6 A parallel framework for radio-network planning and optimization

In this chapter, a parallel framework for radio-coverage simulation is presented. The objective of the framework is to provide an environment for the radio-coverage prediction of large radio networks. Due to its high performance, the framework also enables the evaluation of more complex optimization problems for radio networks.

The framework is implemented as a module of a geographical information system, since the prediction calculation employs digital elevation models and land-usage data. Following a master-worker parallel paradigm over a message-passing communication model proved to be a bottleneck for the performance of the parallel module. A new approach, that overcomes this performance constraint, is introduced in this chapter. The efficiency improvement is based on overlapping process execution and communication. This minimizes the idle time of the worker processes and thus improves the overall efficiency of the system. To this end, the intermediate calculation results are saved into an external database (DB) instead of sending them back to the master process. This approach is implemented as part of a parallel radio-prediction tool (PRATO) for the open-source Geographic Resources Analysis Support System (GRASS) [114]. An extended analysis of the experimental results is provided, which are based on real data from an LTE network currently deployed in Slovenia. Based on these experiments, which were performed on a computer cluster, the new technique exhibits better scalability than the traditional master-worker approach. Some real-world data sets are presented, the coverage predictions of which are calculated in a shorter time while saturating the hardware utilization.

The content of this chapter extends the research work published by the author in [17]. The rest of this chapter is organized as follows. Section 6.2 gives an overview of the relevant publications, describing how they relate to the presented work. Section 6.3 gives a description of the radio-coverage prediction problem, including the radio-propagation model. Section 6.4 concentrates on the design and implementation of the radio-propagation tool, for both the serial and parallel versions. Section 6.5 discusses the experimental results and their analysis.

6.1 Motivation

Although Gordon Moore's well-known and often cited prediction still holds [108], the fact is that for the past few years, CPU speeds have hardly been improving. Instead, the number of cores within a single CPU is increasing. This situation poses a challenge for software development in general and research in particular: a hardware upgrade will, most of the time, fail to double the serial-execution speed of its predecessor. However, since this commodity hardware is present in practically all modern desktop computers, it creates an opportunity for the parallel exploitation of these computing resources in order to enhance the performance of complex algorithms over large data sets. The challenge is thus to deliver the computing power of multi-core systems in order to tackle a computationally time-consuming problem, the completion of which is unfeasible using traditional serial approaches. Indeed, improving the performance and data-set sizes a well-known approach may handle opens new possibilities for models and implementations of enhanced analytical methods.

A traditional approach, when dealing with computationally expensive problem solving, is to simplify the models, thus reducing the time needed for their calculation. Clearly, this method increases the introduced error level, which is not an option for a certain group of simulations, e.g., those dealing with disaster-contingency planning and decision support [84, 180]. The conducted simulations during the planning phase of a radio network also belong to this group. Their results are the basis for the decision making prior to physically installing the BSs and antennas that will cover a certain geographical area. A larger deviation of these results increases the probability of making the wrong decisions at installation time, which may considerably increase the costs or even cause mobile-network operators to incur losses.

Various researchers have successfully deployed high-performance computing (HPC) systems and techniques to solve different problems dealing with spatial data [6, 10, 71, 84, 99, 180, 119, 156, 157, 158, 174]. Their work confirms that a parallel paradigm such as master-worker, and techniques like work pool (or task farming) and spatial-block partitioning are applicable when dealing with parallel implementations over large spatial data sets. However, it is well known that parallel programming and HPC often call for area experts in order to integrate them into a given environment [36]. Moreover, the wide range of options currently available creates even more barriers for general users wanting to benefit from HPC.

In the following, a high-performance, radio-propagation prediction tool for GSM (2G), UMTS (3G) and LTE (4G) networks is presented. It can be used for planning the different phases of a new radio-network installation, as well as a support tool for maintenance activities related to network troubleshooting in general and optimization in particular. Specifically, automatic radio-coverage optimization requires the evaluation of millions of radio-propagation predictions in order to find a good solution set. This is often unfeasible using other serial implementations of academic or commercial tools [81, 106, 124].

