DOI: 10.14529/jsfi240407

# Leveraging OpenMP Tasks for Efficient Parallel Modeling of the Elastic Eave Propagation in Multi-mesh Problems

*Nikolay I. Khokhlov*[1,2], *Vladislav O. Stetsyuk*[1]

This paper presents a new algorithm for parallelizing the grid-characteristic method in shared-memory systems. The OpenMP task parallelism mechanism is used for parallelization. A modification of the grid-characteristic method is considered that uses a set of overlapped grids to determine a complex heterogeneous structure of the computational domain. The complexity of parallelizing the algorithm is represented by the presence of many different-sized grids. The proposed algorithm is described and compared with basic parallelization algorithms. Basic algorithms mean separate parallelization within each computational grid using the loop parallelization mechanism. An analysis of the efficiency of the post-doubling and parallel algorithms is performed. The advantage of the proposed algorithm for a number of problems is demonstrated. The results of testing and calculating the propagation of wave disturbances in a fractured layer are presented. Each crack in the example is specified by a separate computational grid, which significantly increases the multi-scale problem and the number of computational grids. Work is underway to transfer the algorithm to the three-dimensional case.

*Keywords: grid-characteristic method, OpenMP, task based parallelism, overset meshes, geological fractures.*

## Introduction

Modeling of elastic and acoustic processes in two-dimensional and three-dimensional media is a frequently encountered problem. Thus, questions of propagation of dynamic wave disturbances arise in a wide range of problems of mathematical physics. These include problems of seismic exploration, geophysics, non-destructive testing, ultrasound and others. For the numerical solution of this kind of problem, a sufficiently large number of approaches already exist. The best known approaches include finite difference method, finite element method and spectral method. They all have their strengths and weaknesses, as well as the classes of problems they are best suited for. There are methods on unstructured and structured grids. For unstructured grids, finite element methods, spectral methods and discontinuous Galerkin method [13] are more typical. Finite difference methods are more common on structured grids [11, 25]. In some cases, the use of one or another approach is preferable. The works [5, 18] show that for solving seismic problems on sufficiently large computational grids, it is preferable to use structured grids. The method we use is called the grid-characteristic method [9] on the rectangular grids. It is well suited for computer simulations and is characterized by the simplicity of setting the area of integration and the ability to work with multiple meshes (chimera grid method) [23]. This method is also widely used to solve various problems of mathematical physics [15, 16] and in some cases has advantages over other calculation methods [5].

When solving numerical problems of dynamic disturbances in heterogeneous media, there is a need to use sufficiently large computational grids. Using such grids leads to significant time costs for performing calculations. To ensure acceptable time for solving the problem, it is necessary to use modern parallelization technologies. For modeling we use a software package developed by us that supports parallel execution in shared memory systems and distributed

[1]Moscow Institute of Physics and Technology, Dolgoprudny, Russian Federation
[2]Scientific Research Institute for System Analysis of the National Research Centre "Kurchatov Institute", Moscow, Russian Federation

clusters [12]. This software package is parallelized on systems with distributed memory using MPI technology [14, 22]. On systems with shared memory, OpenMP technology is used [10]. Previously, when using this technology, parallelization was implemented using mechanisms for parallelizing cycles of the for type. The purpose of this paper is to investigate the possibility of using the task model that appeared in new versions of the OpenMP specification and to compare the efficiency of the implementation using this model with the previously performed implementation using the functionality of older OpenMP standards.

The work is organized as follows. Section 1 describes the numerical methods and mathematical model. Section 2 describes the implementation of the numerical algorithm and parallelization. Section 3 presents the results of testing the parallel algorithm. Practical Example presents example of calculating a seismic model with fractures. Conclusion summarizes the study and points directions for further work.

## 1. Computational Method

### 1.1. Elastic Wave Model

Elastic wave propagation is described using the Cauchy-Green tensor [24]

$$\epsilon_{i,j} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \sum_l \frac{\partial u_i}{\partial x_l}\frac{\partial u_j}{\partial x_l}\right). \tag{1}$$

We are assuming all displacements to be small, therefore second derivatives can be ignored. Newton's second law and Hooke's law take the following forms

$$\rho\frac{\partial^2 u_i}{\partial t^2} - \sum_j \frac{\partial \sigma_{ij}}{\partial x_i} - f_i = 0, \tag{2}$$

$$\sigma_{ij} = \sum_{k=1}^{3}\sum_{l=1}^{3} C_{ijkl}\epsilon_{kl}. \tag{3}$$

Tensors $\epsilon$ and $\sigma$ are symmetric, so $C$ can be transformed to a matrix using Voigt notation [26]. Moreover, in isotropic media this matrix can be fully defined using only two parameters: $\lambda$ and $\mu$, known as Lamé parameters

$$C_{\alpha\beta} = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}. \tag{4}$$

Defining equation system can be simplified to the following

$$\begin{aligned} \rho\frac{\partial \vec{v}}{\partial t} &= (\nabla\cdot\sigma)^{\intercal} + \vec{f}, \\ \frac{\partial \sigma}{\partial t} &= \lambda(\nabla\cdot\vec{v})\mathbf{I} + \mu(\nabla\otimes\vec{v} + (\nabla\otimes\vec{v})^{\intercal}). \end{aligned} \tag{5}$$

