MPI+OpenMP Parallelization for Elastic Wave Simulation with an Iterative Solver

Mikhail Belonosov[1](✉), Vladimir Tcheverda[2], Victor Kostin[2], and Dmitry Neklyudov[2]

[1] Aramco Research Center - Delft, Aramco Overseas Company B.V., Delft, The Netherlands
mikhail.belonosov@aramcooverseas.com
[2] Institute of Petroleum Geology and Geophysics SB RAS, Novosibirsk, Russia
{cheverdava, kostinvi, neklyudovda}@ipgg.sbras.ru

Abstract. In this paper, we propose and study the hybrid (MPI and OpenMP) parallelization for our novel approach to 3D numerical simulation of elastic waves with Krylov-type iteration method. The quality of the parallelization is justified by weak and strong scaling analysis.

Keywords: Parallelization · MPI · OpenMP · Elastic equation

1 Introduction

Accurate and fast estimation of the subsurface parameters is of vital importance in the oil and gas industry. A potential candidate to handle this task is a frequency-domain full waveform inversion (FWI) (see e.g. [6]) that has been actively developing in the last decades. Due to advances in supercomputing technology, even 3D elastic inversion, that may bring the most valuable information about the subsurface, seems to be feasible. The most time consuming part of this process is the forward modeling performed several times at each iteration. The efficiency of this process is strongly dependent on how optimally the process is parallelized.

In this effort, we consider a frequency-domain elastic iterative solver proposed in [3]. It is based on a Krylov-type iteration method [5] with a special preconditioner. This method demonstrates a fast convergence at low frequencies, needed for FWI applications. In this paper, we explain an approach to parallelize it using a hybrid parallelization: MPI and OpenMP. Its quality is justified by weak and strong scaling analysis. We also illustrate, that this parallel method allows simulation in big models, including a modified 2.5D Marmousi model comprising 90 million cells, for a feasible time.

The original version of this chapter was revised: The abstract was updated. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-48340-1_64

2 A Preconditioned 3D Elastic Equation

Consider an elastic equation written in the velocity-stress form, describing propagation of a monochromatic component of a wave in a 3D isotropic heterogeneous medium

$$\left[i\omega\begin{pmatrix}\rho\boldsymbol{I_{3\times3}} & 0\\ 0 & \boldsymbol{S_{6\times6}}\end{pmatrix} - \begin{pmatrix}0 & \hat{P}\\ \hat{P}^T & 0\end{pmatrix}\frac{\partial}{\partial x} - \begin{pmatrix}0 & \hat{Q}\\ \hat{Q}^T & 0\end{pmatrix}\frac{\partial}{\partial y} - \gamma(z)\begin{pmatrix}0 & \hat{R}\\ \hat{R}^T & 0\end{pmatrix}\frac{\partial}{\partial z}\right]\boldsymbol{v} = \boldsymbol{f}, \quad (1)$$

where vector of unknowns $\boldsymbol{v}$ comprises nine components. These components include the displacement velocities and components of the stress tensor. ω is the real time frequency, $\rho(x,y,z)$ is the density, $\boldsymbol{I_{3\times3}}$ is 3 by 3 identity matrix, $\hat{P}$, $\hat{Q}$ and $\hat{R}$ are constant matrices, $\boldsymbol{S_{6\times6}}(x,y,z) = \begin{pmatrix}A & 0\\ 0 & C\end{pmatrix}$ is 6 by 6 compliance matrix, and

$$A = \begin{pmatrix}a & -b & -b\\ -b & a & -b\\ -b & -b & a\end{pmatrix}, C = \begin{pmatrix}c & 0 & 0\\ 0 & c & 0\\ 0 & 0 & c\end{pmatrix}. \quad (2)$$

Coefficients $a(x,y,z)$, $b(x,y,z)$ and $c(x,y,z)$ are related to the Lame parameters. $\boldsymbol{f}$ is the right-hand side representing the seismic source. $\gamma(z)$ is an attenuation function. Equation (1) is solved in a cuboid domain of $N_x \times N_y \times N_z$ points with free surface top boundary and attenuation layers on the other boundaries.

Introducing preconditioner L_0 (for details refer to [3]), we arrive at equation

$$\left(I - \delta L L_0^{-1}\right)\tilde{\boldsymbol{v}} = \boldsymbol{f}, \text{ with } \boldsymbol{v} = L_0^{-1}\tilde{\boldsymbol{v}}, \ \delta L = L - L_0, \quad (3)$$