As a reference implementation, the publicly available radio-coverage prediction tool, presented in [81], was used. The authors developed a modular radio-coverage tool that performs separate calculations for radio-signal path loss and antenna-radiation patterns, also taking into account different configuration parameters, such as antenna tilting, azimuth and height. The output result, saved as a raster map, is the maximum signal level over the target area, in which each point represents the received signal from the best-serving transmitter (or cell). This work implements some well-known radio-propagation models, e.g., Okumura-Hata [75], the description of which is latter presented in Section 6.3.2. Regarding the accuracy of the predicted values, the authors reported comparable results to those of an industrial tool [81]. To ensure that the presented implementation is completely compliant with this reference, a comparison test was designed, that consists of running both tools with the same set of input

parameters. The test results from PRATO and the reference implementation were identical in all the tested cases.

6.2 Related work

There are few examples of radio-network simulators in the literature [81, 106, 123, 124, 134, 179]. Most of these tools were developed for academic research, thus not targeting industrial-sized environments.

A Matlab-based LTE simulator was proposed in [106]. It implements a standard LTE downlink physical layer, including Adaptive Modulation and Coding (AMC), multiple users, MIMO transmission and scheduler. Despite being open source and freely available, the fact of being implemented in Matlab makes it restrictive in terms of tackling big problem instances of real networks.

A promising tool from the performance point of view was presented in [124], where the authors implemented a full-stack LTE system in C++. Although the tool has no graphical capabilities for displaying the simulation progress or outcome, they might be included, since the source code is available. In our opinion, the main drawback of this tool is the lack of documentation, which makes it very difficult to continue extending this work without the direct help of some of the original authors.

As an extension to the well-known NS-2 network simulator, Filiposka and Trajanov [52] introduced a module for radio-propagation predictions, which takes the terrain profile into account. In this case, the authors focus on the relief, leaving out signal loss due to land-usage, which is an important factor when targeting realistic radio-propagation scenarios.

The task-parallelization problem within the GRASS environment was addressed by several authors in a variety of studies. For example, in [27], the authors presented a collection of GRASS modules for a watershed analysis. Their work concentrates on different ways of slicing raster maps to take advantage of a Message Passing Interface (MPI) implementation.

In the field of HPC, the authors of [6] presented MPI and Ninf-G implementation examples of a GRASS raster module, that processes vegetation indexes from satellite images. The authors acknowledge a limitation in the performance of their MPI implementation for big processing jobs. The restriction appears due to the computing nodes being fixed to a specific spatial range, since the input data are equally distributed among worker processes, creating an obstacle for load balancing in heterogeneous environments.

Using a master-worker technique, the work presented in [82] abstracts the GRASS data types into its own *struct* and MPI data types, thus not requiring the GRASS in the worker nodes. The data are evenly distributed by row among the workers, with each one receiving an exclusive column extent to work on. Their test cluster contained heterogeneous hardware configurations. The authors noted that data-set size is bounded by the amount of memory on each of the nodes, since they allocate the memory for the whole map as part of the set-up stage, before starting the calculation. Regarding the data sets during the simulations, the largest one contained 3,265,110 points. They concluded that the data-set size should be large enough for the communication overhead to be hidden by the calculation time, so that the parallelization pays off.

In [156], the authors employed a master-worker approach, using one worker process per worker node. The complete exploitation of the computing resources of a single computing node is achieved with OpenMP. The experimental environment featured one host. The horizon-composition algorithm presents no calculation dependency among the spatial blocks. Consequently, the digital elevation model (DEM) was divided into separate blocks to be independently calculated by each worker process. The authors presented an improved algorithm

that can also be used to accelerate other applications like visibility maps. The tasks are dynamically assigned to idle processes using a task-farming paradigm over the MPI.

Similar to [156], in [157] there was no calculation dependency among the spatial blocks. The experimental evaluation was made over multiple cores of one CPU and a GPU, and a master-worker setup was used for process communication.

In [180], the authors presented a parallel framework for GIS integration. Based on the principle of spatial dependency, they lowered the calculation-processing time using a knowledge database, delivering the heavy calculation load to the parallel back-end if a specific problem instance is not found in the database. There was an additional effort to achieve the presented goals, since the implementation of a fully functional GIS (or "thick GIS" as the authors call it) was required on both the desktop client and in the parallel environment.

An agent-based approach for simulating spatial interactions was presented in [65]. This technique decomposes the entire landscape into equally-sized regions, i.e., a spatial-block division as in [156], which are in turn processed by a different core of a multi-core CPU. This work used multi-core CPUs instead of a computing cluster.

Some years ago, grid computing received a lot of attention from the research community. It appeared to be a good alternative for accessing the extra computational power needed for the spatial analysis of large data sets [10, 171, 173]. However, several obstacles are still preventing this technology from being widely used. In particular, its adoption requires not only hardware and software compromises with respect to the involved parts, but also a behavioral change at the human level [10].

