–108, doi: 10.1063/1.526755.

Beylkin, G., and R. Burridge, 1990, Linearized inverse scattering problems in acoustics and elasticity: Wave Motion, **12**, 15–52, doi: 10.1016/0165-2125(90)90017-X.

Cabezas, J., M. Araya-Polo, I. Gelado, N. Navarro, E. Morancho, and J. M. Cela, 2009, High-performance reverse time migration on GPU: International Conference of the Chilean Computer Science Society, 77–86.

Cerjan, C., D. Kosloff, R. Kosloff, and M. Reshef, 1985, A nonreflecting boundary condition for discrete acoustic and elastic wave equations: Geophysics, **50**, 705–708, doi: 10.1190/1.1441945.

Claerbout, J. F., 1971, Towards a unified theory of reflector mapping: Geophysics, **36**, 467–481, doi: 10.1190/1.1440185.

Claerbout, J. F., 1985, Imaging the Earth's interior: Blackwell Scientific.

Clapp, R. G., 2009, Reverse time migration with random boundaries: 79th Annual International Meeting, SEG, Expanded Abstracts, 2809–2813.

Dablain, M. A., 1986, The application of high-order differencing to the scalar wave equation: Geophysics, **51**, 54–66, doi: 10.1190/1.1442040.

der Vorst, H. A. V., 1992, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems: SIAM Journal on Scientific and Statistical Computing, **13**, 631–644, doi: 10.1137/0913035.

de Zeeuw, P. M., 1990, Matrix-dependent prolongations and restrictions in a blackbox multigrid solver: Journal of Computational and Applied Mathematics, **33**, 1–27, doi: 10.1016/0377-0427(90)90252-U.

Elman, H. R., O. G. Ernst, and D. P. O'Leary, 2001, A multigrid method enhanced by Krylov subspace iteration for discrete Helmholtz equations: SIAM Journal on Scientific Computing, **23**, 1291–1315, doi: 10.1137/S1064827501357190.

Engquist, B., and A. Majda, 1977, Absorbing boundary conditions for numerical simulation of waves: Mathematics of Computation, **31**, 629–651, doi: 10.1090/S0025-5718-1977-0436612-4.

Erlangga, Y., and R. Nabben, 2008, Multilevel projection-based nested Krylov iteration for boundary value problems: SIAM Journal on Scientific Computing, **30**, 1572–1595, doi: 10.1137/070684550.

Erlangga, Y. A., 2005, A robust and efficient iterative method for the numerical solution of the Helmholtz equation: Ph.D. thesis, Delft University of Technology.

Erlangga, Y. A., and F. J. Herrmann, 2008, An iterative multilevel method for computing wavefields in frequency-domain seismic inversion: 78th Annual International Meeting, SEG, Expanded Abstracts, 1957–1960.

Erlangga, Y. A., C. W. Oosterlee, and C. Vuik, 2006, A novel multigrid based preconditioner for heterogeneous Helmholtz problems: SIAM Journal on Scientific Computing, **27**, 1471–1492, doi: 10.1137/040615195.

Erlangga, Y. A., C. Vuik, and C. Oosterlee, 2004, On a class of preconditioners for solving the Helmholtz equation: Applied Numerical Mathematics, **50**, 409–425, doi: 10.1016/j.apnum.2004.01.009.

Erlangga, Y. A., C. Vuik, and C. W. Oosterlee, 2003, On a class of preconditioners for solving the discrete Helmholtz equation: Mathematical and numerical aspects of wave propagation: University of Jyväskylä, Finland, 788–793.

Fornberg, B., 1988, Generation of finite difference formulas on arbitrarily spaced grids: Mathematics of Computation, **51**, 699–706, doi: 10.1090/S0025-5718-1988-0935077-0.

Fu, H., R. G. Clapp, O. Lindtjorn, T. Wei, and G. Yang, 2012, Revisiting finite differences and spectral methods on diverse parallel architectures: Computers & Geosciences, **43**, 187–196, doi: 10.1016/j.cageo.2011.09.017.

George, A., and J. W. H. Liu, 1981, Computer solution of large sparse positive definite systems: Prentice-Hall.

Griewank, A., and A. Walther, 2000, Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation: ACM Transactions on Mathematical Software, **26**, 19–45, doi: 10.1145/347837.347846.

Hagedoorn, J. G., 1954, A process of seismic reflection interpretation: Geophysical Prospecting, **2**, 85–127, doi: 10.1111/j.1365-2478.1954.tb01281.x.

Ji, Q., S. Suh, and B. Wang, 2012, GPU based layer-stripping TTI RTM: 82nd Annual International Meeting, SEG, Expanded Abstracts, 1–5.

Knibbe, H., C. W. Oosterlee, and C. Vuik, 2011, GPU implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method: Journal of Computational and Applied Mathematics, **236**, no. 3, 281–293, doi: 10.1016/j.cam.2011.07.021.

