GEOPHYSICS®

DIRECT FREQUENCY-DOMAIN 3D ACOUSTIC SOLVER WITH INTERMEDIATE DATA COMPRESSION BENCHMARKED AGAINST TIME-DOMAIN MODELING FOR FWI APPLICATIONS

Journal:	*Geophysics*
Manuscript ID	GEO-2018-0465.R1
Manuscript Type:	Technical Paper
Keywords:	3D, acoustic, finite difference, frequency-domain, parallel
Area of Expertise:	Seismic Modeling and Wave Propagation

Direct frequency-domain 3D acoustic solver with intermediate data compression benchmarked against time-domain modeling for FWI applications

Victor Kostin[1], Sergey Solovyev[1], Andrey Bakulin[2] and Maxim Dmitriev[2]

Right Running Head: Direct 3D Helmholtz solver

[1]Institute of Petroleum Geology and Geophysics, SB RAS, 3, Akad. Koptyug ave., Novosibirsk, Russian Federation.
E-mail: KostinVI@ipgg.sbras.ru,
E-mail: SolovyevSA@ipgg.sbras.ru.

[2]Geophysics Technology, EXPEC ARC, Saudi Aramco, Building 137, Dhahran 31311, Saudi Arabia.
E-mail: andrey.bakulin@aramco.com,
E-mail: maxim.dmitriev@aramco.com

ABSTRACT

We present a fast direct solver for numerical simulation of acoustic waves in 3D heterogeneous media. The Helmholtz equation is approximated by a 27-point finite-difference stencil of second order accuracy that is optimized to reduce the numerical dispersion. Due to the optimization, dispersion errors less than 1%, are achieved with model discretization of only five points per wavelength. Wave propagation problem requires solving a large system of linear equations with complex sparse symmetric coefficient matrix with seismic shots representing the right hand side. The first step is the triangular factorization of the coefficient matrix followed by solving systems of linear equations with triangular coefficient matrices as a second step. Having defined the triangular factors, the second step is very cheap, and its linear scaling with respect to the number of shots is the main advantage of direct methods. To reduce memory consumption and computational time at the factorization step, the lower triangular factors are compressed using low-rank approximation of their nonzero blocks. The compression enables rapid solving of systems of more than 10^8 equations corresponding to realistic geophysical models. Accuracy and performance comparison of our solver with a highly optimized time-domain solver proves that these approaches complement each other - depending on the problem size and computing configuration either solver may be preferable.

INTRODUCTION

Numerical simulation of acoustic wavefields in the frequency domain represents an important capability to solve problems arising in exploration geophysics. In particular, it serves as an engine for acoustic Frequency-Domain Full Waveform Inversion (FD FWI) (Mulder and Plessix, 2004; Shin and Cha, 2008; Virieux et al., 2009; Etienne et al., 2014). For macro velocity reconstruction, such simulation is usually performed many times for a number of low frequencies (up to 20 Hz) at each iteration of this process.

In acoustic simulations, the pressure wavefield is excited by a point source working as a harmonic oscillator at a particular frequency. In exploration seismics, the number of shots can be 10,000's or more, with receivers also numbering in the 10,000's, especially with the proliferation of high-channel-count land seismic acquisition systems, some of which reach one million channels (Ourabah et al., 2015).

The time or frequency domain approaches can be used to perform seismic modeling and FWI (Vigh, and Starr, 2008; Virieux, and Operto, 2009; Plessix, 2017). A conventional solver is based on time-domain simulation followed by a Fourier transform to convert the time-domain wavefields to the frequency-domain. A common strategy consists of distributing the seismic sources over processors. There is no need to exchange data between processors, and the strategy becomes very effective. Good scalability and moderate memory requirements make the approach attractive for industrial purposes. However, to be efficient in parallel processing tens of thousands of seismic sources, the approach requires significant computational resources. In this paper, we provide some experimental data on the solver developed by the Seiscope consortium (see https://seiscope2.osug.fr).

Alternatively, seismic modeling and FWI can be performed in the frequency-domain. A problem on time harmonic solutions of the wave equation transforms into a boundary value problem for the Helmoltz equation. Absorbing boundary conditions are set to reduce reflections of waves from the boundaries. Only a few discrete frequencies are needed to build a reliable macrovelocity model, and this is the main advantage of using the frequency-domain approach. From a practical perspective, in 3D simulations, this requires solving a system of linear equations with huge and sparse coefficient matrix which is some kind of approximation of the boundary value problem. The right-hand sides of the system represent the seismic sources.

Iterative methods can be used to solve the system of linear equations. The Krylov subspace iteration methods demonstrate good efficiency (Plessix, 2007; Erlangga and Nabben, 2008; Belonosov et al., 2017; Belonosov et al., 2018) and moderate memory requirements, but they require good preconditioners and sometimes convergence may be slow or even lost.

With increased computational power of modern computers (especially distributed memory systems, or clusters), it is now becoming more feasible to apply direct methods for solving systems of linear equations with sparse coefficient matrices. The LDLT decomposition of the coefficient matrix is first performed before computing the solutions by forward/backward substitutions. The factorization is computationally expensive, but it is performed only once per frequency because it is independent of the right-hand-side term. The LDLT factorization leads to significant fill-in of the matrix (i.e. the number of nonzeros increase). The fill-in strongly depends on ordering the columns and rows of the matrix. To reduce the fill-in, the Nested Dissection (ND) method to order the matrix entries is used. For 3D problems solved with a finite-difference (FD) method, use of the ND method results in reduction of the fill-in by one order of magnitude ($O(n^4)$ versus $O(n^5)$), where n stands for the dimension of a 3D n^3 cubic grid (George, 1973, George, et al., 1994).

A wide class of matrices arising as discretization from partial differential equations have a so called low-rank property. That is, the fill-in in its direct factorization has low rank off-diagonal blocks. Based on this property, the fill-in can be approximated by low-rank matrices or matrices of special hierarchical structure with low-rank blocks (Chandrasekaran et al., 2006; Xia et al., 2010; Ghysels et al., 2016). Structured methods (Xia, 2013; Wang et al., 2016) are based on using the structural low-rank approximation, and this helps to improve theoretical performance.

In this paper, we present a multifrontal direct method of solving a system of linear equations that arises as an optimal 27-point FD approximation of a boundary value problem for the Helmholtz equation. In this method, the coefficient matrix constructed using special kind of ordering is factorized in a product of triangular matrices. To find the unknowns, two systems with triangular coefficient matrices need to be solved.

Compared to time-domain methods or the frequency-domain iterative methods, the method has an advantage: compared to the cost of the matrix factorization extra right-hand sides do not appreciably increase the computational cost (Bakulin et al., 2018). To reduce the issue of inordinate memory consumption typical for direct solvers, we apply data compression based on the Low-Rank approximation and hierarchical formats of storing data (Aminfar et al., 2016). We further implement Message Passing Interfaces-based (MPI) parallelization inside the solver in order to optimize execution on Distributed Memory High-Performance Computers (clusters).

Several ways of utilizing the low-rank property in algorithms for numerical solution of frequency-domain wave equations are known. Block Low Rank methods (Amestoy et al., 2015, Pichon et al., 2018) were proposed as simple alternatives to structured methods. Although they cannot reach the performance of the structured methods, simplicity of implementation is their attractive feature. Hierarchically Semiseparable (HSS) formats (Xia et al., 2010; Wang et al., 2011; Xia, 2013) were proposed to achieve the best theoretical performance. We apply plain low-rank approximations for subdiagonal blocks of the triangular factor of the LDLT factorization and use the Hierarchical Off-Diagonal Low Rank (HODLR) format for approximation of the diagonal blocks. A similar approach was used by Glinskiy, et al. (2017) but MPI optimization in that paper was targeted to minimize the peak memory consumption. In the current paper the goal is performance improvement.

This paper is organized as follows: The first section discusses the setting of the boundary value problem for the Helmholtz equation, use of Perfectly Matched Layers (PML) to diminish wave reflections from the boundaries and a FD approximation of the boundary value problem. We describe a procedure for minimization of numerical dispersion and use it to construct the optimal 27-point stencil. Compared to classical seven-point stencil, the optimization results in a 1.5x decrease of the number of points per wavelength needed to keep the numerical dispersion error within 1%. In 3D, this decrease leads to a 3x reduction of the coefficient matrix size and even bigger factor for the peformance. We describe a special ordering of the grid points defined by the ND method which results in a special structure of the matrix.

Factorization and solving steps are the main ingredients of our direct method of solving systems of linear equations. The factorization process includes compression of matrix blocks based on the low-rank approximation. The second section is devoted to details of the software implementation. In the last section, results of the numerical experiments are given. Using two realistic subsurface models we demonstrate the accuracy of the solver, its MPI scalability and performance. We compare the solver to the industrial time-domain solver provided by Seiscope consortium and demonstrate that both solvers complement each other.

METHODS

A finite-difference scheme for the Helmholtz equation

In the frequency domain, acoustic wave propagation in R^3 is governed by the Helmholtz equation

$$\Delta u(x) + k^2(x)u(x) = f(x), \qquad (1)$$

where Δ is the Laplace operator, $u(x) = u(x_1, x_2, x_3)$ is the unknown function (pressure), $k(x) = \omega/c(x)$ is the wavenumber, ω is the angular temporal frequency, and $c(x)$ is the sound velocity at point x [1]. The pressure is excited by a source represented by function $f(x)$ in the right hand side . For a seismic shot located at source point $x^s = (x_1^s, x_2^s, x_3^s)$ the right hand side has a form of delta-function

$$f(x) = \delta(x - x^s). \qquad (2)$$

If equation 1 is considered in an unbounded domain, for unique resolvability some condition (e.g. radiation condition (Vainberg, 1966)) should be posed at infinity.

In practice, the solution is to be found in some parallelepiped D where the velocity function is given and the source point is located. To reduce the influence on the solution of the limited computational domain, we apply PML technique (see, e.g. Berenger, 1994; Bermúdez et al., 2007). For this purpose, D is immersed in some bigger parallelepiped $\tilde{D}$ as shown schematically on Figure 1. If free-surface boundary conditions need to be imposed, the respective faces of D and $\tilde{D}$ lie in

[1] To avoid cumbersomeness in formulas, we use x along with triples (x_1, x_2, x_3) and (x, y, z) as notations for independent spatial variables. It should be clear from the context which notation is used, and we hope that such shortcut would not confuse the reader.

the same plane. The differential equation is extended to $\tilde{D}$ in the form of second-order partial differential equation

$$\sum_{i=1}^{3} \alpha_i(x_i)\frac{\partial}{\partial x_i}\left(\alpha_i(x_i)\frac{\partial u}{\partial x_i}\right) + k^2(x)u(x) = f(x), x \in \tilde{D}. \qquad (3)$$

In this equation, $k(x)$ is smoothly extended to D_{PML} , $f(x)$ is extended by zero. Functions

$$\alpha_i(x_i) = \frac{i\omega}{i\omega + d_i(x_i)} \qquad (4)$$

are complex-valued defined via damping functions $d_i(x_i)$ which are zeros for points within D. Respectively, $\alpha_i(x_i)$ are equal to one in D, and equation 3 coincides with equation (1). To provide attenuation of the solution, damping functions are positive within respective PMLs. To keep symmetry of the coefficient matrix in the FD approximation, we divide the partial differential equation 3 by $\alpha(x) = \alpha_1(x_1)\alpha_2(x_2)\alpha_3(x_3)$ and obtain

$$\sum_{i=1}^{3} \frac{\alpha_i(x_i)}{\alpha(x)}\frac{\partial}{\partial x_i}\left(\alpha_i(x_i)\frac{\partial u}{\partial x_i}\right) + \frac{k^2(x)}{\alpha(x)}u(x) = \frac{f(x)}{\alpha(x)}. \qquad (5)$$

On the boundary of domain $\tilde{D}$ the homogeneous Dirichlet condition is posed

$$u(x)|_{x \in \partial\tilde{D}} = 0. \qquad (6)$$

Differential equation 5 (or 3) along with boundary condition 6 is the boundary value problem used for numerical simulation of acoustic waves in the frequency domain.