6.3 Radio-coverage prediction for mobile networks

6.3.1 Background

As it was mentioned in Chapter 3, radio communications in a mobile network take place between a BS (or fixed transmitter) and a number of UEs (or mobile receivers). The effects of signal propagation limit the performance of a mobile-radio system. For this reason, the characterization and modeling of radio propagation is considered a fundamental aspect in radio-network planning [4]. Consequently, understanding the mathematical modeling of a frequency channel is necessary to accurately predict the system performance and to provide a mechanism to analyze the effects caused by signal propagation [122].

The coverage planning of radio networks is a key problem that all mobile operators have to deal with. Moreover, it has proven to be a fundamental issue, not only in LTE networks, but also in other standards for mobile communications [132, 137, 148, 166]. One of the primary objectives of mobile-network planning is to efficiently use the allocated frequency band to ensure that some geographical area of interest can be satisfactorily reached with the BSs of the network. To this end, radio-coverage prediction tools are of great importance, as they allow network engineers to test different network configurations before physically implementing the changes. Radio-coverage prediction is a complex task, mainly due to the several combinations of hardware and configuration parameters that have to be analyzed in the context of different environments. The complexity of the problem means that radio-coverage predictions are computationally-intensive and time-consuming, hence the importance of using fast and accurate tools (see Section 6.4.4 for a complexity analysis of the algorithm). Additionally, since the number of deployed transmitters keeps growing with the adoption of modern standards [132], there is a clear need for a radio-propagation tool that is able to cope with larger work loads in a feasible amount of time (see Section 6.4.4 for the running time of the serial version).

6.3.2 Radio-propagation model

PRATO uses a modified version of the well-known Okumura-Hata model for radio-propagation predictions [75]. Other accurate methods exist, like the ones based on ray tracing [38, 170]. However, these methods are more sensible to deviations in input data, like DEMs and buildings, and are still inefficient in terms of the computational effort required to achieve satisfying results. Empirical methods for radio-propagation predictions, like Okumura-Hata, give acceptable results within a feasible amount of time. For this reason, they have become the industry standard for non-deterministic signal-propagation calculations [16, 35, 76, 137, 149].

The Okumura-Hata model has been largely studied and shown to be suitable for predicting the radio propagation in LTE networks [4, 137]. In its primary form, the model distinguishes the distance from the receiver to the transmitter, the frequency used and the effective antenna height, i.e., the height of the antenna above the receiver's level. These variables are taken into account in order to calculate the path loss in line-of-sight (LOS) conditions. Additionally, for distinguishing non-line-of-sight (NLOS) conditions, the terrain profile and Earth shape are added to the original formula. In this context, a NLOS situation appears when the first Fresnel zone is obscured by at least one obstacle [176]. Equation (6.1) describes the path loss when there is LOS between the transmitter and the receiver.

$$pl_{\mathrm{LOS}}(d_{(x,y)},\beta) = a_0 + a_1\log(d_{(x,y)}) + a_2\log(H_{\mathrm{A}}) + a_3\log(d_{(x,y)})\log(H_{\mathrm{A}}) - 3.2\left[\log(11.75\cdot H_{\mathrm{R}})\right]^2 + 44.49\log(F) - 4.78\left[\log(F)\right]^2, \quad (6.1)$$

where $\beta = (a_0, a_1, a_2, a_3)$ is the vector containing the control parameters of the model, $d_{(x,y)}$ is the distance (in kilometers) from the transmitter to the topography point with coordinates (x, y), H_{A} is the effective antenna height (in meters) of the transmitter, H_{R} is the antenna height (in meters) of the receiver, and F is the frequency, expressed in MHz. In NLOS conditions, the path loss is calculated as in Equation (6.2):

$$pl_{\mathrm{NLOS}}(d_{(x,y)}) = \sqrt{\left[\alpha K(d_{(x,y)})\right]^2 + E(d_{(x,y)})^2}, \quad (6.2)$$

where α is the knife-edge diffraction control parameter, the value of which is calculated based on the level of obstruction of a Fresnel zone, $K(d_{(x,y)})$ is the knife-edge diffraction loss (in dB), and $E(d_{(x,y)})$ is the correction due to the Earth sphere (in dB). All three values depend on the characteristics of the topography point with coordinates (x, y).

