Parallel radio-wave propagation modeling with image-based ray tracing techniques

T.E. Athanaileas a,*, G.E. Athanasiadou b, G.V. Tsoulos b, D.I. Kaklamani a

a National Technical University of Athens, School of Electrical and Computer Engineering, Heroon Polytechiou 9 St., Athens 15773, Greece
b University of Peloponnese, Department of Telecommunications Science and Technology, Terma Karaiskaki, Tripolis 22100, Greece

Article info
Article history:
Received 3 July 2009
Received in revised form 23 April 2010
Accepted 7 August 2010
Available online 17 August 2010

Keywords:
Wireless communications
Radio-wave propagation
Ray tracing
Performance evaluation
Load balancing
Message passing interface

Abstract
Ray tracing is a technique based on the numerical simulation of geometrical optics and the uniform theory of diffraction, two well-known approximate methods for estimating a high-frequency electromagnetic field, based on the ray theory of field propagation. Radio-wave propagation prediction models based on ray tracing play an important role in wireless network planning, as they take into account diverse physical phenomena such as reflection, diffraction and foliage attenuation and are considered critical for the analysis of long term evolution (LTE) systems, which requires a detailed description of the wireless channel. A major practical drawback of these models is that they can easily become very computationally expensive, as the required level of accuracy and the corresponding areas of study increase.

In this paper, a parallel ray tracing algorithm for radio-wave propagation prediction based on the electromagnetic theory of images is presented. The implementation of the algorithm is based on the message passing interface (MPI). The decomposition of the problem is conducted by partitioning the image tree, while dynamic load balancing techniques are employed by means of the master–worker and the work–pool patterns. The performance of the parallel implementation is studied for different problems and task assignment schemes, showing that high speedups can be achieved.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

The major limiting factor imposed on wireless systems is due to the radio channel characteristics. In order to develop and deploy a wireless system, knowledge of the radio channel is essential. Simple propagation and channel models play a critical role in the development, planning and deployment of mobile radio systems. However, with mobile telephony being used by ever increasing percentages of the population, in most urban areas capacity has replaced coverage as the most important issue. Additionally, with the advancement of various modern communication systems, such as fixed wireless access (FWA) and multiple input–multiple output (MIMO) systems, in-depth channel characteristic studies in site-specific environments are becoming increasingly important. Propagation models are now required to provide accurate prediction of radio-wave propagation, by taking into account the exact position, orientation and electrical properties of individual buildings, as well as other characteristics of the environment (e.g. foliage).

* Corresponding author. Tel.: +30 2107722287.
E-mail addresses: thanat@esd.ntua.gr (T.E. Athanaileas), gathanas@uop.gr (G.E. Athanasiadou), gtsoulos@uop.gr (G.V. Tsoulos), dkaklam@mail.ntua.gr (D.I. Kaklamani).

0167-8191/– see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2010.08.002
Ray tracing is a technique used for producing deterministic channel models that operate by processing user-defined environments. The major drawback of deterministic modeling of the radio propagation channel has been the computational burden compared to that of statistical modeling [1]. In order to reduce the required computation time, a number of approaches have been proposed in the literature. Techniques that are typically used include database pre-processing, visibility determination, image tree pre-calculation, as well as other approximation techniques that manage to reduce execution time at the expense of prediction accuracy. Another approach that does not trade accuracy for speeding up computation times and hence can provide an efficient solution to the above problem is parallel processing. A major advantage is that a parallel implementation has the potential of scaling with the increase of computational resources, thus proving efficient for the solution of increasing problem sizes.

In this paper, the implementation of a parallel ray tracing model for radio propagation prediction based on the electromagnetic theory of images [2,3] is presented. The main concept of this model is the creation of a tree of images of the base station, considering all walls and obstacles as potential reflectors. The main concern for the design and development of the parallel implementation is scalability and efficiency. A number of issues have been considered towards this goal, including task granularity, reduction of communication costs, task distribution and load balancing. The nature of the prediction problem poses certain difficulties to load balancing, since it is typical that a certain site may consist of one or more hot-spots (areas illuminated by a large number of rays) that require significantly more computations compared to other areas. In order to tackle this problem, the adopted parallelization approach entails the partitioning of the creation of the image tree, resulting in finer-grained and dynamically created tasks. Task distribution is based on the well-known master–worker and work–pool paradigms and the performance of different assignment schemes is studied. The implementation is based on the message passing interface specifications, in order to support portability to different hardware architectures. Metrics such as speedup, resource utilization and workload expansion ratio are used for performance analysis and evaluation and results from several experiments on propagation prediction problems of different sizes show that the proposed implementation is scalable for varying problem sizes.

The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 briefly presents the sequential ray tracing model for radio-wave propagation prediction that has been previously developed. Section 4 presents considerations regarding the parallelization of the sequential algorithm, as well as implementation details. Results from several experiments for different database configurations are presented in Section 5 along with a performance evaluation, while Section 6 concludes the paper and discusses future work.

