The Development of a Parallel Ray Launching Algorithm for Wireless Network Planning

Zhihua Lai, University of Bedfordshire, UK

Nik Bessis, University of Bedfordshire, UK

Guillaume De La Roche, University of Bedfordshire, UK

Pierre Kuonen, University of Applied Science of Western Switzerland, Switzerland

Jie Zhang, University of Bedfordshire, UK

Gordon Clapworthy, University of Bedfordshire, UK

ABSTRACT

Propagation modeling has attracted much interest because it plays an important role in wireless network planning and optimization. Deterministic approaches such as ray tracing and ray launching have been investigated, however, due to the running time constraint, these approaches are still not widely used. In previous work, an intelligent ray launching algorithm, namely IRLA, has been proposed. The IRLA has proven to be a fast and accurate algorithm and adapts to wireless network planning well. This article focuses on the development of a parallel ray launching algorithm based on the IRLA. Simulations are implemented, and evaluated performance shows that the parallelization greatly shortens the running time. The COST231 Munich scenario is adopted to verify algorithm behavior in real world environments, and observed results show a 5 times increased speedup upon a 16-processor cluster. In addition, the parallelization algorithm can be easily extended to larger scenarios with sufficient physical resources.

Keywords: *Intelligent Ray Launching, Network Planning, Parallelization, Propagation Prediction, Simulation*

INTRODUCTION

Propagation modeling serves as a fundamental input in the wireless network planning and optimization process. Especially, in order to determine the interferences for an indoor femtocell base station with the outdoor macrocell, accurate coverage predictions have to be obtained via propagation modeling (Zhang & De La Roche, 2010). Planning and optimization of a wireless network usually requires simulation of hundreds of User Equipments (UE) and the path loss between these UEs and base stations

DOI: 10.4018/jdst.2011040101

are obligatory to investigate the best servers and handovers etc.

Current propagation models can be divided into three categories: empirical models, semi- deterministic and deterministic models. Empirical models are the simplest models; which are usually based on simple factors such as the carrier frequency and distance. They are extremely fast because of statistical model environmental factors. The semi-deterministic models are enhanced by introducing relevant deterministic factors in the computation. Such models provide higher accuracy than empirical models, thus running time of semi-deterministic models is usually realistically acceptable upon conventional computing power, such as PCs.

The deterministic models consider environmental factors, e.g., buildings and walls, which are time-consuming compared to empirical and semi-deterministic models. However, the deterministic models provide the highest accuracy out of these categories.

Ray-based methods belong to deterministic models and they are based on geometry path finding algorithms (Haslett, 2008). Ray-based methods in general are divided into two subcategories: ray tracing and ray launching. Ray tracing adopts a backward path search technique, which guarantees that exact paths between transmitters and receivers can be computed (Glassner, 1989). Ray tracing offers high accuracy but it is extremely time consuming. The complexity grows exponentially with the number of objects and the maximum ray iterations (Nagy, Dady, & Farkasvolgyi, 2009). Ray tracing is used for precise point-to-point predictions. Several acceleration techniques such as pre-processing (Wolfle, Gschwendtner, & Landstorfer, 1997) or the use of a General Purpose Graphic Processing Unit (GPGPU) (Rick & Mathar, 2007) have been proposed. The performance of ray tracing is usually limited by the inherent complex ray-object intersection tests and many techniques have been proposed over the past years to speed up computation (Degli-Esposti, Fuschini, Vitucci, & Falciasecca, 2009). Ray launching emits the rays from sources; which are separated by a small angle. This method is efficient in an area prediction because the rays are actively followed. However, this approach leads to two inherent problems. The first problem is angular dispersion of ray launching. The distant pixels are less likely to be visited by rays because rays disperse as they are propagated. For example, a distant small object may be missed by rays because a fixed angle is used to separate rays. Secondly, the ray double counting arises when a sample pixel is marked twice by the same rays, which should be avoided because it reduces the accuracy of ray launching. Ray launching is usually faster than ray tracing with less accuracy. The complexity of ray launching grows linearly with the number of objects and maximum ray iteration (Nagy et al., 2009).

In (Lai, Bessis, De La Roche, Song, Zhang, & Clapworthy, 2009), a new model based on discrete ray launching, namely the Intelligent Ray Launching Algorithm (IRLA), has been proposed to obtain fast propagation prediction (path loss and multipath components) within a realistic time scale. In (Lai et al., 2010), the authors extended this model to indoor prediction, which accurately predicts the multipath propagation in indoor environment. The IRLA model has been validated with measurement campaigns (Lai et al., 2010), which has led to the effective development for network applications. In (Lai et al., 2009), the authors proposed an efficient method to improve the accuracy of IRLA by solving angular dispersion problem of ray launching. This method has effectively improved the accuracy and avoids ray double counting. In (Lai et al., 2009), a parallel algorithm of IRLA is implemented based on a toolkit named Parallel Object-oriented Programming in C++ (POP-C++). Preliminary promising results have been presented, which show that parallel IRLA has improved the performance. This article is an extension of this work: issues related to performance and accuracy will be further addressed in this work. This article contributes to present a parallel propagation algorithm that accelerates the time-consuming prediction. The components of the IRLA model are analyzed so that the most time-consuming

components are parallelized. Results show that with 16 processors, the performance can reach up to 5 for certain scenarios.

The rest of this article is organized as follows. At First, the IRLA model is briefly introduced. Secondly, the complexity of IRLA is studied, which serves as the fundamentals to develop the efficient parallel IRLA model. Then, the issues related to parallelization are detailed, which is followed by results that conclude this work.

THE RAY LAUNCHING MODEL: IRLA

IRLA is a discrete ray launching model that aims to provide highly improved prediction in terms of path loss and multipath components for wireless propagation prediction within a short time. In outdoor urban scenarios, a specific procedure has been developed to accelerate the computations of urban rooftop diffractions. IRLA can be easily extended to indoor, indoor-to-outdoor and outdoor-to-indoor scenarios due to the well designed mechanisms to avoid duplication of rays and angular dispersion (Lai et al., 2009). IRLA is based on discrete cubic data set, which can be extracted from vector building data. Typically, building data for outdoor scenarios are simplified to 2.5-D which are described as polygon-shaped buildings with height information. For outdoor scenarios, the IRLA separates roof-top diffractions from horizontal diffractions and reflections. The algorithm quickly checks the number of roof-top diffractions required between the transmitter and receiver. The components of IRLA and their relationship are depicted in Figure 1. Given the input data (building, antenna, and network configuration), the discrete data set is built, based on which Line-of-Sight (LOS) component obtains secondary pixels for reflections and diffractions.

Horizontal-Reflection-Diffraction (HRD) and Vertical- Diffraction (VD) are independent of each other and thus can be run in parallel. When these two components are completed, a post-processing procedure is carried out (such as antenna pattern adjustment and indoor coverage prediction) and final outputs include path loss and multipath components.

