The Journal of Supercomputing, , 1–18 ()

Seismic ray-tracing and Earth mesh modeling on various parallel architectures

MARC GRUNBERG marc.grunberg@eost.u-strasbg.fr
EOST, ULP, 5 rue R. Descartes, F-67084 Strasbourg

STÉPHANE GENAUD genaud@icps.u-strasbg.fr
CATHERINE MONGENET mongenet@icps.u-strasbg.fr
LSIIT-ICPS, UMR 7005, CNRS-ULP, Bd S. Brant, F-67400 Illkirch

Abstract. A major research topic in geophysics deals with the modelization of the Earth interior, and seismic tomography is a means to improve knowledge in this field. In order to improve the accuracy of existing methods, huge quantities of information must be computed. We present in this paper the design of a software program implementing a fast seismic ray-tracing in an Earth mesh. We show that massively parallel computational resources may be used to process huge quantity of data. We present experimentations on several computational equipments with various hardware architectures (a parallel computer, a cluster and a Grid) and report which parallel programming paradigm best suits each of these equipments.

Keywords: Parallel MPI application, geophysics, tomography, ray-tracing, earth mesh.

1. Introduction

One of the main domains in geophysics concerns seismic tomography. Its objective is to build a global seismic wave velocity model of the Earth interior. Seismic waves can be classified accordingly to their propagation mode, namely the compressional mode and the shear mode. In such a model any point (defined by its latitude, longitude and depth) holds some geophysical properties such as the local propagation velocities (V_p and V_s for compressional and shear modes respectively) of seismic waves. From these velocities, physical rock properties can be inferred.

In order to build such a model we use seismic events information recorded in databases by international organizations such as ISC (*International Seismic Center*). Seismic events are recorded by networks of stations (or seismic captors) located all around the world. After such an event, the seismogram data are analyzed in order to localize the earthquake hypocenter. Each database record corresponds to one event and lists all the waves arrival times at the different stations. These databases contain a huge amount of information: for instance, the ISC catalog contains several millions of such arrival times.

A seismic wave is modeled by a set of *rays*. Each ray represents the wavefront propagation from the hypocenter (source) to one station. The final objective of the seismic tomography process is to build an accurate seismic velocity model from an initial one, given ray travel times computations. The first step is therefore to trace these seismic ray paths using information extracted from the databases and then to compute their travel times. These computed times are then compared with the measured ones (from the databases). In the

second step, the initial velocities model is refined in order to get a better fit between the computed and the measured travel times.

The ray tracing algorithm is an iterative process that builds the ray path step by step using linear segment increments. Thus the ray path is a set of discretized points with appropriate information (incidence angle, seismic wave propagation mode). In a global Earth model a ray path is usually discretized using several hundred to several thousands of points depending on the ray length and its nature. Since the ISC database contains millions of such rays and because of these discretizations, we have to compute billions of information items. Therefore parallel computation is required.

The final objective of our project is to build an adaptive mesh of the Earth given a set of seismic rays. The size of a cell in the mesh will be adapted depending on the "illumination" quality, that is a region with few crossing rays will be modeled with large cells whereas a region sampled by many rays in various directions will yield small cells.

In this paper we focus only on ray-tracing in a regular Earth mesh, and propose an efficient solution to implement this problem on parallel systems. Section 2 explains the requirements of the ray tracing process, while section 3 shows how to parallelize the computation. Experimental results as well as comparisons to related works are given in sections 4 and 5, concluding remarks and future work are discussed in the last section.

2. The seismic ray tracing process

The ray path modelization represents seismic wave front propagation from one source (seismic hypocenter) to one receiver (station). Each time a ray reaches a geological interface (lower mantle-outer core, outer core-inner core, ...) it can be either transmitted or reflected. Moreover its propagation mode may convert from P (compressional mode) to S (shear mode) and vice-versa. All these changes during the ray propagation define the ray *signature*. The signature is a combination of symbols which tells the history of the ray. An example signature is PcS, which means the ray propagated in compressional mode (P) downto to interface mantle-kernel (c) and reflected towards the surface in shear mode (S).

The seismograms recorded at stations are analyzed to pick out the different arrival times of the waves and the corresponding ray is given its signature. Table 1 lists the set of possible ray signatures for a given source-receiver pair. Graphical representation of several ray-paths with distincts signatures are presented on figure 1.

The Earth velocity model retained as the initial one is based on eleven layers given by the 1D $\mathrm{IASPEI91}$ model [9] which depends only on the depth. The ray-tracing algorithm is based on the Snell-Descartes law in spherical geometry and describes the variation of seismic wave velocities:

$$p = r.\frac{\sin(i)}{v(r)} \tag{1}$$

where p is the ray parameter which is constant on any point of the ray, $v(r)$ is the wave propagation velocity at depth r and i is the incidence angle of the ray at depth r (figure 2).

The ray path computation relies on the initial incidence angle (i_0, given by the IASP library), and on the ray signature which is used to guide the ray-tracing process. (The

Table 1. The 29 seismic signatures given by the IASPEI91 model for an angular distance (source,receiver) of 130°, and 0 km source depth.

