Research Note

Parallelization of 3D Pseudo-Bending Algorithm for Seismic Ray Tracing

Madineh Banihashem Kalibar[1], Hossein Sadeghi[2*], and Sayyed Keivan Hosseini[3]

1. M.Sc. Graduate, Earthquake Research Center, Ferdowsi University of Mashhad, Mashhad, Iran
2. Associate Professor, Department of Geology, Faculty of Science, Ferdowsi University of Mashhad, Mashhad, Iran, * Corresponding Author; email: sadeghi@um.ac.ir
3. Assistant Professor, Earthquake Research Center, Ferdowsi University of Mashhad, Mashhad, Iran

Received: 16/04/2019
Accepted: 01/09/2020

ABSTRACT

Bending ray tracing is a technique for finding the shortest travel path from a fixed source to a fixed receiver. Ray tracing is a time-consuming computing technique in applications such as tomography, which involves a large number of source-receiver pairs. In this regard, parallel programming makes it possible to reduce the running time of a serial program significantly by breaking it into a discrete series and solve it by different processing units simultaneously. Along with the rapid development of parallel computing technologies in both hardware architecture and system software, parallel computing is growing rapidly in a broad range of scientific computing applications. In this paper, the parallelization of pseudo-bending ray tracing algorithm is presented using both task and data parallelization strategies. In the task parallelization, the bending calculation of each path section is distributed to different processors, while in the data parallelization, due to the independent calculation for each pair of source-receiver, the data parts are distributed to different processors. The performance results of the parallelizations of the pseudo-bending algorithm for ray tracing in a 3D velocity model are shown using OpenMP, which is an application programming interface for shared memory multiprocessing programming. The advantage of OpenMP programming model is its simplicity to parallelize an existing serial code. This is especially useful now that multi-core CPUs are common. The results show the effectiveness and efficiency of the approach. A significant speedup in the ray tracing implementation is achieved. This reduction in computation time allows more rays to be traced, which directly affects the accuracy of tomography results. Sufficient ray coverage is needed to obtain tomography images with perfect resolution.

Keywords:
Ray tracing; Bending; Tomography; Parallel programming; Multiprocessor; OpenMP

1. Introduction

The present study was conducted to investigate the parallelization of the ray tracing algorithm in order to reduce its computation time. Ray tracing determinesþ a seismic ray path between a source and receiver and its travel time in three-dimensional non-homogeneous isotropic media.

Ray tracing is based on Eikonal equation (Eq. 1), which describes the kinematic propagation of high-frequency waves:

$$\frac{d}{dl}(s\frac{dr}{dl}) = \nabla s \quad (1)$$

where s is slowness, l is ray path length, and r is position vector on the path. In a homogeneous medium, the rays from a fixed source emit with uniform angular distribution and simply through

straight-line paths. However, non-homogeneity medium causes to change the emitting angle and thereby imposing different curvature of the ray paths (Figure 1). The phenomenon of multipathing is another critical problem. There may be more than one path connecting a source to a receiver and therefore defining a valid path will be the problem. In addition to its accuracy and validity, the speed of calculation is another important feature of a ray tracing method, which is of great importance due to the growing need for a fast calculating technique, especially in complex three-dimensional media. For decades, researchers have been trying to introduce reasonably fast approximate solutions to the traditional exact methods. The solvers are divided computationally into two main types; i.e., grid-based and ray-based [1]. Using the grid-based method, it is possible to calculate and store travel times for all points in the model. Then, following the travel time gradient from source to receiver, the Eikonal solution can be obtained by different approaches such as finite difference (e.g., Vidale [2], Rawlinson and Sambridge [3]). Although the grid-based methods, even in extremely non-homogenous media show high stability, they are time-consuming. Therefore, researchers have paid particular attention to the ray-based methods, which are commonly known as the shooting and bending methods (e.g., Langan et al. [4]; Sun [5]; Mao and Stuart [6]; Cores et al. [7]; Xu et al. [8]).

In the shooting, a ray with a desired take-off angle is shot toward the receiver. Afterward, using the ray equation (Equation 1) as an initial value problem and following the Snell's law at the successive interfaces, a complete ray path is traced. The shooting method is computationally simple but rather time-consuming. Mohammadzaheri et al. [9] introduced a distributed map-reduced algorithm on a cluster in order to reduce the computation time of the shooting method.