In this work, as well as in [52], the terrain profile is used for LOS determination, i.e., an obstacle obstruction in the first Fresnel zone of the transmitter. In order to adequately predict signal-loss effects due to foliage, buildings and other fabricated structures, additional loss factors based on the land usage (clutter data), are included. This technique is adopted by several propagation models for radio networks [3, 16, 113]. Consequently, an extra term is introduced for signal loss due to clutter, thus defining the model-predicted path loss as

$$pl(d_{(x,y)},\beta) = pl_{\mathrm{LOS}}(d_{(x,y)},\beta) + pl_{\mathrm{NLOS}}(d_{(x,y)}) + pl_{\mathrm{CLUT}}(d_{(x,y)}), \quad (6.3)$$

where $pl_{\mathrm{CLUT}}(d_{(x,y)})$ represents the clutter loss at the topography point with coordinates (x, y), expressed in dB.

6.4 Design and implementation

6.4.1 Geographic Resources Analysis Support System

GRASS [114], a free and open-source software project that implements a Geographical Information System (GIS), is used as the software environment for PRATO. This GIS software was originally developed at the US Army Construction Engineering Research Laboratories and is a full-featured system with a wide range of analytical, data-management, and visualization capabilities. Currently, the development of GRASS GIS is supported by a growing community of volunteer developers.

The use of GRASS GIS as an environment for PRATO presents many advantages. First, the current development of GRASS is primarily Linux-based. Since the field of HPC is dominated by Linux and UNIX systems, an environment with Linux support is critical for this work. Software licensing is another important consideration for choosing GRASS, since it is licensed under the GNU Public License [150] that imposes the availability of the source code. This allows to make potential modifications to the system, thus adapting it for the parallel computation environment. Moreover, GRASS provides a great deal of built-in functionality, capable of operating with raster and vector topological data that can be stored in an internal format or a DB.

6.4.2 Multi-paradigm parallel programming

The implementation methodology adopted for PRATO follows a multi-paradigm, parallel programming approach in order to fully exploit the resources of each node in a computing cluster. This approach combines a master-worker paradigm with an external DB. To efficiently use a shared memory multi-processor on the worker side, and to effectively overlap the calculation and communication, PRATO uses POSIX threads [26].

To use the computing resources of a distributed memory system, such as a cluster of processors, PRATO uses the MPI [70]. The MPI is a message-passing standard that defines the syntax and semantics designed to function on a wide variety of hardware. The MPI enables multiple processes, running on different processors of a computer cluster, to communicate with each other. It was designed for high performance on both massively parallel machines and on workstation clusters.

PRATO also supports the execution of the most computationally-intensive parts of the radio-propagation algorithm on a GPU. Moreover, the GPU hardware is used if it is available on the computing nodes that host the worker processes (see Section 6.4.5.3 later in this chapter for a discussion about the GPU implementation).

In order to make the text clearer and to differentiate between the programming paradigms used from here on, a POSIX thread will be refered to simply as a 'thread' and a MPI process as a 'process'.

6.4.3 Design of the serial version

This section describes the different functions contained in the serial version of PRATO, which is implemented as a GRASS module. Their connections and data flow are depicted in Figure 6.1, where the parallelograms of the flow diagram represent the input/output (I/O) operations.

The design follows a similar internal organization as the radio-planning tool presented in [81], but with some important differences. Specifically, the design presented here employs a direct connection to an external database server for intermediate result saving, instead of the slow, built-in GRASS database drivers. To explicitly avoid tight coupling with a specific

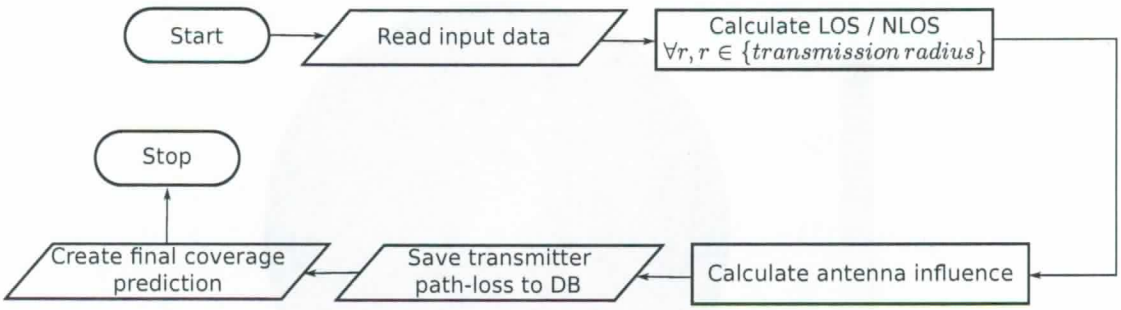

Figure 6.1: Flow diagram of the serial version.

database vendor, the generated output is formatted in plain text, which is then forwarded to the DB. Any further processing is achieved by issuing a query over the database tables that contain the path-loss results for each of the processed transmitters.