Pdiff	$PKPdf$	$PKiKP$
PP	$SKPbc$	$PKSbc$
$SKPab$	$PKSab$	$PKSdf$
$SKPdf$	$SKiKP$	$SKSac$
$SKSdf$	$SKKSac$	$PKKPdf$
Sdiff	PS	SP
$PKKSbc$	$SKKPbc$	$PKKSdf$
$SKKPdf$	$SKKSac$	$SKKSdf$
$P'P'df$	SS	$S'S'ac$
$S'S'ac$	$S'S'df$	

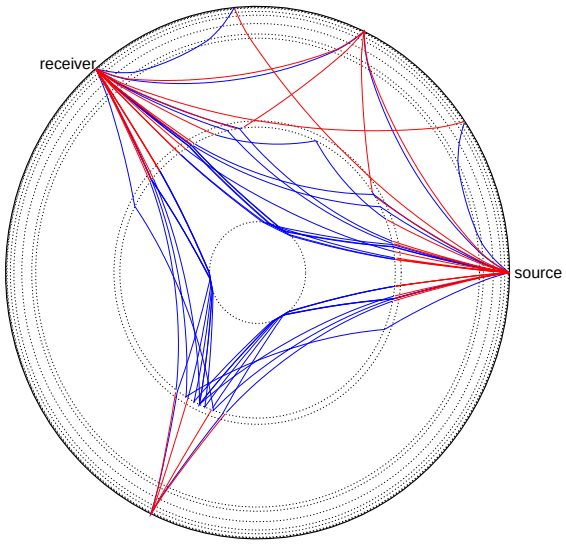


Figure 1. Visualization of 27 seismic ray paths. Each ray path is computed given an initial angular incidence i_0 and ray signature from table 1. (diffracted rays Pdiff and Sdiff were not computed). Blue paths symbolize compression waves, while the red ones are shear waves.

parsing of the signature string gives an immediate outline of the ray path, as mentioned in section 2).

The computation does not depend on the azimuth. The ray is therefore two-dimensional in a plane defined by the source, the receiver and the Earth center. The ray path computation

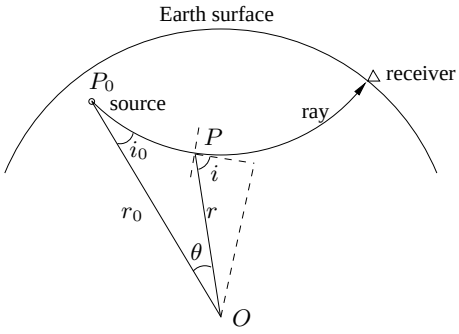


Figure 2. Each ray point P is defined by its depth r, angular incidence i, and $\theta = (\widehat{P_0OP})$. The first point P_0 (the source) is defined by (r_0, i_0, $\theta_0 = 0$). O is the Earth center.

is done in this plane and then three successive rotations are applied to set the rays in a 3D space.

The 2D ray path computation is done using an iterative method which propagates the ray segment by segment using either elementary equal-length segments or equal-angle segments depending on the incidence angle. When the incidence angle is close to 90° the algorithm based on equation (1) switches from equal-length segments to equal-angle ones. During this process when a segment reaches an interface, the information extracted from the ray signature is used to guide the ray: it can be transmitted or reflected, or changes the nature of the seismic phase.

Since our final goal is to build an adaptive mesh, the ray-tracing process not only computes the discretized rays paths and travel times, but also stores all the required information associated with a ray in the cells of an initial regular mesh.

The cell geometry of this mesh results first from a decomposition of the sphere into a given number of layers from the surface to the center (in table 3, there are 8 layers). Each layer is decomposed into regular angular sectors (both in latitude and in longitude) issued at the center of the sphere. Each elementary volume thus obtained defines a cell that can be approximated by an hexahedron. This information will be used in a second step to refine the mesh by merging cells whose "illumination" is not good enough. The mesh adaptation will be realized by cells merging and hence the mesh data structure must manage neighborhood information through a set of links in all six directions. The ray information is mainly composed by the length of the ray in the cell, the coordinates of its first and last point in the cell and its incidence angle. Moreover other information is computed on each cell to guide the mesh adaptation such as the ray density (*i.e.* the number of rays crossing the cell with respect to the cell size), the average ray length, the number of ray hits by cell face, and the cell fill rate, which measures the quality of the spatial repartition of rays in the cell.

3. Parallelization

3.1. Parallelization strategy

The difficulty of parallelizing the application lies in keeping information related to both rays and mesh in each processor's local memory.

One possibility is to allocate to each processor a distinct part of the mesh based on a geographic decomposition (for example we could attribute the two hemispheres to two processors). Each ray would then be traced by the processor dealing with the geographic area the ray is traveling through.

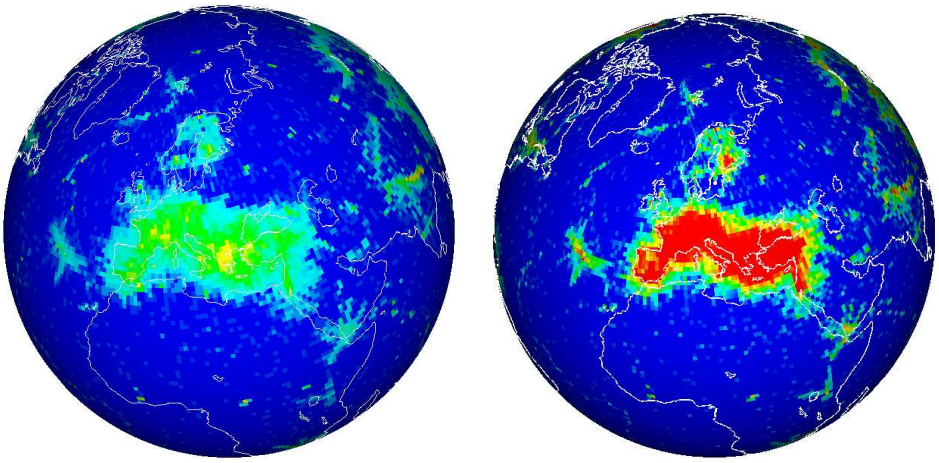
The pitfall of this approach is two-fold: first, the ray tracing computations would be largely unbalanced as the captors are geographically dispersed in a disparate manner at the globe surface (oceans have very few captors for example) and hence some processors would see very few rays passing in their zone. Secondly, inter-processor communications would have to take place every time the ray would pass from one zone to another. Furthermore, increasing the number of processors would also increase the number of communications and lead to catastrophic performances.