2. Related work

Two widely used techniques for radio-wave propagation modeling are the shooting and bouncing ray (SBR) method [4,5] and ray tracing [6]. According to the SBR method, rays are launched with a specific angular separation from source points, so that no building faces are missed at large distances from the receiver, and their paths are traced until a certain power threshold is reached. Techniques based on ray tracing consider all walls and obstacles as potential reflectors and evaluate the location of their base station images. Generally, deterministic models for radio-wave propagation prediction are computationally very expensive compared to statistical modeling [1] and hence, in order to reduce the required computation time, a number of approaches have been proposed. In [7], techniques for reducing the database complexity by means of building footprint simplification are presented. In [8], authors exploit several approximation methods for different propagation models (e.g. multipaths are disregarded if the distance between the mobile station and the base station surpasses a certain threshold). In [9], authors present progressive and approximate techniques for SBR systems, according to which intermediate prediction results are fed back to users continuously, while the workload is dynamically adjusted during the prediction process. A common drawback of the aforementioned approaches is that the reduction of the required computation time is achieved at the expense of the accuracy of the prediction model. Also, methods that do not affect accuracy have been proposed. In [6], visibility determination techniques based on the concept of illumination zones of the images have been presented in order to prevent the exponential increase of images in higher orders of reflection. The aforementioned methods manage to improve runtime efficiency; nevertheless they do not provide a sufficient solution to increasing problem sizes.

Parallel processing has been identified as a solution that has the potential to address the requirements of deterministic radio-wave propagation prediction models, as the ability to scale with increasing computational resources proves very efficient for larger problems. Several approaches to parallel propagation prediction based on the SBR method have been proposed. SBR is an intrinsically parallel technique, because the rays that are launched by the transmitting antenna are independent to each other, allowing the SBR code to be directly applicable in the parallel programming paradigm. In [10], authors present a parallel outdoor propagation model for microcells for the Cray T3E supercomputer, based on the message passing paradigm. Although the computation time is reduced compared to the sequential version, the model does not achieve efficient scalability. In [11], a parallel three-dimensional radio-wave propagation modeller running on a 200-node Beowulf cluster of Linux workstations has been developed based on the message passing interface, and utilized for globally optimal transmitter placement for indoor wireless communication systems. Each processor has a complete copy of the building database locally and the model considers only reflections and transmissions. In [12], authors propose a combination of
the phase-parallel and manager-workers paradigms as the underpinning framework for implementing the SBR technique on a network of workstations. The original computation is partitioned into multiple small tasks based on either raypath-level or source-point-level granularity, while dynamic load balancing scheduling schemes are employed to allocate the resulting tasks to the workers. The parallel implementation is shown to achieve nearly linear speedup for different problem sizes and for an increasing number of computer nodes. In [13], a parallel model based on 3D radio-wave propagation prediction is presented. The total workload is divided according to the initial rays to be launched and field points to be evaluated, and the resulting tasks are statically assigned to the nodes of the parallel system randomly. While there is much published work on parallel models based on the SBR method, the parallel processing of image-based ray tracing for radio-wave propagation prediction has not been studied in the literature.

A field related to the work presented herein is ray tracing in the context of computer graphics applications. A number of approaches based on parallel processing have been presented for dealing with large computational requirements of the corresponding models. In [14,15], authors present different approaches for the realization of a parallel graphics ray tracing system. In [14], a hybrid scheduling approach is presented that combines demand driven and data parallel techniques, taking into consideration the data intensity of the task and the amount of data locality that will be present in the task, while in [15] the reduction of communication costs is discussed. In [16], authors study the effects of different task partitioning schemes in a parallel ray tracing implementation based on MPI. In [17], a parallel progressive ray tracing approach for interactive-rate animation generation is presented, which exploits the temporal coherence between frames in order to achieve efficient load balancing. Recently, research has been concentrated on parallelization of specialized data structures used for fast, real-time ray tracing. For example, in [18], a technique for the parallel construction of the kd-tree structure is presented, while in [19], authors present a parallel variant for building bounding volume hierarchies (BVHs), in order to better exploit the multi-core architecture of modern CPUs. Early approaches essentially consider their computational problem as almost embarrassingly parallel, which can be divided into a number of components that can be independently processed. The problem of radio propagation modeling with image-based ray tracing entails concepts such as the image tree, which contains certain dependencies and thus these approaches are inapplicable in this case. Moreover, radio waves are electromagnetic waves of lower frequencies compared to light, which means that different propagation mechanisms are presumed (for example corner diffraction is not considered for computer graphics).

In this paper, a novel parallel model for radio-wave propagation prediction with an image-based ray tracing technique is presented. Naturally, the image tree is totally concatenated, hindering the splitting of the tasks and the load balancing among the processor nodes. The parallel model presented herein proposes a scheme for partitioning the creation of the image tree, resulting in the twofold advantage of minimizing sequential parts in the algorithm and achieving well balanced workloads, as it will be further explained in the next sections.

3. Sequential ray tracing model

3.1. Imaging technique

The propagation algorithm takes into consideration all walls and obstacles as potential reflectors and evaluates the location of their base station images. This imaging technique works by conceptually generating an image tree for each base station location. This image information is then stored and used for computing the channel characteristics at each user location. In order to reduce the number of base station images, each image that is computed is associated with a certain illumination zone, which is the area for which the specific image can give a valid path [6]. Only walls and obstacles inside the illumination zone of the image can be used for the formation of new images. Moreover, once the reflection images are formed, they are not valid for the entire wall, but only for the part of the wall illuminated by the previous image.