### 1.2. Grid-characteristic Method

Equation system (5) can be written as single matrix equation with variable vector [9]

$$\mathbf{q} = \begin{bmatrix} v_1 & v_2 & v_3 & \sigma_{11} & \sigma_{22} & \sigma_{33} & \sigma_{23} & \sigma_{13} & \sigma_{12} \end{bmatrix}^{\mathsf{T}}. \tag{6}$$

After grouping derivatives along coordinates, it takes the following form

$$\frac{\partial}{\partial t}\mathbf{q} - \mathbf{A_1}\frac{\partial}{\partial x_1}\mathbf{q} - \mathbf{A_2}\frac{\partial}{\partial x_2}\mathbf{q} - \mathbf{A_3}\frac{\partial}{\partial x_3}\mathbf{q} = 0. \tag{7}$$

We can split this equation into 3, i.e. 1 along each axis [17]

$$\frac{\partial}{\partial t}\mathbf{q} = \mathbf{A_i}\frac{\partial}{\partial x_i}\mathbf{q}. \tag{8}$$

Original grid-characteristic method [21] is based on solving the equation system using the transfer along the characteristics, but we are using the modification, which is more suitable for computer modelling. Since the original equation system is hyperbolic, all matrices $\mathbf{A_i}$ have a full set of eigenvalues and eigenvectors. Therefore, they can be diagonalized: $\mathbf{A_i} = \mathbf{\Omega_i}^{-1}\mathbf{\Lambda_i}\mathbf{\Omega_i}$. We can replace the variable again: $\omega_\mathbf{i} = \mathbf{\Omega_i}\mathbf{q}$ and the equation takes the following form

$$\frac{\partial}{\partial t}\omega_\mathbf{i} + \mathbf{\Lambda_i}\omega_\mathbf{i} = 0. \tag{9}$$

Matrix $\mathbf{\Lambda_i}$ is a diagonal matrix of eigenvalues, so the matrix equation can be split into independent equations for each component of $\omega$. Moreover, each of those equations is an advection equation, which can be easily solved using finite-difference schemes.

### 1.3. Boundary Conditions

Applying the procedure described above to the nodes lying near the boundary of the integration area involves some additional steps. As it was mentioned, grid-characteristic method is based on transferring the values along the characteristrics. For boundary nodes some of the characteristics are outgoing (i.e., the values we need to transfer are coming from outside the grid). To do a simulation step in these nodes, we need to define the boundary conditions and calculate the values using them.

We model only linear boundary conditions. They have the following form [7]

$$B\mathbf{q}(t+\tau) = b. \tag{10}$$

Now we can split the replacement into two summands, corresponding to the inner and outer characteristics

$$\begin{aligned} \vec{q}(t+\tau,\vec{x}) &= \mathbf{\Omega}^{int}\vec{\omega}^{int}(t+\tau,\vec{x}) + \mathbf{\Omega}^{(*)out}\vec{\omega}^{out}(t+\tau,\vec{x}) = \\ &= \vec{q}^{\,int}(t+\tau,\vec{x}) + \mathbf{\Omega}^{(*)out}\vec{\omega}(t+\tau,\vec{x}). \end{aligned} \tag{11}$$

Expressing $\vec{\omega}^{out}(t+\tau,\vec{x})$ from the boundary condition formula gives the following form of the equation

$$\vec{q}(t+\tau,\vec{x}) = \vec{q}^{\,int}(t+\tau,\vec{x}) + \mathbf{\Omega}^{(*)out}\left(B\mathbf{\Omega}^{(*)out}\right)^{-1}(b - B\vec{q}^{\,int}(t+\tau,\vec{x})). \tag{12}$$

During the modeling the summand corresponding to the inner characteristics is calculated from the known node values and the summand corresponding to the outer characteristics is calculated and taken into account later during the value correction. It is made possible by the fact that we do not use $\omega$ values in the boundary conditions, all the operations are done using the "real" values ($\mathbf{q}$).

Some boundary conditions and the process of their modeling is described with more details in [8, 9]. It is also worth mentioning, that modeling an absorbing boundary is complicated. Modeling it with linear boundary conditions is possible, but the boundary has a noticeable reflection. A better way to model it is to use a Perfectly Matched Layer (PML) as described in [2, 20, 27].

### 1.4. Chimera Grids

Grid-characteristic method can be used for simulation using both structured and unstructured grids. However, simulations in unstructured grids require significantly more computational resources. Moreover, often the simulation area is mostly homogeneous and only contains a few subareas, where inhomogenieties are located. Accurate modelling in these subareas requires covering them with grids that have a small spatial step, but covering the whole integration area with a grid with a step this small is a waste of computational resources. Methods, that use unstructured meshes, like finite elements method [19, 28], solve this by simply making a grid more detailed where it is needed, but we are working with rectangular and preferably structural grids, so cannot follow this path.

The method we are using is known as a chimera grid method [3, 4]. It is based on using multiple grids: a coarse grid, known as "main", that covers the whole area and a set of fine grids, that cover the areas where we need to do the modeling more accurately. The main grid is structured, while additional grids (we call them overset grids) can be structured or unstructured. We are using interpolation to transfer the values between grids.

It is worth mentioning that we are not limited to just one singular main grid, it can be decomposed into blocks or we can have several main grids and transfer the values between them. Moreover, overset grids can be nested, and this is useful in situations like modeling a crack cluster.