Our strategy of parallelization is based on the replication of the mesh information on each processor. To overcome memory constraints due to such a replication, we only compute some minimal mesh information on each processor, ignoring the links between cells and allocating memory for cells only when needed. We call such a mesh a *light mesh*. In the Euro-Mediterranean area treated in section 5 using the light mesh instead of the full one spares 13.5 Mbytes. For a global Earth mesh we spare more than 200 Mbytes per process (nearly half the memory available for one processor in our node's cluster).

The parallel execution may involve any number of processors and we usually map one process per processor. Each process receives from the root process a description of the mesh (such as the description presented in table 3) covering the area under investigation and a set of rays to trace (which are equally distributed among the processes). Each process then starts to compute its rays. At the beginning of each ray segment, the process tests if the ray path has entered a new mesh cell and tests to which cell the segment endpoint belongs. If the ray is the first to enter the cell, memory is allocated for this cell in order to store information related to the ray traversal, otherwise the existing cell data is updated with information brought by the ray.

Once all rays have been computed by all the processes, information for a given cell may potentially be distributed over several processes. We therefore need to merge, for each cell, the distributed data. Each process is assigned a geographic distinct subset of the mesh, called a *sub-domain*. Each process is made responsible for gathering information corresponding to its sub-domain from other processes. An all-to-all communication phase follows in which each process first sends the data it has computed to the appropriate processes and then receives data computed by others and related to its sub-domain. Notice that this phase is extremely costly: for n processes there are n^2 sub-domains, and for typical executions like the ones in section 5 a total of approximately 1.5 GBytes of data is exchanged. Once the processes have received all their data, they individually merge the results so as to compute a unique *score* for each cell corresponding to its illumination quality, and send back their

results to the master process (figure 4). Figure 3 shows the visual results of the whole process in an Earth mesh. These scores are computed from the seismic activity of year 1999.



(a) (b)

Figure 3. Scores computed in a global Earth mesh (first layer) : figure (a) represents the ray density and the figure (b) shows the cell fill rate. The color scale goes from blue for poorly informative cells, to red for highly exposed cells.

The whole application scheme can be summarized as follows :

1. **data transmission**	*one to all comm.*	the ray data as well as the mesh description are distributed by the master processor to all the processors.
2. **ray-tracing**	*computation*	each processor computes the ray paths from the received ray data.
3. **mesh update**	*computation*	on each processor, ray paths are analyzed and cells of the local light mesh are enriched with ray data information.
4. **data exchange**	*all to all comm.*	information exchange so that each process owns all the information related to its sub-domain computed by other processes.
5. **merge**	*computation*	cell information is merged.
6. **score computation**	*computation*	each cell is given a score reflecting its illumination.
7. **concatenation**	*all to one comm.*	all score information is sent back to the master process.

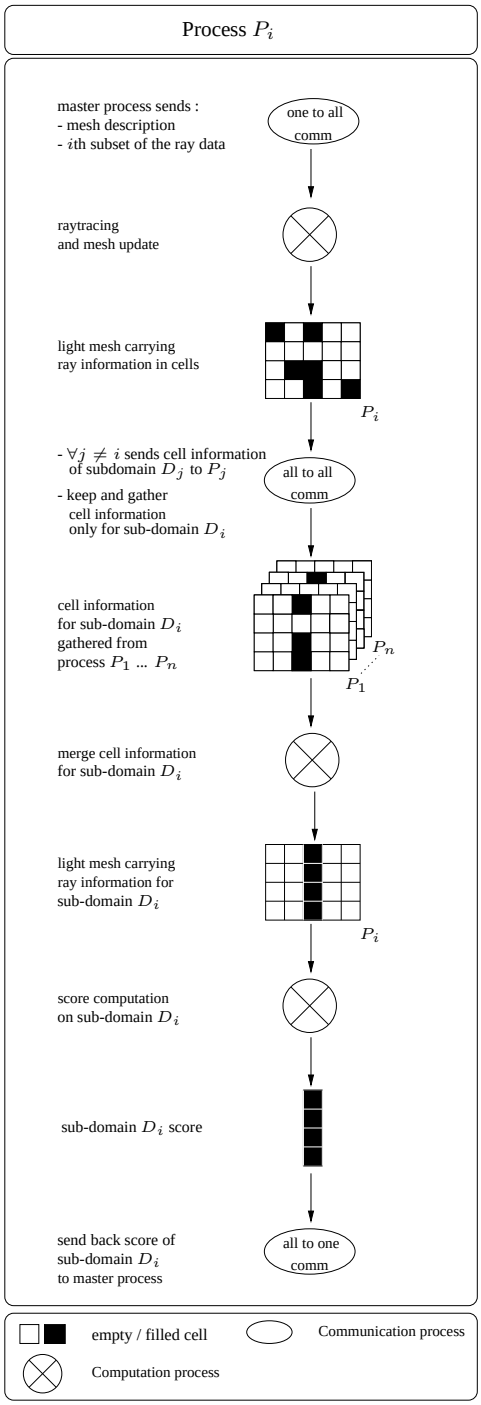


Figure 4. Computation and communication scheme of processor P_i responsible for sub-domain D_i

3.2. Implementation

It is important that the design and the implementation of the code be modular and portable. It is modular as each functionality is written as a library: the mesh feeding uses the library that implements the ray-tracing algorithm based on the Snell-Descartes law, which in turn uses a library implementing the IASP91 model and another one that provides ray signature parsing (to guide the ray-tracing process). Thus, other ray-tracing algorithms could be easily plugged in the application in the future. The software organization is described in figure 5.