Fig. 1 depicts the basic concept of image generation and path tracing. The base station (BS) is initially reflected about the first reflecting wall to generate the image BS\textsubscript{1}. This image is then reflected about the second reflecting wall to generate the image BS\textsubscript{2}. The path itself is then re-created in reverse order. Initially a line is drawn between the mobile position (MS) and BS\textsubscript{2} and the point of interaction is found (point A). A line is then drawn from point A back to BS\textsubscript{1} and the second reflection point (point B) is found. Point B is then traced back to the base station and a purely reflected path is said to exist. In Fig. 1, the illumination zones of the reflection images are also shown. The illumination area is the area for which the specific image can give a valid path. Hence, except for the position and the wall of each image, the area illuminated by the image is also calculated and stored. The number of images is significantly reduced since only images which are capable of producing valid paths are generated and stored in the image table. In the first place, only walls inside the illumination zone of the image can be used for the formation of new images. Moreover, once the reflection images are formed, they are not valid for the entire wall, but only for the part of the wall illuminated by the previous image.

According to this imaging algorithm, a simple example of an image tree for a base station placed in a rectangular room is depicted in Fig. 2. The figure shows all possible permutations of walls and indicates which reflection images are not valid when taking into account their illumination zones.
3.2. Ray tracing model

The parallel model presented in this paper is based on a 3D ray tracing propagation model that calculates the electromagnetic field of each ray according to the uniform theory of diffraction (UTD) and the geometrical optics (GO) and takes into account the illumination zones technique.
account the losses incurred due to partial Fresnel zone blocking when radio waves travel above buildings and foliage [3]. The model works with raster terrain as well as 3D vector building and foliage databases. It also considers reflections off building walls, off-axis roof top and terrain diffractions and calculates the attenuation of each ray when it passes through foliage and when the Fresnel zone is blocked.

The main steps of the incorporated algorithm (Algorithm 1) are the following:

1. **Loading and processing of the building, foliage and terrain databases**: a database containing all terrain, foliage and building data is loaded into the main memory.
2. **Creation of the BS image tree**: the mirror images of the BS with respect to the building walls are computed, according to the algorithm described in Section 3.1.
3. **Path tracing**: after the image tree of the BS has been created, the heights of the terrain points and buildings along the path from the MS to the BS antenna are calculated for each point in the receiving points grid, hence the vertical profile along the path is generated. The vertical profile is used in order to examine whether there is a line-of-sight path or any obstructions in the path, in which case the exact coordinates of the diffraction points on buildings and terrain are computed. Rays that propagate through foliage or above foliage are also traced. Reflected rays are traced in a similar way: for each image that can produce a valid path, the reflection points are found and the vertical profile along the whole path is generated.
4. **Field calculations**: when the exact geometry of a ray has been found, its field strength is calculated. The angles of arrival (azimuth and elevation) at the BS and MS antennas are calculated and the respective antenna gains are considered during the field calculations. The model computes the diffraction and reflection coefficients along each path as a function of the incident and departing angles. Effects such as the depolarisation of the reflected power and the foliage attenuation as a function of the path length inside vegetation are also taken into account. The above calculations are performed according to UTD and GO, while between the ray interactions free space loss is considered. For propagation over roof tops, the model also examines any obstructions of the Fresnel zones and calculates any excess path loss, according to the Fresnel zones theory. The set of results consists of received power with and without Fresnel zone blocking, time of arrival, angles of departure and arrival (θ and ϕ) on both antennas and loss due to diffraction on top of trees.

**Algorithm 1. Sequential radio-wave propagation algorithm with image-based ray tracing**

1: load area database into main memory and read input file
2: generateImages(BS)
3: for (each point X in the point grid) do
4:   for (each image Y in the image tree) do
5:     create vertical profile between X and Y
6:     perform ray tracing and field calculations
7:   end for
8: end for
9: deliver prediction results
10:
11: function generateImages(image X)
12: if (current order of reflection < max order of reflection) then
13:   for (each wall Y in the illumination zone of image X) do
14:     compute corresponding image Z
15:     generateImages(Z)
16:   end for
17: store X in the image table
18: end if
19: end function

A more elaborated description of the imaging technique and the sequential ray tracing model used herein is out of the scope of the paper. Interested readers may refer to [2] and [3] for more details.

4. **Parallel ray tracing model**

It is rather evident from the description in the previous section that the image-based ray tracing model is not intrinsically parallelizable. The main obstacle is the fact that the model is based on the creation of an image tree, the parallelization of which is not a straight-forward procedure.
An approach to parallelizing the ray tracing algorithm would be to perform the image tree generation in a sequential fashion before calculating the rays at the receiving points in parallel. If \( z \) is the fraction of the workload that corresponds to image tree generation, then, as Amdahl’s law \([20]\) points out, the speedup \( S_n \) that can be attained with \( n \) processors is:

\[
S_n = \frac{n}{1 + (n - 1)z}
\]

When considering a very large number of processors, the maximum speedup that can be achieved is:

\[
\lim_{n \to \infty} S_n = \frac{1}{z}
\]

In image-based ray tracing, \( z \) can typically become quite large, which greatly restricts the scalability of such a parallelization approach (e.g. for \( z \) equal to 10\%, the maximum possible speedup is 10).