## 2. Implementation

### 2.1. Solver Design

The modeling software package was designed to support both elastic and acoustic processes, multiple finite-difference schemes and flexible configuration without rebuilding. The class, responsible for modeling iterations is the `Solver` class. This class owns a set of `GridContainer` class instances, which represent grids, and a set of `GridCorrector` instances for grid contacts. Each `GridContainer` holds everything needed to make a simulation step in a grid: a `Grid`, a `Schema` and 8 sets of `GridCorrectors` (1 set of correctors and 1 set of fillers for each coordinate axis, a set of fillers and a set of correctors that are applied during a step over all axes).

`Grid` class is responsible for storing data, related to grid nodes or the whole grid. It supports both structured and unstructured curvelinear rectangular grids. Node data is stored in ZYX order (X is the fast axis) and uses array-of-structs layout.

`Scheme` class is responsible for doing a simulation step on the nodes of a grid. It is a template class, parametrized by `Transformer` and `Reconstructor`. During the simulation step, `Transformer` class is used to convert node values to $\omega$. After that `Reconstructor` is used to calculate new $\omega$ values, and finally `Transformer` is used to calculate new node values from them. This split allows us to reuse finite-difference schemes, implemented as `Reconstructors` for modeling other physical processes.

Our implementation distinguishes between three types of `GridCorrectors`: fillers, correctors and contacts. Fillers are used to fill ghost nodes – nodes that lay near the edges of a grid chunk. They are mainly used to transfer values between parts of a grid in multi node cluster environments. Correctors are manipulating values in grid nodes and are used to implement boundary conditions, material property changes, destruction and inhomogeneities. Contacts are affecting multiple grids at once. Their use cases include interpolation in chimera grid approach and contacts between elastic and acoustic media.

## 2.2. Parallel Implementation

For parallel execution in shared memory we are using OpenMP [6].

As described above, we are doing simulation steps in turns over different axes, and a step for each axis is split into several subtasks: applying predictor and corrector contacts, applying boundary conditions and fillers, doing the step in grids.

Predictor and corrector contact application is similar, and the only difference is when it is done (before or after the grid step). Correctors are different from everything else, because they impact multiple grids at once. To avoid data races and ensure the correct results, we are enforcing the order of corrector applications. This is done using OpenMP task dependencies. Each corrector creates a task and declares that this task depends on the grids it is using in some way. For example, interpolation from one grid to the other declares that the first grid is an input dependency and the second is an output dependency. This allows OpenMP runtime to construct a task dependency graph, preventing multiple contacts from trying to write into the same grid at the same time or using the data that should be but is not yet updated by some contact as an input for the other contact. These constraints are actually very strict and can be relaxed a bit in some cases, for example if the node values near the left border of the grid are interpolated, values in nodes near the right border can be used in some other corrector. On the other hand, this dependency planning upgrade is hard to implement, because to do it, we need some kind of decomposer, like the one used for MPI. From our observations, applying grid contacts accounts for too small a percentage of the total simulation time for this feature to be a high priority, that is why we have not implemented it yet.

Boundary conditions are applied independently for each grid and are generally only taking a little time, therefore we are generating 1 task per grid to apply all boundary conditions to it. If in future boundary condition parallelism disbalance starts being an issue, we can switch to generating separate tasks for each boundary condition, but for now it does not really affect the simulation time.

Simulation step in each grid is also done independently from other grids, but there are some nuances. Step on the X axis is relatively simple. We are using a 5-point scheme, so for each grid node we update the value using 2 nodes to the left of it and 2 nodes to the right. Grid-characteristic method involves three phases for each node: transformation, reconstruction and increment. During the transformation phase, stress and velocity in the node and two of its neighbors on each side are converted to interim values called $\omega$. We will refer to these values as *ppw*, *pw*, *w*, *nw* and *nnw*. During the reconstruction phase, a new $\omega$ value is calculated for the node. Finally, the increment phase calculates new stress tensor and velocity from the new $\omega$. After that this process is repeated for the next node. It is important to note that for this node we have already calculated 4 out of 5 $\omega$ values: $pw$, $w$, $nw$ and $nnw$ for the previous node are $ppw$, $pw$, $w$ and $nw$ for the next one correspondingly. Therefore we only need to calculate the value of $nnw$, and that will suffice for the reconstruction and increment. Parallel implementation of the step along the X axis consists in splitting the grid into stripes and generating an OpenMP task for each stripe. The task is doing a step for each row in the grid in its stripe. Since stripes do not overlap, these tasks are independent. Doing the step over the Y axis is more tricky. Since we are storing grid nodes in row-major layout (X is the fast axis), doing the step over the X axis naively is utilizing the CPU cache properly: next values, fetched into the cache from memory correspond to the next nodes we are going to work with. This is no longer true for the Y axis: trying to do a steps over all nodes in a column results in reading from 5 different memory locations and therefore many cache misses. To address this issue, we are using a different approach. Initially, the grid is again split into stripes on the Y axis, and each stripe is assigned its own OpenMP task. Inside the task we are doing the same phases: transformation, reconstruction and increment, but they are done not for individual nodes, but for whole rows at once. Figure 1 shows how this split works.