Concerning portability, the code is written in C and contains Fortran routines borrowed from Kennett [9]. The library used to parallelize the code is MPI [10] (note however that the user can easily choose at compile time if he wants a sequential or parallel version). All these technical choices come from portability requirements since the application must be built and run on any Unix system: we discuss in section 3.3 various execution environments at which the application can be targeted. The versatility of the environments leads to strong portability requirements.

Currently, the application runs on Linux (various distributions), IRIX, SunOS and HPUX systems. Our application also generates VTK [12] files to visualize the results.

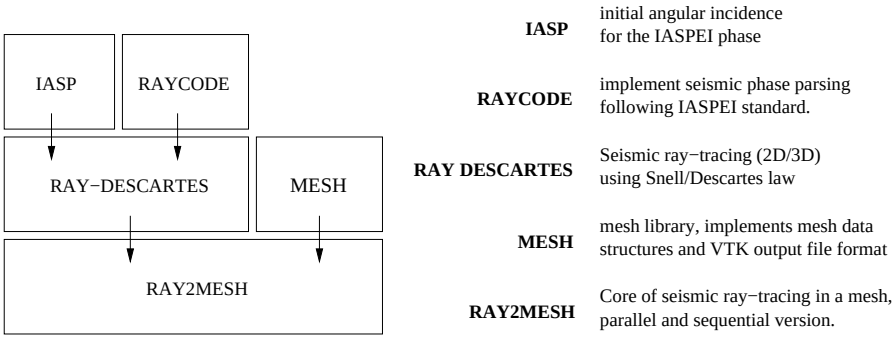


Figure 5. Software is organized as collaborating packages

3.3. Experimentations and target architectures

In the following, we proceed with two kinds of experiments. The first one, in section 4, evaluates the ray-tracing performance on a parallel computer and we compare these results to related published works. As the ray-tracing process is the core part of the application, we need to assess that the ray-tracing speed is fast enough to enable the computation of huge quantities of rays.

The second experiment in section 5, evaluates the performances of the whole application. The computing resources we first had in mind for the application execution were parallel computers. In [8], we ran a series of tests on an SGI Origin 3800 parallel computer. Since then, the data exchange and merge phases (described at the end of section 3.1 and on figure 4) have been added, and new experimentations have been conducted. Note that each appli-

cation phase assesses different hardware features: ray-tracing depends on processor speed, mesh update depends on the available quantity of RAM, and data exchange performance relies on fast network.

Therefore, it is informative to test the application on various architectures to analyze how each of these hardware features behaves on the different test platforms. In order to better assess the application behavior with these new features, we also measure the performances on a cluster of PCs, and we report in this paper these complementary results. Further work has consisted in experimenting the application behavior in an heterogeneous environment commonly referred to as *computational Grids* [6]. The heterogeneity is due both to the processors speeds and to network bandwidths and latencies.

Since MPI is available on all these platforms, the application can be executed nearly without code modification in these new environments. However, without further modification, the performances obtained in a Grid environment are bad due to the heavy load imbalance. From the study of the application's behavior carried out in [3], it came out that new programming paradigms had to be used. We have therefore integrated several enhancements in the application, which greatly improve the performances obtained in heterogeneous environments. The improvements come from the use of a *master-slave* (also referred to as task-farming) paradigm which brings a good load-balance of the computation resources, and from the compression of the messages sent during the data-exchange phasis.

4. Ray-tracing experimentations

We carried out experiments on two test-cases to evaluate the performances of the ray-tracing computations.

The first test-case concerns the study of a regional area (the Euro-Mediterranean area). Figure 6 illustrates the ray-tracing performed in this area. We have used the data set from Granet and Trampert [7] of about 17000 rays with a P signature, which reduces to 10751 rays after removing those which get out the geographic boundaries of the studied area described in table 3.

In the second case, we traced all the seismic events of year 1999, as recorded in the world-wide ISC databases. The data set is composed of about 827000 rays, of which 325749 can be traced (the remaining rays do not have a precise enough signature).

In the following, we try to evaluate the number of rays computed by our ray-tracing algorithm in a given time unit. We call this metric the *ray-tracing speed*.

Table 2 and 4 present the ray-tracing speed obtained on an SGI Origin 3800 parallel computer with 500 MHz Mips R14K processors. We have chosen a 2 km long discretization step all along the ray-tracing to get a very good precision. We call np the number of processors involved (first column in table). All times are in seconds, min and max refer to the shortest and longest (resp.) computation times observed on the different processors. These times measure both the data transmission and ray computation (I/O operations are excluded from the measurements). The absolute speed $speed_a$ is the number of rays computed divided by max, while the relative speed is $speed_r = speed_a/np$. Column e indicates the efficiency

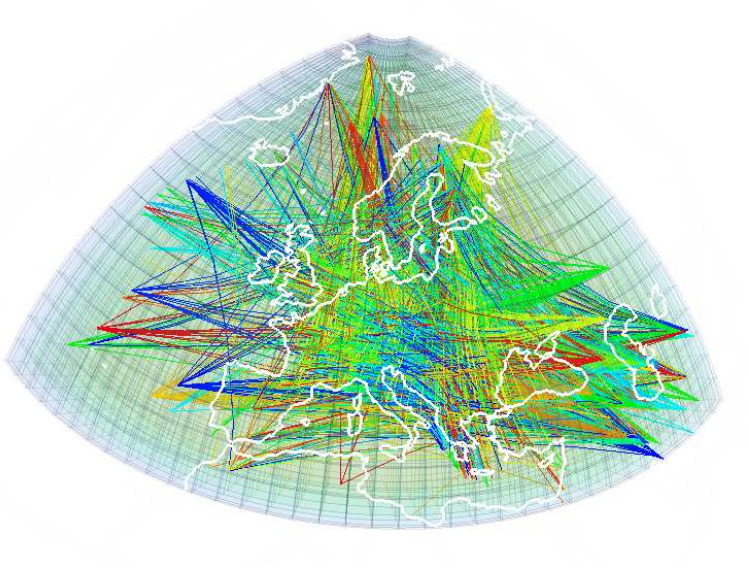


Figure 6. VTK image of 2500 rays in a mesh of the Euro-Mediterranean area.

of the parallel runs depending on np. All times are average times based on measures from two to four runs.