For numerical solution of this boundary value problem we first introduce a rectangular equidistant grid in $\tilde{D}$. The grid points (x_i,y_j,z_k) are defined as multiples of integer triplets (i,j,k) with grid step h. A discrete analog u_{ijk} of the unknown function $u(x,y,z)$ is defined as $u_{ijk} = u(x_i,y_j,z_k)$ and discretization f_{ijk} of the right-hand side $\frac{f(x)}{\alpha(x)}$ is similarly done.

Differential equation 5 is approximated using FD schemes. To reduce impact of numerical dispersion, one can use FD schemes of higher order of approximation (Dablain, 1986, Liu and Sen, 2011). Such schemes use wider stencils which may result in cost increase at the stage of matrix factorization.

To approximate the differential equation at point (x_i,y_j,z_k), we use a 27-point FD approximation of second-order accuracy

$$\sum_{i_0=-1}^{1}\sum_{j_0=-1}^{1}\sum_{k_0=-1}^{1}\beta_{i,i_0,j,j_0,k,k_0}u_{i+i_0,j+j_0,k+k_0}=f_{ijk} \tag{7}$$

to boundary value problem 5 and 6. To define coefficients $\beta_{i,i_0,j,j_0,k,k_0}$ we represent the left-hand side of the equation 7 as a linear combination of seven terms with weight coefficients

$$\gamma_1\Delta_1 u_{ijk}+\gamma_2\Delta_2 u_{ijk}+\gamma_3\Delta_3 u_{ijk}$$
$$+k^2(x_i,y_j,z_k)\left(w_1 u_{ijk}+\frac{w_2}{6}\sum_{(2)}u_{i_1,j_1,k_1}+\frac{w_3}{12}\sum_{(3)}u_{i_1,j_1,k_1}+\frac{w_2}{8}\sum_{(4)}u_{i_1,j_1,k_1}\right). \tag{8}$$

Operator Δ_j, $(j=1,2,3)$ in sum 8 is a second-order accuracy FD approximation of the Laplace operator represented by stencil marked (j) in Figure 2. In fact, in Figure 2 only $z-$parts of respective stencils are depicted. Obviously, to get an approximation of the Laplace operator, the sum of weights γ_j should be equal to one. Likewise, sums in parentheses in expression 8 are the second order accuracy approximations of $u(x_i,y_j,z_k)$ that correspond to stencils depicted in Figure 3. In these stencils, the red ball denotes the central point and does not participate in approximations. Respective weights should also sum up to one.

For FD operator 8, the dispersion analysis gives an explicit expression c_{ph} $(\varphi,\theta,G;\gamma_1,\gamma_2,\gamma_3,w_1,w_2,w_3,w_4)$ for the phase velocity of a plane wave propagating in direction defined by spherical angles φ,θ. Here, $G=\frac{\omega h}{2\pi c}$ is the reciprocal of the number of grid points per wavelength. Function

$$\Psi(\gamma_1,\gamma_2,\gamma_3,w_1,w_2w_3w_4)=\iiint\limits_D \left|\frac{c_{ph}(\varphi,\vartheta,G;\gamma_1,\gamma_2,\gamma_3,w_1,w_2w_3w_4)}{c}-1\right|^2 d\varphi d\theta dG \quad (9)$$

represents the squared dispersion error averaged with respect to directions and values of G. Domain of integration in integral 9 is defined as

$$D=\left\{(\varphi,\theta,G):0\le\varphi\le 2\pi,\ -\frac{\pi}{2}\le\theta\le\frac{\pi}{2},G_{\min}\le G\le G_{\max}\right\}. \quad (10)$$

Minimizing function 9 with respect to weights we find the optimal values. In Table 1 one can find the optimal weights for range of frequencies 4 - 10 Hz.

Dispersion curves for 50 plane waves propagating in randomly chosen directions are depicted in Figure 4. It is clear, that eight points per wavelength is sufficient to keep the dispersion error less than 1%. To justify our choice of using a 27-point optimal stencil, we note that for the classical seven-point approximation to reach the level of 1% dispersion error, twelve points per wavelength is required. Optimization of FD stencils for minimization of the numerical dispersion is a popular topic of many papers (Jo, et al., 1996; Hustedt et al., 2004; Operto et al., 2007).

A system of linear algebraic equations

FD approximation 7 of boundary value problem 5, 6 takes the form of a system of linear equations

$$\mathbf{Au}=\mathbf{f}, \quad (11)$$

where $\boldsymbol{u}$ stands for a column of values u_{ijk} at grid points of the unknown function, and $\mathbf{f}$ has components f_{ijk}. For any grid point (i,j,k) there are at most 27 non-zero coefficients in equation

7. In other words, matrix $\mathbf{A}$ is sparse, complex valued (due to PMLs) and symmetric (but not Hermitian). Coefficients $\beta_{i,0,j,0,k,0}$ come to the diagonal of the matrix whereas the locations of remaining 26 non-zero coefficients depend on the ordering of the grid points.

To order the grid points, we use the Nested Dissection (ND) algorithm characterized by substantial reduction in memory consumption (George, 1973; George et al., 1994; Davis et al., 2016) compared to conventional ordering. For convenience, we illustrate the ND algorithm in Figure 5 where the computational domain is depicted as a parallelepiped. The grid points are not shown for simplicity. The algorithm deals with subsets of the finite set of grid points, but, for visibility, we operate with geometrical objects and terms. We sequentially divide the domain into subdomains using separators until the sizes of the subdomains reach computationally manageable level.

At the first step, in Figure 5(a), the *separator* is shown as a horizontal gray plane labeled #15. The plane passes through the grid points and divides the parallelepiped into approximately equal subdomains. The entire set of grid points disjoins into three subsets: the upper (the first subset) and the lower (the second subset) subdomains, and the separator itself (the third subset).

Let us sequentially order the grid points (subset one, subset two, subset three). At this point, ordering within the subsets is not specified.

A stencil centered at some grid point (i,j,k) connects the center with 26 *neighbors*, i.e. points whose grid coordinates differ from the center coordinates by not more than one. Obviously, for a center from the upper subdomain the neighbors belong to the upper subdomain, or, possibly to the separator. From this observation, we get the following block structure

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{\mathrm{uu}} & 0 & \mathbf{A}_{\mathrm{u},\mathrm{s}_{15}} \\ 0 & \mathbf{A}_{\mathrm{ll}} & \mathbf{A}_{\mathrm{l},\mathrm{s}_{15}} \\ \mathbf{A}_{\mathrm{s}_{15},\mathrm{u}} & \mathbf{A}_{\mathrm{s}_{15},\mathrm{l}} & \mathbf{A}_{\mathrm{s}_{15},\mathrm{s}_{15}} \end{pmatrix}. \tag{12}$$

of the coefficient matrix.

The process can be repeated recursively. In Figure 5(b), a vertical gray plane partitions the subdomains. This vertical plane contains two separators #13 and #14. We order the grid points so that points from separators #13 and #14 and #15 are put at the end of the list and grid points from four subdomains are arranged first, and we have a block 7×7-matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & 0 & 0 & 0 & \mathbf{A}_{1,\mathrm{s}_{13}} & 0 & \mathbf{A}_{1,\mathrm{s}_{15}} \\ 0 & \mathbf{A}_{22} & 0 & 0 & \mathbf{A}_{2,\mathrm{s}_{13}} & 0 & \mathbf{A}_{2,\mathrm{s}_{15}} \\ 0 & 0 & \mathbf{A}_{33} & 0 & 0 & \mathbf{A}_{3,\mathrm{s}_{14}} & \mathbf{A}_{3,\mathrm{s}_{15}} \\ 0 & 0 & 0 & \mathbf{A}_{44} & 0 & \mathbf{A}_{3,\mathrm{s}_{14}} & \mathbf{A}_{4,\mathrm{s}_{15}} \\ \mathbf{A}_{\mathrm{s}_{13},1} & \mathbf{A}_{\mathrm{s}_{13},2} & 0 & 0 & \mathbf{A}_{\mathrm{s}_{13},\mathrm{s}_{13}} & 0 & \mathbf{A}_{\mathrm{s}_{13},\mathrm{s}_{15}} \\ 0 & 0 & \mathbf{A}_{\mathrm{s}_{14},3} & \mathbf{A}_{\mathrm{s}_{14},4} & 0 & \mathbf{A}_{\mathrm{s}_{14},\mathrm{s}_{14}} & \mathbf{A}_{\mathrm{s}_{14},\mathrm{s}_{15}} \\ \mathbf{A}_{\mathrm{s}_{15},1} & \mathbf{A}_{\mathrm{s}_{15},2} & \mathbf{A}_{\mathrm{s}_{15},3} & \mathbf{A}_{\mathrm{s}_{15},4} & \mathbf{A}_{\mathrm{s}_{15},\mathrm{s}_{13}} & \mathbf{A}_{\mathrm{s}_{15},\mathrm{s}_{14}} & \mathbf{A}_{\mathrm{s}_{15},\mathrm{s}_{15}} \end{pmatrix}. \qquad (13)$$

In this formula matrix block indices 1, 2, 3, 4 indicate grid points from subdomains, whereas s_{13}, s_{14}, s_{15} relate to points from the respective separators.

The third level separators #9, #10, #11, #12 are shown in Figure 5(c). Every higher level separator partitions subdomains obtained at the previous level in two smaller subdomains and the separator itself. So, k steps of such process produce 2^k subdomains and $2^k - 1$ separators. The grid points are ordered so that separators in reverse ordering follow the points from subdomains. The coefficient matrix becomes a block $(2^{k+1} - 1) \times (2^{k+1} - 1)$ matrix. For $k = 3$ the structure is depicted in the left image of Figure 6.

LDLT factorization with compression

To solve the system of linear equations 8 with a sparse complex symmetric coefficient matrix by a direct method, LDLT factorization is first applied:

$$\hat{\mathbf{A}} = \mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^{\mathbf{t}} = \mathbf{L} \cdot \mathbf{D} \cdot \mathbf{L}^{\mathbf{t}}. \qquad (14)$$

Here $\mathbf{P}$ denotes a permutation matrix that appear due to pivoting, $\mathbf{D}$ is a block-diagonal matrix with blocks of size one or two, $\mathbf{L}$ is a lower triangular matrix with units on the diagonal, 't' indicates the transposed of the matrix. Factorization 14 is considered as a variant of Gauss elimination (see e.g. Bunch et al., 1976; Duff et al., 1979; Duff et al., 1983; Schenk and Gärtner, 2006).