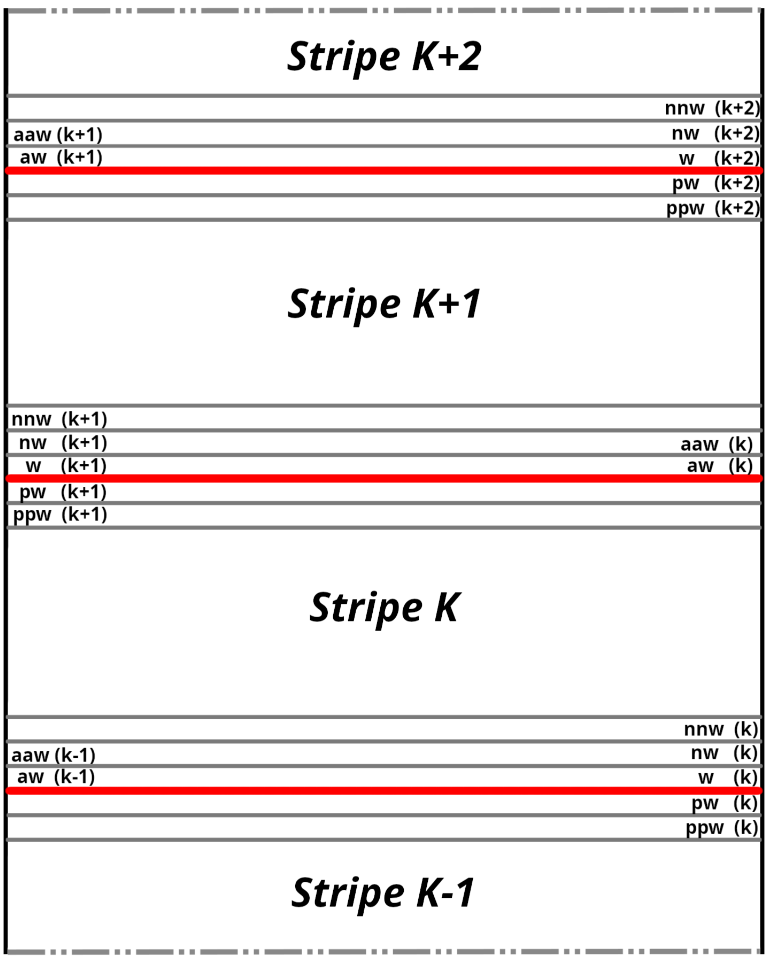

**Figure 1.** Illustration of splitting mesh to stripes and lines for which $\omega$ values are calculated at the initialization for each stripe

Since we are using the 5-node scheme, updating the last 2 rows in a stripe depends on the $\omega$ values of two first rows in the next stripe, referred to as $aw$ and $aaw$ in Fig. 1. We need to calculate these values in advance, since later they can be overwritten by a different task. But the $\omega$ values for these rows are also used as $w$ and $nw$ for the first 2 rows of the next stripe. This allows us to avoid computing them twice and simply copy them at the beginning. To balance the workload between the threads better, the number of tasks assigned for each grid is dynamically computed. At the beginning of the simulation we are only assigning 1 task for each grid, and measuring the time this task takes to complete. After that we use this time as a weight, and assign the number of tasks for each grid according to these weights. To avoid the unnecessary overhead, we also factor in sizes of grids, because small grids cannot efficiently utilize many tasks.

## 3. Testing

### 3.1. Problem Statement

To test the solver performance, we simulated a two-dimensional elastic wave propagation in homogeneous medium. Integration area was split into 20 tiles, each tile was described by a single grid. Each tile contained three nested grids with half the spacing, within one of which were also three nested grids with half the spacing (i.e., quarter the spacing of the main grid). An illustration of grid arrangement is presented in Fig. 2.

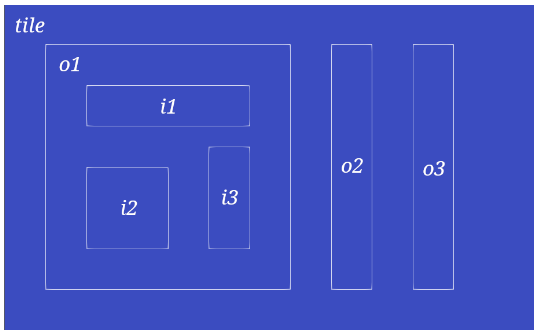

**Figure 2.** Problem statement illustration

**Table 1.** Grid sizes

| Grid | Spacing (meters) | Size (nodes) |
|---|---|---|
| tile | 2 | $260 \times 160$ |
| o1 | 1 | $240 \times 240$ |
| o2 | 1 | $40 \times 240$ |
| o3 | 1 | $40 \times 240$ |
| i1 | 0.5 | $320 \times 80$ |
| i2 | 0.5 | $160 \times 160$ |
| i3 | 0.5 | $80 \times 200$ |

## 3.2. Testing Environment

Performance testing was conducted on a server with 256Gb DDR4 RAM and two 12-core Intel Xeon processors (hyperthreading enabled), running `Linux 6.8.0` kernel and `Ubuntu 24.04`.

Two compilers were used for testing: `g++ 13.2.0` from the GNU Compiler Collection and `icpx 2024.2.0` from the Intel OneAPI kit (based on `clang++` from the LLVM project). These compilers use different OpenMP runtimes, and we were interested in checking whether it will have a significant impact on parallelization.

We did not use any manual CPU affinity configuration and allowed system scheduler and OpenMP runtimes to select processors and cores for threads according to their algorithms.