Another approach would be to parallelize the propagation model based on data decomposition techniques. The site area can be partitioned into smaller subareas and each processor can be assigned the task of generating the partial image tree for the obstacles residing in the specific subarea it is assigned with. However, in order to compute images in the next orders of reflection, information about the walls and obstacles in subareas residing in other processing nodes are required. Specifically, image data computed by a certain processor need to be made available to all other processors for computing images in the next order of reflection. This results in large data transfers and consequently large communication overheads, thus this scheme is not considered to be efficient.

The approach adopted in this paper consists in an alternative fine-grained partitioning of the image tree creation process. The image tree is successively and hierarchically built using the tree nodes computed in previous steps as input and computing images in the next order of reflection for the whole database. Ray calculations at the receiver points are also computed in the same task only for the images computed before, as will be thoroughly described in Section 4.1. The main objective is the development of a scalable and efficient parallel implementation for an increasing number of utilized processors. In order to achieve this goal, a number of concerns have been taken into consideration, such as computation decomposition, reduction of communication overhead and task scheduling. These concerns are discussed in detail in the following sections.

4.1. Computation decomposition – task formulation

As stated earlier, the main concern for the parallelization of the image-based ray tracing is the parallel creation of the image tree. It is possible to perform this process efficiently in parallel by building the image tree gradually, using information from images computed for a given order of reflection for computing images for the subsequent order. In the proposed parallel algorithm, each computed image (which corresponds to a tree node) serves as a new parent image and all obstacles in the database are considered for creating child images (according to the algorithm described in Section 3.1) in a single task. Due to the small sizes of the resulting subtrees, calculation of the rays for all points in the database for the images computed in a specific task, is also performed in the same task. However, if the number of points is large (due to a large database or a high resolution of the grid of points), it is possible that the computing requirements of the tasks can become quite heterogeneous, resulting in load imbalance problems. A solution to this potential constraint is further partitioning the process of computing the rays by segmenting the receiving points in subgroups, thus creating finer grained jobs. In this case, the corresponding subtree needs to be made available to the computing nodes along with information regarding the grid points, which might result in communication overheads. Nevertheless, as shown in Section 5, the approach of computing the rays for all points in the database for a given subtree is sufficient even for moderately large databases and a dense grid of points, due to the small size of the subtrees. The parallelization process described above is demonstrated in Fig. 3 for the case of the image tree shown in Fig. 2. In this figure, T0–T16 are the tasks into which the image creation process is decomposed according to the method described before.

![Fig. 3. Computation decomposition.](image-url)
According to this parallelization scheme, images in the first order of reflection need to be computed in a sequential fashion, considering the root node corresponding to BS as the parent image. Unlike images, which have a confined illumination zone [6], the BS illuminates the whole area, resulting in a far larger number of images. This sequential part restricts the scalability of the parallel model, as described previously. In order to overcome this restriction, a data decomposition scheme is used for the computation of images in the first order of reflection: the area database is split into subregions, which are distributed across available processors in order to compute first order images in parallel, with each processor working on the walls and obstacles contained in the specific subregion only. Since no a priori knowledge about hot-spot areas is available, the site database is equally divided in a number of subregions. This can of course lead to load imbalance between processing nodes, especially when larger subregions are considered. However, this load imbalance exists only in the process of computing first order images and is compensated for during computations for higher orders of reflection, where tasks are finer grained. In any case, this is a far more efficient approach than having to perform the creation of first order images in a single task. The above concept is represented by tasks T0.0–T0.3 in Fig. 3, where the creation of first order images reflected from different walls is performed in a different task (although in actual cases these tasks handle the generation of images reflected from all the walls in a specific subregion).

4.2. Task scheduling and load balancing

Clearly, the parallelization scheme described in the previous section results in the dynamic creation of tasks. For each image that is computed, a new task needs to be created for computing images in the subsequent order of reflection, unless the maximum order indicated at the set-up has been reached. In order to cope with dynamically created tasks, the master–worker and work–pool paradigms have been employed. One processing node is appointed as the master and all others are the workers. The master process is responsible for the input processing, computation task distribution, result generation and for workers coordination. Each worker repeatedly requests new jobs from the master, carries out the processing, and ultimately sends back the results to the master. Evidently, there is no direct communication among workers. Remaining tasks to be computed are kept in a work–pool, which is managed by the master process. Each time the master receives a result from a worker, it adds the computed images to the work–pool, until the maximum order of reflection has been reached. These images will be used as parent images for calculating higher order images. The computation is terminated when the work–pool is empty and when no pending tasks exist.

At this point it is noted that in order to perform vertical profiles generation and field calculations (as described in Section 3.2), a worker process requires information about the images computed in the path from the BS to the current parent image. This information is sent to the worker as input during task assignment. However, information about the previous images is not required for the production of a new subtree, except of course for the parent image in the current task.