Mathematically, the bending method is a two-point boundary value problem in which an initial arbitrary path between a source and a receiver is estimated, followed by iteratively bending the path in a 3D velocity model to satisfy Fermat's principle of stationary time. The original bending algorithms (Jacob [10], Wesson [11], Julian and Gubbins [12], Pereyra et al. [13]) solve the ray equation exactly and therefore need complicated computations. Um and Thurber [14] proposed a pseudo-bending method as an alternative algorithm for the bending method, which determines the ray path by perturbing a series of points to speed up the calculation. Later, Zhao et al. [15] and Koketsu and Sekine [16] extended the pseudo-bending method to account for velocity discontinuities and spherical coordinate, respectively. To overcome the convergence problem, Sadeghi et al. [17] developed a method that employs genetic algorithms to search for the globally minimum time path between two fixed points. The pseudo-bending method was improved by Koulakov [18] for applicability to any parameterization of the velocity distribution.

The significant progress toward simplicity and computation time reduction has made the bending method as a widely used ray tracing method in seismic tomography. However, intensive calculations are needed to gain high-quality and high-resolution images; therefore, increasing the computational speed is always of high importance. In recent years,

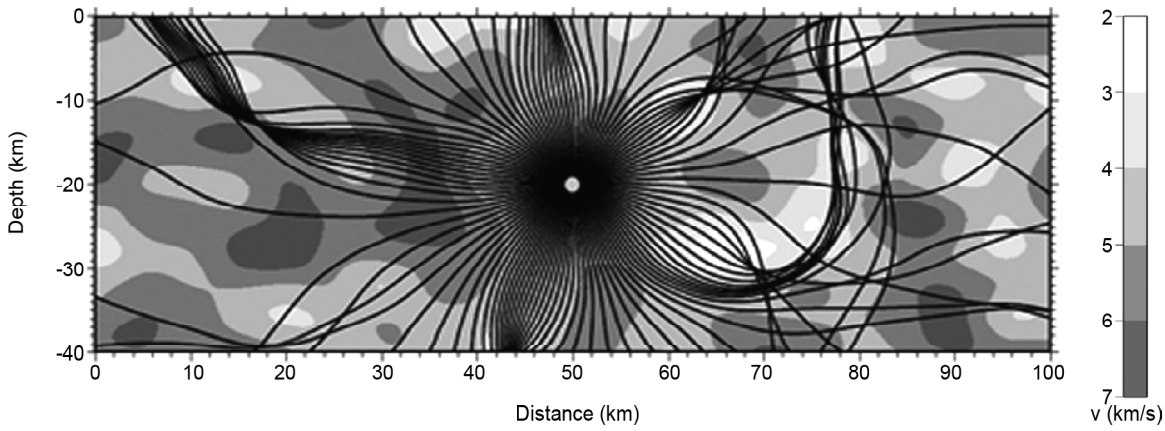

Figure 1. An angular uniform emission rays by a source point (gray dot) in a smoothly varying heterogeneous medium [1].

parallel computing on a cluster of computers or computers with multicore processors has been the subject of intense interest for researchers to overcome the excessive computational time requirements. This paper aims to parallelize the pseudo-bending ray tracing algorithm. Traditionally, a program is written for sequential computation on a single processor. The processor performs the program's instructions one after another. Therefore, an approximate run time of the serial program can be calculated by multiplying the number of instructions by the average time per instruction. To decrease the run time of this program without changing the code, it is needed to enhance the speed of the processor. However, it increases the power required to operate the processor and increase the amount of heat generated. In this regard, it is much more difficult for the heat sink to be removed at a reasonable speed [19]. As a fundamental drawback of the processor speed, it is not capable of increasing the speed at a constant rate. As a result, there is no significant speed improvement for a single processor computer. Instead, today's computers have multiple processors and can run instructions in parallel and simultaneously. These computers lead to a significant increase in computation speed.

Multiple *N* processors, in an ideal world, have an *N*-fold speed-increasing effect. However, the actual speedup will be less than *N* folds. Amdahl's law [20] is a formula that provides the maximum speedup to a parallel program based on the fraction of the code that can be parallelized. Therefore, proper parallelization of code requires expertise in parallel programming. Accordingly, researchers and industry vigorously seek to develop tools that can facilitate parallel programming or the conversion of serial programs to parallel. Some tools and techniques have been developed to help parallel programming. Among them, the oldest and still applicable Application Program Interface (API) for shared memory parallel computing is Open Multi-Processing (OpenMP), which helps to create more easily multi-threaded codes of existing serial programs [21]. In the present study, we use OpenMP to parallelize a sequential bending algorithm for ray tracing in a 3D velocity model and to show the speedup of the parallel executions on a multi-core computer.