Knibbe, H., C. Vuik, and C. W. Oosterlee, 2013, 3D Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method on multi-GPUs, *in* Proceedings of ENUMATH 2011, the 9th European Conference on Numerical Mathematics and Advanced Applications, Leicester, 653–661.

Lailly, P., 1983, The seismic inverse problem as a sequence of before stack migration: Proceedings of the Conference on Inverse Scattering: Theory and Applications, SIAM, 206–220.

LGM 2012, The Little Green Machine: Massive many-core supercomputer at low environmental cost: http://www.littlegreenmachine.org.

Liu, H., B. Li, H. Liu, X. Tong, Q. Liu, X. Wang, and W. Liu, 2012, The issues of prestack reverse time migration and solutions with graphic processing unit implementation: Geophysical Prospecting, **60**, 906–918, doi: 10.1111/j.1365-2478.2011.01032.x.

Louis, A., P. Maas, and A. Rieder, 1997, Wavelet: Theory and applications: John Wiley and Sons.

Mallat, S., 2008, A wavelet tour of signal processing, 3rd edition: The sparse way: Academic Press.

Marfurt, K. J., and C. S. Shin, 1989, The future of iterative modeling of geophysical exploration: Supercomputers in seismic exploration: Pergamon Press, 203–228.

Micikevicius, P., 2009, 3D finite difference computation on GPUs using CUDA: GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, 79–84.

Mulder, W. A., and R.-E. Plessix, 2002, Time- versus frequency-domain modelling of seismic wave propagation: Presented at the 64th Annual Conference and Exhibition, EAGE, Extended Abstract E015, 27–30.

Mulder, W. A., and R.-E. Plessix, 2004a, A comparison between one-way and two-way wave-equation migration: Geophysics, **69**, 1491–1504, doi: 10.1190/1.1836822.

Mulder, W. A., and R.-E. Plessix, 2004b, How to choose a subset of frequencies in frequency-domain finite-difference migration: Geophysical Journal International, **158**, 801–812, doi: 10.1111/j.1365-246X.2004.02336.x.

Operto, S., J. Virieux, P. Amestoy, J.-Y. L'Excellent, L. Giraud, and H. B. H. Ali, 2007, 3D finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study: Geophysics, **72**, no. 5, SM195–SM211, doi: 10.1190/1.2759835.

Plessix, R.-E., 2008, A Helmholtz iterative solver for 3D seismic-imaging problems: Geophysics, **72**, no. 5, SM185–SM194, doi: 10.1190/1.2738849.

Plessix, R.-E., and W. A. Mulder, 2004, Frequency-domain finite-difference amplitude-preserving migration: Geophysical Journal International, **157**, 975–987, doi: 10.1111/j.1365-246X.2004.02282.x.

Riyanti, C., Y. Erlangga, R.-E. Plessix, W. A. Mulder, C. Vuik, and C. W. Oosterlee, 2006, A new iterative solver for the time-harmonic wave equation: Geophysics, **71**, no. 5, E57–E63, doi: 10.1190/1.2231109.

Riyanti, C. D., A. Kononov, Y. A. Erlangga, C. Vuik, C. W. Oosterlee, R.-E. Plessix, and W. A. Mulder, 2007, A parallel multigrid-based preconditioner for the 3D heterogeneous high-frequency Helmholtz equation: Journal of Computational Physics, **224**, 431–448, doi: 10.1016/j.jcp.2007.03.033.

Saad, Y., 2003, Iterative methods for sparse linear systems: SIAM.

Saleh, B., 2011, Introduction to subsurface imaging: Cambridge University Press.

Shen, X., and R. G. Clapp, 2011, Random boundary condition for low-frequency wave propagation: 81st Annual International Meeting, SEG, Expanded Abstracts, 2962–2965.

Stevens, W. R., 1998, UNIX network programming, Volume 1, 2nd ed.: Networking APIs: Sockets and XTI: Prentice Hall.

Symes, W. W., 2007, Reverse time migration with optimal checkpointing: Geophysics, **72**, no. 5, SM213–SM221, doi: 10.1190/1.2742686.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, **49**, 1259–1266, doi: 10.1190/1.1441754.

van Gijzen, M. B., Y. A. Erlangga, and C. Vuik, 2007, Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplace: SIAM Journal on Scientific Computing, **29**, 1942–1958, doi: 10.1137/060661491.

Wang, S., M. V. de Hoop, and J. Xia, 2010, Acoustic inverse scattering via Helmholtz operator factorization and optimization: Journal of Computational Physics, **229**, 8445–8462, doi: 10.1016/j.jcp.2010.07.027.

Wang, S., M. V. de Hoop, and J. Xia, 2011, On 3D modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver: Geophysical Prospecting, **59**, 857–873.

Whitemore, N. D., 1983, Iterative depth imaging by backward time propagation: 53rd Annual International Meeting, SEG, Expanded Abstracts, 382–385.

Zhebel, E., 2006, A multigrid method with matrix-dependent transfer operators for 3D diffusion problems with jump coefficients: Ph.D. thesis, Technical University Bergakademie Freiberg.