## 3.3. Testing Results

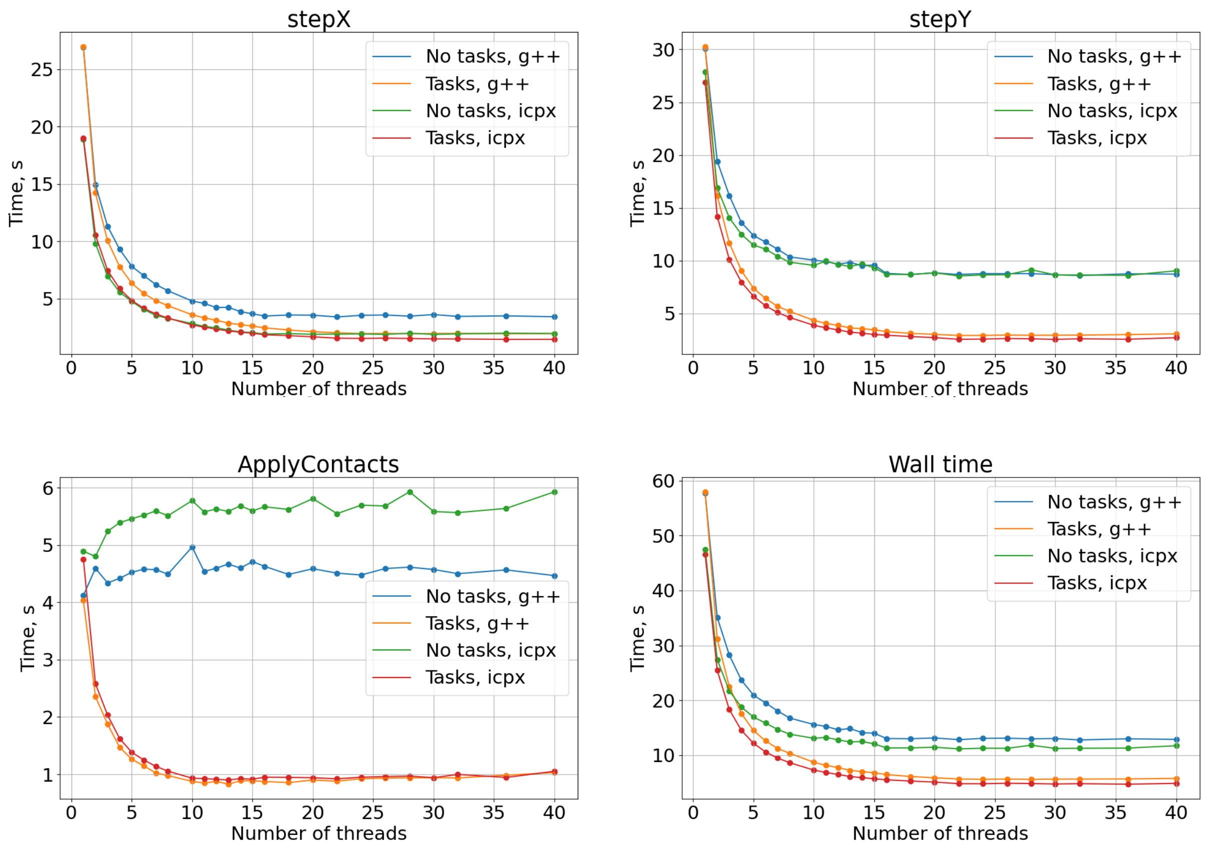

**Figure 3.** Testing result plots

We have conducted the testing varying the number of threads used by the program from 1 to 40. We have also tried using more threads, but the results were noisy and unreliable. Final time for each number of threads was calculated as an average of 3 runs to accommodate for possible external condition changes like CPU clock multiplier decrement caused by overheat. Since the simulation consists of multiple repeated steps, we do not need many runs for averaging the time and preventing the impact of short-term fluctuations.

We have measured the durations of different simulation stages separately, as well as the total simulation time, the plots are presented in Fig. 3. As we can see, task-based parallelism shows similar results to the simple `parallel for` of `stepX`. On `stepY` it significantly outperforms the naïve version, and has roughly the same perofrmance as the `stepX`. `ApplyContacts` stage was not parallelized before, and plots show that while it can benefit from parallel implementation, the number of threads it can use efficiently is low, and the time save is also relatively small.

We can also see that `icpx`-compiled solver performs slightly better than `g++`-compiled, but they do not have any significant difference in parallelizm efficiency. From this we assume that different OpenMP runtimes do not have any significant impact on the performance of our application, and the difference in performance is caused by `icpx` doing optimization and vectorization slightly better.

## 3.4. Performance Analysis

To identify the bottlenecks, we have conducted an additional performance analysis. For this purpose we have disabled the hyper threading and ran a simulation with 20 threads limitation with an increased number of time steps.

According to the Intel VTune [1] statistics, serial execution time is approximately 9% of the total simulation duration, therefore most performance-critical operations are already executed in parallel regions, and the performance cannot be significantly improved by parallelizing serial regions.