6.4.3.1 Input parameters

All input data are read in the first step (see "Read input data" in Figure 6.1). Their formats differ based on the data they contain, i.e.,

- GRASS raster files are used for the DEM and clutter data, whereas
- a text file is used for the transmitter configurations and other service-dependent options.

Since the module accepts a considerable amount of input parameters, they are read from a text-based initialization (INI) file. This is far more practical than passing them as command-line parameters, which would make them error-prune and difficult to read. Besides, the INI file may contain configuration parameters for many transmitters. The user selects which one(s) to process at run-time by passing a command-line option.

6.4.3.2 Isotropic path-loss calculation

This step starts by calculating which receiver points, r, are within the specified transmission radius (see "*transmission radius*" in Figure 6.1). The transmission radius is defined around each transmitter in order to limit the radio-propagation calculation to a reasonable distance. For these points, the LOS and NLOS conditions are calculated with respect to the transmitter (see "Calculate LOS/NLOS" in Figure 6.1). The following step consists of calculating the path loss for an isotropic source (or omni antenna). This calculation is performed by applying the radio-propagation model, which was previously defined in Equation (6.3), to each of the points within the transmission radius around the transmitter.

Figure 6.2 shows an example of the isotropic path-loss calculation, only including the map area within the transmission radius. The color scale is given in dB, indicating the signal loss from the isotropic source of the transmitter, located at the center. Notice the hilly terrain is clearly distinguished due to LOS and NLOS conditions from the signal source.

6.4.3.3 Antenna diagram influence

This step considers the antenna radiation diagram of the current transmitter and its influence over the isotropic path-loss calculation (see "Calculate antenna influence" in Figure 6.1). Working on the in-memory results generated by the previous step, the radiation diagram

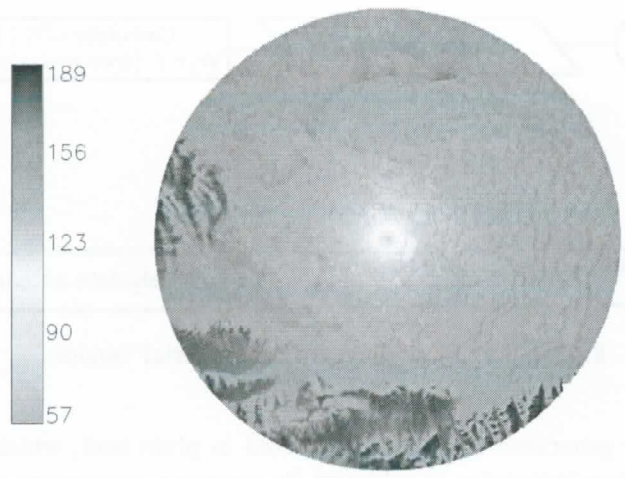

Figure 6.2: Example of a raster map, showing the result of a path-loss calculation from an isotropic source.

of the antenna is taken into account, including the beam direction, the electrical and the mechanical tilt. Figure 6.3 shows the map area within the transmission radius, where this calculation step was applied to the results from Figure 6.2. Notice the distortion of the signal propagation that the antenna has introduced.

Figure 6.3: Example of a raster map, showing the antenna influence over the isotropic path-loss result, as depicted in Figure 6.2.

6.4.3.4 Transmitter path-loss prediction

In this step, the path-loss prediction of the transmitter is saved in its own database table (see "Save transmitter path-loss to DB" in Figure 6.1). This is accomplished by connecting the standard output of the GRASS module with the standard input of a database client. Naturally, the generated plain text should be understood by the DB itself.