Let the matrix be represented in a block form as

$$\hat{\mathbf{A}} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1K} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{K1} & \mathbf{A}_{K2} & \cdots & \mathbf{A}_{KK} \end{pmatrix}, \qquad (15)$$

with $n_i \times n_j -$ blocks $\mathbf{A}_{ij}$. Diagonal blocks $\mathbf{A}_{ii}$ are square though may have different orders n_i. Block structure of matrices $\boldsymbol{L}$ and $\boldsymbol{D}$ is as follows

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_{11} & & \cdots & \\ \mathbf{L}_{21} & \mathbf{L}_{22} & \cdots & \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}_{K1} & \mathbf{L}_{K2} & \cdots & \mathbf{L}_{KK} \end{pmatrix}, \quad \mathbf{D} = \begin{pmatrix} \mathbf{D}_1 & & \cdots & \\ & \mathbf{D}_2 & \cdots & \\ \vdots & \vdots & \ddots & \vdots \\ & & \cdots & D_K \end{pmatrix}. \qquad (16)$$

Diagonal blocks $\mathbf{L}_{jj}$ are lower triangular square matrices of order n_j with units on the diagonals, $\mathbf{D}_j$ are diagonal block matrices with blocks of size one or two.

A pseudo-code of a block LDLT algorithm (see Algorithm 1) uses function FLDLT that solves the LDLT factorization problem for diagonal blocks of sizes $n_i \times n_i$. These problems are considered as 'elementary' with respect to block structure of matrix $\hat{\mathbf{A}}$.

```
for j=1:K
    for p =1:j-1
        $\mathbf{A}_{jj} = \mathbf{A}_{jj} - \mathbf{L}_{jp}\mathbf{D}_p\mathbf{L}_{jp}^t$        (17)
    end
    $[\mathbf{L}_{jj}, \mathbf{D}_j, \mathbf{P}_j] = \mathrm{FLDLT}(\mathbf{A}_{jj})$        (18)
    for i=j+1:K
```

```
        for $p = 1{:}j-1$
            $\mathbf{A}_{ij} = \mathbf{A}_{ij} - \mathbf{L}_{ip}\mathbf{D}_p\mathbf{L}_{jp}^t$        (19)
        end
        $\mathbf{L}_{ij} = \mathbf{A}_{ij}\mathbf{P}_j^t\mathbf{L}_{jj}^{-t}\mathbf{D}_j^{-1}$        (20)
    end
end
```

Algorithm 1: Pseudo-code of a block version of LDLT factorization. Sparsity of the matrix is taken into account by skipping values of indices i and p that correspond to fully zero blocks in equations 17, 19 and 20.

In equations 18 and 20, $\mathbf{P}_j$ is a permutation matrix that appears due to 'restricted' pivoting applied to diagonal blocks of the matrix being decomposed in order to improve numerical stability. Low impact on performance is the main reason why we choose restricted pivoting. However, restricted pivoting is expected to be less efficient for numerical stability than the partial pivoting that involves the elements of the whole panel. To attest the method, we refer to Schenk and Gärtner, 2006 where similar technique is described. Our numerous experiments confirm numerical stability of the results.

Factorization traverses from the tree bottom to its top. Matrix $\mathbf{L}$ inherits the block structure of the lower triangle of matrix $\hat{\mathbf{A}}$, but its blocks have more nonzero elements than blocks of $\hat{\mathbf{A}}$ (*fill-in* phenomenon). The phenomenon can be easily understood from equation 19 for Schur updates. Gray shading in Figure 6 (left) is used to illustrate the 'fill-in factor' (ratio of the number of nonzero elements in the block to the total number of the block elements). The fully zero blocks are white colored, and those in black mean the fill-in factor is close to one. As illustrated in Figure 6 (left) blocks on the lower and right side of matrix L have bigger fill-in factors and may be larger in size.

Figure 6 depicts the structure of factor **L**, but to some extent, this also describes the lower triangle of matrix $\hat{\mathbf{A}}$ (the upper triangle can be obtained by reflecting the picture with respect to the main diagonal). The elimination tree (ET) for matrix $\hat{\mathbf{A}}$ has a binary structure (Figure 7). Recall that for LDLT factorization, dependencies among panels are indicated by the ET edges, and these dependencies denote directions how updates move from a 'son' to its 'father', 'grandfather' and beyond up to the tree 'root' along the tree branches. For example, Figure 7 shows that panel #13 depends on panels #9 and #10, which in turn depend on #1, #2, #3, #4. Panel #13 receives updates not only from its 'sons' #9 and #10 but also from 'grandsons' #1, #2, #3, #4. These updates are independent of #9 and #10 - this additional parallelism can be used for performance optimization.

Fortunately, for the FD approximation of the Helmholtz equation, matrix L blocks lying below the diagonal possess data sparsity property, i.e. they are Low-Rank. $M \times N$-matrix **B** is called *data sparse* if it can be approximated by a matrix $\mathbf{B}^r$ of rank $r \ll \min(M,N)$. Such an approximation can be written as

$$\mathbf{B} = \mathbf{B}^r + \boldsymbol{\delta}\mathbf{B} = \mathbf{U} \cdot \mathbf{V}^t + \boldsymbol{\delta}\mathbf{B},\ \|\boldsymbol{\delta}\mathbf{B}\| < \varepsilon\|\mathbf{B}\|\ , \qquad (21)$$

with $M \times r-$matrix **U**, $N \times r-$matrix **V** and $\varepsilon > 0$ being some threshold parameter. Matrices **U** and **V** are stored instead of **B**. Approximation 21 is used for data compression in the LDLT factorization algorithm we use; the lower the sum $r/M + r/N$, the bigger is the compression effect. The compression also helps to reduce floating point operations count.

The Low-Rank approximation in equation 21 provides a foundation for data compression. For spectral matrix norm or Frobenius norm, the truncated Singular Value Decomposition (SVD) (see Golub and Loan, 1996; Godunov et al., 2013) can be used to solve the minimization problem

$$\mathbf{B}^r = \arg \min_{\operatorname{rank} \mathbf{X} = r} \|\mathbf{B} - \mathbf{X}\|^2. \quad (22)$$

The solution can be considered as the optimal approximation of matrix **B** by rank r matrices. SVD provides a constructive solution $\mathbf{B}^r$ to the approximation problem 22 in the form of matrices **U** and **V** such that $\boldsymbol{B}^r = \mathbf{U} \cdot \mathbf{V}^t$ (cf. 21). The SVD algorithm is numerically stable but time consuming. Many alternative approaches can be found in the literature. We use an approach based on randomized sampling (Martinsson and Voronin, 2011) because of its high performance and robustness.

Data sparsity in matrix **L** blocks is schematically illustrated in Figure 6 (right) where some blocks are shown as containing two narrow black blocks schematically representing respective matrices **U** and $\mathbf{V}^t$ that are stored instead of the block they approximate. Usually these blocks are dense, which is why they are black in the Figure. Obviously, due to some overhead, approximation 21 of matrix **L** blocks makes sense for high enough fill-in factors (Figure 6, right) whereas some pale gray blocks remain 'as is'.

There is one more possibility to compress matrix **L**: diagonal blocks of this matrix in Figure 6 (right) can be approximated by Hierarchically Off-Diagonal Low Rank (HODLR) matrices (Aminfar et al., 2016; Glinskiy et al., 2017; Kostin et al., 2017). To get this structure, respective diagonal blocks of **L** are represented as 2×2 block matrices with sub-blocks of half size and approximation (21) is applied to the off-diagonal sub-block. In our solver, this procedure is applied recursively to the diagonal blocks several times until a small size is reached.

Solving step

Having defined factors in factorization 11, one can solve the system of linear equations 14 by inverting triangular matrices

$$\begin{aligned}&\mathbf{w} = \mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{P}\mathbf{f};\\ &\mathbf{u} = \mathbf{P}^t\mathbf{L}^{-t}\mathbf{w}.\end{aligned} \tag{23}$$

In a case of several right-hand sides, vectors $\mathbf{f}, \mathbf{y}, \mathbf{z}, \mathbf{u}$ should be treated as matrices comprised of respective column-vectors, and computations 23 can be parallelized with respect to columns.

During computation of factorization 14 matrix $\mathbf{L}$ is compressed to reduce memory consumption. The compressibility factor depends on threshold ε, frequency ω and some other parameters; in our experiments compressibility factor varied from 3 to 7. Due to compression, equation 14 is no longer valid and should be replaced with

$$\hat{\mathbf{A}} = \mathbf{L} \cdot \mathbf{D} \cdot \boldsymbol{L}^t + \boldsymbol{\delta}\hat{\mathbf{A}} \tag{24}$$

where $\boldsymbol{\delta}\hat{\mathbf{A}}$ denotes a perturbation due to compression errors. The solution obtained by formulas 21 may become inaccurate. To improve the accuracy one can carry out a few steps of the iterative refinement (Wilkinson, 1964). Having computed factorization 14, the overhead due to inclusion of the iterative refinement in the solving process is low.

Provided the norm of perturbation $\boldsymbol{\delta}\hat{\boldsymbol{A}}$ is small enough and the coefficient matrix is 'not-too-badly-conditioned', the iterative refinement converges. The threshold ε directly impacts the compressibility factor - the bigger its value, the bigger is the compressibility factor, suggesting that the threshold value should be increased. However, a larger threshold value implies an increase in the norm of perturbation $\boldsymbol{\delta}\hat{\mathbf{A}}$ in 24 which may lead to loss of convergence of the iterative refinement, or at least require more iterations to converge. Therefore, unjustifiably large threshold values should be avoided. Due to this trade-off, in practice, the threshold value is established by trial and error.

SOFTWARE IMPLEMENTATION DETAILS

Some standard functionality needed for our solver is taken from the Intel® Math Kernel Library (https://software.intel.com/en-us/intel-mkl). This library provides highly optimized BLAS (www.netlib.org/blas) and LAPACK (www.netlib.org/lapack) functions. In our solver we use OMP parallelization implicitly by calling threaded versions of BLAS and LAPACK functions. This is the only way of using OMP threading in the solver, there is no explicit OMP parallelization in it.

Our solver is targeted for distributed memory systems (clusters) using MPI. The binary structure of the ET of the LDLT factorization algorithm provides good MPI parallelization opportunities. Figure 8 shows mapping of the ET structure on compute nodes for the case of four nodes. Each subtree (denoted by dashed ellipses) is assigned to a particular compute node. The nodes are assumed to have big enough RAM to contain respective input data and results. It is clear that up to the level 3 (in Figure 8, this level is marked with a dashed line passing the ET nodes) of the ET, the compute nodes process their portions of data independently and there is no data exchange between them.

However, this kind of parallelism decreases while the process traverses up along the tree, and at a particular level of the tree, the number of nodes becomes less than the number of cluster nodes. Moreover, while moving up along the ET, fill-in factors of respective panels increase with increasing memory requirements and numbers of floating point operations.

Upon reaching the level where the ET width becomes less than the number of cluster nodes, finer granulation of the matrix being processed is applied. In other words, the panels are subdivided into subpanels which are assigned to separate cluster nodes. Unfortunately, such subdivision does not provide a binary structure of the ET suitable for parallelization. However, it allows delegating computation of some Schur updates to the lower part of the tree, thus improving parallelism.

One of costly operations in the LDLT factorization is the Schur update

$$\mathbf{A}_{ij} = \mathbf{A}_{ij} - \sum_{p<j} \mathbf{L}_{ip}\mathbf{D}_p\mathbf{L}_{jp}^t \tag{25}$$

(see the $p-$loop in Algorithm 1). For $j-$th node of the ET, equation 25 means gathering this node updates from sons, grandsons, etc. The number of summands in this formula may be rather large, and matrices $\mathbf{L}_{ip}$ are given factorized in products of 'thin' matrices and they are distributed along different cluster nodes. To apply the Schur update 25, we use different strategies depending on the position of ET node with respect to the dashed line.