Computational Complexity

The discrete data set size is (N_x, N_y, N_z), which represents the number of cubes for X, Y, and Z dimensions respectively. The numbers of building cubes are known as N_{ground}, N_{wall} and N_{roof}, which represent the number of building ground, walls and roofs respectively. Therefore the total number of representing buildings can be denoted as:

$$N_{\text{buildings}} = N_{\text{ground}} \cup N_{\text{wall}} \cup N_{\text{roof}}$$

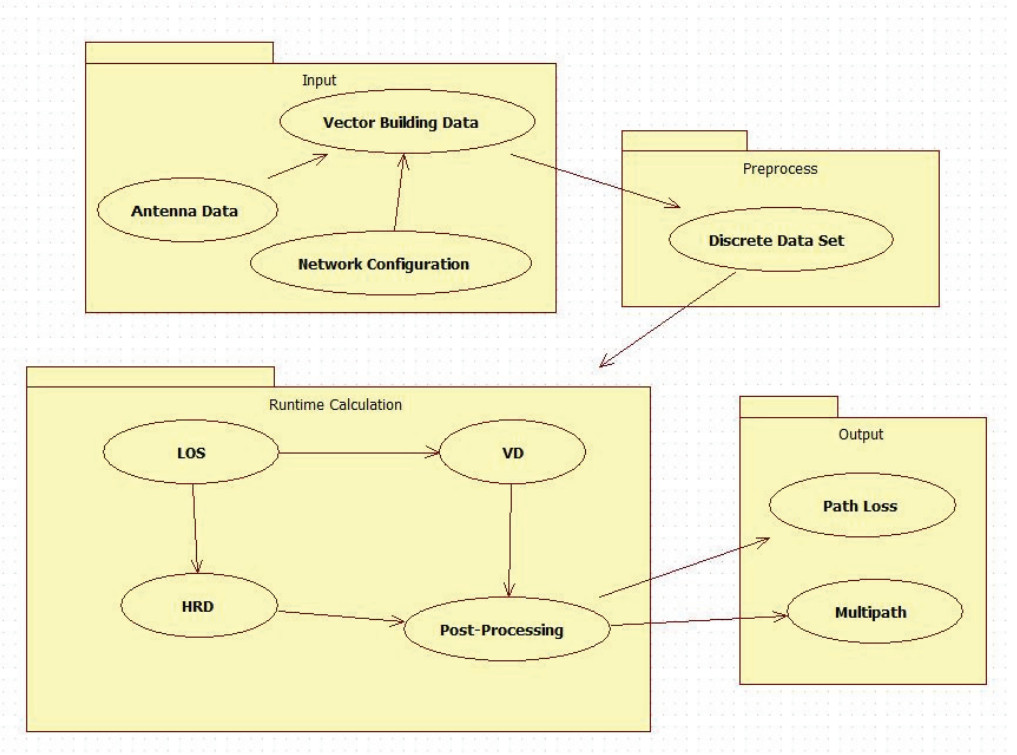
For example, there are cubes; which are joint edges of walls and roofs. $N_{buildings}$ depends on the size of the scenario, the number of buildings and the resolution used for building the discrete data. $N_{buildings}$ usually impacts on the computation complexity. For example, greater $N_{buildings}$ causes larger computational complexity and vice versa.

The complexity of IRLA thus can be modeled by five parts: C_{pre}, C_{post}, C_{los}, C_{vd} and C_{hrd}, which represent the computation complexity for pre-calculation, post-processing, component LOS, VD and HRD respectively. Let C be the total complexity of IRLA, then it is can be obtained as following:

$$C = C_{\text{los}} + C_{\text{vd}} + C_{\text{hrd}} + C_{\text{pre}} + C_{\text{post}}$$

C_{los} can be approximated based on the number of cubes on the fringe of scenario. The process of IRLA prediction starts with launching rays in all 3-D directions. Based on the discrete data set, the resolution and the number of cubes along each dimension (X, Y and Z) are known. Therefore the number of discrete rays required can be obtained by connecting the transmitter to all the cubes at the fringe of the scenarios (Lai et al., 2009), which is:

Figure 1. Structures of the ray launching model



$$N_{\text{fringe}} = 2N_x N_y + 2(N_z - 2)(N_x + N_y - 2)$$

where:

N is the number of discrete rays.

N_x, N_y and N_z are the number of cubes in dimension X, Y and Z respectively.

This formula ensures no pixels are missing due to angular dispersion of ray launching (Lai et al., 2009) from component LOS. The use of such ray launching mechanism is useful in distribution of rays. N_{fringe} is the number of discrete rays launched by LOS. Suppose the transmitter is placed in cubic position (T_x, T_y, T_z), the distance function $D(x_1, y_1, z_1, x_2, y_2, z_2)$ acknowledges for the number of cubes that have to be checked on a particular discrete ray starting from (x_1, y_1, z_1) and the ending at position (x_2, y_2, z_2) (one of the fringe cubes). The maximum value of D is obtained if there is no obstacle found along the discrete ray being checked. In this case, D can be calculated as:

$$D(x_1, y_1, z_1, x_2, y_2, z_2) = \max(|x_1 - x_2|, |y_1 - y_2|, |z_1 - z_2|)$$

The worst case for LOS occurs if it is an empty scenario (free space). Every single cube has to be checked. In this case, $N_{buildings} = 0$, C_{los} can be roughly approximated to:

$$C_{\text{los}} = \sum_{i=1}^{N_{\text{fringe}}} D(T_x, T_y, T_z, x_i, y_i, z_i)$$

where:

x_i, y_i, z_i represents the cube coordinates of fringe at index i.
N_{los} denotes the number of secondary cubes obtained via checking cubes on discrete rays if there are obstacles.

$$N_{\text{los}} = \sum_{i=1}^{N_{\text{fringe}}} H(T_x, T_y, T_z, x_i, y_i, z_i)$$

where:

x_i, y_i, and z_i represents the cube coordinates of fringe at index i.

$$H(T_x, T_y, T_z, x_i, x_y, z_i) = \begin{cases} 1 \text{ discrete ray } (T_x,\ T_y,\ T_z) \rightarrow \\ \quad (x_i,\ y_i,\ z_i) \text{ is blocked by obstacles} \\ 0 \text{ otherwise} \end{cases}$$

For indoor scenarios, $C_{vd} = 0$ because component VD (for rooftop diffractions) is not activated. For outdoor scenarios, N_{vd} represents the number of cubes that are checked by VD and can be approximated as:

$$N_{vd} = \sum_{i=1}^{2(Nx+Ny-2)} D(T_x, T_y, T_z, x_i, x_y, z_i) N_z$$

where:

D is assumed to reach its maximum value (no obstacles along the discrete ray).

For each cube in N_{vd}, a discrete scan-line is launched from the transmitter. The building blocks between these two cubes are checked. C_{vd} can thus be approximated by:

$$C_{vd} = \sum_{i=1}^{Nvd} C_{vd\ -\ scan}(T_x, T_y, T_z, x_i, y_i, z_i)$$

where:

x_i, y_i, z_i represents the cube coordinates at index i being checked.