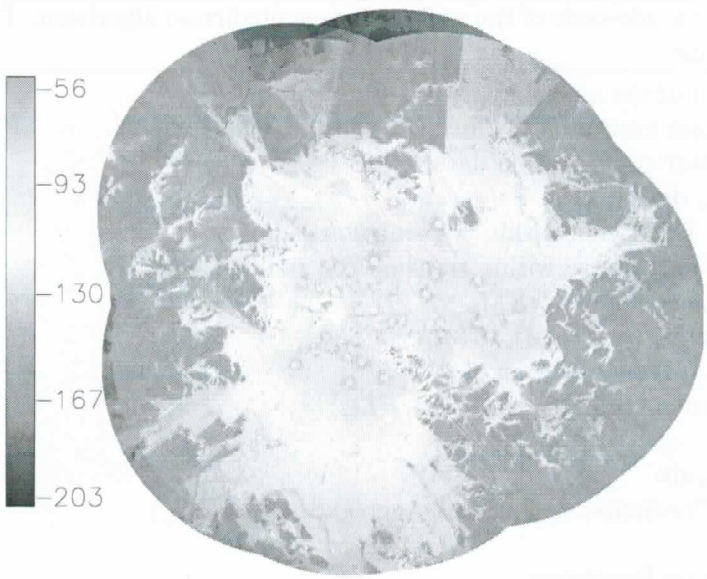

Figure 6.4: Example of a raster map, displaying the final coverage prediction of 136 transmitters over a geographical area. The color scale is given in dBm, indicating the received signal strength. Darker colors denote areas with a reduced signal due to the fading effect of the hilly terrain and clutter.

6.4.3.5 Coverage prediction

The final radio-coverage prediction, containing the aggregation of the partial path-loss results of the involved transmitters, is created in this step (see "Create final coverage prediction" in Figure 6.1). The received signal strength from each of the transmitters is calculated as the difference between its transmit power and the path loss for the receiver's corresponding position. This is done by executing an SQL query over the tables containing the path-loss predictions of each of the processed transmitters. Finally, the output is generated using the GRASS built-in modules *v.in.ascii* and *v.to.rast*, which create a raster map using the query results as the input. The resulting raster map contains the maximum received signal strength for each individual point, as shown in Figure 6.4. In this case, the color scale is given in dBm, indicating the strongest received signal strength from the transmitters.

6.4.4 Computational complexity

In this section, the time complexity of the radio-coverage prediction algorithm is presented, the pseudo-code of which is listed in Algorithm 6.1.

The algorithm starts by loading the input DEM and clutter data. Both RSGs should account for the same area and resolution, consequently containing the same number of pixels, M. The transmitter data is then loaded into set T_x, the cardinality of which is denoted as $n = |T_x|$. For each transmitter $t_x \in T_x$, a smaller subarea of the DEM and clutter data, denoted as DEM_{t_x} and $Clut_{t_x}$, respectively, is delimited around t_x, based on a given transmission radius. The number of pixels within this sub-area is denoted as m, and its value is the same for all $t_x \in T_x$. The visibility for an RSG pixel is computed using the *LineOfSight* function, by walking from the antenna of the transmitter to the given pixel element, along the elements intersected by a LOS, until either the visibility is blocked, or the target is reached [45]. Regarding the *PathLoss* function, whenever a receiver point is in NLOS, the walking path from the transmitter has to be inspected for obstacles, calculating

Algorithm 6.1 Pseudo-code of the radio-coverage prediction algorithm. The time complexity is given per line.

$DEM \leftarrow$ DEM of the whole area.	$\triangleright\ O(M)$
$Clutter \leftarrow$ signal losses due to clutter of the whole area.	$\triangleright\ O(M)$
$T_x \leftarrow$ transmitter configuration data.	$\triangleright\ O(n)$
for all $t_x \in T_x$ **do**	$\triangleright\ O(n \cdot m^2)$
$DEM_{t_x} \leftarrow$ DEM area within transmission radius of t_x	$\triangleright\ O(m)$
$Clut_{t_x} \leftarrow$ Clutter area within transmission radius t_x	$\triangleright\ O(m)$
$LoS_{t_x} \leftarrow$ LineOfSight (DEM_{t_x})	$\triangleright\ O(m^2)$
$PL_{t_x} \leftarrow$ PathLoss $(DEM_{t_x}, Clut_{t_x}, LoS_{t_x})$	$\triangleright\ O(m^2)$
$Diag_{t_x} \leftarrow$ Antenna diagram of t_x	$\triangleright\ O(1)$
$PL_{t_x} \leftarrow$ AntennaInfluence $(Diag_{t_x}, PL_{t_x})$	$\triangleright\ O(m)$
end for	
for all $t_x \in T_x$ **do**	$\triangleright\ O(n \cdot m)$
$CoveragePrediction \leftarrow$ PathLossAggregation (t_x, PL_{t_x})	$\triangleright\ O(m)$
end for	
return $CoveragePrediction$	