We solve Eq. (3) via the biconjugate gradient stabilized method (BiCGSTAB) [7]. This assumes computing several times per iteration the product of the left-hand side operator of Eq. (3) by a particular vector $\boldsymbol{w}$, i.e. computing $\left[\boldsymbol{w} - \delta L L_0^{-1}\boldsymbol{w}\right]$. Computations of $L_0^{-1}w$ takes the most of runtime. To solve $L_0 q_1 = \boldsymbol{w}$ we assume that function $\boldsymbol{w}(x,y,z)$ is expanded into a Fourier series with respect to x and y with coefficients $\hat{\boldsymbol{w}}\left(k_x,k_y,z\right)$, where k_x and k_y - spatial frequencies. $\hat{\boldsymbol{w}}$ are solutions to equation

$$\left[i\omega\begin{pmatrix}\rho_0\boldsymbol{I}_{3\times3} & 0\\ 0 & \boldsymbol{S}_0\end{pmatrix} - ik_x\begin{pmatrix}0 & \hat{P}\\ \hat{P}^T & 0\end{pmatrix} - ik_y\begin{pmatrix}0 & \hat{Q}\\ \hat{Q}^T & 0\end{pmatrix} - \gamma(z)\begin{pmatrix}0 & \hat{R}\\ \hat{R}^T & 0\end{pmatrix}\frac{\partial}{\partial z}\right]\hat{\boldsymbol{v}} = \hat{\boldsymbol{w}}, \quad (4)$$

with the same boundary conditions as for Eq. (1). Here ρ_0 and S_0 are some averaging of ρ and S. We solve it numerically, applying a finite-difference approximation, resulting in a system of linear algebraic equations with a banded matrix. Computation of $\hat{\boldsymbol{w}}$ we perform via the 2D Fast Fourier Transform (FFT) and after $\hat{\boldsymbol{v}}$ are found, $L_0^{-1}\boldsymbol{w}$ is computed via the inverse 2D FFT.

3 Parallelization

Four computational processes including BiCGSTAB, the 2D FFTs and solving (4), mainly drive the solver. We decompose the computational domain along one of the horizontal coordinates and parallelize these processes via MPI: using parallel BiCGSTAB function from PETSc [2], 2D FFT from Intel Math Kernel Library [4], and each MPI process, corresponding to a certain subdomain, solves boundary value problems (4) for its own set of spatial frequencies k_x and k_y, independently of other MPI processes. The main exchanges between the MPI processes are while performing FFTs.

Following this strategy, each MPI process would independently solve its own set of $N_x \cdot N_y / N$ (N – number of MPI processes) problems. We solve them in a loop, parallelized via OpenMP. Schematically, our parallelization strategy is presented in Fig. 1.

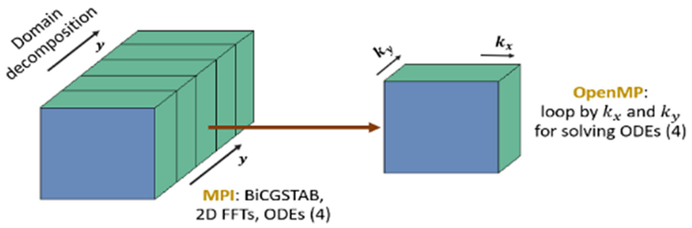
Domain decomposition
y
y
MPI: BiCGSTAB, 2D FFTs, ODEs (4)
k_y
k_x
OpenMP: loop by k_x and k_y for solving ODEs (4)

Fig. 1. Parallelization scheme.

To investigate the properties of this parallelization we construct a 2.5D land model (left image of Fig. 2) from the open source 2D Marmousi model. It is discretized with a uniform grid of 551 × 700 × 235 points. In the right image of Fig. 2 we illustrate the 10 Hz monochromatic component of the computed wavefield for this model. Using 9 nodes with 7 MPI processes per node and 4 cores per process, the total computation time is 348 min.

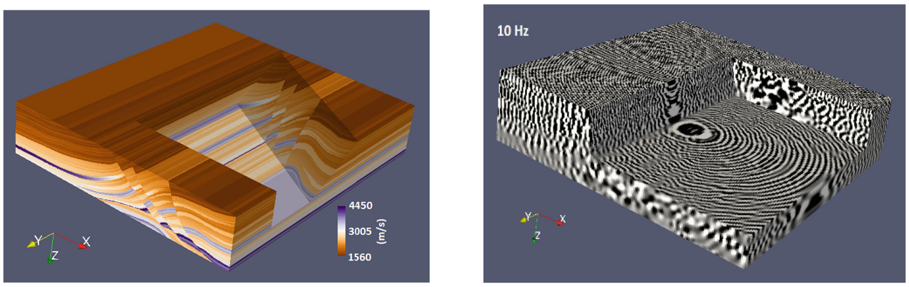

Fig. 2. Left - 2.5D P-velocity model; right - 3D view of a computed wavefield.

MPI strong scalability of the solver is defined as ratio t_M/t_N, where t_M and t_N are elapsed run times to solve the problem with N and $M > N$ MPI processes each corresponding to a different CPU. Using MPI, we parallelize two types of processes. First, those scaling ideally (solving problems (4)), for which the computational time with N processes is $\frac{T}{N}$. Second, the FFT, that scales as $\frac{T_{FFT}}{\alpha(N)}$, with coefficient $1 < \alpha(N) < N$. The total computational time becomes $\frac{T}{N} + \frac{T_{FFT}}{\alpha(N)}$ (here we simplify, assuming no need of synchronization) with scaling coefficient $\frac{T+T_{FFT}}{\frac{T}{N}+\frac{T_{FFT}}{\alpha(N)}}$, that is greater than $\alpha(N)$. This is why, we expect very good scalability of the algorithm, somewhere between the scalability of the FFT and the ideal scalability. We did not take into account OpenMP, which can be switched on for extra speed-up. It is worth noting, that we can not use MPI instead of OpenMp here, since then the scaling would degrade. MPI may have worked well if $T \gg T_{FFT}$, but this is not the case.