For the ET nodes lying above the dashed line, or, equivalently, for the values j, we split the total number of updates (i.e. indices p) into pairs. Each pair is summed up on a separate cluster node. The process is repeated several times until all updates are summed up. Finally, the sum of all updates is added to block $\mathbf{A}_{ij}$. The number of steps is proportional to the logarithm of the total number of updates.

For the ET nodes below the dashed line, all operations are performed on the same compute node but there are also some peculiarities. As it was already mentioned above, nonzero subdiagonal blocks in the leftmost panels of the triangular factor are stored 'as is', without applying low-rank approximation. This is beneficial both from the performance and memory consumption points of view. So, to compute Schur update 25 with blocks in the conventional format, standard BLAS functionality (with some modifications due to sparsity of the blocks) is needed.

Starting from some level, the low-rank approximation is applied to the blocks, and the low-rank arithmetic is switched on. Let $m_1 \times n-$block $\mathbf{L}_{ip}$ and $m_2 \times n-$block $\mathbf{L}_{jp}$ be represented by low-rank approximants $\mathbf{U}_{ip} \cdot \mathbf{V}_{ip}^t$ and $\mathbf{U}_{jp} \cdot \mathbf{V}_{jp}^t$ where $\mathbf{U}_{ip}$ and $\mathbf{V}_{ip}$ have sizes $m_1 \times r_1$ and $n \times r_1$, whereas , $\mathbf{U}_{jp}$ and $\mathbf{V}_{jp}$ have sizes $m_2 \times r_2$ and $n \times r_2$, respectively. Assuming $r_1 \leq r_2$, update $\mathbf{L}_{ip}$

$\mathbf{D}_p\mathbf{L}_{jp}^t$ becomes a product of $m_1 \times r_1 -$ matrix $\mathbf{U}_{ip}$ and $r_1 \times m_2 -$ matrix $\left(\mathbf{U}_{jp}\left(\mathbf{V}_{jp}^t\left(\mathbf{D}_p^t\mathbf{V}_{ip}\right)\right)^t\right)^t$ (if the opposite inequality $r_1 > r_2$ takes place matrix $\mathbf{L}_{ip}\mathbf{D}_p\mathbf{L}_{jp}^t$ is decomposed in a product of $m_1 \times r_2 -$ matrix and $r_2 \times m_2 -$ matrix).

Summing up rank r_1 update $\mathbf{U}_1 \cdot \mathbf{V}_1^t$ and rank r_2 update $\mathbf{U}_2 \cdot \boldsymbol{V}_2^t$ can be obtained by concatenation of matrices

$$\mathbf{U}_1 \cdot \mathbf{V}_1^t + \mathbf{U}_2 \cdot \mathbf{V}_2^t = (\mathbf{U}_1 \quad \mathbf{U}_2)(\mathbf{V}_1 \quad \mathbf{V}_2)^t. \tag{26}$$

The result has the desired form but probably the representation is not optimal - matrices $(\mathbf{U}_1 \quad \mathbf{U}_2)$ and $(\mathbf{V}_1 \quad \mathbf{V}_2)$ have $r_1 + r_2$ columns which may be greater than their ranks. In other words, the product needs to be recompressed

$$(\mathbf{U}_1 \quad \mathbf{U}_2)(\mathbf{V}_1 \quad \mathbf{V}_2)^t = \mathbf{U}\mathbf{V}^t. \tag{27}$$

This is the same problem as the compression problem.

3D NUMERICAL EXPERIMENTS

In this section, we describe a few numerical experiments to validate our implementation. The experiments were performed on the Linux supercomputer Shaheen II (https://www.hpc.kaust.edu.sa/content/shaheen-ii) at King Abdullah University of Science and Technology (KAUST, Saudi Arabia). Top 500 Linpack performance of this cluster is 5.5 Pflops/s (https://www.top500.org/list/2018/11/?page=1). Each compute node is supplied with two processors Intel® Xeon® CPU E5-2698 v3 @ 2.3 GHz (32 cores in total) and 128 GB RAM; this configuration provides theoretical peak performance of 1.2 Tflops/s per node.

The threshold parameter was taken from the range 10^{-5}-10^{-4}. We used one MPI process per cluster node. The stopping criterion for the iterative refinement was set to achieve the following inequality

$$\frac{\|\mathbf{Au}-\mathbf{f}\|}{\|f\|} < 10^{-5}. \quad (28)$$

Models

To demonstrate the accuracy of the method first we compare the numerical solution for a homogeneous model to the analytical solution. For a synthetic model with the velocity being linearly dependent on the depth we compare solutions obtained with different grid steps.

Second subsurface model is a marine transition zone model referred below as the TZ model with dimensions 18,000 × 23,500 × 7,000 m. In this model, velocity varies between 1,042 m/s and 7,626 m/s (Figure 9). FD discretization with a grid size of 30 m results in a system of 1.3×10^8 linear equations.

As a third subsurface model, we use a portion of the SEG Overthrust (referred below as OT) model (Aminzadeh et al., 1997) with dimensions of 9870 × 9870 × 4620 m. In this model the velocity varies between 2,286 m/s and 6,000 m/s. FD discretization with a grid of 30 m in all directions results in a system of 2.4×10^7 linear equations.

Accuracy

First, we benchmark numerical solution of the Helmholtz equation

$$(\Delta + k^2)v = -\delta(x) \quad (29)$$

with the analytical solution available for a homogeneous medium

$$v(x,y,z) = \frac{e^{-ik\sqrt{x^2+y^2+z^2}}}{4\pi\sqrt{x^2+y^2+z^2}}. \quad (30)$$

We compute numerical solutions for different values of the grid step. Accuracy of the approximate solution $u^h(x,y,z)$ is measured by metrics

$$\beta_k(r) = \frac{\|u^h(x,y,z) - v(x,y,z)\|_{B(r_0,r),k}}{\|v(x,y,z)\|_{B(r_0,r),k}} \tag{31}$$

depending on the size r of the domain. In definition 31, index k stands for the type of the norm ($k = 1,2$) and

$$B(r_0,r) = \left\{(x,y,z) |\, r_0 \le \sqrt{x^2 + y^2 + z^2} \le r\right\} \tag{32}$$

denotes the concentric spherical layer where the norms are computed. Parameter r_0 is used to exclude the small singularity region around the origin. Metrics $\beta_k(r)$ are simple relative accuracies of the solution for the respective norms. Note that the accuracy also depends on the grid step h.

We provide numerical results for the homogeneous model with dimensions of 5120 × 5120 × 2560 m and sound velocity c_0=1280 m/s. The wavefield was excited by a monochromatic 4 Hz point source placed at the center of the model (2560 m, 2560 m, 1280 m). We used three different grid steps, $h = 64$, 32 and 16 m, that correspond to 5, 10 and 20 points-per-wavelength respectively. The width of PMLs was taken to be 15 grid steps. In Figure 10 one can find graphs of functions $\beta_k(r)$ for the described problem setting. It is clear, that halving the grid step results in a four times reduction of the relative errors β_1 and β_2 which confirms the second-order approximation of the method. For β_∞ while transitioning from $h/2$ to $h/4$ the reduction is worse which probably can be explained by stronger sensitivity of the L_∞ norm to perturbations.

For the second test, we used a 3D model with velocity being a linear function of depth and investigated convergence of the solution with decreasing grid step. For the same velocity model, we computed solutions u^h, $u^{h/2}$, $u^{h/4}$ for steps h, $h/2$ and $h/4$, respectively. The solution $u^{h/4}$ was used as a substitute for the exact solution. In Figure 11 one can find respective functions $\beta_k(r)$

which in this case provide information on deviation of solutions $u^h, u^{h/2}$from $u^{h/4}$. The second-order approximation is clearly confirmed for β_1 and β_2, but not for β_∞.

Strong scaling

For a problem which can be solved with an MPI solver and parallelization on different number of cluster nodes, the ‘scaling factor’ is defined as ratio t_{seq}/t_P where runtimes t_{seq} and t_P stand for one and P cluster nodes, respectively.

For the OT velocity model and different values of frequencies, we measured the scaling factors of our solver (Figure 12). The FDFD solver scales well for moderate cluster sizes, but scaling becomes worse as the number of cluster nodes increases beyond about 4 to 8 nodes. In addition, scaling factors deteriorate with increasing frequency. The problem size also impacts the scaling, with better scaling for larger models as illustrated in Figure 13 where scaling factors for the OT model and the TZ models are compared.

Scaling behavior can be explained as follows. The ET is less balanced on lower levels than on upper levels. The amount of computations defined by dashed ellipses in Figure 8 may vary from one ellipse to another. In other words, different cluster nodes are assigned different amounts of work, becoming unbalanced. Spatial variations of the sound velocity may cause unbalancing because numerical ranks depend on the local velocity and the frequency. At the highest level of the ET, the subtrees represent big parts of the domain that appear as results of the ND algorithm. Most likely the ‘average velocity’ for these big parts experiences minor variation from one part to another. However, if we move down along the tree the granulation becomes finer and variation of the velocity becomes higher. This leads to larger difference of numerical ranks, and associated unbalancing of the cluster nodes.

A similar cause of unbalancing also appears at lower levels of the ET when the parts of the computational domain become small parallelepipeds. These parallelepipeds have faces of two different types - the faces that are parts of the outer boundary of the domain and the faces that are parallel to separators. A small parallelepiped may have from zero to three faces that are parts of the outer boundary and from six to three faces parallel to separators. These differences cause different amount of computations associated with the different nodes of the ET.

Solution of a problem with the TZ model requires at least 32 cluster nodes, and we cannot use the definition of the scaling factor described above. In Figure 13, we show graphs of the ratios $t_{32}/t_P$where t_{32} stands for runtime on 32 cluster nodes. Figure 13 also shows data for the smaller OT model revealing that scaling and hence performance is better for larger systems of equations using the FDFD approach.

Comparison to time-domain solver

We compare our solver with a Time-Domain Finite-Difference solver (referred below as TDFD) developed by the Seiscope consortium (see https://seiscope2.osug.fr). To simulate wave propagation in time domain, the initial-boundary value problem for the wave equation is solved by TDFD. To eliminate reflection of waves from the boundaries, the domain is surrounded with PML's. The differential equation is approximated using fourth-order explicit FD scheme. The MPI implementation of the TDFD solver uses parallelization with respect to shots, so that one shot is assigned per single cluster compute node. No data exchange between nodes is performed because solutions for different shots are fully independent of each other. Dependence of the TDFD solver runtime on the number of seismic shots and the number of compute nodes can be described by formula

$$t_{\mathrm{TDFD}}(N_{\mathrm{nodes}}, N_{\mathrm{shots}}) = t_{\mathrm{TDFD}}(1,1)\left\lceil \frac{N_{\mathrm{shots}}}{N_{\mathrm{nodes}}} \right\rceil, \tag{33}$$

where $\lceil a \rceil$ stands for the least integer greater than or equal to a, and $t_{\mathrm{TDFD}}(1,1)$ depends on some other parameters like the problem size, the sound velocity variability, the hardware parameters, etc.

To get the solution in the frequency domain, Fourier transform is applied to the time domain solution. In TDFD, the frequency spectrum of the shots contains all time frequencies, so, multiple frequency components are obtained all at once.