the diffraction losses for each of them, i.e., α and $K(d_{(x,y)})$ from Equation (6.2). Hence, its quadratic complexity, which dominates the complexity of the algorithm, together with *LineOfSight*, resulting in an algorithmic complexity denoted by:

$$O(M + n \cdot m^2). \tag{6.4}$$

Although n will generally be many orders of magnitude smaller than m^2, its computational-time complexity is relevant for practical use. For example, assuming the radio-coverage prediction for one transmitter completes in around 15 seconds using a serial implementation, the prediction for a mobile network comprising 10,240 transmitters would have an execution time of almost two days.

6.4.5 Design of the parallel version

The focus here is on the practical usability and performance of PRATO. The parallel implementation aims at overcoming the computational-time constraints that prevent a serial implementation from tackling bigger problem instances in a feasible amount of time.

A major drawback of the GRASS as a parallelization environment is that it is not thread-safe, meaning that concurrent changes to the same data set have an undefined behavior [24]. One technique to overcome this problem is to abstract the spatial data from the GRASS. For example, in [82], the authors achieved the GRASS abstraction by introducing a *Point* structure with four *double* attributes, where each pixel of the RSG is mapped to an instance of this structure. Another possibility is for one of the processes, e.g., the master, to read entire rows or columns of data before dispatching them for processing to the workers [6, 82]. In this case, an independence between row/column calculations is required, which is a problem-specific property. Here, abstraction from the GRASS is achieved by loading each spatial-data set into a separate 2D matrix of basic data-type elements, e.g., *float* or *double* depending on the desired accuracy. Each matrix is then assigned the geographical location of the closest corner to the origin of the map-projection system used, e.g., the lower-left corner for the transverse Mercator map projection over central Europe. It follows that the geographical location of any element within the matrix is calculated as the combination of the geographical location of the matrix and the offset of the target element (see Figure 6.5). The advantage of

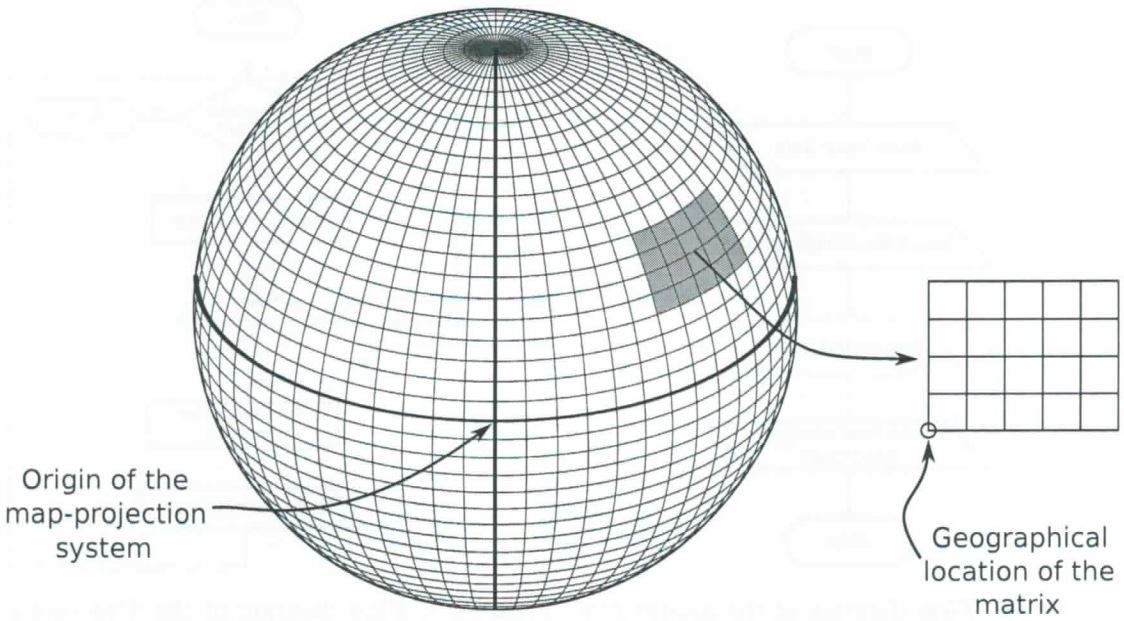

Figure 6.5: Example of a geographical-location mapping of the input-spatial data into a 2D matrix, the lower-left corner of which indicates the nearest point to the origin of the map-projection system.

this technique is having the geographical location of a pixel readily available with a minimum memory footprint. Moreover, a convenient consequence of this abstraction schema is that worker processes are completely independent of the GRASS, thus significantly simplifying the deployment of the parallel implementation over multiple computing hosts.