The procedure $C_{vd\text{-}scan}$ is to check the number of rooftop diffractions. In the worst case, each scan-line involves multiple visibility checks between two building blocks which are costly. In this case, the computation complexity can be approximated by counting the number of checks and their corresponding ray lengths.

$$C_{\text{vd-scan}} \simeq \sum_{i=1}^{N_{\text{checks}}} L_i$$

where:

N_{checks} represents the number of visibility checks.

L_i represents length (the number of cubes) on discrete ray segment i.

However, due to caching techniques and the intelligence of using geometry to avoid possible checks, $C_{vd\text{-}scan}$ can be often be reduced to the complexity of constant $O(1)$.

IRLA incorporates the engine HRD to virtually launch and follow discrete rays. The number of rays is denoted as N_{los}, which is obtained from the LOS component. Depending on the complexity of scenario, current signal strength carried by discrete rays, the threshold and the number of ray iterations, the complexity varies from constant to exponentials i.e. the ray generates many secondary diffraction rays or a reflection ray. This can be greatly accelerated by the intelligent marking scheme; which avoids double marking and angular dispersion. C_{hrd} can be approximated to.

$$C_{\text{hrd}} = \sum_{i=1}^{N_{\text{ros}}} C_{\text{hrd-ray}}(i)$$

where:

$C_{hrd\text{-}ray}(i)$ returns the computational complexity of discrete ray i.

Pre-calculation C_{pre} and post-processing C_{post} usually involve operations on the entire discrete data set. In this case, C_{pre} and C_{post} can be approximated to $N_x N_y N_z$. The complexity of C is calculated based on one transmitter. Given n transmitters, the complexity can be sum to $\sum_{i=1}^{n} Ci$.

Parallelization

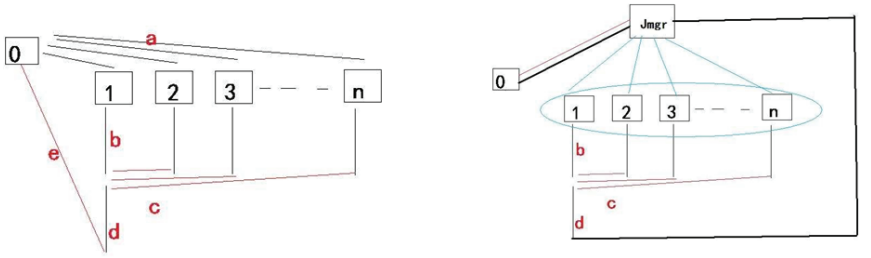
The components prototype of the IRLA model has been depicted in Figure 1. The HRD and VD components are dependent on the discrete data set but both can be executed in parallel. The outputs of these two components are merged and a post-processing procedure is carried. Since these two components are most time-consuming out of all other IRLA components, parallelization via splitting data or instructions has to be performed so that overall speed up can be observed. From the micro aspects of the view, parallelization can be possible even within components, e.g., HRD can easily be parallelized by distributing the rays among processors. These two possibilities offer speed up in the following two manners:

Single-Instruction-Multiple-Data (SIMD) (Silberschatz & Galvin, 2006): From a micro aspect, computation-intense components can be parallelized via splitting the data. Each individual processor shares the same instructions but performs calculations on different portions of data (e.g. different rays). This can be efficiently and advantageously applied to components that are easily- parallelizable. For example, the inverse operation of an image can be parallelized by cutting images into pieces that are sent to parallel processors. The IRLA model contains such similar components. For example, the calculation of HRD can be narrowed down to trace each discrete ray that can be treated in parallel. However, this approach requires different specific treatment for different components (i.e. parallelization implementation is different). A significant parallelization speedup is often gained when this approach is employed on data-intense components. In most of the cases, the data split causes the problem of simultaneously accessing the same piece of information by parallel objects/threads. Therefore, the success of this approach depends on the implementation of locks to critical sessions (i.e. a lock prevents other parallel objects/threads accessing important/critical information).

Multiple-Instruction-Multiple-Data (MIMD) (Bisseling, 2004): From a macro aspect, different components can be scheduled on different processors, e.g., one or more processors handle HRD while at the same time the others handle VD. If two or more components are independent from each other, this approach introduces a light-weight (as compared to SIMD approach) parallelization technique. Independent models can be scheduled to different processors for computation simultaneously. However, if the running time from these models is largely different, some processors will be kept idle because usually a barrier is used. This can be avoided by continuous data/instructions fetch from a central node (for example, job manager or resource scheduler in distributed grid environment). However, this will increase the complexity and may increase the need of communication overhead.

Parallelization can be combined by both SIMD and MIMD approaches, which introduces a two-level parallelization scheme. For example,

Figure 2. Parallel IRLA with & without job manager



(a) Parallel IRLA without Job Manager

(b) Parallel IRLA with Job Manager

some faster processors target more data-intense components and the rest are handled by slower processors (MIMD), thus processors are virtually grouped into two. Inside each group, the second level of parallelization (SIMD) can be applied. Finally, the results from both groups are merged. This is advantageous because it is more grid like and can be easily/slightly modified to suit a distributed grid environment.

Figures $2_{(a)}$ and $2_{(b)}$ display the overall parallel model of IRLA with and without a job manager respectively. A job manager is a scheduler that is responsible for deploying computation to available work nodes. If no job manager is used, worker nodes have to be manually given in the first place. This scheme is usually used within a cluster; which is locally limited and not flexible to extend. Without the central control of the job manager, the communication between user's node (N_0) and work nodes (from N_1 to N_n) are visible. In Figure 2(a), stage a represents the messages sent from user's node to work nodes. *b* corresponds to the stage where work nodes carry the parallel computation. *c* corresponds to the stage where all work nodes are stopped by a barrier. *d* corresponds to the stage where results are collected from work nodes and merged. Finally, at stage *e*, the results are sent to user's node. By contrast, if a job manager is used, N_0 is only visible to the job manager. In Figure 2(b), stage *b*, *c* and *d* are the similar to the stages in Figure 2(a) except that the results are sent to job manager instead of user's node. This scheme is often used in scalable and distributed grid environment (Foster & Kesselman, 2003) where the number of work nodes can be easily extended.

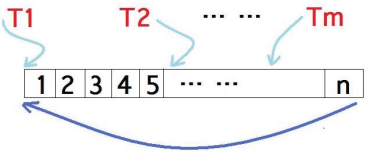
Multithreading

In general, more threads increase the probability of resource competition. But this can be reduced by proper assignment of parallel sub-tasks. For example, the total number of tasks for VD and HRD can be determined before-hand. Each thread obtains a piece of the computation task. In order to reduce the conflict, threads handle pieces of rays that are far located, i.e., discrete rays are separated (greater than a resolution pixel) and unlikely to conflict with each other.

The computers have been equipped with multi-cores technology; which shares the memory via a high-speed system bus (Silberschatz & Galvin, 2006). This enables efficient message exchange between threads. The static data distribution scheme for threads can be described as follows.