4.1. Regional area

For the regional area study, table 2 shows that we achieved a ray-tracing speed of 3821 rays$/s$ on 64 processors. Using only one processor, the ray-tracing speed is about 168 rays$/s$.

Table 2. Ray-tracing speed of 10751 P-type rays, with a 2 kms ray discretization in a regional mesh.

np	min	max	$speed_a$	$speed_r$	e
1	65.36	65.36	164.49	164.49	100%
2	33.16	33.18	324.07	162.03	98%
8	8.25	8.67	1240.50	155.06	94%
16	4.27	4.79	2245.64	140.35	85%
32	2.36	3.11	3462.48	108.20	66%
64	1.75	2.81	3821.45	59.71	36%

```
<?xml version="1.0"?>
<mesh
 lat-unit-size="0.5" lon-unit-size="0.5"
 lat-min="30"  lat-max="87"
 lon-min="-42" lon-max="66">
 <layer name="l1" zstart="0"    zend="40"   lat-unit="1" lon-unit="1" />
 <layer name="l2" zstart="40"   zend="100"  lat-unit="1" lon-unit="1" />
 <layer name="l3" zstart="100"  zend="250"  lat-unit="1" lon-unit="1" />
 <layer name="l4" zstart="250"  zend="400"  lat-unit="1" lon-unit="1" />
 <layer name="l5" zstart="400"  zend="650"  lat-unit="1" lon-unit="1" />
 <layer name="l6" zstart="650"  zend="900"  lat-unit="1" lon-unit="1" />
 <layer name="l7" zstart="900"  zend="1200" lat-unit="1" lon-unit="1" />
 <layer name="l8" zstart="1200" zend="1500" lat-unit="1" lon-unit="1" />
</mesh>
```

Table 3. The mesh description of the regional Euro-Mediterranean area

We can compare this result to the ray-tracing algorithm of Bijwaard and Spakman [1] whose speed is claimed to be 15 rays$/s$ on a Mips R8K. Other recent results with a ray-tracing method based on minimization of travel times come from Cores *et al.* [2] who obtain a maximum speed of about 12.5 rays$/s$ on a Sparc-Ultra 1. However, they only tested their algorithm on small data sets. Sambridge and Gudmunsson [11] obtained a ray-trace speed at about 12.5 rays$/s$ on a Sun Sparcstation 10/40.

Obviously, these speeds would be much faster on a MIPS R14K processor. However notice that it is difficult to compare these results more precisely because the algorithms are different and mainly because we select a 2 km step whereas Bijwaard and Spakman [1] used a step varying from 10 km to 5 km depending on depth.

In terms of speed-up, we can see from results in table 2 that the algorithm is scalable up to 16 processors (Bijwaard and Spakman [1] used a maximum of 3 processors for the computation, hence we cannot compare this aspect). Our parallel algorithm shows an efficiency of 85% and 65% with 16 and 32 processors respectively, and decreases dramatically to 36% with 64 processors.

We must also note that the processor load imbalance (let us define it by $(max-min)/max$) is correlated with the efficiency. It remains acceptable up to 16 processors (about 10%) but increases up to 37% with 64 processors.

This behavior can be explained by the relatively small size of the data set: with 64 processors, each processor has only 168 rays to compute. In this case, the overhead due to data transmission and the imbalance in ray length distribution to processors becomes predominant. The experiment at the Earth scale with many more rays, described in section 4.2, confirms this explanation.

4.2. Global area

The second test-case deals with the computation of rays issued from seismic events which occurred during the year 1999, with rays traveling through the whole Earth model. Unlike the Euro-Mediterranean experiment, we have a very large set of ray signatures as we treat rays with P, S, Pn, Pg, pP, PKP, PcP, $pPKP$, ScP, *etc*, signatures. Table 4 gives

the ray-tracing speed for more than 3.10^5 rays (about 30 times more than in the previous experiment).

Table 4. Ray-tracing speed of 325749 rays (large set of ray signature), with a 2 km ray discretization in a global mesh.

np	min	max	$speed_a$	$speed_r$	e
2	960.70	1158.58	281.14	140.56	97%
8	244.99	308.43	1056.15	132.01	81%
16	132.47	171.02	1904.74	119.04	73%
32	86.32	108.63	2998.70	93.70	57%
64	49.90	69.35	4697.17	73.39	45%

The overall ray-tracing speed is slightly slower in this experiment, except for 64 processors. It is not surprising as the dataset contains rays with higher lengths, like PKP rays, which take more time to compute. However, we can notice that the efficiency decreases less quickly: from 57% to 45% with 32 and 64 processors, correlated with a load imbalance that remains more stable in this experiment, between 20% and 28% on 8 and 64 processors respectively. This can be explained by the fact that each processor receives a bigger set of rays (more than 5000 rays per processor for 64 processors) and hence, the data transmission overhead (step 1 of the application) is not significant compared to the ray computation time.