To underline the different approaches, our method is referred as the FDFD solver which stands for Frequency-Domain Finite-Difference solver. Results of accuracy tests are provided in Figure 14 (the OT model) and Figure 15 (the TZ model) in terms of functions $\beta_k(r)$ introduced in the previous section. In Figures 14 and 15 these functions are used to demonstrate the difference between the solution obtained with TDFD solver and the solution obtained with the FDFD solver.

For both models, solutions were computed using parameters listed in Table 2. Note that for each model the solutions were computed for three different frequencies. The corresponding numbers of grid steps for the FDFD solver are shown in columns N_x, N_y and N_z, whereas the total number of grid points is in column N and the number of points-per-wavelength is listed under "ppw" column. For the TZ model, the snapshots of real part of the solution computed along the horizontal plane at a depth of one grid step are shown in Figure 16.

Time-domain solution obtained via the TDFD solver contains components that correspond to all three values of the frequency. So, the solver was run once with a grid step of 30 m. The modeling time interval was [0, 10 s].

For particular pairs of values $(N_{\text{nodes}}, N_{\text{shots}})$, the measured run times (in seconds) are provided in Tables 3 and 4 in the form of fractions, where run time $t_{\text{TDFD}} = t_{\text{TDFD}}(N_{\text{nodes}}, N_{\text{shots}})$ is put as the numerator, whereas $t_{\text{FDFD}}(N_{\text{nodes}}, N_{\text{shots}})$ is put as the denominator. Note that value of time for FDFD comprises of the sum of the runtime to solve the problem for all three frequencies. Ideal scalability of the TDFD solver with respect to N_{nodes} was assumed to fill in the table instead of repeating multiple shot.

Tables 3 and 4 summarize computation times of the two solvers in terms of N_{shots} and N_{nodes} (Figure 17). For few shots, TDFD solvers are relatively efficient, but for larger numbers of shots and nodes, FDFD becomes relatively more efficient per shot. The thick line (Figure 17) is the 'line of equal performance' of the two solvers. For a given number of N_{nodes}, this line defines the number of shots that is 'big enough' to fully reap the benefits of the FDFD solver and reach or exceed the numerical performance of TDFD solvers. For example, for a problem with a comparatively small number of shots that has to be solved on a particular cluster (a point close to point C of line segment CD), the fastest way would be using the TDFD solver. As the number of shots increases (the point moves to D along CD), starting from some number of shots, the FDFD direct solver becomes faster than TDFD solver. Another way to look at it is to fix the number of shots, such as defined by horizontal line AB. Then moving from point A towards B along horizontal segment AB means solving the same problem on clusters with increasing number of nodes. For a fixed number of shots, the FDFD direct solver would usually be faster for comparatively smaller clusters, but on bigger clusters the TDFD solver performs better. Such behavior can be explained with a help the following reasoning. For the TDFD solver, the runtime is roughly proportional to N_{shots} with coefficient $t_{\text{TDFD}}(1,1)/N_{\text{nodes}}$ (cf. 28). The runtime for the FDFD solver can be described by a sum $t_{\text{fact}}(N_{\text{nodes}}) + t_{\text{solve}}(N_{\text{nodes}}, N_{\text{shots}})$ representing time for the factorization and time for the solving step correspondingly. In this sum, the first summand does not depend on the number of shots. To some extent, the dependence of the summand can be thought as approximately linear $t_{\text{solve}}(N_{\text{nodes}}, N_{\text{shots}}) \cong t_{\text{solve}}(N_{\text{nodes}}, 1) \cdot N_{\text{shots}}$. Values of coefficients $t_{\text{TDFD}}(1,1)/N_{\text{nodes}}$ and $t_{\text{solve}}(N_{\text{nodes}}, 1)$ can be taken from Tables 3 or 4. For example, for the OT model $t_{\text{TDFD}}(1,1) = 161$, $t_{\text{solve}}(16, 1) = 0.4$. This simple insight explains that if the number of shots increases, then FDFD starts to win from a certain value of shots.

In general, the necessary and sufficient condition for existence of points $(N_{\text{nodes}}, N_{\text{shots}})$ where the FDFD wins is the following

$$\frac{t_{\text{TDFD}}(1,1)}{N_{\text{nodes}}} > t_{\text{solve}}(N_{\text{nodes}}, 1). \qquad (34)$$

To satisfy this condition the right side of this inequality should be inversely proportional to N_{nodes}, or, in other words, the solving step is MPI parallelized.

Performance: comparison to Intel® MKL Parallel Direct Sparse Solver for Clusters

To estimate the efficiency of a solver, its performance data in GFlops (giga flops per second) are used. Among other parameters (the hardware, the software, the number of MPI processes, etc.) these data depend on the problem (more precisely, size of the model and some parameters that characterize the problem). To get performance data for the solver, one has to count the number of floating point operations used to compute the solution of the problem and measure the runtime. This approach works well for comparatively simple algorithms where the flops counts can be calculated. Sparsity of the matrix, use of the Low-Rank approximation, incorporating the iterative refinement in computations makes impossible rigorous counting of the floating point operations.

Comparison to another solver is an alternative approach to get impression on the solver efficiency. We compared our solver to Parallel Direct Sparse Solver for Clusters from Intel® MKL (Intel® Parallel Studio XE 2017 Update 4) referred below as PARDISO. The comparison was made on a small homogeneous model of size $201 \times 201 \times 101$ referred below as SH and the medium size OT model described above (see Table 5). RAM data in Table 5 is the peak memory usage per node. It is clearly seen that our solver has essential advantage in memory usage but loses in performance on small problems. Starting from moderate size problems the solver outperforms MKL Cluster PARDISO.

CONCLUSIONS

We present a direct method for parallel solution of the acoustic wave equation in 3D heterogeneous media. To reduce memory consumption, the method utilizes intermediate data sparsity for compression. The compression technique is based on Low-Rank approximation of fill-in blocks. It is applied directly to blocks lying below the diagonal of the triangular factor. The diagonal blocks are also compressed using HODLR format. These methods help to solve a system of ~ 10^9 equations corresponding to large velocity models of interest to geophysical exploration. The strong scalability of the solver is good for a moderate number of cluster nodes and large enough problems. This direct solver is successfully used for geophysical modeling applications while some challenges remain to be resolved. Poor MPI scalability of the LDLT factorization for the number of compute nodes beyond 16 is one of the challenges.

Comparing the FDFD solver with a TDFD solver reveals that each method has its strengths and weaknesses. Detailed numerical experiments with real-world scenarios using a transition zone model and Shaheen II supercomputer demonstrate the existence of the “line of equal performance” defined in terms of number of shots and available nodes. The time-domain solver performs better for comparatively large ratios of N_{nodes}/N_{shots} , whereas the direct solver is relatively more efficient for smaller ratios of N_{nodes}/N_{shots}.

While generally larger number of nodes become available with time (as computing power becomes cheaper), seismic acquisition experiences significant growth in trace density per square kilometer enabled by larger number of both sources and receivers. Concurrent “data tsunami” and increase in compute power may lead to diverse scenarios that geophysical community needs to be prepared to handle. Depending on available computing power and number of shots for a particular imaging or FWI problem, one or the other solver may be significantly more efficient. Therefore, the optimal FWI toolbox should contain both solvers and apply the most efficient for the specific scenario based on the “line of equal performance”.

In the future, the techniques used for solving the Helmholtz equation are planned to be extended to more complicated media (elasticity with possible anisotropy and viscosity). Current version of the solver can solve around 10^9 equations. Additional optimization of factorization step should allow computations for a larger models in the near future.

ACKNOWLEDGEMENTS

We are thankful to King Abdullah University of Science and Technology (KAUST) for providing us access to the supercomputer Shaheen II. We greatly appreciate editors Valentina Socco, Stig Hestholm and anonymous reviewers whose comments helped to improve the quality of the paper.

REFERENCES

Amestoy, P.R., C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L’Excellent, and C. Weisbecker, 2015, Improving multifrontal methods by means of block low-rank representations: SIAM Journal on Scientific Computing, **37,** no**.**3, A1451–A1474.

Amestoy, P., R. Brossier, A. Buttari, J.-Y. L'Excellent, T. Mary, L. Métivier, A. Miniussi, and S. Operto, S., 2016, Fast 3D frequency-domain full-waveform inversion with a parallel block low-rank multifrontal direct solver: Application to OBC data from the North Sea: Geophysics **81**, no. 6, R363-R383.

Aminfar, A., S. Ambikasaran, and E. Darve, 2016, A fast block low-rank dense solver with applications to finite-element matrices: Journal of Computational Physics, **304**, 170–188.

Aminzadeh, F., J. Brac, and T. Kuntz, 1997, 3-D salt and overthrust models, *in* SEG/EAGE modelling series: SEG Book Series.

Bakulin, A., M. Dmitriev, V. Kostin, and S. Solovyev, 2018, Benchmarking 3D time-and frequency-domain solvers for FWI applications for different cluster sizes and variable number of sources: *in* SEG Technical Program Expanded Abstracts 2018, 3888-3892.

Belonosov, M., M. Dmitriev, V. Kostin, D. Neklyudov, and V. Tcheverda, 2017, An iterative solver for the 3d Helmholtz equation: J. of Comput. Physics, **345**, 330-344.

Belonosov, M., V. Kostin, D. Neklyudov, and V. Tcheverda, 2018, 3D numerical simulation of elastic waves with a frequency-domain iterative solver: Geophysics, **83**, no. 6, T333-T344.

Berenger, J.-P., 1994, A perfectly matched layer for the absorption of electromagnetic waves: J. of Comput. Physics, **114**, 185-200.

Bermúdez, A., L. Hervella-Nieto, A. Prieto, and R. Rodríguez, 2007, An optimal perfectly matched layer with unbounded absorbing function for time-harmonic acoustic scattering problems: J. of Comput. Physics, **223**, 469-488.

Bunch, J. R., L. Kaufman, and B. N. Parlett, 1976, Decomposition of a symmetric matrix: Numerische Mathematik, **27**, 95-109.

Chandrasekaran, S., M. Gu, and T. Pals, 2006, A fast ULV decomposition solver for hierarchically semiseparable representations: SIAM Journal on Matrix Analysis and Applications, **28**, no. 3, 603–622.

Dablain, M.A., 1986, The application of high-order differencing to the scalar wave equation: Geophysics, **51**, no. 1, 54-66.

Davis, T. A., S. Rajamanickam, and W. M. Sid-Lakhdar, 2016, A survey of direct methods for sparse linear systems: Acta numerica, **25**, 383-566.

Duff, I. S., and J. K. Reid, 1983, The multifrontal solution of indefinite sparse symmetric linear systems: ACM Transactions on Mathematical Software (TOMS), **9**, 302-325.

Duff, I. S., J. K. Reid, N. Munksgaard and H.B. Nielsen, 1979, Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite: IMA Journal of Applied Mathematics, **23,** no. 2, 235-250.

Erlangga, Y. A. and R. Nabben, 2008, On a multilevel Krylov method for the Helmholtz equation preconditioned by shifted Laplacian: Electronic Transactions on Numerical Analysis, **31**, 403-424.

Etienne, V., T. Tonellot, P. Thierry, V. Berthoumieux, and C. Andreolli, 2014, Optimization of the seismic modeling with the time-domain finite-difference method: 84th SEG Annual Meeting Expanded Abstracts, 3536 -3540.

George, A., 1973, Nested dissection of a regular finite element mesh: SIAM J. Numer. Anal., **10**, 345-363.