2. OpenMP Parallel Mode

The OpenMP is a shared memory multi-thread compiler that is used as a portable programming interface for applications written in C, C+ +, and Fortran languages. The first version was released in October 1997 and is the most commonly used parallel programming language today. The industry works for more cores on a piece of a computer; however, the final performance comes from the software. It means that it should go in software and make it run in parallel. There is no magical compiler that takes a serial code and automatically creates a parallel code, and users have to parallelize their existing serial programs by themselves. The main work occurred with finding the concurrency and understanding the parallel algorithm strategy. The users have to look into the code and think about the problem and decide where the concurrency is and how it could be executed in parallel. The purpose of OpenMP is to easily convert an existing serial code to a multithreaded code by adding some commands. It is relatively easy to learn for users who are familiar with general programming concepts even without a background in parallel programming. Some of the advantages of OpenMP include: (1) simple parallelization; which can be as simple as taking a serial program and placing OpenMP directives, (2) incremental parallelization; which provides the capability to implement parallelization for both loops at a time or even small segments of a code at a time, and (3) portability; which is standard among a variety of shared memory platforms and is supported by a large number of compilers.

The execution model of OpenMP is a Fork/Join (Split/Merge) model, which is a parallelizing form that starts with a single thread (the master thread). The program's code is divided into serial and parallel sections. The master thread, as the only thread in the beginning, runs the serial sections of the code and forks a specified number of slave threads and the system splits the parallel sections of the code among them. The parallel sections are executed by every thread. When all threads completed execution and finished their share of computational work, they merge and join the master thread. Again, the program is assigned to the master thread for implementing the Fork/Join instructions (Figure 2).

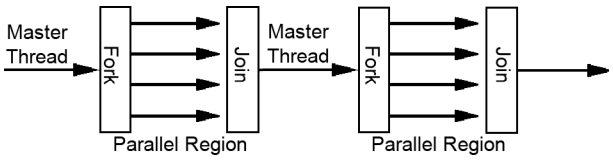

Figure 2. Fork-Join model for OpenMP parallel mode.

3. Reference Algorithm

Although the bending is computationally more complex than the other methods of ray tracing, it is faster in computation time. The computation time is the main reason why this method has been used extensively. In addition to the exact bending methods, the pseudo-bending methods have also been developed to obtain simple and efficient algorithms that reduce the computation time while maintaining the accuracy. The pseudo-bending algorithms use the Fermat's principle of travel time minimization without a direct solution of the ray equation. In a 3D velocity model, the ray path between a specified pair of source and receiver is constructed by starting a three-point guess to the ray path and perturbing it geometrically using the following equations [14]:

$$\mathbf{n}' = \nabla v - \frac{\left[\nabla v.\left(X_{k+1} - X_{k-1}\right)\right]\left(X_{k+1} - X_{k-1}\right)}{\left|X_{k+1} - X_{k-1}\right|^2} \tag{2}$$

$$R = \frac{\left(cv_{mid} + 1\right)}{\left(4\,c\mathbf{n}.\left(\nabla \mathrm{v}\right)_{mid}\right)} + \left[\frac{\left(cv_{mid} + 1\right)^2}{4c\mathbf{n}.\left(\nabla v\right)_{mid}^{\ 2}} + \frac{L^2}{2cv_{mid}}\right]^{\frac{1}{2}} \tag{3}$$

where $= \frac{\mathbf{n}'}{|\mathbf{n}'|}, L = \left|X_{k+1} - X_{mid}\right|$, and $c = (1/\mathrm{v}_{k+1} + 1/\mathrm{v}_{k-1})/2$. X_{k+1} and X_{k-1} are position vectors for k+1 and k-1 points, ∇v is velocity gradient, and $\left(\nabla v\right)_{mid}$ is the velocity gradient at the mid-point X_{mid}; $\mathbf{n}$, and R are the direction and amount of perturbation for the mid-point, respectively. Koketsu and Skenie [16] showed that the pseudo-bending method is numerically more stable than the exact bending methods that use the finite difference method (FDM) to solve directly the ray equation.