5. Mesh experimentations

In the following series of experimentations, we complete the ray-tracing process with the subsequent steps of the application (steps 3 to 7 in section 3.1). The data set used for these experiments describes the seismic rays of year 1999. In addition we use a global Earth mesh (described in table 5) made of 11 layers, all its cells being $1^\circ \times 1^\circ$ wide so that there are a total of 712800 cells. Note that, though we use cells of identical size, the cells could be of arbitrary sizes in latitude and longitude in a given layer.

The experiments have been conducted on three kinds of equipments which greatly differ in their hardware architectures. We first present results obtained on a parallel computer, which was our primary target. Then we compare these results with the execution on a cluster of PCs. Finally, we briefly explain the application behavior in a Grid environment.

The perspective of using computing environments with heterogeneous resources led to the development of two distinct communication schemes for the distribution of rays. In the first alternative, the rays are distributed in *scatter* mode: one process called *root*, distributes an equal share of the rays to be traced to all processes (itself included) in a single communication round. Once their share received, processes perform the rays computation without interruption. The second alternative uses the *master/slave* paradigm. In this case, one master process reads small chunks of the ray data set and distributes these chunks to slave processes, which compute the received piece of data. As soon as a slave process has

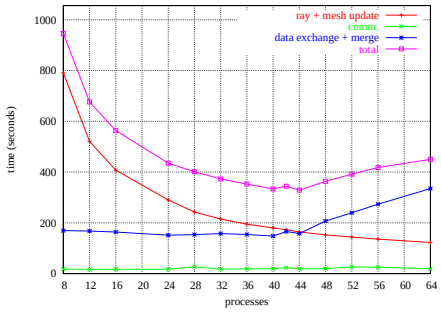
```
<?xml version="1.0"?>
<mesh
lat-unit-size="1" lon-unit-size="1"
lat-min="-90" lat-max="90"
lon-min="0"   lon-max="360">
 <layer name="l0"  zstart="0"    zend="20"   lat-unit="1" lon-unit="1"/>
 <layer name="l1"  zstart="20"   zend="35"   lat-unit="1" lon-unit="1"/>
 <layer name="l2"  zstart="35"   zend="120"  lat-unit="1" lon-unit="1"/>
 <layer name="l3"  zstart="120"  zend="210"  lat-unit="1" lon-unit="1"/>
 <layer name="l4"  zstart="210"  zend="410"  lat-unit="1" lon-unit="1"/>
 <layer name="l5"  zstart="410"  zend="660"  lat-unit="1" lon-unit="1"/>
 <layer name="l6"  zstart="660"  zend="760"  lat-unit="1" lon-unit="1"/>
 <layer name="l7"  zstart="760"  zend="2740" lat-unit="1" lon-unit="1"/>
 <layer name="l8"  zstart="2740" zend="2889" lat-unit="1" lon-unit="1"/>
 <layer name="l9"  zstart="2889" zend="5153" lat-unit="1" lon-unit="1"/>
 <layer name="l10" zstart="5153" zend="6371" lat-unit="1" lon-unit="1"/>
</mesh>
```

Table 5. The mesh description of the global area

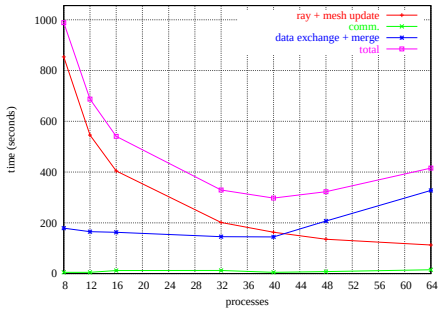
finished its computations, it asks the master for a new chunk. The process repeats until the master sends a termination message to the slaves. Note that the master process is only assigned the administrative task of distributing data and does not compute any ray. Still, the workload in this scheme is naturally balanced as faster processors ask for more work.

5.1. On a parallel computer

The parallel computer is the SGI Origin 3800 with 512 R14K/500Mhz processors. The MPI library used is the vendor MPI. Though processors power approximatively compares to an Intel Pentium III/800 MHz, the processors interconnection network and I/O operations are extremely efficient.



(a) World mesh computation with 325749 rays in *scatter* mode.



(b) Computation with 325749 rays in *master/slave* mode.

Figure 7. World mesh computation benchmarks on the Origin 3800.

The performance results are presented on figure 7. The graphs draw the longest times observed: for instance the total execution time represented for a given number of processes is the end time of the slowest process. First, the scatter and the master/slave mode show comparable performances, except for a small number of processes, in which case the fact that the master process does not participate to computations is not negligible. We can notice that the total execution time decreases up to 40 processors, but increases when more processors are involved. It seems that the data set is not big enough for more than 40 processors. To check this, we doubled our original data set and observed that the optimal number of processors is 56 in this case, as shown on figure 8. The overall results exhibited

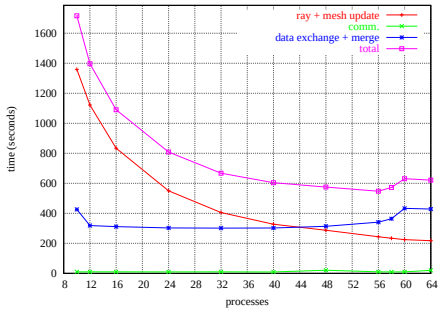


Figure 8. World mesh computation on the Origin 3800 with twice more rays in *master/slave* mode.