George, A., J. Liu, and E. Ng, 1994, Computer solution of sparse linear systems: Florida Academic Press.

Ghysels, P., X. S. Li, F.-H. Rouet, S. Williams, and A. Napov, 2016, An efficient multicore implementation of a novel hss-structured multifrontal solver using randomized sampling: SIAM Journal on Scientific Computing, **38,** no 5, S358–S384.

Glinskiy, B., N. Kuchin, V. Kostin, and S. Solovyev, 2017, Parallel computations for solving 3D Helmholtz problem by using direct solver with low-rank approximation and HSS technique, *in* Lecture Notes on Computer Sciences, **10187**, Springer, 342-349.

Godunov, S. K., A. Antonov, O. Kiriljuk, and V. Kostin, 2013, Guaranteed accuracy in numerical linear algebra: Springer Science & Business Media.

Golub, G., and C. V. Loan, 1996, Matrix computations, 3rd ed.: The Johns Hopkins University Press.

Hustedt, B., S. Operto, and J. Virieux, 2004, Mixed-grid and staggered-grid finite-difference methods for frequency-domain acoustic wave modelling: Geophysical Journal International, **157**, no. 3, 1269-1296.

Jo, C.-H., C. Shin, and J.H. Suh, 1996, An optimal 9-point, finite-difference, frequency-space, 2-D scalar wave extrapolator: Geophysics, **61**, no. 2, 529-537.

Kostin, V., S. Solovyev,   H. Liu, and A. Bakulin, 2017, HSS cluster-based direct solver for acoustic wave equation: 88th Annual International Meeting, SEG, Expanded Abstracts, 4017-4021.

Liu, Y. and M. K. Sen, 2011, Finite-difference modeling with adaptive variable-length spatial operators: Geophysics, **76**, no. 4, T79-T89.

Martinsson, P.-G. and S. Voronin, 2011, A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices: SIAM Journal on Scientific Computing, 38, no. 5, S485-S507.

Mulder, W., and R.-E. Plessix, 2004, How to choose a subset of frequencies in frequency domain finite-difference migration: Geophysical Journal International, **158**, 801 - 812.

Operto, S., J. Virieux, P. Amestoy, and J.Y. L'Excellent, L. Giraud, and H. Ben Hadj 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.

Ourabah, A., J. Keggin, C. Brooks, D. Ellis, J. Etgen, et al., 2015, Seismic acquisition, what really matters?: Presented at the 2015 SEG Annual Meeting, Society of Exploration Geophysicists.

Pichon, G., E. Darve, M. Faverge, P. Ramet, and J. Roman, 2018, Sparse supernodal solver using block low-rank compression: Design, performance and analysis: Journal of Computational Science, **27**, 255 – 270.

Plessix, R.-E., 2007, A Helmholtz iterative solver for 3D seismic-imaging problems: Geophysics, **72**, SM185 - SM194.

Plessix, R.-E., 2017, Some Computational Aspects of the Time and Frequency Domain Formulations of Seismic Waveform Inversion, *in* Modern Solvers for Helmholtz Problems: Birkhauser, Cham, 159-187.

Schenk, O., and K. Gärtner, 2006, On fast factorization pivoting methods for sparse symmetric indefinite systems: Electronic Transactions on Numerical Analysis, **23**, 158-179.

Shin, C., and Y. H. Cha, 2008, Waveform inversion in the Laplace domain: Geophysical Journal International, **173**, 922 - 931.

Vainberg, B. R., 1966, Principles of radiation, vanishing attenuation and limit amplitude in the general theory of partial differential equations: Usp. Mat. Nauk, **21**, 115 - 194. (in Russian).

Vigh, D. and E.W. Starr, 2008, Comparisons for waveform inversion, time domain or frequency domain?: *in* SEG Technical Program Expanded Abstracts 2008, 1890-1894.

Virieux, J., S. Operto, H. Ben-Hadj-Ali, R. Brosssier, V. Etienne, F. Sourber, L. Giraud, and A. Haidar, 2009, Seismic wave modeling for seismic imaging: The Leading Edge, **28**, 538 - 544.

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

Wang, S., M. 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.

Wang, S., X. S. Li, F.-H. Rouet, J. Xia, and M. V. de Hoop, 2016, A parallel geometric multifrontal solver using hierarchically semiseparable structure: ACM Transactions on Mathematical Software (TOMS), **42**.

Wilkinson, J. H., 1964, Rounding errors in algebraic processes: Prentice Hall.

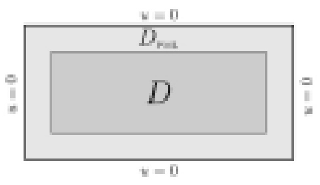
Xia, J., 2013, Efficient structured multifrontal factorization for large sparse matrices: SIAM J. Sci. Comput., **35**, A832-A860.

Xia, J., S. Chandrasekaran, M. Gu, and X. S. Li, 2010, Fast algorithms for hierarchically semiseparable matrices: Numerical Linear Algebra with Applications, **17**, no. 6, 953–976.

LIST OF FIGURES

LIST OF TABLES

Table 5: Performance and memory comparison of our solver and MKL Cluster PARDISO. Data obtained 129 compute nodes using 129 MPI processes, 20 OMP threads. Each node has 128 GB RAM. The threshold for our solver was 10^{-4}.



1

Figure 1: Schematic view of domain $\tilde{D}=D \cup D_{PML}$. In $D_{PML,}$ the solution is attenuated by introducing complex valued functions $\alpha_i(x_i)$ in equation (3). Homogeneous Dirichlet conditions are posed on boundary $\partial\tilde{D}$..

215x279mm (300 x 300 DPI)

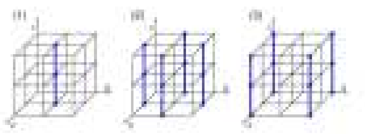


Figure 2: Three finite difference stencils for the approximation of the partial derivative u_{zz}.

215x279mm (300 x 300 DPI)

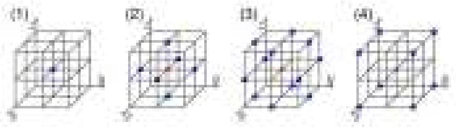


Figure 3 Four stencils for the approximation of k^2 (x)u(x).

215x279mm (300 x 300 DPI)

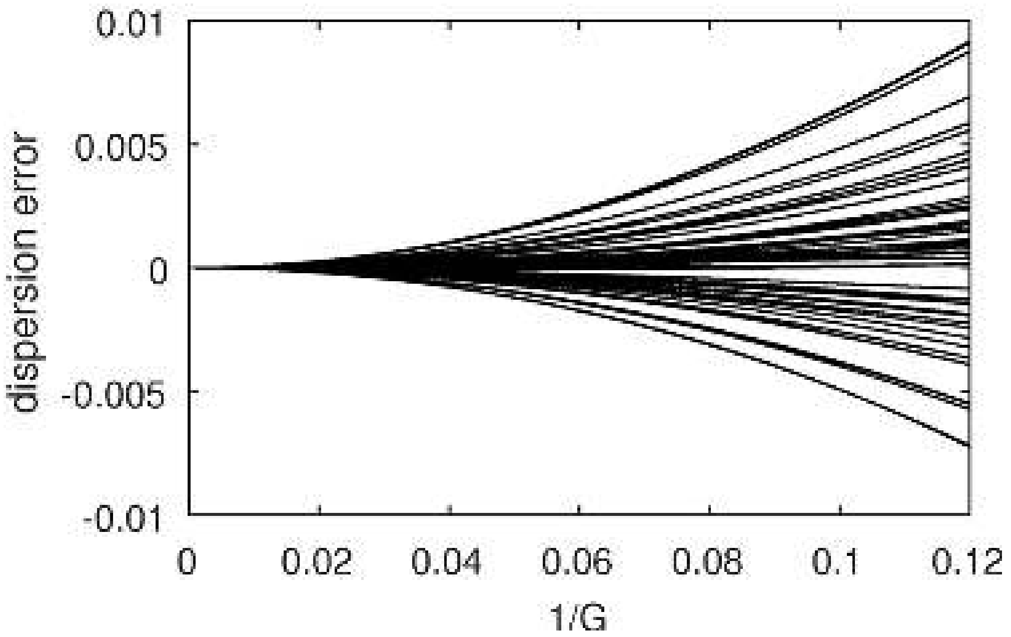
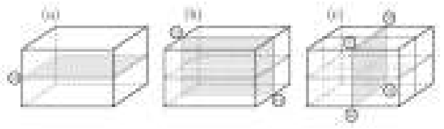


Figure 4: For 50 randomly chosen directions of propagation of plane waves, dispersion error curves are plotted as functions of the reciprocal of the number of grid points per wavelength. The FD approximation is defined by operator 8 with optimal parameters from Table 1.

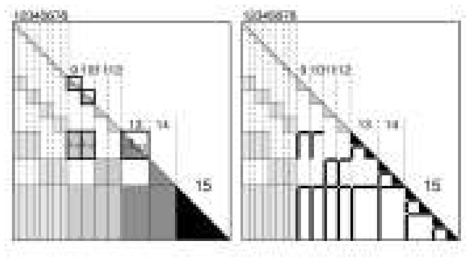
75x47mm (300 x 300 DPI)



1

Figure 5. A schematic view of the ND algorithm comprising (a) first level separator #15 partitioning the domain into the upper and the lower subdomains, (b) second level separators #13 and #14 partitioning the domain into four subdomains, and (c) further partitioning by third level separators #9, #10, #11, #12 resulting in 8 subdomains.

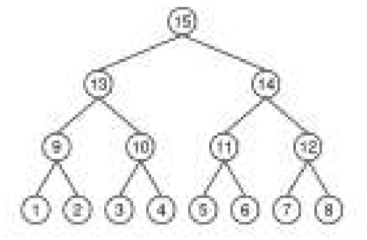
215x279mm (300 x 300 DPI)



1

Figure 6. **L** factor structure showing (left) the original and (right) the Low-Rank approximation. Fill-in is shown by grey shading. See respective ET in Figure 7.

215x279mm (300 x 300 DPI)



1

Figure 7: The ET corresponding to the matrix structure in Figure 6. The tree nodes correspond to the panels in Figure 6. Dependence of panel node #13 on nodes #9 and #10 is illustrated in Figure 6, left, as respective substructure of the matrix.

215x279mm (300 x 300 DPI)

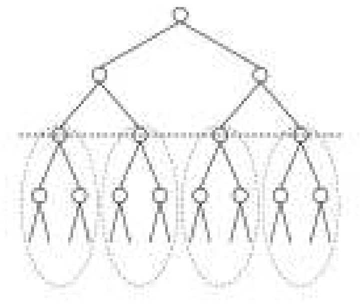


Figure 8: Mapping of the ET to cluster nodes. Dashed ellipses denote the ET nodes combined together and assigned to the same cluster node.

215x279mm (300 x 300 DPI)

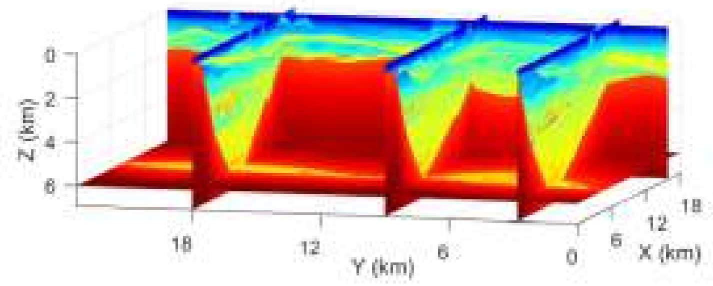


Figure 9: Cross-sections of the TZ velocity model.