Conceptually, dynamically created tasks can result in idle times, when one or more processing nodes are starved for tasks while waiting for another node to complete a computation that will produce additional tasks. This can be mitigated by giving priority to jobs that correspond to lower orders of reflection (tasks for computing first order images are dispatched before dispatching tasks for second order images, etc.), as well as utilizing moderately small task sizes, so that tasks in a given order of reflection n are computed before tasks in order n + 1 have been exhausted. As shown in Section 5, this approach proves efficient for various problem sizes, completely eliminating the potential bottlenecks described above.

Three different schemes have been considered for assigning tasks to the worker nodes. Sending many small messages can cause network latency to dominate communication overheads. Often it is more efficient to package small messages into a larger message, thus increasing the effective communication bandwidth. The different schemes that have been considered are [12,16,14]:

(a) fixed task-size,
(b) reducing task-size and
(c) variable task-size.

In the fixed task-size scheme, the master always assigns the same number F of parent images to a worker request. If \( T_{rem} \) is the remaining number of units to be computed, then the number of units assigned to a worker is \( A = \min(T_{rem}, F) \).

In the reducing task-size scheme, if \( n \) is the number of computation nodes, the master starts with assigning \( A = F \) computation units to the workers. When \( T_{rem} \) drops below \( nF \), the master assigns a single computation unit to each worker request, in order to minimize worker idle time and achieve better load balancing.

In the variable task-size, the master process assigns \( A = \lfloor T_{rem} R/n \rfloor \) units to the worker, where \( R \) is an adjustment factor (0 < \( R < 1 \)). In each case, the remaining number of computation units after an assignment is \( T_{rem} = T_{rem} - A \). The effects of the different task assignment schemes in the performance of the parallel implementation are studied in Section 5.

Algorithm 2 is the basic form of the parallel propagation algorithm.
Algorithm 2. Basic parallel propagation algorithm

1: load area database into main memory and read input file
2: if (master process) then
3: while (work–pool not empty OR tasks pending) do
4: receive message from worker; store results to file system;
5: if (max order of reflection not reached) then
6: add created images to the work–pool as parent images
7: end if
8: get next parent image(s) from work–pool, giving priority to lower order images
9: send message with input image(s) to worker
10: end while
11: else
12: while (not termination message received) do
13: receive assignment message with parent images \( X_i \) from master process
14: for (each \( X_i \)) do
15: for (each wall \( Y \) in the illumination zone of image \( X_i \)) do
16: generate images of \( X_i \) for the next order of reflection only
17: end for
18: end for
19: for (each point \( K \) in the point grid) do
20: (each image \( L \) in images generated previously in the task) do
21: create vertical profile between \( K \) and \( L \)
22: perform ray tracing and field calculations
23: end for
24: end for
25: send results to master process
26: end while
27: end if

4.3. Computation–communication overlapping

One of the most important issues in order to achieve efficiency and scalability is the reduction of communication overheads. A widely used and very efficient technique is overlapping computation and communication. The approach adopted in the parallel implementation presented herein is the utilization of two threads in each processor, as suggested in [12]: a computation thread and a communication thread. The computation thread is in charge of carrying out CPU intensive tasks such as computing the image tree and calculating rays at receiving points, while the communication thread handles requests for transferring data from/to other sites. Communication can be performed either by blocking or non-blocking calls to the message passing library, the latter being used in our parallel implementation. By using two threads in the master process, it follows that along with coordinating computations among the workers, the master node also performs useful ray tracing calculations, thus exploiting resources that would otherwise be idle.

4.4. Implementation

The prototype implementation of the sequential radio propagation model is based on C++. The MPICH implementation [21,22] of the MPI standard has been used for the development of the parallel propagation model. The development of the threaded functionality described in the previous section is based on the POSIX threads API [23].

5. Experimental results – evaluation

A number of experiments has been conducted in order to evaluate the proposed parallel propagation prediction model. Performance tests were performed on three 8-core SMP nodes with 2 GHz Intel Xeon CPUs and 8 GB of RAM, connected by 1 Gb Ethernet and operating on Debian Linux. In order to have a homogeneous communication environment, one of the nodes is utilized only for the execution of the master process (albeit only one CPU core in the node is used) and worker processes are executed on the two other nodes. Consequently, experiments have been conducted while using up to 17 processes. Additionally, in order to ensure that each process and the corresponding threads run constantly and exclusively on the same core, the cpu-affinity-mask based scheduling feature of Linux has been exploited.

Three area databases with different sizes have been used in the experiments, which are parts of the wider area of Cambridge, UK [2] (Fig. 4):
Experiments have been conducted for all three databases with up to three orders of reflection. Furthermore, experiments have been conducted for database B with up to 3, 4 and 5 orders. The problem set-up in all cases is shown in Table 1. In order to perform the first order image generation in parallel, as described in Section 4.1, database A has been partitioned in sub-regions with a size of 50 m \times 50 m. Due to the fact that buildings on the bounds of a subregion cannot be segmented, the size of the subregions is not exactly 50 m \times 50 m and the database is partitioned in 98 subregions in total. Databases B and C have been divided in 100 m \times 100 m subregions (a total of 116 subregions for database B and 255 subregions for database C). The selection of the partitioning size is made so that during computations of first order images, enough tasks are available for all processes and the different assignment schemes.

