Solution of Large-scale Seismic Modeling Problems

Nikolay Khokhlov¹, Nikolay Yavich¹, Mikhail Malovichko¹, and Igor Petrov¹

Moscow Institute of Physics and Technology, Dolgoprudny, Moscow Region, Russia
khokhlov.ni@mipt.ru

Abstract
The grid-characteristic method is commonly used to accurately model elastic waves propagation within the application of mechanical stress tests, collision of solid bodies and others. In this note, we discuss the application of the grid-characteristic method to numerical simulation of seismic waves which is an important problem in geophysics. A computational code for this application has been developed in our team. We studied its low-level optimization with streaming SIMD instructions (SSE and AVX). Beyond, we parallelized our code with MPI and reached parallel efficiency of up to 70% when using 16'000 computational cores.

Keywords: MPI, SSE, AVX, grid-characteristic method, hyperbolic partial differential equation, linear elasticity

1 Introduction

Numerical simulation of wave phenomena in solids is of major interest in a wide set of applications. Those applications include geophysical exploration for oil and gas as well as seismic wave propagation generated by earthquakes. Mathematical formulation of the respective physical problem requires incorporation of various geological features: layered structures, cracks, caverns. Problems of this type demand intense computations since the typical size of a heterogeneity can be near meters while the size of computational domain may reach few kilometers. Consequently, spatial grids with up to ten billion cells are commonly needed in practice.

Currently several approaches to numerical simulation of such large-scale problems in heterogeneous geological media are known. Their analysis was presented in the following review articles [3, 20, 21, 13]. Generally researchers tend to use higher order discretization schemes [14] because those imply smaller numerical dispersion. The later is crucial for accurate simulation of the wavefront at distance of hundred and thousand wavelengths. Consequently, finite-difference, spectral element and discontinuous Galerkin methods are generally applied in computational geophysics [5, 7, 12]. Though less commonly cited, the grid-characteristic method [4, 9, 15] also provides high-order accuracy and does not fit to any the three mentioned groups. More importantly, monotonic schemes could be designed on its base [8] what is quite advantageous relative to other approaches.
We implemented a computational code for numerical solution of time-dependent equations of linear elasticity using 2D and 3D structured block grids taking into account geological heterogeneities explicitly. Discretization of the partial differential equation system is based on the grid-characteristic [11, 16, 8, 17, 6] and finite-volume methods [18] second to fourth order accurate.

In this paper, we first present the governing equations of linear elasticity and describe discretization of them with the grid-characteristic method (Section 2). Next we share our experience with optimization of single-threaded (Section 3) and parallel (Section 4) versions of our code. In Section 5, we apply our code for numerical modeling of seismic waves due to an earthquake and show results illustrating correctness of the computed responses.

2 Mathematical Model

Governing equations of linear elasticity describe the state of an infinitesimal volume under small deformations. In a Cartesian right coordinate system \((x_1, x_2, x_3)\) they are of the form [19],

\[
\begin{align*}
\rho \frac{dv}{dt} &= \text{div } \sigma, \\
\frac{d\sigma}{dt} &= \lambda \text{div } v I + \mu (\nabla v + (\nabla v)^T),
\end{align*}
\]

where \(\rho\) is the medium density, \(v\) is the velocity vector, \(\sigma\) is the Cauchy stress tensor, \(\lambda\) and \(\mu\) are Lamé coefficients, \(I\) is the identity tensor. Since \(\sigma\) is a symmetric tensor with only six independent entries, we have overall nine unknowns which could be formed into a vector \(u\),

\[u = \{v_1, v_2, v_3, \sigma_{11}, \sigma_{12}, \sigma_{13}, \sigma_{22}, \sigma_{23}, \sigma_{33}\}.
\]

Now the governing equations (1) can be equivalently rewritten in a matrix form as

\[
\frac{\partial u}{\partial t} = \sum_{j=1}^{3} A_j \frac{\partial u}{\partial x_j},
\]

where \(A_j, j = 1, 2, 3\) are three \(9 \times 9\) matrices. To solve this system of equations we apply the grid-characteristic method on curvilinear hexahedral computational grids [16]. In this method, the system (2) is transformed to the following,

\[
\frac{\partial u}{\partial t} = \sum_{j=1}^{3} \tilde{A}_j \frac{\partial u}{\partial \xi_j}, \quad \tilde{A}_j = \sum_{i=1}^{3} \frac{\partial \xi_j}{\partial x_i} A_i,
\]

where \((\xi_1, \xi_2, \xi_3)\) are splitting directions in the transformed space. Next, we approximate (3) with the conventional operator splitting technique and obtain three one-dimensional systems of the next form,

\[
\frac{\partial u}{\partial t} = \tilde{A}_j \frac{\partial u}{\partial \xi_j}, \quad j = 1, 2, 3.
\]