182x128mm (300 x 300 DPI)

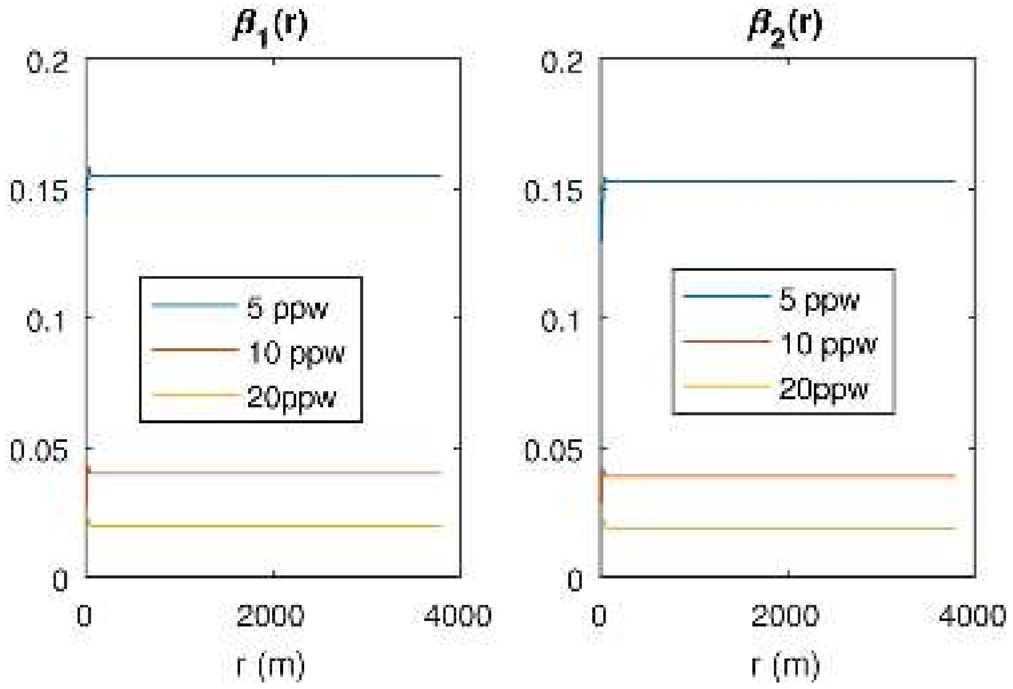


Figure 10 Relative errors of numerical solutions computed for grid steps *h,h/2* and *h/4* (64, 32, 16 m) in a homogeneous medium as a function of the domain radius r.

92x62mm (300 x 300 DPI)

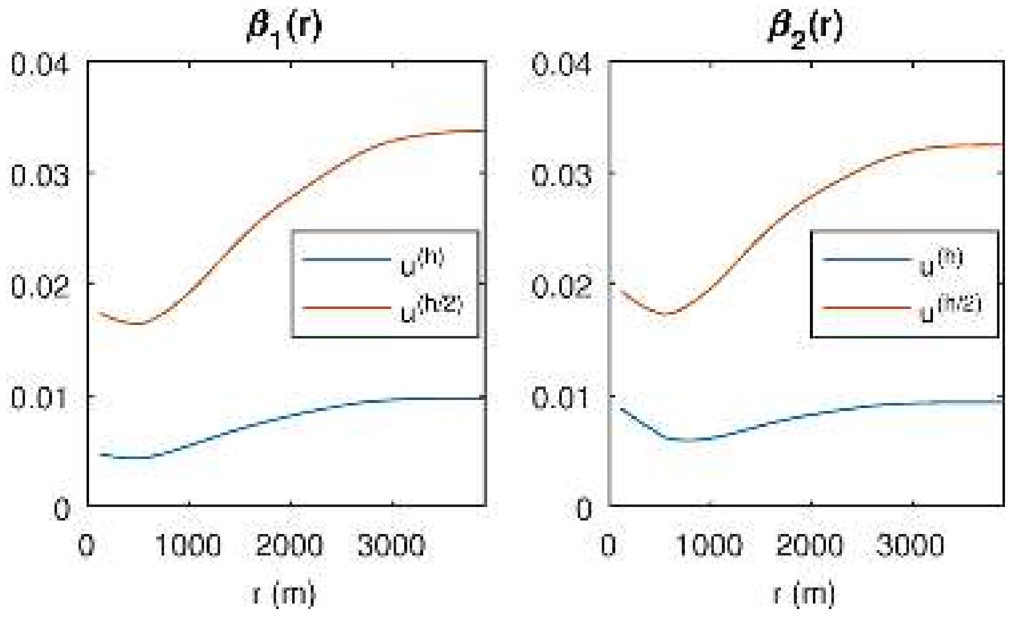


Figure 11: Relative deviation of solutions u^h, $u^{(h/2)}$ from $u^{(h/4)}$ for a model with the velocity being a linear function of the depth.

90x54mm (300 x 300 DPI)

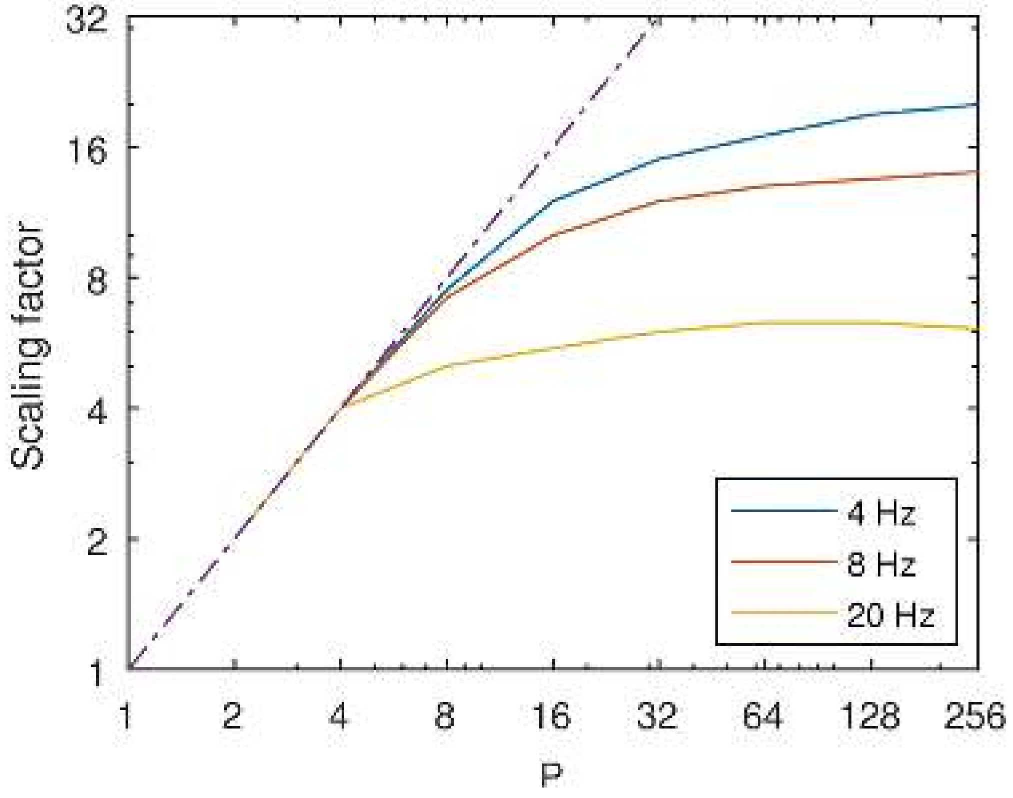


Figure 12: The scaling factors for direct solver as functions of the number *P* of cluster nodes shown for OT velocity model.

75x59mm (300 x 300 DPI)

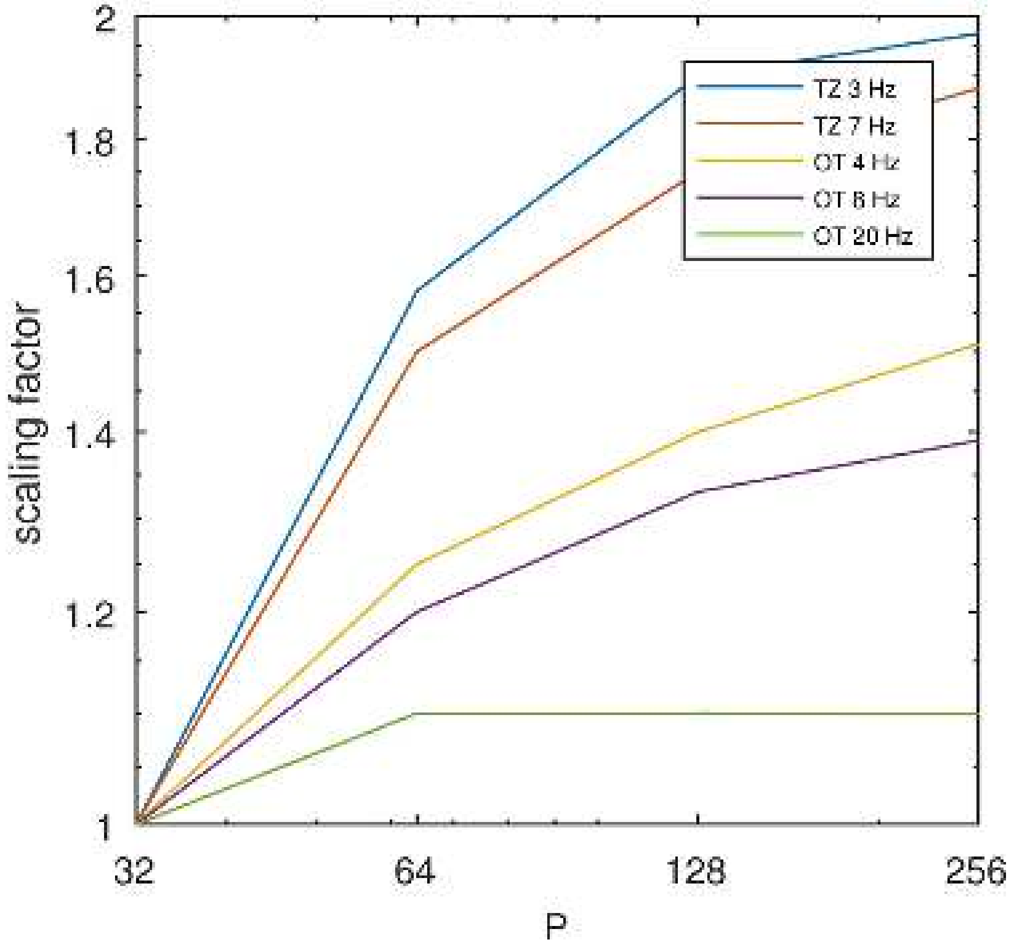


Figure 13: Scaling factors t_{32}/t_{P} for the TZ (1.8×10^8 equations) and OT models (2.4×10^7 equations). Observe higher scalability for larger TZ model.