In order to evaluate the performance of the proposed parallel prediction model, we use the metrics of (a) speedup, (b) efficiency, (c) resource utilization and (d) workload expansion ratio. Assuming that $T_{seq}$ is the computation time when using only one CPU, $t_i$ is the finish time for the $i$th CPU, $n$ is the number of CPUs employed, $T_{max,n}$ and $T_{avg,n}$ are the maximum and average finish times among the $n$ CPUs, while $T_{sum,n}$ is the sum of finish times for all CPUs, then:

$$T_{max,n} = \max_{i=1}^{n} t_i,$$

$$T_{sum,n} = \sum_{i=1}^{n} t_i \text{ and}$$

$$T_{avg,n} = \frac{\sum_{i=1}^{n} t_i}{n}.$$

### Table 1
Set-up of experiments.

<table>
<thead>
<tr>
<th>BS antenna location</th>
<th>(1870, 5250)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS antenna height</td>
<td>15 m</td>
</tr>
<tr>
<td>Frequency</td>
<td>2.5 GHz</td>
</tr>
<tr>
<td>MS antenna height</td>
<td>1.5 m</td>
</tr>
<tr>
<td>BS antenna polarization</td>
<td>Vertical</td>
</tr>
<tr>
<td>MS antenna polarization</td>
<td>Vertical</td>
</tr>
<tr>
<td>Transmitted power</td>
<td>30 dBm</td>
</tr>
<tr>
<td>Resolution of grid analysis</td>
<td>15 m</td>
</tr>
</tbody>
</table>
The speedup $S_n$, the efficiency $E_n$, the workload expansion ratio $W_n$ and the resource utilization $U_n$ can be computed as:

$$S_n = \frac{T_{seq}}{T_{max,n}} = \frac{T_{seq}}{\max_{i=1}^{n} t_i},$$

$$E_n = \frac{S_n}{n} = \frac{T_{seq}}{\left( \sum_{i=1}^{n} t_i \right)},$$

$$W_n = \frac{T_{sum,n}}{T_{seq}} = \sum_{i=1}^{n} t_i / T_{seq},$$

and,

$$U_n = \frac{T_{avg,n}}{T_{max,n}} = \sum_{i=1}^{n} t_i / (nT_{max,n}).$$

5.1. Implementation evaluation

Tables 2–4 show detailed results for databases A, B and C, respectively, when utilizing the fixed task-size assignment scheme with $F = 1$. The columns “# of subregions – time (s)”, “# of 1st order images – time (s)” and “# of 2nd order images – time (s)” refer to the number of tasks handled by each worker in each computation phase and the duration between the start of the first task and the finish of the last task handled by each process in each corresponding phase. The total number of images computed in the 3rd order of reflection is 6934, 85,486 and 634,648 for databases A, B and C, respectively.

In the case of database A, the sequential part of loading the database into the main memory (approximately 1 s) is comparable to ray tracing computations and thus the achieved speedup is constrained by it. The column “ideal speedup” in Table 2 refers to the maximum possible speedup taking into account this sequential part. In the case of databases B and C, this sequential part is negligible and thus it is not considered. For brevity, results for 1–6 processes are presented in these tables.

Execution statistics for the case of 17 processes are depicted in Figs. 5–10 for all the databases. These graphs show the number of tasks handled by each process in each computation phase and the corresponding duration, as in Tables 2–4.

A first observation regarding the previous tables and graphs is that processing times for different calculations even in the same order of reflection are quite different for all databases. For instance, in database C, the maximum computation time for processing subregions is 9.66 s, the minimum computation time is 0.38 s and the average time is 0.76 s. The corresponding times for processing 1st order images are 29.17 s, 0.37 s and 0.46 s and for 2nd order images 10.61 s, 0.37 s and 0.43 s. However, the dynamic scheduling of tasks managed by the master–worker pattern achieves efficient distribution of workload in all phases of the computation, as the differences in finish times in each phase among processes are very small. Most importantly, idle times for the processes are small in all cases, indicating that the overall computational load has been well balanced among them. For database C and 6 processes the maximum idle time is 0.50 s, while for 17 processes the maximum idle time is 1.20 s. Although each task assigned to a worker includes ray calculations for the whole database, tasks
are still fine-grained enough so as to enable efficient load balancing. The same conclusions can be drawn for the cases of databases A and B.

Fig. 11 depicts the speedups achieved for all three databases for three orders of reflection and for a varying number of processes. For databases B and C, the speedup is almost linear with the number of utilized processes. In the case of database A, the achieved speedup is very close to the theoretical maximum pointed out by Amdahl’s law. The corresponding efficiency graph is shown in Fig. 12. The speedups and efficiency achieved for database B and different orders of reflection (3, 4 and 5) are shown in Figs. 13 and 14, respectively. Speedup in these cases is again very close to linear and gets marginally better as the order of reflection (and thus workload) is increased. The workload expansion ratios and resource utilization for the three databases and three orders of reflection are depicted in Figs. 15 and 16 respectively. The workload expansion ratio is a metric that indicates the amount of extra computation time that the parallel execution adds to the serial execution. For databases B and C, values are close to unity for any number of utilized processes. However, in the case of database A there is a noticeable