Every one-dimensional system is of the hyperbolic type and posses a complete eigenvector basis and real eigenvalues, \(\tilde{A}_j = \Omega_j^{-1} \Lambda_j \Omega_j\), where \(\Omega_j\) is the eigenvector matrix of \(\tilde{A}_j\), \(\Lambda_j\) is a diagonal matrix of its eigenvalues. We can find an analytical expression for \(\Lambda_j\), namely, for each direction \(\xi_j\), we have
\[ \Lambda_j = \text{diag} \{ c_1 l_j, -c_1 l_j, c_2 l_j, -c_2 l_j, c_2 l_j, -c_2 l_j, 0, 0, 0 \}, \]

where

\[ c_1 = \sqrt{\frac{\lambda + 2\mu}{\rho}}, \quad c_2 = \sqrt{\frac{\mu}{\rho}}, \quad w^j = \nabla \xi_j, \quad l_j = |w^j| = \sqrt{(w^j_1)^2 + (w^j_2)^2 + (w^j_3)^2}. \]

Coefficients \( w^j \) are computed using the inverse Jacobian matrix. It has to be computed in every computational cell either analytically (if the transform \( \xi = \xi(x) \) is available) or numerically using the second order finite-difference formula,

\[ (w^j_i)_m = \frac{(\xi_j)_{m+1} - (\xi_j)_{m+1}}{\Delta x_i}. \]

Let us switch from variable \( u \) in (3) to \( p = \Omega u \). Then, the matrices involved in the equations will be diagonalized,

\[ \frac{\partial p}{\partial t} + \Lambda \frac{\partial p}{\partial x} = 0, \]

here and later we will skip \( j \) index.

The obtained one-dimensional convection equations are solved using the characteristic method [8]. At every time-step \( n + 1 \), we first find \( p^{n+1} \) and after receive \( u^{n+1} \),

\[ u^{n+1} = \Omega^{-1} p^{n+1}. \]

Vector \( p^{n+1} \) is computed from \( p^n \) using explicit finite-difference schemes. For example, in case of the fourth order discretization, the following formula is used,

\[ p^{n+1}_m = p^n_m - \kappa (\Delta_1 - \kappa (\Delta_2 - \kappa (\Delta_3 - \kappa \Delta_4))), \]

with \( \Delta_1 = \frac{1}{24} (-2p^{n+2}_m + 16p^{n+1}_m - 16p^{n-1}_m + 2p^n_{m-2}), \]

\[ \Delta_2 = \frac{1}{24} (-p^{n+2}_m + 16p^{n+1}_m - 30p^n_m + 16p^{n-1}_m + 2p^n_{m-2}), \]

\[ \Delta_3 = \frac{1}{24} (2p^{n+2}_m - 4p^{n+1}_m + 4p^n_m - 2p^{n-1}_m), \]

\[ \Delta_4 = \frac{1}{24} (p^{n+2}_m - 4p^{n+1}_m + 6p^n_m - 4p^{n-1}_m + p^n_{m-2}), \]

where \( \kappa \) is the Courant number.

We implemented the described computational scheme in our research code. Next we discuss optimization of the single-threaded version of the code.

### 3 Optimization of the single-threaded code

Since we applied explicit temporal discretization (5), to obtain values of the unknown at some step, we perform several arithmetical operations with neighboring values from the previous step. This transfer requires a double loop to cover the computational grid. We tested such a realization and noticed multiple cache misses when sufficiently large grids are used. Those cache misses implied slower execution time.
To mitigate this problem, we changed the way algorithm paces through the grid as well as used extra storage arrays and forced data caching. This optimized memory usage resulted in a faster performance for large computational grids. The obtained speedup depended on the CPU type. Figure 1 illustrates dependence of the number of CPU cycles per grid node on the grid size for a 2D problem. We checked two CPUs (L2 cache size given) and compilers: Intel Xeon 5110, 4Mb, icc-11.1 and Intel Core2 Quad Q8200, 2Mb, gcc-4.4.4. We used -O3 compiler optimization in both cases. Depending on the CPU, the obtained speedup reached five times.

We attempted to further improve execution time of our code by using Streaming SIMD Extensions instruction set (SSE) [10]. SSE operates on several extended 128-bit registers (eight on present-day CPUs) or on 256-bit registers if AVX is supported. Algorithms benefit from this functionality if the same operations have to be performed on different data sets. In this case, SSE parallelizes computations within the data sets.

For a 2D problem discretized on a $4096 \times 4096$ computational grid, Figure 2 shows execution time for different versions of our code and different CPUs and compilers. We tested performance of the original code, its cache-optimized version as well as versions of the code employing SSE and AVX streaming instructions. Each of the presented values is an arithmetical mean of execution time collected after ten runs. We used the following processors and compilers: Intex Xeon E5-2690 with gcc-4.4.6, Intex Xeon E5-2690 with icc-12.1.0, AMD Opteron 6276 with gcc-4.7.2. We also varied the floating-point data type: from single-precision (float) to double-precision (double).