Given the total number of jobs (e.g. discrete rays) n (N_1 to N_n), and the number of threads to be used is represented by T (T_1 to T_t). Assume each thread obtains approximately equal size of jobs, the size of jobs can be calculated by $J = N / P$. Assume adjacent jobs (N_i and N_{i+1}) represent adjacent rays. Define indices $j = (i - 1)J + 1$ and $k = j + 1$. Hence, each thread T_i obtains an array of jobs from N_j to N_k. This approach is easy to implement but has the disad-

Figure 3. Dynamic data distribution of multithreading



vantage of keeping threads idle due to unequal computation time. For example, some threads may finish computations early and they have to be kept waiting until the rest of the threads have finished. In order to maximize CPU computation usage, more threads have to be created. However, this will lead to the increase of resource competition among threads and will possibly slow down computation. To solve this, a flexible and dynamic data distribution method is proposed, which eliminates the problem and is far more efficient.

Like static data distribution scheme, threads are assigned with a start index and the number of jobs to compute based on the total job number and the number of threads. However, the total number of jobs for each thread is not fixed in the dynamic distribution scheme. Threads continuously fetch next available job index until all computation jobs have been computed. The total job indices are treated as a virtual circular queue, as displayed in Figure 3. In order to reduce the possibility of resource competition from threads, continuous blocks of job indices are assigned to threads. Since the memory is effectively accessed by threads, synchronization techniques such as semaphores (Silberschatz & Galvin, 2006) are employed. Threads are computing simultaneously and when each job index is finished, a pointer indicating next job for each thread is incrementing. The current job index is checked if being locked by other threads and if it has been computed. In this case, each thread will not be kept waiting unless there are no more jobs. It was verified by experiments that (Tabel 1, using 3 threads on T9300, 4GB RAM), on average, this parallelization scheme yields from 140% to 160% speedup over static data distribution scheme depending on the scenarios. The number of threads that is considered optimal in practice can be set to the number of physical cores because nearly all the time all the threads are active, which can be mapped to each core.

POP-C++

Parallel Object-oriented Programming in C++ (POP-C++) is a parallel-object oriented programming language in C++ (Nguyen, 2004). POPC++ is an extension of C++ which makes it easy to program parallel applications. It eliminates the need to explicitly invoke and handle message-passing between distributed nodes by introducing a parallel object model. All communication is handled via implicit object calls; which makes it efficient and flexible. Parallel objects represented in POP-C++ (Nguyen & Kuonen, 2007) are logical independent but can be geographically distributed. This provides parallelism via asynchronous methods invocation (asynchronous methods return immediately upon invocation).

Objects created by the POP-C++ runtime system carry the computation in parallel. There are two major schemes.

The first scheme is to create a central node (manager); which is responsible for splitting the data/instructions to available nodes and wait for the returned results. This can be considered as a flexible master-worker scheme where the master node is in control of job splitting, scheduling and data merging. This scheme leads to a large amount of communication because message-passing to send and receive results between master and worker nodes have to be considered. However, data splitting is dynamically accomplished at runtime, which is efficient because worker nodes following send/receive principle can largely avoid the idle processors.

The second scheme eliminates the requirement for a central control. At the first stage, computation tasks are divided according to available nodes' capability (power, memory etc). Each node has been assigned for a piece of work. The nodes start computation. They send back results to the assigned node once the computation finishes. This scheme has a low communication overhead (there is no message-passing between work nodes). However, the parallel efficiency (resource utilization) largely depends on the static data distribution scheme. If faster nodes do not have a larger piece of a computation job, they idle and efficiency is compromised. Assume there are N nodes available during runtime and their performance indices P are known and calculated based on the CPU speed, memory and etc.

P can thus be define as:

$$P_i = uM_i + vC_i$$

where i is the index for nodes, u, v are the weighting for the scores of memory M and CPU speed C, respectively. A percentage p to represent the portion of jobs for each node can be calculated as:

$$p_i = \frac{P_i}{\sum_{i=1}^{N} P_i}$$

Base on p_i, Node N_i can thus compute the portion p_i of jobs and sends the results to an pre-assigned node which collects and merges the results. This scheme has no central control and thus can be easily extended if work nodes are increased. The idling time can be largely reduced by introducing the job splitting calculation.

In theory, IRLA can be parallelized via these two approaches. However, taking consideration of efficiency and flexibility, the parallelization of IRLA is accomplished via the second scheme described. The reasons are detailed as follows.

In a distributed grid environment (Coco, Laudani, & Pollicino, 2009), IRLA will benefit from an efficient grid resource scheduler that utilizes the resources.

A master-worker scheme is not flexible and cannot be easily adapted in a distributed grid environment where grid resources are usually dynamical. The use of a master node is inflexible and has the disadvantages of high-overload and overhead of communication. If the master node is faulty, the parallel simulation would crash or the performance would be degraded until an alternate master server is set up. The communication overhead would slow down the overall calculation time if data exchange is high.

Parallelization of the Components

The main computation components of IRLA are LOS, VD, and HRD. Low complex components such as post-processing are not parallelized because simply distributing the jobs of this module will not improve the overall performance rather it will incur extra communication overhead.

The objects are created in parallel. On creation, they are given an ID. Building data, antenna data and network configurations have to be loaded by all objects before actual simulation starts. This is ensured by setting up a barrier. As the time of loading data can usually be trivial, the cost of this barrier can usually be neglected. Because LOS engine has a lower computation complexity compared to other components, it will only be performed fully on the node where the result is stored (in this case, on the master node), while the rest of the nodes would simply just obtain LOS pixels for the use of a HRD engine. This will avoid unnecessary communication overhead spent on trivial tasks.

The following details the parallelization of each components of IRLA.

Parallelization of LOS: LOS marks the visibility and collects direct paths from the transmitter. This component has low complexity and nowadays can be handled very fast on standard PCs. This component is expected to run with full functionality at the node; which is used to save results but a more light-weighted LOS component is accomplished at other worker nodes. The modified light-weighted LOS component

does not calculate path loss at all and thus can be executed faster. However, at all nodes, LOS component marks the visibility area and collects secondary pixels for the HRD and VD. In this case, communication can be avoided and all processors can collect secondary cubes for the use of HRD.

Parallelization of VD: VD is an independent component mainly used for outdoor scenarios. The complexity of this component is $O(_n^3)$ (n denotes the number of border cubes at X-Y planes) i.e. $z = 0$ and ($x = 0$ or $x = N_x$ or $y = 0$ or $y = N_y$) where x, y, z represent the co-ordinates of cubes and N_x and N_y denotes the X and Y dimensions of scenario. By connecting the transmitter and these cubes, scan-lines are formed virtually. The principle thus can be easily parallelized because these scan-lines are independent from each other and they can be processed in parallel. The scan-line consists of building blocks comprising of a stack of pixels, which should be handled by only one scan-line. In a distributed environment, a processor shares global static information by message passing or accessing to a central node; which keeps the shared information. Message passing is costly and should be avoided wherever possible. The design of parallel IRLA is not centralized. The requirement to share global static variables is removed by a static data distribution scheme. In this case, there will be overlap of jobs assigned to each node because at this stage, nodes do not check if building blocks have been processed by other nodes. At the end of the calculations, results are sent to a node for collection and merged. Overlapping is also checked and only one piece of the result is considered for one building block. In order to avoid simultaneous access to the same building blocks, locks are used.