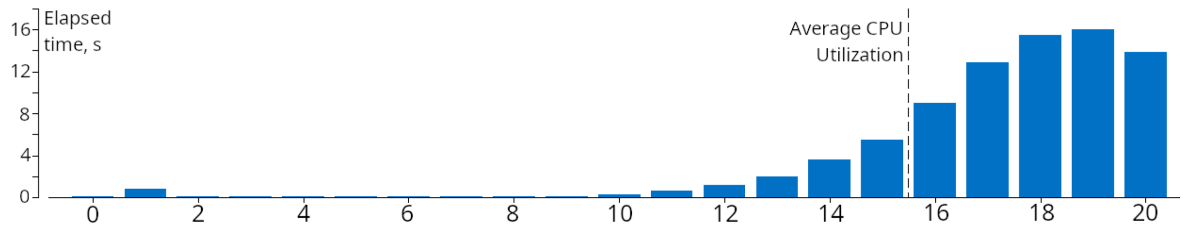

**Figure 4.** Thread activity histogram

Figure 4 shows the histogram of thread activity. From this histogram we can see that thread workload imbalance is noticable, but not critical. This means that our heuristics might need some tuning in future, but this is not the main performance bottleneck. Intel VTune estimates the potential performance gain here to be around 8%.

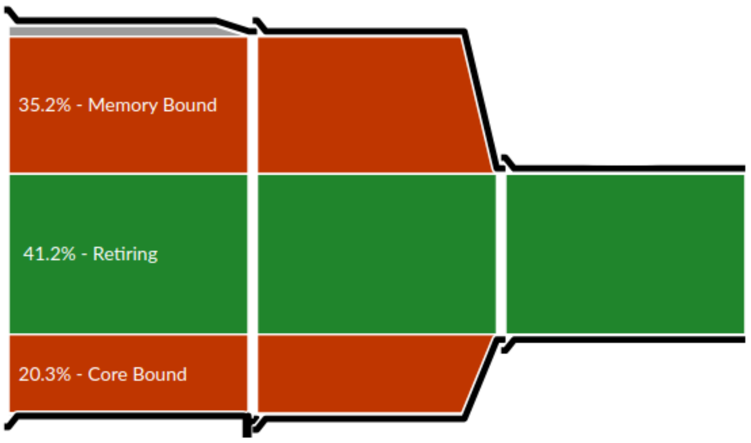

**Figure 5.** Microarchitecture exploration

Microarchitecture analysis shows (Fig. 5), that there are many backend-bound pipeline slots. Since the simulation consists of different stages, it is important to explore the bottlenecks in each of them. Table 2 contains the detailed data.

**Table 2.** Microarchitecture metrics of different operations

| | OmegaX | OmegaY | reconstruct | incrementX | incrementY |
|---|---|---|---|---|---|
| Retiring | 7.10% | 16.70% | 68.70% | 53.30% | 61.40% |
| Frontend Bound | 0.20% | 1.40% | 0.50% | 0.80% | 0.90% |
| Bad speculation | 1.10% | 1.70% | 0.00% | 6.00% | 3.40% |
| Backend Bound | **91.50%** | **80.20%** | 30.80% | 39.90% | 34.30% |
| Memory | **74.10%** | **61.10%** | 9.30% | 11.40% | 7.10% |
| L1 Bound | 14.70% | 0.00% | - | - | - |
| L2 Bound | **39.20%** | 33.90% | - | - | - |
| L3 Bound | 0.00% | 6.40% | - | - | - |
| DRAM Bound | 15.80% | 30.70% | - | - | - |
| Bandwith | 41.90% | **80.40%** | - | - | - |
| Latency | 49.80% | 11.60% | - | - | - |
| Store Bound | 0.00% | 1.10% | - | - | - |
| Core | 17.40% | 19.10% | 21.50% | 28.50% | 27.20% |
| Divider | 0.00% | 0.80% | 0.30% | 20.00% | 27.90% |
| Port utilization | 16.40% | 20.30% | 49.10% | 32.80% | 41.20% |
| 0-port | 66.60% | 68.80% | 21.60% | 12.30% | 14.30% |
| 1-port | 19.50% | 15.40% | 24.70% | 16.50% | 10.90% |
| 2-port | 13.90% | 7.00% | 24.10% | 17.10% | 26.70% |
| ≥3-port | 7.50% | 16.30% | 25.10% | 48.10% | 51.50% |

From this table we can see that `OmegaX` and `OmegaY` (transformation phase) are memory-bound, while `reconstruct`, `incrementX` and `incrementY` are utilizing the microarchitecture efficiently, but could benefit from better vectorization.

NUMA remote accesses stand for approximately 44% of all memory access operations, but NUMA access latency does not play a significant role in overall solver performance.

## Practical Example

Using the software we developed, we have simulated the elastic wave propagation in a fractured layer. The simulated area size was $3000 \times 500$ meters. The area contained 30 fractures, 80 to 82 meters long, all at the same depth, but varying angles. Apart from fractures, the medium was isotropic with $C_p = 2698$ m/s, $C_s = 1730$ m/s and $\rho = 2259$ kg/m$^3$.

The main grid size was $15360 \times 2560$ nodes with a grid step 0.196 $m$ along both axes. Each fracture was modelled using an additional $417 \times 9$ nodes grid.

The time step of the simulation was $5 \cdot 10^{-5}$ s, and a total of 12000 time steps were modelled. The wave originated from a point source near the middle of the area. Ricker wavelet was used as a source function.

Figure 6 shows the wavefield images 50, 100, 150, 200, 250, 300 and 400 ms after the source activation.