These experiments indicated a speedup of up to seven times for single-precision data and up to two times for double-precision data. Depending on the compiler version and CPU, the performance was different partially due to availability of automatic vectorization. Performance of the code employing AVX is quite close to that of SSE. This presumably happens since time spent on loading data into AVX and SSE registers dominates over actual computational time. Though we plan to investigate this feature further.
4 MPI Parallelization

For our code to benefit from distributed memory environment common to present-day computational clusters, it was parallelized with MPI [2]. We followed the conventional approach for decomposition of the computational domain into subdomains and for boundary cells communication. During parallelization, our major goal was to obtain efficiency when using a substantial number of computational cores (e.g. thousands).

To test our parallelized code, we considered a 3D problem discretized on $1000 \times 1000 \times 1000$ grid and ran it in on HECToR Supercomputer [1]. Each cluster node had two 16-core AMD Operton 2.3GHZ CPUs. In our tests, we fixed the number of threads per cluster node: either 16 or 32. Figure 3 shows the speedup received depending on the number of computational cores occupied (from 128 to 16'384). Figure 4 illustrates efficiency of the parallelized code. We picked such a large-scale problem to make 16 thousand cores work for some measurable time. On the other hand, it was impossible to simulate this example with less than 4 cluster nodes (128 cores) due to memory limitation.

These tests indicated an efficiency of 70-80% which is a fairly good result given the number of cores used. The efficiency is different for 16- and 32-threaded tests though. This is presumably due to design of cluster nodes: in the 32-threaded tests, the threads compete for access to the memory bus slowing performance of each other. Another possible reason to the observed performance is that the threads compete for FPU resources. These are hardware features of this particular cluster and not a drawback of our code.

Another test we performed was a study of scalability when both problem size and number of computational cores were increased. In other words, number of grid cells per computational core was fixed to 16 million while the number of cores varied from 1 to 4096. Our results are presented in Figure 5. We observed a noticeable drop of the 32-threaded line which is due to the cluster hardware features discussed earlier.

Our code proved to provide a good speedup and thus applicable for large-scale problems.
We will present numerical modeling examples in the next Section.

5 Wave Propagation Modeling Example

In this Section, we will study elastic wave propagation originating from an earthquake in a heterogeneous medium. The earthquake is assumed to be caused by instantaneous strain release along a fault in the Earth’s crust. We tested two types of faults at earthquake’s origin: one with a horizontal slip, the other with a vertical slip. To match our setup with an actual event (Guadalupe Victoria earthquake, Mexico, July 6, 2010) we located the hypocenter 1.5km below
Figure 5: Speedup when both problem size and number of computational cores were increased (number of grid cells per computational core was fixed to 16 million while the number of cores varied from 1 to 4096).

the surface. The Earth model (other than near the fault) was assumed to be layered. All the layers had their density equal to 2500\,kg/m^3; thicknesses and primary and secondary wave velocities are given in Table 1.

<table>
<thead>
<tr>
<th>$H$, m</th>
<th>$V_p$, km/s</th>
<th>$V_s$, km/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>4.19</td>
<td>2.79</td>
</tr>
<tr>
<td>400</td>
<td>4.65</td>
<td>3.1</td>
</tr>
<tr>
<td>500</td>
<td>5.85</td>
<td>3.5</td>
</tr>
<tr>
<td>$\infty$</td>
<td>6.13</td>
<td>3.9</td>
</tr>
</tbody>
</table>

Table 1: Layers thicknesses $H$ and primary $V_p$ and secondary $V_s$ wave velocities

Figure 6 illustrates velocity amplitude distribution at two instances. We observe a strong second wavefront of the transverse wave (Figure 6, Panel 4); it is slower and reaches the Earth surface later. We can also observe reflections from boundaries of geological bodies. Velocity distributions due to the horizontal and vertical slips are different and are well-correlated with physical principles.

6 Conclusions

We implemented, optimized, and tested a computational code aimed at numerical simulation of seismic waves. The use of SSE and AVX instructions made our code five times faster in some cases. Our MPI parallelization indicated good scalability and efficiency when using up to 16’000 computational cores. The code managed to tackle very large discrete problems – with billions of unknowns. We applied it to simulate seismic waves generated by an earthquake and received results matching physical principles. Next we plan to further investigate performance of our code and apply it to geophysical modeling problems.
Figure 6: Lateral view of velocity amplitude distribution at two instances; seismic waves due to horizontal slip (near the hypocenter, Panel 1, and surface, Panel 2) and vertical slip (near the hypocenter, Panel 3, and surface, Panel 4)

Acknowledgments

The study was funded by Ministry of Education and Science of the Russian Federation under grant agreement No. 14.575.21.0084 on October 20, 2014 (the unique identifier PNI: RFMEFI57514X0084) for the Moscow Institute of Physics and Technology (State University).

References