Parallelization of HRD: The number of discrete rays needed to be launched from the transmitter is known as N_{fringe}. As long as double counting is avoided, these rays can be considered independently, which offers the parallelism. The roughly-divide-and-solve approach as used in parallel VD can be also applied to HRD. Rays are roughly divided at the beginning of parallelization and they are calculated in independent memory space of the worker node. Double counting is avoided at each worker node. However, this approach does not guarantee the removal of all redundant pixels because rays may be repeatedly calculated at the worker nodes simply because close rays are launched at two nodes but there is no communication between them to avoid double counting. This can be solved at the last stage where results are collected at one node.

Efficiency. Assume D represents the number of conflicts caused by duplicated jobs (rays, building blocks etc) that have been produced due to distributed parallel simulation. Then smaller D leads to better efficiency and vice versa because duplicated jobs cost unnecessary computation time and cause an overhead of results sending and merging. It is preferable to mark continuous rays thus they can be efficiently computed locally on one node. Distributed HRD and VD employ similar strategy as allocating threads. Approximately, suppose job space is J_1 to J_n (n denotes the total number of jobs), and there are P distributed processors, then $D = P$.

Assume the overall performance of IRLA depends on N modules noted as $M_1 .. M_N$. The approximate running time (percentage) for these modules is represented as $p_1, p_2 .. p_N$. Thus,

$$\sum_{i=1}^{N} p_i = 1$$

The theoretical maximum speed up of M_i can be denoted as S_i and calculated by Amdahl's Formula (Bisseling, 2004). Hence max(S_i), $i \in$

Figure 4. Parallel IRLA with & without communication

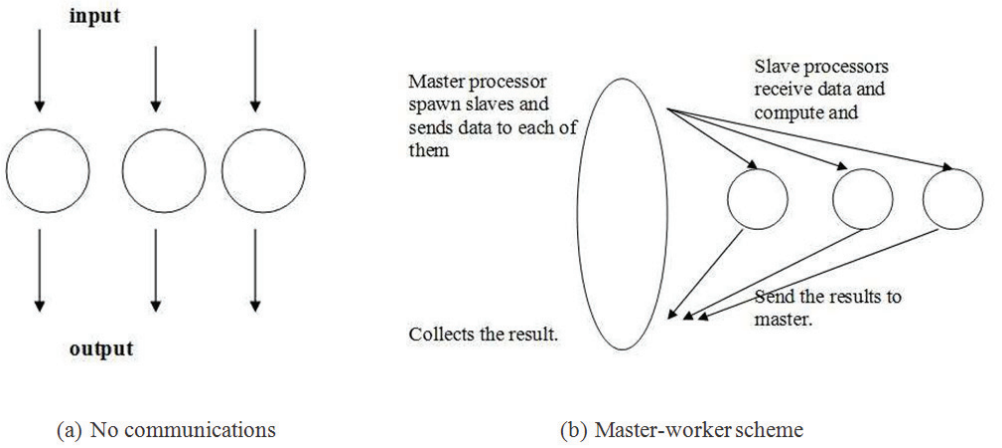


Figure 5. Optimization via using shared-memory

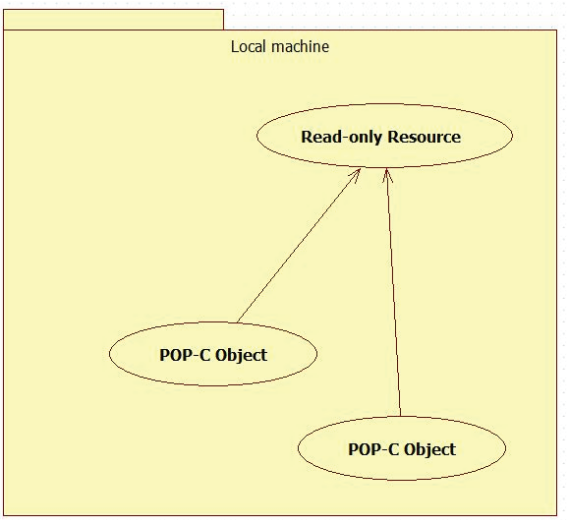


Table 1. Speedup of multithreading parallelization scheme

Speedup	Scenario
1.52	Munich
1.43	Paris

[1, N] gives the most important component (with priority) that is optimised.

Optimization. Figures 4(a) and 4(b) depict two structures that can be applied to parallel IRLA: No-communications and master-worker schemes. The no-communication scheme (Figure 4(a)) does not require any communication between processors. All the results are stored on local machines as files and if necessary, the results are copied and merged after simulation. This eliminates the costly message-passing and processors are independent to each other. The master-worker scheme (Figure 4(b)) requires one-time collection from the master node at the end of simulation, which may cause delay if the message-passing takes time (if the data to send and

Figure 6. Running time via parallelization

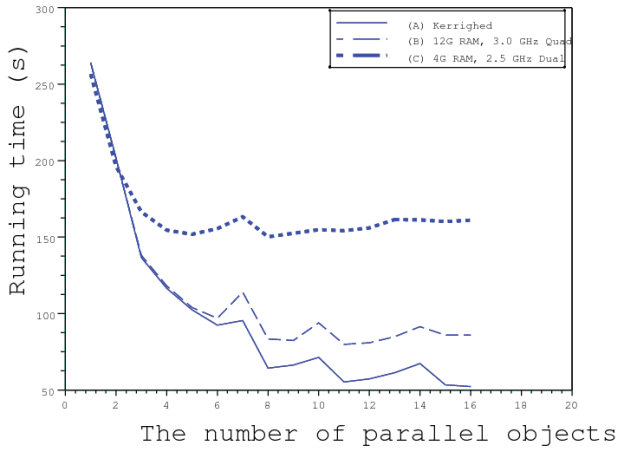
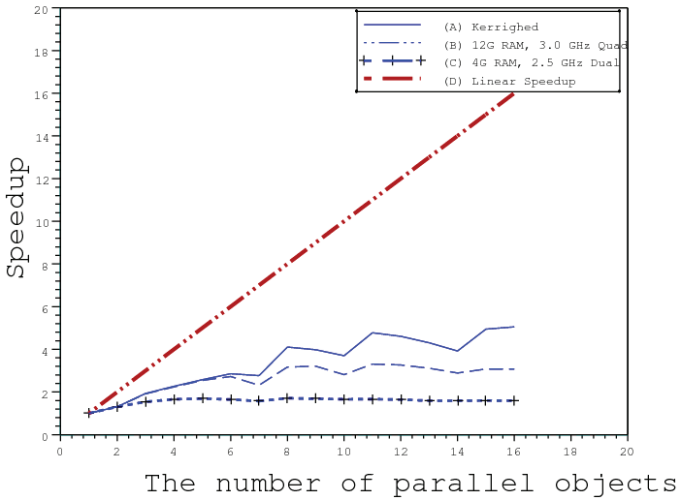