In this work, we parallelize a modified version of the pseudo-bending algorithm presented by Koulakov [18]. The important feature of the approach of Koulakov is that it only requires defining uniquely one positive velocity values at any point in the model. So, it can use any velocity model parameterization such as nodes, cells, and polygons, or analytical laws, or any other related ways. Figure (3) shows the basic principle of the algorithm. The starting ray path is a straight line between two fixed endpoints (i.e., A and B), and point 1 in the middle of the path is used for the bending to reach a minimum travel time. Bending is done in two directions perpendicular to the ray path (i.e., in and across the plane of the ray) and the amount of shift is linearly dependent on the length of the path segments. In the next step, the three points (i.e., A, B, and C) are fixed and the bending of the ray path is done in two segments at middle points 1 and 2, followed by searching the point 3 by fixing the points 1 and 2. The act of dividing between two endpoints continues until the path segments reach the predefined minimum length.

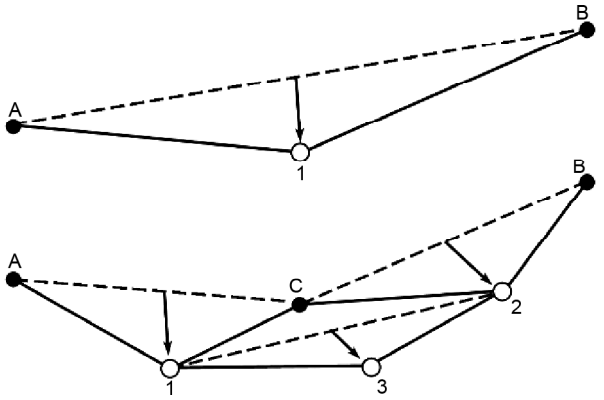

Figure 3. The principle of the bending reference algorithm for the ray tracing (adapted from Figure (2b) of Um and Thurber [14]).

Portions of this algorithm run independently while the others have a dependency chain; however, they eventually are merged to estimate a new ray path. The existence of loops with independent iterations allows us to parallelize the algorithm. There are two kinds of parallelizing strategies; task parallelism (partition independent tasks) and data parallelism (partition independent data). The two kinds of parallelization were studied for the bending ray tracing. In the task parallelism approach, the perturbation computations of ray path sections are partitioned between different processors of a parallel machine (Figure 4). The parallel algorithm has the following structure:

- Define an initial path for a given position of a source and receiver in the 3D model.

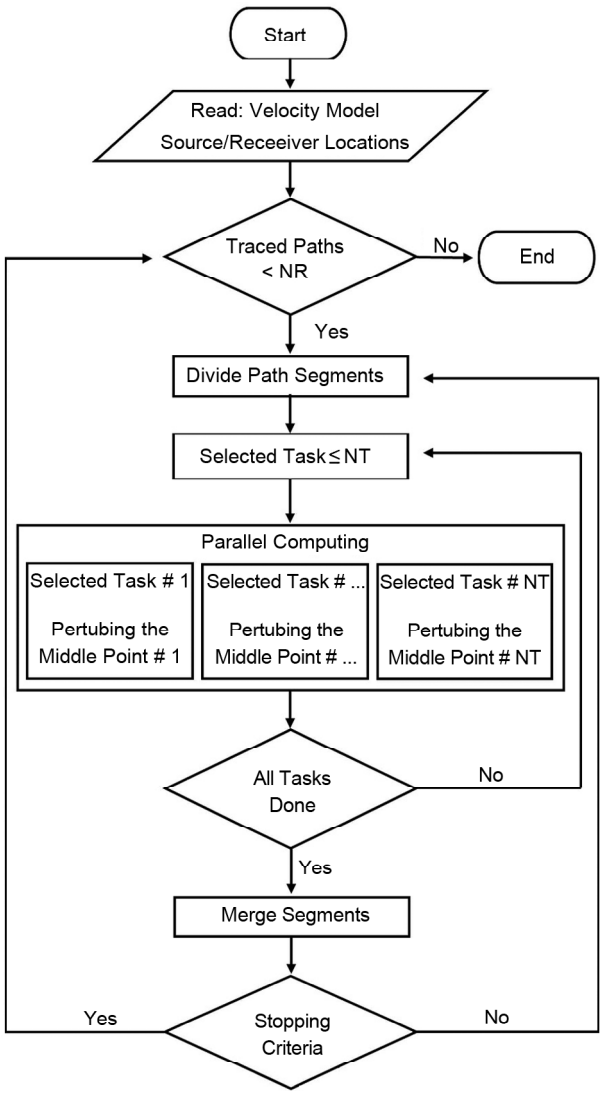

Figure 4. Flowchart of the task parallelization scheme; the NR is the total number of paths between source-receiver pairs. The NT is the number of tasks, and the NP is the number of processors. The optimal condition is NT ≤ NP.