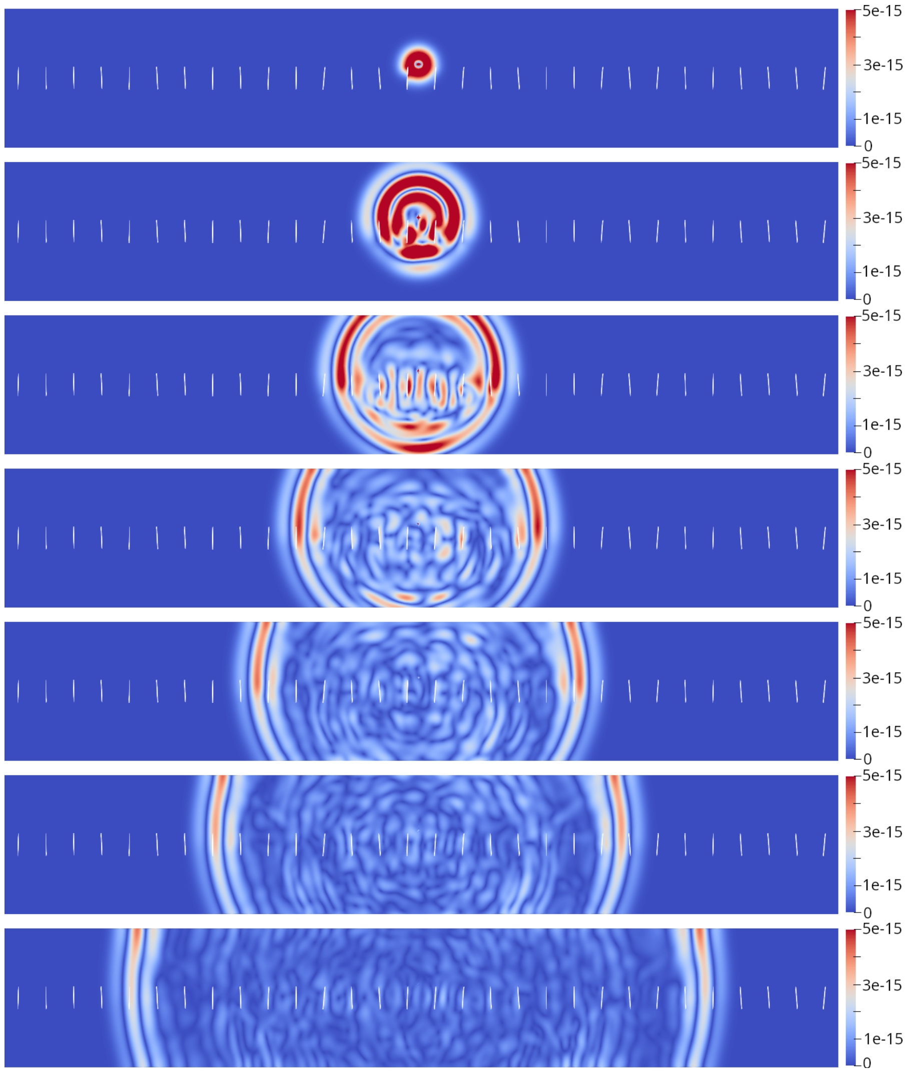

**Figure 6.** Wavefield snapshot at different time moments

Since the main grid has significantly more nodes than all crack grids combined, we did not observe any substantial reduction of the total simulation time. Nevertheless, we have confirmed that the new implementation does not introduce any computational errors or performance degradation. We have also observed the performance improvement in the crack and contact simulation phases, as it was expected.

## Conclusion

This paper proposes a novel algorithm for the parallelisation of dynamic wave processes in heterogeneous media. The grid-characteristic method was employed as a numerical technique for the resolution of the resulting system of partial differential equations. The distinctive feature of this approach was the utilization of overset chimera meshes. The aforementioned factors contributed to the complexity of parallelisation, given the unstructured nature of the links between the computational meshes and the variability in mesh size. The proposed algorithm is based on the OpenMP task parallelisation mechanism and has been implemented and tested. The results of this testing have demonstrated the efficacy of the proposed algorithm. Work is currently underway to extend the algorithm to three-dimensional scenarios.

## Acknowledgements

The work was carried out within the framework of the state task of the NRC "Kurchatov Institute" – SRISA on the topic No. FNEF-2024-0002 "Mathematical modeling of multi-scale dynamic processes and virtual environment systems" (1023032900401-5-1.2.1).