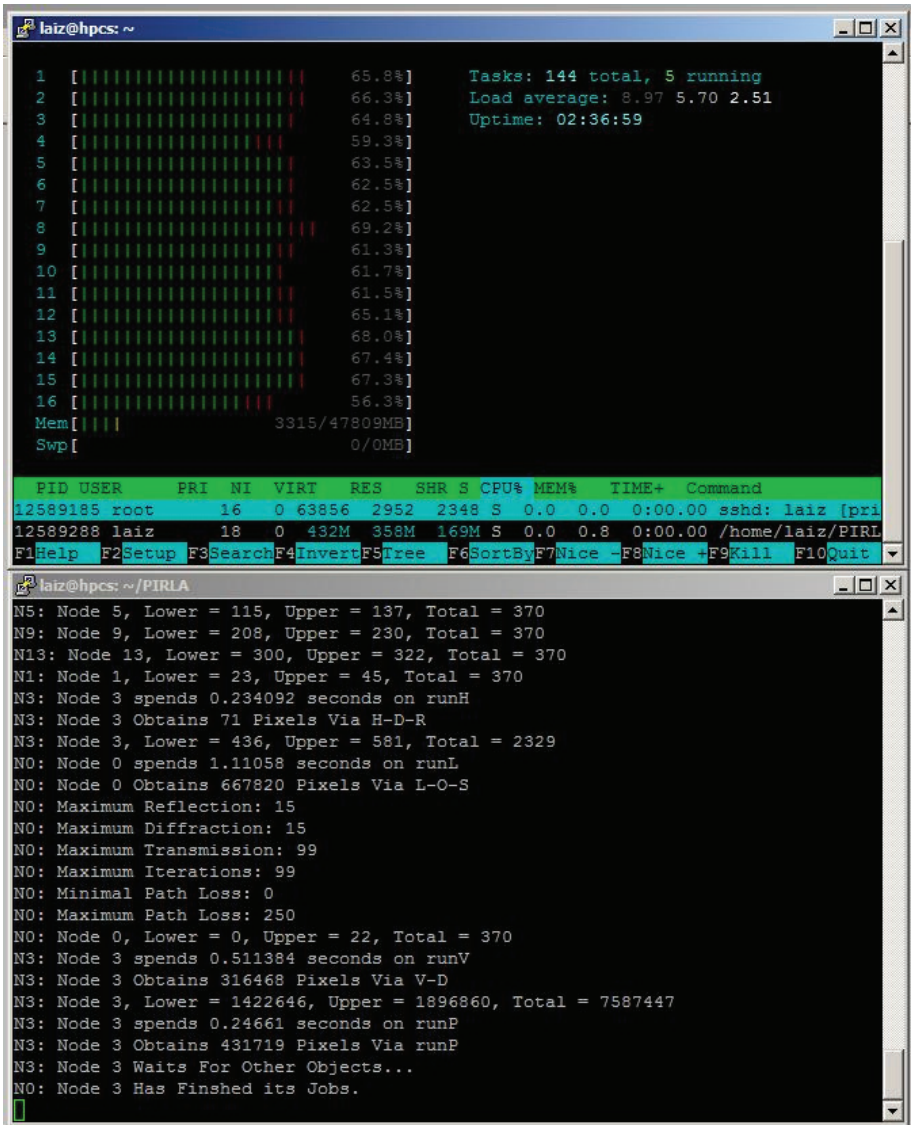
Figure 7. Speedup via parallelization



receive is large). Usually, more processors to split the computation, less data is required to be sent from worker node at the end of simulation. This is due to the job splitting scheme, in which the total computations are virtualized as pieces of small work, which then are distributed among available processors.

Usually, if there are many parallel objects created on the same physical machine, they are considered as independent processors; which have independent memory space. This causes waste of memory because usually these objects are opening the same input data (building data, antenna, network parameters etc). Furthermore, files (resources) are treated as read-only and

Figure 8. Run simulation on Kerrighed



will not change during computation. Larger scenario (or higher resolution) will cause larger discrete data set, which needs to be loaded by each object. It will limit the performance and the number of objects that can be created on the same machine. To solve this, shared-memory between processes are adopted (Figure 5). Parallel objects (processes) will check if the resources are available before they load it. And they will make the resources visible to other objects if they are created on the same physical machine. In this manner, memory consumption is reduced and the number of objects that can be created on the same machines is increased.

Simulations

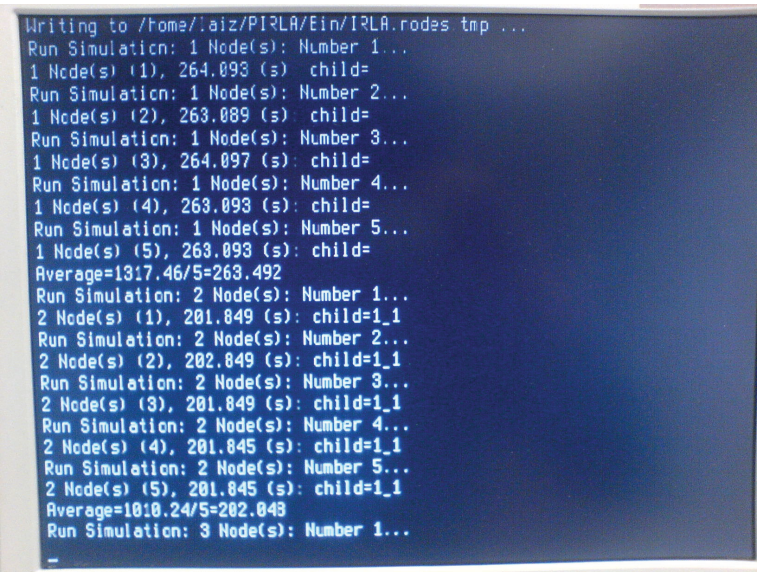
In order to test the parallelization efficiency of the parallel IRLA model via multithreading and distributed computing technologies, simulations are carried out on three platforms and results are analyzed. The specifications of machines (type A, B, C) are listed in Table 1, in which "Estimated power" is an estimation score calculated via POPC++ runtime system.

The simulation scenario is based on COST231-Munich (Universitat Karlshrue, n. d.). In this scenario, the size (N_x, N_y, N_z) is equal to (483, 683, 23) when the resolution is set to 5 meter. In order to analyze the results more clearly, the ray-signal threshold is increased to 250 dB, which will increase the computation complexity.

In order to assure a relative accurate timing result, simulations are required to run several times and the average results are adopted (Figure 9). A simulation on the Kerrighed (1998) that is a distributed-shared-memory architecture is displayed in Figure 8.

The running time is displayed in Figure 6 and its corresponding speedup is displayed in Figure 7. It is observed that multi-threaded simulation generally dominates the single-threaded (the number of parallel objects is one) because the resources are more efficiently utilized by the system. However, when the number of threads increases, the performance has reached the peak and tends to degrade, which is limited by physical resources and possibly the resource competition tends to occur more often.