- Bend the initial path by a single processor.
- Divide the bent path into two sections, assigning the bending of each section to two distinct processors.
- Double the path sections, assigning the bending task of each section to a specific processor.
- Continue the path dividing and processor assigning respectively.
- If the number of sections exceeds the number of processors, assigning the bending of an extra section to the processor will finish its task early.
- Repeat (1) to (6) until ray paths are traced for all source and receiver pairs.

This algorithm can be easily implemented using OpenMP. Parallelization of the serial code can be simply specified by adding a directive '# pragma omp ...' to the place of the code where we want to execute in Parallel. The program starts as a single thread (the master thread). It working on an initial path and it comes to the point of dividing path, where an additional thread can help out. The master thread forks in two threads at that point, step (3) in the algorithm. Now is gone from serial part of the program to parallel region as the two threads work together in parallel as a team. When they finished their work, join back together into one thread to merge the results. The master thread continues for step (4), doubling the ray path sections, forks off another team with the double number of threads, and divides the bending of each path section among them. All the threads run concurrently on each processor and independent of each other, but before joining, if the number of sections exceeds the number of processors, the bending of extra sections wait for processors that finish their jobs earlier to complete all the ray path sections.

Since the bending algorithm rays are traced independently between fixed endpoints, it can also be easily distributed over multiple processors by data parallelization, in which each processor only reads a part of the data. Figure (5) shows a schematic example. It should be noted that although a 2D model is presented in this figure, the algorithm is designed for the 3D model. The data parallelism strategy itself can be done either by partitioning the all paths into P parts equal to the number of processors and just executing the ray tracing on each of the processors or by partitioning based on the all possible paths from a source or to a station (Figure 6). In this case, if there are M sources and N stations, the parallel algorithm has the following structure:

- Partition data into M or N parts, assigning one per processor.
- Execute the ray tracing on each of the processors.
- If the number of parts exceeds the number of processors, assign the remaining parts to the processor that will carry out their job earlier.
- Execute the ray tracing until all ray paths are traced.

This algorithm can also be easily implemented using OpenMP. Unlike the previous algorithm, here the master thread forks into several worker threads

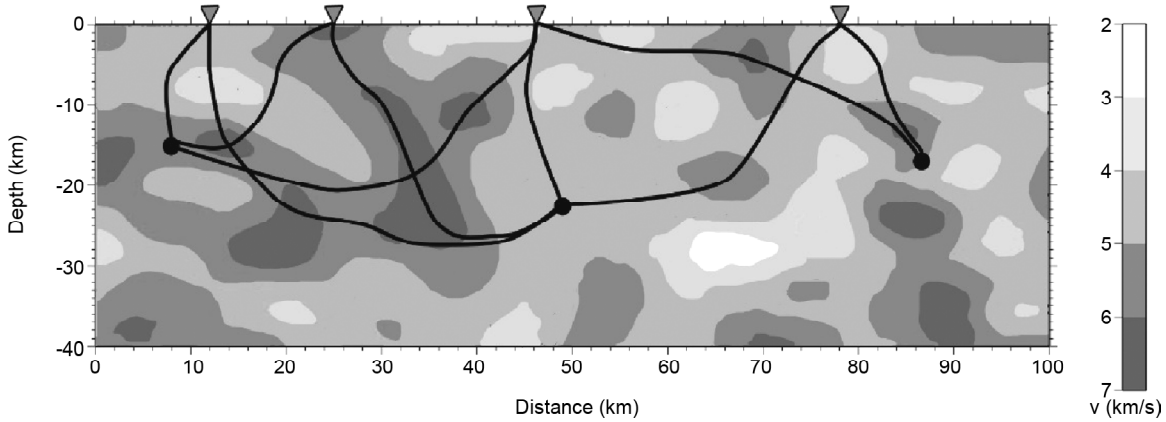

Figure 5. To parallelize the bending ray tracing algorithm, the paths connecting all possible source-receiver pairs, or the rays connecting an individual source (black circles) to all possible stations (gray triangles), and vice versa can be distributed among the processors.