## References

1. Intel VTune Profiler User Guide – intel.com. https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top.html, accessed: 2024-07-01
2. Appelö, D., Kreiss, G.: A new absorbing layer for elastic waves. Journal of Computational Physics 215, 642–660 (2006). https://doi.org/10.1016/J.JCP.2005.11.006
3. Benek, J., Buning, P., Steger, J.: A 3-D chimera grid embedding technique. American Institute of Aeronautics and Astronautics (1985). https://doi.org/10.2514/6.1985-1523
4. Benek, J.A., Steger, J.L., Dougherty, F.C., Buning, P.G.: Chimera. a grid-embedding technique (1986), https://apps.dtic.mil/sti/citations/ADA167466
5. Biryukov, V.A., Miryakha, V.A., Petrov, I.B., Khokhlov, N.I.: Simulation of elastic wave propagation in geological media: Intercomparison of three numerical methods. Computational Mathematics and Mathematical Physics 56(6), 1086–1095 (jun 2016). https://doi.org/10.1134/S0965542516060087
6. Board, O.A.R.: OpenMP Application Programming Interface Specification 5.2 (2021), https://www.openmp.org/specifications/, accessed: 2024-07-01
7. Chelnokov, F.B.: Explicit representation of grid-characteristic schemes for the elasticity equations in two- and three-dimensional space. Mathematical modeling 18, 96–108 (2006)
8. Favorskaya, A.V., Khokhlov, N.I., Petrov, I.B.: Grid-characteristic method on joint structured regular and curved grids for modeling coupled elastic and acoustic wave phenomena in objects of complex shape. Lobachevskii Journal of Mathematics 41, 512–525 (4 2020). https://doi.org/10.1134/S1995080220040083/FIGURES/16
9. Favorskaya, A.V., Zhdanov, M.S., Khokhlov, N.I., Petrov, I.B.: Modelling the wave phenomena in acoustic and elastic media with sharp variations of physical properties using the grid-characteristic method. Geophysical Prospecting 66, 1485–1502 (10 2018). https://doi.org/10.1111/1365-2478.12639
10. Furgailo, V., Ivanov, A., Khokhlov, N.: Research of Techniques to Improve the Performance of Explicit Numerical Methods on the CPU. In: 2019 Ivannikov Memorial Workshop (IVMEM). pp. 79–85. IEEE (sep 2019). https://doi.org/10.1109/IVMEM.2019.00019
11. Galis, M., Moczo, P., Kristek, J.: A 3-D hybrid finite-difference-finite-element viscoelastic modelling of seismic wave motion. Geophysical Journal International 175(1), 153–184 (oct 2008). https://doi.org/10.1111/j.1365-246X.2008.03866.x

12. Ivanov, A.M., Khokhlov, N.I.: Parallel implementation of the grid-characteristic method in the case of explicit contact boundaries. Computer Research and Modeling 10, 667–678 (2018). https://doi.org/10.20537/2076-7633-2018-10-5-667-678

13. Käser, M., Dumbser, M.: An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes - I. The two-dimensional isotropic case with external source terms. Geophysical Journal International 166(2), 855–877 (aug 2006). https://doi.org/10.1111/j.1365-246X.2006.03051.x

14. Khokhlov, N., Petrov, I.: Application of the grid-characteristic method for solving the problems of the propagation of dynamic wave disturbances in high-performance computing systems. Proceedings of the Institute for System Programming of the RAS 31(6), 237–252 (2019). https://doi.org/10.15514/ISPRAS-2019-31(6)-16

15. Khokhlov, N.I., Favorskaya, A., Furgailo, V.: Grid-Characteristic Method on Overlapping Curvilinear Meshes for Modeling Elastic Waves Scattering on Geological Fractures. Minerals 12(12), 1597 (dec 2022). https://doi.org/10.3390/min12121597

16. Kozhemyachenko, A.A., Petrov, I.B., Favorskaya, A.V., Khokhlov, N.I.: Boundary Conditions for Modeling the Impact of Wheels on Railway Track. Computational Mathematics and Mathematical Physics 60(9), 1539–1554 (sep 2020). https://doi.org/10.1134/S0965542520090110

17. LeVeque, R.J.: Finite Volume Methods for Hyperbolic Problems. Cambridge University Press (8 2002). https://doi.org/10.1017/CBO9780511791253

18. Lisitsa, V., Tcheverda, V., Botter, C.: Combination of the discontinuous Galerkin method with finite differences for simulation of seismic wave propagation. Journal of Computational Physics 311, 142–157 (apr 2016). https://doi.org/10.1016/j.jcp.2016.02.005

19. Liu, W.K., Li, S., Park, H.S.: Eighty years of the finite element method: Birth, evolution, and future. Archives of Computational Methods in Engineering 29, 4431–4453 (6 2022). https://doi.org/10.1007/S11831-022-09740-9

20. Luo, S., Chen, Z.D.: A FDTD-based modal PML. IEEE Microwave and Wireless Components Letters 16, 528–530 (2006). https://doi.org/10.1109/LMWC.2006.882408

21. Magomedov, K., Kholodov, A.: The construction of difference schemes for hyperbolic equations based on characteristic relations. USSR Computational Mathematics and Mathematical Physics 9(2), 158–176 (1969). https://doi.org/https://doi.org/10.1016/0041-5553(69)90099-8

22. Mitskovets, I., Sagan, V., Khokhlov, N.: Parallel Modeling of Elastic Wave Propagation, with Explicit Pore Delineation Using Overset Grids Method. Physics of Particles and Nuclei 55(3), 516–518 (jun 2024). https://doi.org/10.1134/S1063779624030602

23. Mitskovets, I., Stetsyuk, V., Khokhlov, N.: Novel approach for modeling curved topography using overset grids and grid-characteristic method pp. 1–5 (1 2021). https://doi.org/10.3997/2214-4609.202011784

24. Novacki, W.: Theory of Elasticity. MIR (1975)

25. Vishnevsky, D., Lisitsa, V., Tcheverda, V.: Efficient Finite-difference Algorithm for Simulation of Seismic Waves in Models with Anisotropic Formations (apr 2012). https://doi.org/10.3997/2214-4609.20143647

26. Voigt, W.: Lehrbuch der kristallphysik (mit ausschluss der kristalloptik). B.G. Teubner (1910)

27. Yao, H.M., Jiang, L.: Machine-Learning-Based PML for the FDTD Method. IEEE Antennas and Wireless Propagation Letters 18, 192–196 (1 2019). https://doi.org/10.1109/LAWP.2018.2885570

28. Zienkiewicz, O.C., Taylor, R.L.R.L., Zhu, J.Z.: The finite element method: its basis and fundamentals. Elsevier, 7 edn. (2013)