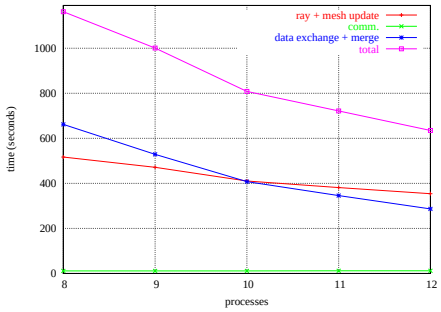
in this last experiment are quite satisfactory: the information that can be extracted from the seismic events of a whole year have been structured into a geographical mesh in less than 6 minutes. This can not be done on standard computing equipment mainly because of RAM requirements: our attempts to run this experiment on a single PC equipped with 2 GBytes rapidly failed because of memory exhaustion.

5.2. On a cluster

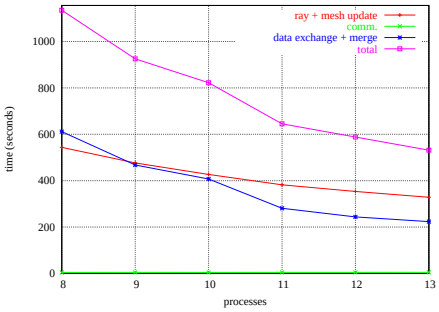
Our cluster of PCs is composed of 6 nodes interconnected via an Ethernet Gigabit switch. Each node is bi-Pentium Xeon 1.7 GHz with 1 GBytes RAM. We use LAM/MPI[13] as communication library.

Results of runs in scatter and master/slave modes are presented on figure 9. Not surprisingly, the CPU hungry ray-tracing and mesh update phases are done faster on the cluster than on the parallel computer: for 12 processors, it takes 354 s on the cluster and 520 s on the R14K processors. Although the cluster processors are faster, the total execution times are roughly equal on the two systems. This is due to the cluster communication subsystem which, even with Giga-bit ethernet cards, cannot rival with the dedicated hardware of the parallel computer: the data exchange between 12 processors takes 286 s versus 167 s on the Origin.

Finally, figure 10 summaries the total execution times of the scatter and master/slave versions on both architectures. The figure (a) shows a slight advantage for the cluster,



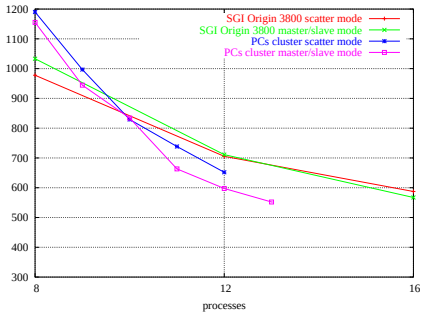
(a) World mesh computation with 325749 rays in *scatter mode* on the PCs cluster.



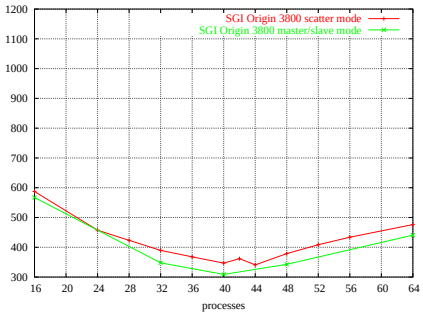
(b) World mesh computation with 325749 rays in *master/slave* mode on the PCs cluster.

Figure 9. World and Euro-Mediterranean mesh computation benchmarks on the PCs cluster.

but we can not say how the application would scale on this kind of platform with more processors. Concerning the parallel paradigms tested, the master/slave mode shows as good performance as the scatter mode, even with homogeneous processors. Though the master/slave mode uses one less computing process, it avoids the bottleneck encountered in the scatter mode. The reason is that computations may begin only once all processes have received their share of work, and the computer which runs the root process can only send the initial data to processes in turn.



(a) *scatter* mode



(b) *master/slave* mode

Figure 10. World mesh computing speed comparison between two kinds of architectures (Origin 3800/Cluster) using two different parallels algorithms (*scatter* mode *vs* master/slave *mode).*

5.3. In a Grid environment

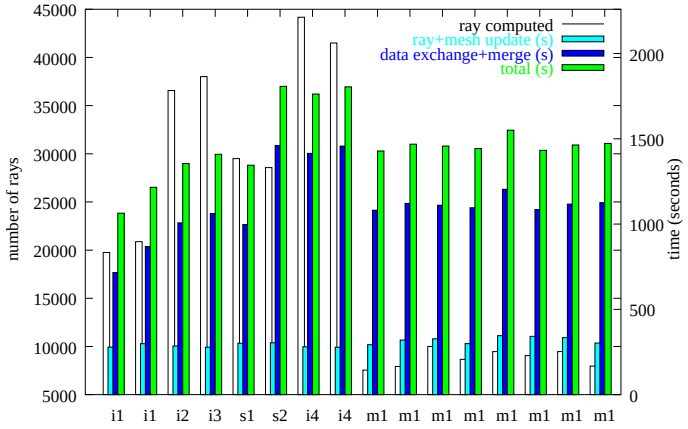
A subset of our Grid testbed, installed with Globus [5] and MPICH-G2 [4] has been used to run the application. We chose the fastest CPUs available to have an aggregate computing power comparable to the two other systems. From benchmarks based on a run of the application, we have given ratings reflecting the relative execution times on several CPUs. A rating of 1 is arbitrarily chosen for an Intel Pentium III/800 MHz processor. The selected machines are distributed over three sites: two campuses of Louis Pasteur university (Illkirch and Strasbourg) and a site at the other end of France (Montpellier), with the Origin 3800 System. At the university, eight PCs were selected whose ratings range from 1 to 1.79, with an average value of 1.34. In addition, some other eight processors were taken from the Origin 3800 with an average rating of 1. The network bandwidth between the two city campuses is typically 3.5 Mbytes/s, whereas the WAN link from the University to the Origin 3800 is about 250 KBytes/s. For our test, we run in master/slave mode since experiments with the scatter mode showed huge imbalances in processors workloads. The input ray dataset is located on one of the PCs that serves as master.