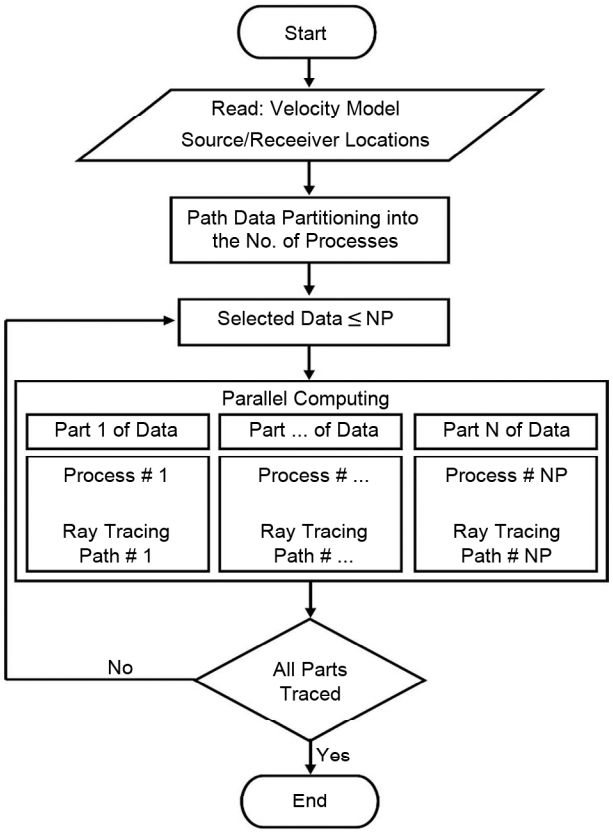

Figure 6. Flowchart of the data parallelization scheme; the NP denotes the number of processors.

equal to the number of CPU cores. Each worker keeps doing its job until all the ray paths are traced.

4. Results

The dataset used in this work is the real data presented in Koulakov [18] for a study in Costa Rica, consisting of 1079 sources and 129 stations. A total number of 24,285 P rays were traced through a 3D synthetic model defined by a checkerboard pattern. The checkerboard consists of vertical columns of altering ±10% of velocity perturbations with respect to the true 1D reference model (see Figure 2, in Koulakov [18]). The checkerboard size pattern is 20×20 km. The model is parameterized by using a 3D grid of nodes with 5 km spacing. P-wave velocity values are defined at grid-nodes and the velocity distribution is interpolated linearly between the nodes. We implemented the parallel ray tracing algorithm by using OpenMP to execute on a multi-core pro-cessor. We tested the proposed parallel algorithms on a hexacore machine (Intel Core i7-2630QM @ 2.0 GHz, 4GB RAM) and measured their performance. A noteworthy issue in the implementation of OpenMP is that a parallel region must be a structured code block, which means that it is not allowed to jump in or out of the parallel region. Therefore, it is needed to replace the GOTO commands in the code with Do loops. The standard OpenMP function, omp_ get_ wtime(), is used to measure the time that has passed during the execution. In order to compute a precise measure of the speedup, the algorithm was executed 100 times and the mean value of the measured times was taken as the result. The results of the parallel algorithms are discussed. The execution time of the parallel code with 2, 3, 4, 5, and 6 cores was reduced in comparison to the serial code (1-core). Moreover, since bending ray tracing for each path between a source-receiver pair is inherently independent of the other paths, and also the pseudo-bending technique perturbs the path segments independently from each other, the parallelization does not affect the results accuracy. Briefly, the calculated ray paths and travel times by the parallel codes are the same as the serial code. Figure (7) presents the