Figure 9. Test parallelization efficiency



The running time can be greatly shortened by increasing the number of processors (the node specification can be found in Table 2) at the beginning. However, performance may degrade due to the unavoidable overhead for each object to load data and sends results at the end of calculations when more and more processors are used. It has been observed that for some scenarios, the job distributed to each object is small and each object is capable of handling it even (because of cache hit in local memory). In this case, a super linear speed up may be observed.

It is also interesting to find that with two or three processors, multithreading may outperform distributed POPC++; which is mainly due to the overhead of communication or pro-

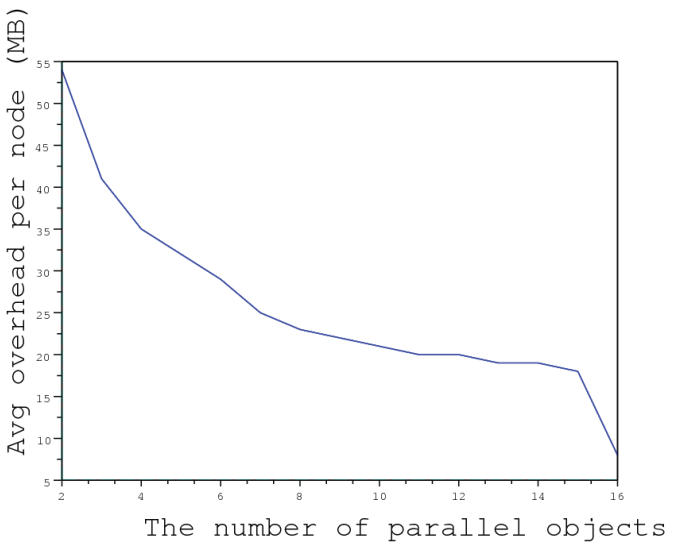
Table 2. Specification

Type	CPU (GHz)	Estimated Power	Cores	RAM (G)	OS
A (C)	3.0	5419	4	12	Fedora 10
B	2.5	4812	2	4	Ubuntu 9

Table 3. Running time of components

Components	Running Time (s)	Percentage (%)
LOS	1.3	9.5
VD	2.9	21.1
HRD	8.6	62.8
Post-processing	0.9	6.6

Figure 10. Communication cost via parallelization



cessor idling time from unfair distribution of jobs.

The running time of IRLA is greatly reduced by deploying parallel computation tasks to available nodes (Figure 8). It is also interesting to find that distributed memory is handled by Kerrighed behind scenes so that all objects created on the cluster virtually see a global large memory space and they can share the same data easily, which consume less memory (Figure 5).

The components of IRLA are of different complexity. Experiments show that different amount of time is spent on these components. For example, given Munich scenario, the running time for LOS, VD, HRD and post-processing is listed in Table 3. Apparently, the most time consuming parts are HRD and VD. It can be derived that the overall maximum speed for IRLA by parallelization is (based on the percentages of these components in Table 3).

$$\frac{1}{(1-0.211-0.628)+\frac{0.211+0.628}{N}}$$

N is the number of processors used; when N approaches infinity, the equation reaches 6.21. Each component can be further optimized by pinpointing the most time-consuming part. However, experiments show that usually the speedup hardly approaches 6.21, which is reasonable because of costly message-passing and the overhead of loading data etc. Figure 7 show that the maximum speedup via Kerrighed cluster (16 objects) is approximately 5, which is far less than linear speedup. The explanations are twofold. The first is due to communication overhead that nodes have to send and collect results. The second is due to unpredictable amount of job tasks (rays distribution) and hence the timing to finish sub-computation tasks at each node is different, which incur barrier synchronization waiting time. This varies from scenario to scenario but at least this experiment indicates the same speedup pattern observed on the same scenario (Figure 7).

The communication overhead (measured in Mega Bytes) decreases as the number of parallel objects grows, Figure 10 indicates that to some extent, when the number of processors employed is high, the communication overhead can be minimized to a constant because the average data amount to be sent over the network is split into small portions which can be sent and received within a short time. Furthermore, the total speedup has a limit because of the aforementioned inherent parallelization strategy of IRLA.

CONCLUSION

Ray launching is extremely time consuming in large scenarios. Solving angular dispersion and avoiding double counting have been proposed in previous work. Intelligent algorithms have been developed to accelerate the computation. Parallelization has been focused in this article where the issues related to performance etc are described. The multithreading and POP-C++ version of IRLA was developed and speedup was obtained (up to five times faster with sixteen processors). Parallelization further reduces the running time of IRLA and this can be further extended to distributed grid environment (Lai, Bessis, Zhang, & Clapworthy, 2007; Lai et al., 2009) in the future work. By using POPC++ toolkit, computation tasks are deployed and performance speedup can be observed. The parallelization also helps to solve a more complex problem which may not be solved on a single computer, i.e., the memory may be a restricting factor for some large scenarios on a single computer.

ACKNOWLEDGMENT

This work was supported by the EU-FP7 iPLAN and FP6 GAWIND under grant number MTKD-CT-2006-042783 (“Marie Curie Fellowship for Transfer of Knowledge”).

REFERENCES

Bisseling, R. (2004). *Parallel scientific computation: A structured approach using BSP and MPI*. New York, NY: Oxford University Press.

Coco, S., Laudani, A., & Pollicino, G. (2009, March). Grid-based prediction of electromagnetic fields in urban environment. *IEEE Transactions on Magnetics*, *45*, 1060–1063. doi:10.1109/TMAG.2009.2012577

Degli-Esposti, V., Fuschini, F., Vitucci, E., & Falciasecca, G. (2009). Speed-up techniques for ray tracing field prediction models. *IEEE Transactions on Antennas and Propagation*, *57*, 1469–1480. doi:10.1109/TAP.2009.2016696

Foster, I., & Kesselman, C. (2003). *The grid2, blueprint for a new computing infrastructure*. Sanfrancisco, CA: Morgan Kaufmann.

Glassner, A. (1989). *An introduction to ray tracing*. San Francisco, CA: Morgan Kaufmann.

Haslett, C. (2008). *Essentials of radio wave propagation*. Cambridge, UK: Cambridge University Press.

Kerrighed. (1998). *What is Kerrighed?* Retrieved from http://www.Kerrighed.org

Lai, Z., Bessis, N., De La Roche, G., Kuonen, P., Zhang, J., & Clapworthy, G. (2009, November). A new approach to solve angular dispersion of discrete ray launching for urban scenarios. In *Proceedings of the Loughborough Antennas & Propagation Conference* Leicestershire, UK (pp. 133-136).

Lai, Z., Bessis, N., De La Roche, G., Kuonen, P., Zhang, J., & Clapworthy, G. (2010, April). On the use of an intelligent ray launching for indoor scenarios. In *Proceedings of the Fourth European Conference on Antennas and Propagation*, Barcelona, Spain.

Lai, Z., Bessis, N., De La Roche, G., Kuonen, P., Zhang, J., & Clapworthy, G. (2010, April). The characterisation of human-body influence on 3.5 GHz indoor path loss measurement. In *Proceedings of the Second International Workshop on Planning and Optimization of Wireless Communication Networks*, Barcelona, Spain (pp. 1-6).