We estimate the strong scaling for modeling in two different models, both of $200 \times 600 \times 155$ points: a subset of depicted in Fig. 2 and the overthrust model [1]. From the left image of Fig. 3 we conclude that our solver scales very well up to 64 MPI processes.

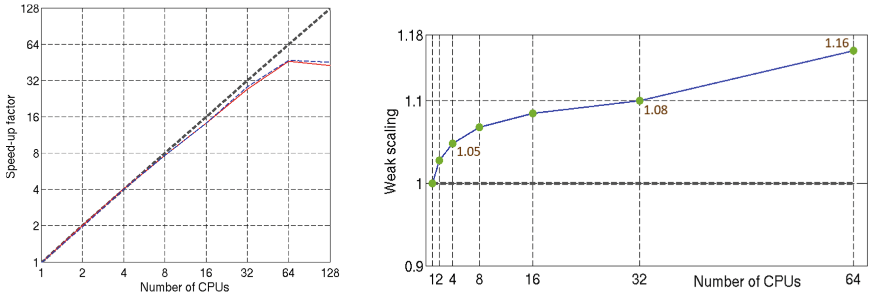

Fig. 3. Left - strong MPI scaling of the solver: blue dashed line - for the Marmousi model, red line - for the overthrust model, the dashed grey line - ideal scalability; right - weak MPI scaling measurements: the blue line - the solver and the dashed grey - ideal weak scaling. (Color figure online)

For weak scaling estimation, we assign the computational domain to one MPI process and then extend the size of the computational domain along the y-direction, while increasing the number of MPI processes. Here, we use one MPI process per CPU. The load per CPU is fixed. For the weak scaling, we use function $f_{weak}(N) = \frac{T(N)}{T(1)}$, where $T(N)$ is the average computational runtime per iteration with N MPI processes. The ideal weak scalability corresponds to $f_{weak}(N) = 1$.

To estimate it in our case, we considered a part of the model presented in Fig. 2 of size $200 \times 25 \times 200$ points with a decreased 4 m step along the y-coordinate. After extending the model in the y-direction 64 times, we arrive at a model of size $200 \times 1600 \times 150$ points. The right image of Fig. 3 demonstrates that for up to 64 MPI processes, weak scaling of our solver has small variations around the ideal weak scaling.

With OpenMP we parallelize the loop over spatial frequencies for solving (4). To estimate the scalability of this part of our solver, we performed simulations in a small part of the overthrust model comprising 660 × 50 × 155 points on a single CPU having 14 cores with hyper-threading switched off and without using MPI. Figure 4 shows that our solver scales well for all threads involved in this example. It is worth mentioning, that we use OpenMP as an extra option applied when further increasing of the number of MPI processes doesn't improve performance any more, but the computational system is not fully loaded, i.e., there are free cores.

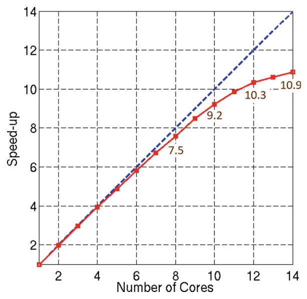

Fig. 4. Strong scalability on one CPU: blue line - ideal and red line is the solver scalability. (Color figure online)

4 Conclusions

Further improvement of the MPI scaling may be achieved by incorporating a domain decomposition along two horizontal directions into the current MPI parallelization scheme. Moreover, the parallelization using domain decomposition along the vertical direction for solving boundary value problems 4 may be applied for accelerating the computational runtime.

Acknowledgments. Two of the authors (Dmitry Neklyudov and Vladimir Tcheverda) have been sponsored by the Russian Science Foundation grant 17-17-01128.

References

1. Aminzadeh, F., Brac, J., Kuntz, T.: 3-D Salt and Overthrust Models: SEG/EAGE Modelling Series, no. 1, SEG Book Series, Tulsa, Oklahoma (1997)
2. Balay, S., Abhyankar, S., Adams, M. et al.: PETSc Users Manual. Argonne National Laboratory, ANL-95/11 - Revision 3.11 (2019). https://www.mcs.anl.gov/petsc
3. Belonosov, M., Kostin, V., Dmitriev, M., Tcheverda, V.: 3D numerical simulation of elastic waves with a frequency-domain iterative solver. Geophysics **83**(6), 1–52 (2018)

4. Intel: Intel®Math Kernel Library (Intel®MKL) (2018). https://software.intel.com/en-us/intel-mkl
5. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
6. Symes, W.W.: Migration velocity analysis and waveform inversion. Geophys. Prospect. **56** (6), 765–790 (2008)
7. Van Der Vorst, H.A.: BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **13**(2), 631–644 (1992)