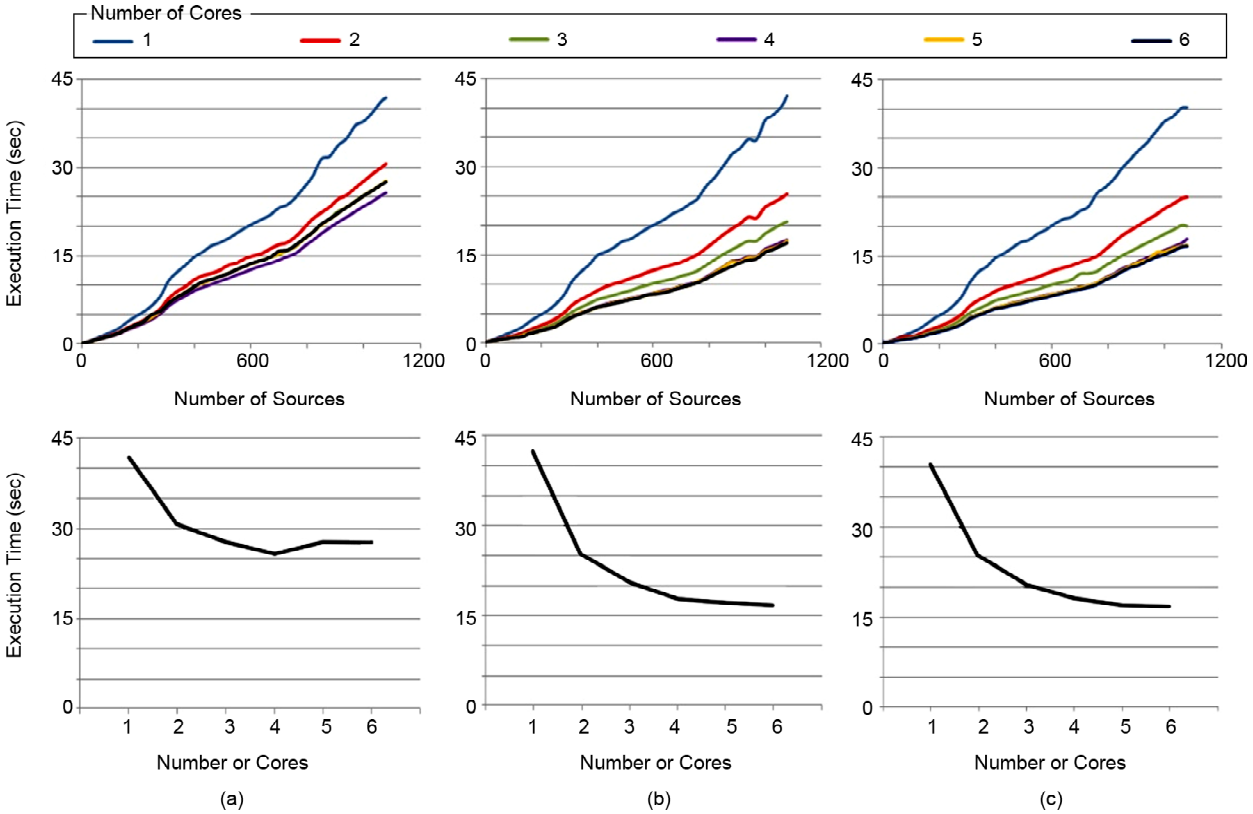

Figure 7. The execution time of sequential and parallel algorithm for different number of sources (top) and the all 1079 sources (bottom) by distributing bending sections on a different number of cores (a), distributing sources on a different number of cores (b), and partitioning all paths connecting the sources to the stations into a number of parts equal to the number of cores (c).

execution time versus the number of sources from 3 to 1079, for a different number of cores. The figure also shows the execution time of the all 1079 sources, separately. In the task parallelism, the bending sections are distributed among different processors (Figure 7a). It is seen that the execution time with four cores is reduced compared to the two and three cores. However, using five and six cores slows down the computation speed. The result can be explained by the time that should be spent on allocating data between the cores and on defining local variables. When the number of data is low, the reduction in the computation time due to parallelization does not compensate the time required for distributions. In the data parallelism, by distributing the sources among different processors, the execution time is reduced by increasing the number of cores (Figure 7b). The data parallelism of the bending code by partitioning the ray paths connecting the sources to the stations into a number of parts equal to the number of processors shows a similar result. By increasing the number of cores, the execution time of the parallel code is reduced more compared to the serial code (Figure 7c).

5. Conclusions

Ray tracing as the forward problem is a time-consuming part of seismic tomography. Accurate tomographic images are obtained when tracing a sufficient number of rays. Therefore, it is required to speed up the ray tracing implementation. Since ray tracing is an iterative process in which the true path is created between each pair of sources and receivers step by step, and, on the other hand, determining the path between each pair of source and receiver is independent of each other, the ray tracing codes support well both data and task parallelism. In this paper, we used the OpenMP to parallel an existing pseudo-bending ray tracing code and to show the advantage of exploiting both data/task parallelisms by its shorter execution time.

Acknowledgement

The authors are very thankful to I. Koulakov for providing the LOTOS code as well as the seismic data, as well as the three anonymous reviewers for their helpful comments and advices. The authors are also deeply indebted to M. Navazandeh for his advice and guidance.