A typical execution of the application with 16 processors is represented on figure 11. On the horizontal axis, the processors are named depending on their geographic location: the first letter refers to the sites, either Illkirch (i), Strasbourg (s) or Montpellier (m). The following number indicates if the processor belongs to a same computer: identical names identify different processors of a same computer. The master/slave paradigm brings an acceptable load-balance in total execution times: the longest execution time is 13% longer than the shortest one. Although, from the tests on this sole configuration, we conclude that much work should be done to reach acceptable performances. While the computation time of rays is a bit faster than on the parallel computer, the data-exchange phasis is seven times longer and the overall execution time is three times longer. We note a small improvement when messages are compressed with *zlib* during the data-exchange phasis (the maximum duration is 1291 s instead of 1460 s without compression). It appears that the all-to-all communication scheme is an unavoidable bottleneck in a Grid environment, and that the design of this phasis should be reworked to alleviate the loss of performance.

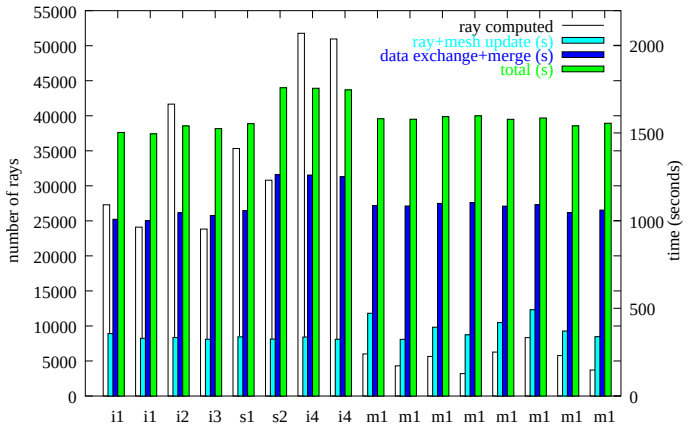
6. Conclusion

In this paper, we have described the design and the performances of the core part of a seismic tomography tool under development. This work is the preliminary step of an ongoing project whose aim is to build an adaptive mesh to model the Earth interior. We put forward the parallel design of the program, since the huge quantity of data to be computed will require massively parallel computations. At this step of the project, we have evaluated and compared the performances obtained to related work.

The results show that the extension to a wider set of data, that is millions of rays traced in a full Earth mesh are attainable. In this case, the use of massively parallel computational resources will be unavoidable. This is the reason why the project is part of a larger initiative called TAG – which stands for *Transformations and Adaptations for the Grid*, http://grid.u-strasbg.fr – whose objectives are to port scientific applications on the grid and to develop tools for efficient execution of programs in this framework.



(a) World mesh computation of 325749 rays, *master/slave*, without *zlib*



(b) World mesh computation of 325749 rays, *master/slave*, with *zlib*

Figure 11. Execution in a Grid environment : 16 computing processors on three geographically distant sites

Acknowledgments

An important part of the experiments have been carried out on the SGI Origin 3800 of the CINES (http://www.cines.fr), and on the PCs cluster of the Institut de Physique du Globe in Strasbourg. We want to thank them for letting us have access to their computational resources.
We are grateful to Michel Granet who provided valuable comments, and to the RENASS, the French National Seismic Network (http://renass.u-strasbg.fr) for providing ISC data.

References

1. Harmen Bijwaard and Wim Spakman. Fast kinematics ray tracing of first and later arriving global seismic phases. *Geophys. J. Int.*, 139:359–369, 1999.
2. Debora Cores, Glenn M. Fung, and Reinaldo J. Michelena. A fast and global two point low storage optimization technique for tracing rays in 2D and 3D isotropic media. *Journal of Applied Geophysics*, 56(45):273–287, September 2000.
3. Romaric David, St´ephane Genaud, Arnaud Giersch, Benjamin Schwarz, andÉric Violard. Source code transformations strategies to load-balance Grid applications. In Manish Parashar, editor, *Proceedings of Grid Computing - Grid 2002 : Third International Workshop*, volume 2536 of *Lecture Notes in Computer Science*, pages 82–87, Baltimore, MD, USA, November 2002. Springer-Verlag.
4. Ian Foster and Nicholas Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. *Supercomputing*, November 1998.
5. Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.
6. Ian Foster and Carl Kesselman. *The Grid, Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.
7. Michel Granet and Jeannot Trampert. Large-scale p-velocity structures in the euro-mediterranean area. *Geophys. J. Int.*, 99:583–594, 1989.
8. Marc Grunberg, St´ephane Genaud, and Catherine Mongenet. Parallel seismic ray-tracing in a global earth mesh. In *Proceedings of PDPTA'02*, pages 1151–1157, June 2002.
9. Brian L.N. Kennett. Iaspei 1991 seismological tables. *Research School of Earth Sciences Australian National University*, 1991.
10. Message Passing Interface Forum. *MPI : A message-passing Interface Standard*, June 1995.
11. Malcom Sambridge and Olafur Gudmundsson. Tomography systems of equations with irregular cells. *J. of Geophys. Res.*, 103(No. B1):773–781, 1998.
12. Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics*. Prentice Hall, December 1997.
13. Jeffrey M. Squyres, Andrew Lumsdaine, William L. George, John G. Hagedorn, and Judith E. Devaney. The interoperable message passing interface (IMPI) extensions to LAM/MPI. In *Proceedings, MPIDC'2000*, March 2000.