### Table 3
Statistics for database B (1000 m × 1000 m).

<table>
<thead>
<tr>
<th># of processes</th>
<th>pid</th>
<th># of subregions – time (s)</th>
<th># of 1st order images – time (s)</th>
<th># of 2nd order images – time (s)</th>
<th>Total time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>116 – 25.20</td>
<td>2930 – 315.73</td>
<td>20328 – 1980.38</td>
<td>2322.36</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>54</td>
<td>12.61</td>
<td>1489 – 158.44</td>
<td>10207 – 998.38</td>
<td>1170.61</td>
<td>1.98</td>
</tr>
<tr>
<td>2</td>
<td>62</td>
<td>12.59</td>
<td>1441 – 158.92</td>
<td>10121 – 997.85</td>
<td>1170.50</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>28 – 8.25</td>
<td>960 – 105.38</td>
<td>6840 – 667.84</td>
<td>782.73</td>
<td>2.97</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>8.09</td>
<td>985 – 105.64</td>
<td>6771 – 667.73</td>
<td>782.64</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>8.86</td>
<td>985 – 107.14</td>
<td>6717 – 665.50</td>
<td>782.61</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4
Statistics for database C (1500 m × 1500 m).

<table>
<thead>
<tr>
<th># of processes</th>
<th>pid</th>
<th># of subregions – time (s)</th>
<th># of 1st order images – time (s)</th>
<th># of 2nd order images – time (s)</th>
<th>Total time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>116 – 25.20</td>
<td>2930 – 315.73</td>
<td>20328 – 1980.38</td>
<td>2322.36</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>54</td>
<td>12.61</td>
<td>1489 – 158.44</td>
<td>10207 – 998.38</td>
<td>1170.61</td>
<td>1.98</td>
</tr>
<tr>
<td>2</td>
<td>62</td>
<td>12.59</td>
<td>1441 – 158.92</td>
<td>10121 – 997.85</td>
<td>1170.50</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>28 – 8.25</td>
<td>960 – 105.38</td>
<td>6840 – 667.84</td>
<td>782.73</td>
<td>2.97</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>8.09</td>
<td>985 – 105.64</td>
<td>6771 – 667.73</td>
<td>782.64</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>8.86</td>
<td>985 – 107.14</td>
<td>6717 – 665.50</td>
<td>782.61</td>
<td></td>
</tr>
</tbody>
</table>
increase in its value while the number of processes increases. This is mainly due to the fact that the time for loading the database is comparable to the overall computation time, thus the additional database loading performed by each additional process creates a non-trivial overhead. As the workload expansion ratio is related to the complexity of the computation, it is expected that more complex computations will demonstrate lower workload expansion ratios. Resource utilization is in all cases very close to unity, indicating that all processes spend little time in idle status.
5.2. Comparison of task assignment schemes

The experiments described in the previous section have been repeated for the three different task assignment schemes described in Section 4.2. More specifically, the values of $F$ for the fixed task-size and the reducing task-size range between $1$ and $5$ ($F_1$–$F_5$ and $R_2$–$R_5$) and the values of $R$ for the variable task-size range between $0.05$ and $0.3$ ($V_0.05$–$V_0.3$) with a step...
The effects of the task assignment schemes seem to be different for different numbers of utilized processes. Tables 5–7 show the best and worst task assignment schemes for each database, for 17, 10 and 3 processes, respectively.

A first observation is that the differences between the best and worst assignment schemes are not very profound, especially for a small number of processes. Even in the case of 17 processes, the maximum improvement in speedup between the

Fig. 11. Speedup for databases A, B and C and three orders of reflection.

Fig. 12. Efficiency for databases A, B and C and three orders of reflection.

Fig. 13. Speedup for 3, 4 and 5 orders of reflection for database B.

size of 0.05. The effects of the task assignment schemes seem to be different for different numbers of utilized processes. Tables 5–7 show the best and worst task assignment schemes for each database, for 17, 10 and 3 processes, respectively.

A first observation is that the differences between the best and worst assignment schemes are not very profound, especially for a small number of processes. Even in the case of 17 processes, the maximum improvement in speedup between the
worst and best task assignment schemes is only approximately 1% for database A. However, these differences present a tendency to become more visible as the number of processes increases and are likely to become considerable for larger numbers of processes. Another observation is that certain schemes seem to be more efficient for specific sizes of the parallel system. As shown in Table 7, the variable task-size scheme with a relatively large value of $R$ has reduced performance compared to
the other schemes. On the other side, the variable task-size scheme performs better for larger number of processes, while fixed task-size schemes are not as efficient, as shown in Table 5.