References

1. Rawlinson, N., Hauser, J., and Sambridge, M. (2007) Seismic ray tracing and wavefront tracking in laterally heterogeneous media. *Advances in Geophysics*, **49**, 203-273.
2. Vidale, J.E. (1988) Finite-difference calculation of travel times. *Bulletin of the Seismological Society of America*, **78**(6), 2062-2076.
3. Rawlinson, N. and Sambridge, M. (2004) Multiple reaection and transmission phases in complex layered media using a multistage fast marching method. *Geophysics*, **69**(5), 1338-1350.
4. Langan, R.T., Lerche, I., and Cutler, R.T. (1985) Tracing of rays through heterogeneous media: An accurate and efficient procedure. *Geophysics*, **50**(9), 1456-1465.
5. Sun, Y. (1993) Ray tracing in 3-D media by parameterized shooting. *Geophysical Journal International*, **114**(1), 145-155.
6. Mao, W. and Stuart, G.W. (1997) Rapid multi-wave-type ray tracing in complex 2d and 3d isotropic media. *Geophysics*, **62**(1), 298-308.
7. Cores, D., Fung, G.M., and Michelena, R.J. (2000) A fast and global two point low storage optimization technique for tracing rays in 2D and 3D isotropic media. *Journal of Applied Geophysics*, **45**(4), 273-287.
8. Xu, T., Zhang, Z., Gao, E., Xu, G., and Sun, L. (2010) Segmentally iterative ray tracing in complex 2D and 3D heterogeneous block models. *Bulletin of the Seismological Society of America*, **100**(2), 841-850.
9. Mohammadzaheri, A., Sadeghi, H., Hosseini, S.K., and Navazandeh, M. (2013) DISRAY: a distributed ray tracing by map-reduce. *Computer and Geosciences*, **52**(0), 453-458.
10. Jacob, K.H. (1970) Three-dimensional seismic ray tracing in a laterally heterogeneous spherical earth. *Journal of Geophysical Research*, **75**(32), 6675-6689.
11. Wesson, R.L. (1971) Travel-time inversion for laterally inhomogeneous crustal velocity models. *Bulletin of the Seismological Society of America*, **61**(3), 729-746.
12. Julian, B.R. and Gubbins, D. (1977) Three-dimensional seismic ray tracing. *Journal of Geophysics*, **43**(1), 95-113.
13. Pereyra, V., Lee, W.H.K., and Keller, H.B. (1980) Solving two-point seismic-ray tracing problems in a heterogeneous medium, Part 1. A general adaptive finite difference method. *Bulletin of the Seismological Society of America*, **70**(1), 79-99.
14. Um, J. and Thurber, C. (1987) A fast algorithm for two-point seismic ray tracing. *Bulletin of the Seismological Society of America*, **77**(3), 972-986.
15. Zhao, D., Hasegawa, A., and Horiuchi, S. (1992) Tomographic imaging of P and S wave velocity structure beneath Northeastern Japan. *Journal of Geophysical Research*, **97**(B13), 19909-19928.
16. Koketsu, K. and Sekine, S. (1998) Pseudo-bend ing method for three-dimensional seismic ray tracing in a spherical earth with discontinuities. *Geophysical Journal International*, **132**(2), 339-346.
17. Sadeghi, H., Suzuki, S., and Takenaka, H. (1999) A two-point, three-dimensional seismic ray tracing using genetic algorithms. *Physics of the Earth and Planetary Interiors*, **113**(0), 355-365.
18. Koulakov, I. (2009) LOTOS code for local earthquake tomographic inversion. Benchmarks for testing tomographic algorithms. *Bulletin of the Seismological Society of America*, **99**(1), 194-214.
19. Kiessling, A. (2009) An introduction to parallel programming with OpenMP, The University of Edinburgh, A Pedagogical Seminar (accessed 24 September 2020), URL: https://www.roe.ac.uk/ifa/postgrad/pedagogy/2009_kiessling.pdf.
20. Amdahl, G.M. (1967) Validity of the single processor approach to achieving large scale computing capabilities. *Proc. of the American Federation of Information Processing Societies* (AFIPS Press, **30**), Washington, DC, 483-485.
21. Klemm, M. and Supinski, B. (2019) *OpenMP Application Programming Interface Specification Version 5.0*. Independently published, 668p.