75x70mm (300 x 300 DPI)

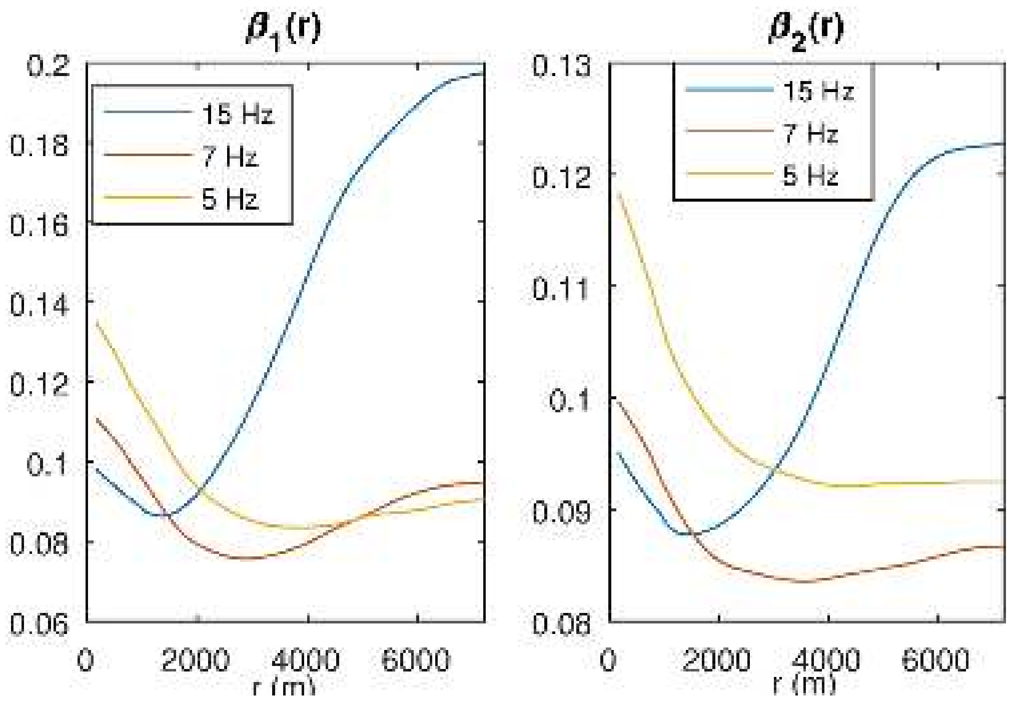


Figure 14: For the OT model, relative deviations of solutions obtained with TDFD and FDFD methods.

90x62mm (300 x 300 DPI)

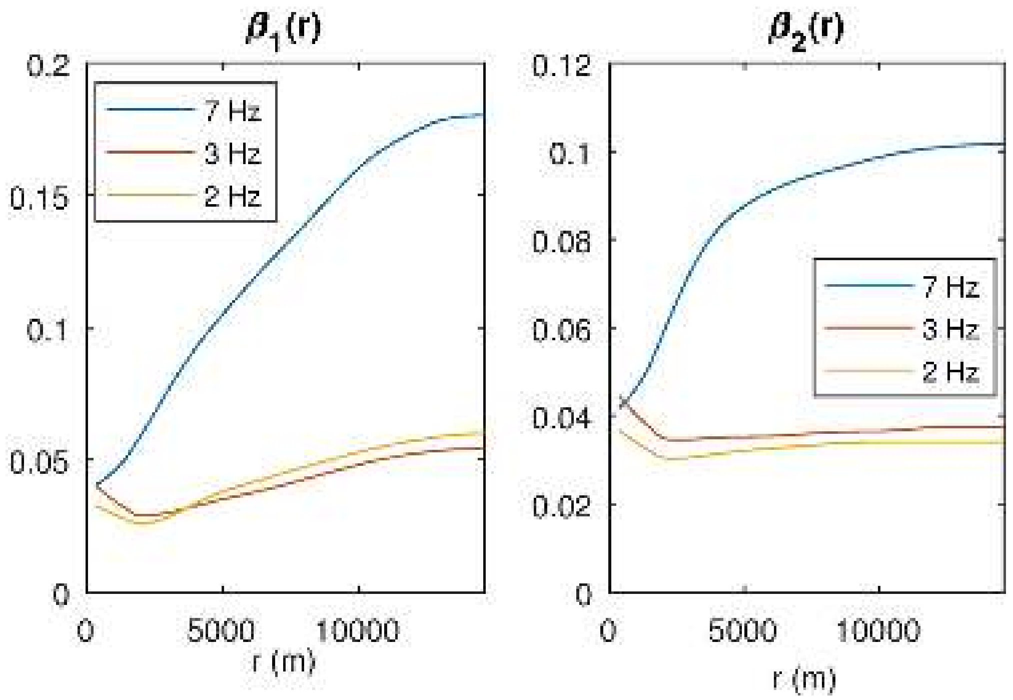


Figure 15: Same as Figure 14 but for TZ model.

90x62mm (300 x 300 DPI)

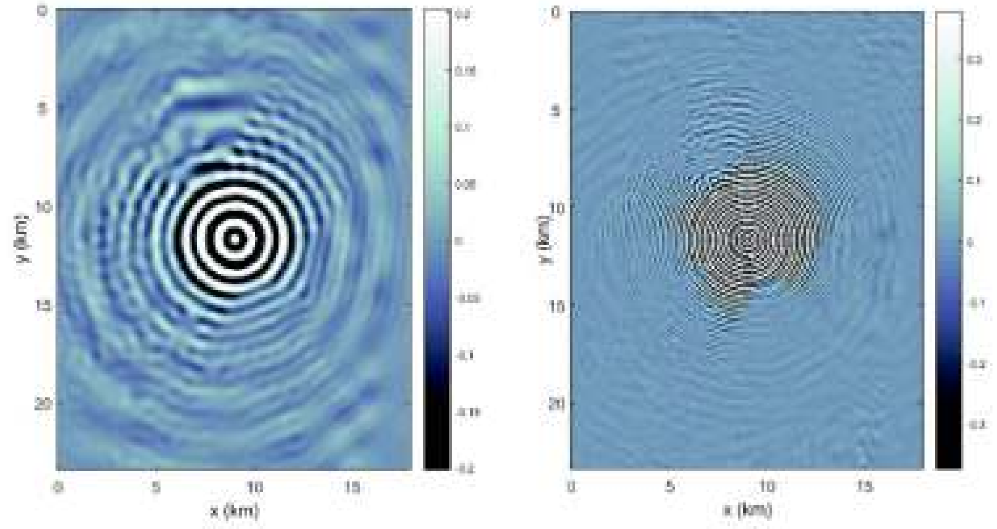


Figure 16: Horizontal snapshots of the real part of the solution for TZ model. Left: frequency v=2 Hz, depth 90 m; Right: v=7 Hz, depth 30 m.

104x56mm (300 x 300 DPI)

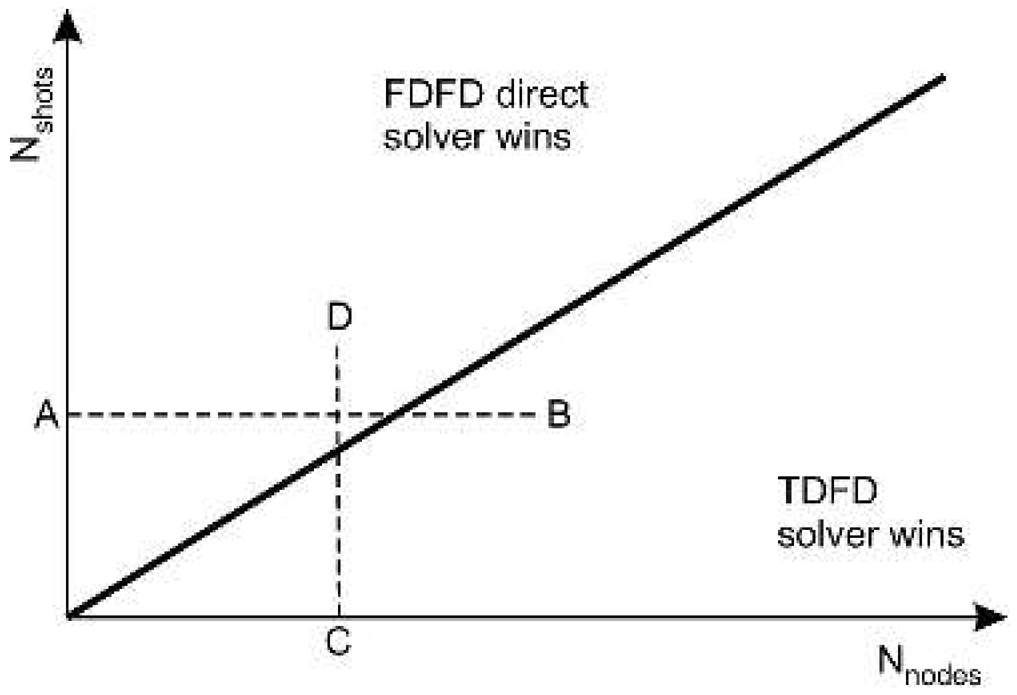


Figure 17: Relative performance of the TDFD and FDFD solvers shown with number of shots (vertical axis) versus number of nodes (horizontal axis). The thick line defines the 'line of equal performance'.

74x50mm (300 x 300 DPI)

Table 1: Values of optimal parameters for stencil 8.

γ_1	0.6558	w_1	0.5756
γ_2	0.2945	w_2	0.1874
γ_3	0.0497	w_3	0.3588
		w_4	-0.1219

Table 2: Parameters of numerical experiments.

Model	ν (Hz)	h (m)	ppw	N_x	N_y	N_z	$N = N_x N_y N_z$
OT	5	90	5.1	110	110	52	629,200
	7	60	5.4	165	165	78	2,123,550
	15	30	5.1	330	330	155	16,879,500
TZ	2	90	5.8	200	261	77	4,019,400
	3	60	5.8	300	391	116	13,606,800
	7	30	5.0	600	781	231	108,246,600

Table 3: Computation times for different combinations of N_{nodes} and N_{shots} in OT model Compute time is summarized as ratios $\frac{t_{\text{TDFD}}}{t_{\text{FDFD}}}$.

		N_{shots}			
		1	128	512	1280
N_{nodes}	2	161/7,377	10,304/7,755	41,216/8,899	103,040/11,186
	4	161/4,294	5,152/4,475	20,608/5,023	51,520/6,117
	8	161/3,295	2,576/3,387	10,304/3,666	25,760/4,223
	16	161/2,927	1,288/2,978	5,152/3,132	12,880/3,439

Table 4: Same as Table 3 but for for TZ model

		N_{shots}			
		1	128	1280	12800
N_{nodes}	32	1,066 / 8,416	4,298 / 8,565	42,737 / 9,917	427,370 / 23,435
	64	1,066 / 6,920	2,159 / 7,014	21,384 / 7,865	213,840 / 16,379
	128	1,066 / 6,093	1,099 / 6,169	10,747 / 6,858	107,470 / 13,752
	256	1,066/ 6,093	1,099/ 6,130	5,450 / 6,475	54,500 / 9,922

Table 5: Performance and memory comparison of our solver and MKL Cluster PARDISO. Data obtained 129 compute nodes using 129 MPI processes, 20 OMP threads. Each node has 128 GB RAM. The threshold for our solver was 10^{-4}.

Model	Measured data	Our solver	PARDISO
SH	Factorization time (s)	967	543
	RAM (GB)	8.1	23
OT	Factorization time (s)	836	2615
	RAM (GB)	17.9	49

DATA AND MATERIALS AVAILABILITY

Data associated with this research are confidential and cannot be released.