6. Conclusions and future work

In this paper, a parallel implementation for radio-wave propagation prediction using an image-based ray tracing technique has been presented. The proposed parallelization scheme consists in the partitioning of the image tree generation process; the image tree is built in a gradual and concurrent manner using the tree nodes computed in previous steps as input and computing images in the next order of reflection for the whole database, while ray calculations at the receiver points are also computed in the same task. The implementation handles the distribution of dynamically created tasks by utilizing the master–worker and work–pool paradigms. Certain issues have been considered for maximizing efficiency, such as computation–communication overlapping and task priorities. In order to enable portability, the development of the algorithm has been based on the MPI standard. Several experiments have been conducted for different areas and varying orders of reflection. Metrics such as speedup, resource utilization and workload expansion ratio have been employed for evaluating the performance of the algorithm. Results indicate that the implementation is scalable and efficient for moderately large databases and for increasing orders of reflection and thus can provide timely information for the detailed description of the wireless channel, which is critical for the design of LTE systems. Additionally, the performance of different task assignment schemes proposed in the literature has been studied. Results show that the difference in performance is not profound, however it seems to become more evident as the number of utilized processes increases.

Work currently under way considers exploiting the parallel implementation in order to conduct more detailed ray tracing studies, which would be difficult to perform otherwise due to overwhelming processing times (e.g. study the effect of taking into consideration higher orders of reflection). Furthermore, future work will consider the integration of the parallel implementation with a grid-enabled simulation tool [24], towards creating an environment for the collaborative simulation and optimization of wireless networks.

### Table 5
Best and worst task assignment schemes for 17 processes.

<table>
<thead>
<tr>
<th>Database</th>
<th>Best</th>
<th>$T_{\text{max}}$</th>
<th>$T_{\text{sum}}$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>V0.3</td>
<td>6.36</td>
<td>107.39</td>
<td>14.01</td>
</tr>
<tr>
<td>B</td>
<td>V0.3</td>
<td>140.33</td>
<td>2375.97</td>
<td>16.35</td>
</tr>
<tr>
<td>C</td>
<td>V0.1</td>
<td>2479.61</td>
<td>42129.4</td>
<td>16.80</td>
</tr>
<tr>
<td>Database</td>
<td>Worst</td>
<td>$T_{\text{max}}$</td>
<td>$T_{\text{sum}}$</td>
<td>Speedup</td>
</tr>
<tr>
<td>A</td>
<td>F2</td>
<td>6.57</td>
<td>110.06</td>
<td>13.85</td>
</tr>
<tr>
<td>B</td>
<td>F4</td>
<td>141.35</td>
<td>2381.23</td>
<td>16.43</td>
</tr>
<tr>
<td>C</td>
<td>F4</td>
<td>2490.25</td>
<td>42158.33</td>
<td>16.73</td>
</tr>
</tbody>
</table>

### Table 6
Best and worst task assignment schemes for 10 processes.

<table>
<thead>
<tr>
<th>Database</th>
<th>Best</th>
<th>$T_{\text{max}}$</th>
<th>$T_{\text{sum}}$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>V0.25</td>
<td>10.02</td>
<td>100.11</td>
<td>8.89</td>
</tr>
<tr>
<td>B</td>
<td>R5</td>
<td>236.61</td>
<td>2363.03</td>
<td>9.82</td>
</tr>
<tr>
<td>C</td>
<td>R2</td>
<td>4209.80</td>
<td>42094.39</td>
<td>9.90</td>
</tr>
<tr>
<td>Database</td>
<td>Worst</td>
<td>$T_{\text{max}}$</td>
<td>$T_{\text{sum}}$</td>
<td>Speedup</td>
</tr>
<tr>
<td>A</td>
<td>F3</td>
<td>10.16</td>
<td>100.79</td>
<td>8.77</td>
</tr>
<tr>
<td>B</td>
<td>R3</td>
<td>236.92</td>
<td>2364.26</td>
<td>9.8</td>
</tr>
<tr>
<td>C</td>
<td>V0.3</td>
<td>4224.95</td>
<td>42242.94</td>
<td>9.86</td>
</tr>
</tbody>
</table>

### Table 7
Best and worst task assignment schemes for three processes.

<table>
<thead>
<tr>
<th>Database</th>
<th>Best</th>
<th>$T_{\text{max}}$</th>
<th>$T_{\text{sum}}$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>V0.15</td>
<td>30.71</td>
<td>92.12</td>
<td>2.90</td>
</tr>
<tr>
<td>B</td>
<td>R4</td>
<td>781.68</td>
<td>2344.81</td>
<td>2.97</td>
</tr>
<tr>
<td>C</td>
<td>F4</td>
<td>13983.25</td>
<td>41948.65</td>
<td>2.98</td>
</tr>
<tr>
<td>Database</td>
<td>Worst</td>
<td>$T_{\text{max}}$</td>
<td>$T_{\text{sum}}$</td>
<td>Speedup</td>
</tr>
<tr>
<td>A</td>
<td>F1</td>
<td>30.88</td>
<td>92.64</td>
<td>2.88</td>
</tr>
<tr>
<td>B</td>
<td>V0.3</td>
<td>784.22</td>
<td>2352.48</td>
<td>2.96</td>
</tr>
<tr>
<td>C</td>
<td>V0.3</td>
<td>14098.45</td>
<td>42294.89</td>
<td>2.96</td>
</tr>
</tbody>
</table>
Acknowledgements

The work presented in this paper is co-financed by E.U.-European Social Fund (80%) and the Greek Ministry of Development-GSRT (20%). Authors wish to acknowledge the Computer Systems Laboratory (CSLab) of the National Technical University of Athens for providing the computational resources that were used for performing the experiments presented in this paper.

References