Lai, Z., Bessis, N., De La Roche, G., Song, H., Zhang, J., & Clapworthy, G. (2009, March). An intelligent ray launching for urban propagation prediction. In *Proceedings of the Third European Conference on Antennas and Propagation*, Berlin, Germany (pp. 2867-2871).

Lai, Z., Bessis, N., Kuonen, P., De La Roche, G., Zhang, J., & Clapworthy, G. (2009, August). A performance evaluation of a grid-enabled object-oriented parallel outdoor ray launching for wireless network coverage prediction. In *Proceedings of the Fifth International Conference on Wireless and Mobile Communications*, Cannes, France (pp. 38-43).

Lai, Z., Bessis, N., Zhang, J., & Clapworthy, G. (2007, September). Some thoughts on adaptive grid-enabled optimisation algorithms for wireless network simulation and planning. In *Proceedings of the UK e-Science, All Hands Meeting*, Nottingham, UK (pp. 615-620).

Nagy, L., Dady, R., & Farkasvolgyi, A. (2009, March). Algorithmic complexity of FDTD and ray tracing method for indoor propagation modelling. In *Proceedings of the Third European Conference on Antennas and Propagation*, Berlin, Germany.

Nguyen, T. (2004). *An object-oriented model for adaptive high-performance computing on the computational grid.* Présentée à la faculté informatique et communications, Zurich, Switzerland.

Nguyen, T., & Kuonen, P. (2007, January). Programming the grid with POP-C++. *Future Generation Computer Systems*, *23*(1), 23–30. doi:10.1016/j.future.2006.04.012

Rick, T., & Mathar, R. (2007, March). Fast edge-diffraction based radio wave propagation model for graphics hardware. In *Proceedings of the 2nd International ITG Conference* (pp. 15-19).

Silberschatz, A., & Galvin, P. (2006). *Operating system concepts with java* (7th ed.). New York, NY: John Wiley & Sons.

Universitat Karlshrue. (n. d.). *COST231 urban micro cell measurements and building data.* Retrieved from http://www2.ihe.uni-karlsruhe.de/forschung/cost231/cost231.en.html

Wolfle, G., Gschwendtner, B., & Landstorfer, F. (1997, May). Intelligent ray tracing - a new approach for the field strength prediction in microcells. In *Proceedings of the IEEE Vehicular Technology Conference,* Phoenix, AZ (pp. 790-794).

Zhang, J., & De La Roche, G. (2010). *Femtocells: Technologies and deployment*. New York, NY: John Wiley & Sons. doi:10.1002/9780470686812

Zhihua Lai finished BSc (Honours, First Class) and Ph.D. in University of Bedfordshire in 2006 and 2010 respectively. He was also a visiting scholar at GRID and Ubiquitous Computing Group, University of Applied Sciences of Fribourg, Switzerland in 2009 when he developed parallel distributed radiowave propagation models. His main research interests include radiowave propagation modelling and distrbuted/parallel algorithms. He has published over 10 papers and has been involved in a number of funded European projects in these areas.

Nik Bessis is currently a Head of Distributed and Intelligent Systems (DISYS) research group, a Professor and a Chair of Computer Science in the School of Computing and Mathematics at University of Derby, UK. He is also an academic member in the Department of Computer Science and Technology at University of Bedfordshire (UK). He obtained a BA (1991) from the TEI of Athens, Greece and completed his MA (1995) and PhD (2002) at De Montfort University (Leicester, UK). His research interest is the analysis, research, and delivery of user-led developments with regard to trust, data integration, annotation, and data push methods and services in distributed environments. These have a particular focus on the study and use of next generation and grid technologies methods for the benefit of various virtual organizational settings. He is involved in and leading a number of funded research and commercial projects in these areas. Prof. Bessis has published widely and is the editor of three books and the Editor-in-Chief of the International Journal of Distributed Systems and Technologies *(IJDST). In addition, Prof. Bessis is a regular reviewer and has served as a keynote speaker, conferences/workshops/track chair, associate editor, session chair and scientific program committee member.*

Guillaume De La Roche has been working as a research fellow at the Centre for Wireless Network Design (CWiND), United Kingdom, since 2007. From 2001 to 2002 he was a research engineer at Infineon, Munich, Germany. From 2003 to 2004 he worked in a small French company where he deployed WiFi networks. From 2004 to 2007 he was with the CITI Laboratory at the National Institute of Applied Sciences (INSA), France. He holds a Dipl-Ing from CPE Lyon, France, and M.Sc. (2003) and Ph.D. (2007) degrees in wireless communications from INSA Lyon. He is a co-author of the book Femtocells: Technologies and Deployment (Wiley, 2010).

Pierre Kuonen received a Master in Electrical Engineering from the Swiss Federal Institute of Technology (EPFL) in 1982. After six years of experience in the industry, he joined the Computer Science Theory Laboratory at EPFL in 1988 and began working in the field of parallel and distributed computing. He received his Ph.D. in computer science from the EPFL in 1993. Since 1994 he has worked regularly in the field of parallel and distributed computing. First at EPFL where he founded and directed the GRIP (Group for Research in Parallel Computing) and then at the University of Applied Sciences of Valais. Since 2003 he is full professor at the University of Applied Sciences of Fribourg in he Information and Communication technologies department (ICT), where he is leading the Grid & Ubiquitous Computing Group. Since 1993, besides his teaching activity, he was constantly involved in national or international research projects particularly for applications in the field of telecommunications or wave propagation.

Jie Zhang is Professor of Wireless Communications and Networks at the Department of Computer Science and Technology, University of Bedfordshire (UoB), UK. He received his MEng and PhD degrees from the Department of Automatic Control and Electronic Engineering, East China University of Science and Technology, Shanghai, China. From 1997 to 2002, he was a Research Fellow with University College London, Imperial College London, and Oxford University. He is the founding Director of the Centre for Wireless Network Design, which is one of the best-funded and leading research groups in wireless network design and femtocell research in Europe. Since 2003, he has been awarded more than 19 projects worth over £ 4.0 million (his share). He has published over 120 journal and conference papers, and is a lead author of the first technical book on femtocells - "Femtocells: Technologies and Deployment*", which was published by Wiley in Jan. 2010.*

Gordon J. Clapworthy received a BSc (Hons., Class 1) in Mathematics and a PhD in Aeronautical Engineering from the University of London, and an MSc, with Distinction, in Computer Science from The City University, London. He is a Professor of Computer Graphics in the Department of Computer Science & Technology and Head of the Centre for Computer Graphics & Visualization (CCGV) at the University of Bedfordshire, UK. His interests include medical visualisation, computer animation, biomechanics, virtual reality, surface modelling, and fundamental computer graphics algorithms. Clapworthy has published over 150 refereed articles and has been involved in 25 international projects in recent years, mostly funded by the European Commission; he coordinated 8 of these. He is a member of the ACM, ACM SIGGRAPH and Eurographics.