In the area of geographical-information science, the master-worker paradigm has been successfully applied by several authors [5, 6, 27, 71, 82, 156, 157]. However, this technique presents certain issues that prevent the full exploitation of the available computing resources when deployed over several networked computers. Additionally, such issues are difficult to measure when the parallelization involves only one computing node [156, 157] (because no network communication is required), or only a few processes deployed over a handful of nodes [82]. Specifically, the problem refers to network saturation and idle processes within the master-worker model. Generally speaking, a single communicating process, e.g., the master, is usually not able to saturate the network connection of a node. Using more than one MPI process per node might solve this problem, but possible rank-ordering problems may appear, thus restricting the full utilization of the network [127]. Another issue appears when the master process executes the MPI code, in which case other processes sleep, making a serial use of the communication component of the system. Consequently, the master process becomes the bottleneck of the parallel implementation as the number of worker processes grows. This situation is also common when dealing with the metadata of a spatial region, which may relate to several elements of a RSG, making it a frequent cause of load imbalance [65, 77, 174]. In PRATO, the transmitter configuration and its antenna diagram represent metadata that are complementary to the sub-region that a transmitter covers.

Hybrid MPI-OpenMP implementations [156, 157], in which no MPI calls are issued inside the OpenMP-parallel regions, also fail to saturate the network [127]. A possible solution to this problem is to improve the communication overlap among the processes. To this end, PRATO features non-blocking point-to-point MPI operations, and an independent thread in the worker process that saves the intermediate results into a DB. One such database system

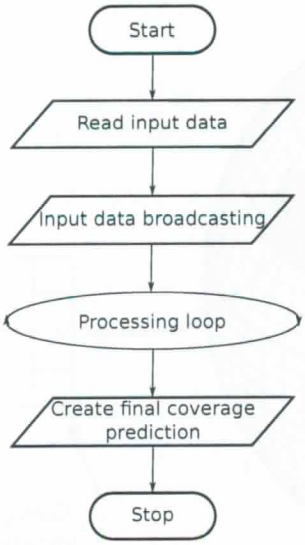

Figure 6.6: Flow diagram of the master process.

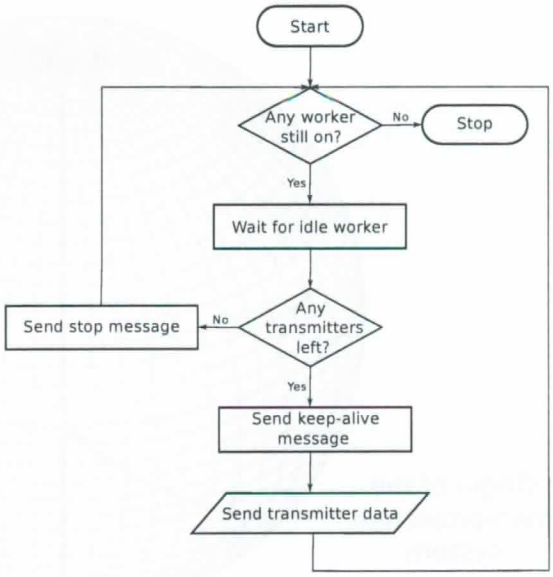

Figure 6.7: Flow diagram of the "Processing loop" step of the master process.

per computer cluster is used, which also serves the input data to the GRASS, in order to aggregate the partial results of the path-loss predictions or to visualize them. It is important to note that any kind of DB may be used, e.g., relational, distributed [121] or even those of the NoSQL type [152]. Nevertheless, a central, relational-database system is used here, since they are the most popular and widely available ones. Additionally, the non-blocking message-passing technique used to distribute the work-load among the nodes provides support for heterogeneous environments. As a result, computing nodes featuring more capable hardware receive more work than those with weaker configurations, thus ensuring a better utilization of the available computing resources despite hardware diversity, and improved load balancing.

6.4.5.1 Master process

The master process, the flow diagram of which is given in Figure 6.6, is the only component that runs within the GRASS environment. As soon as the master process starts, the input parameters are read. This step corresponds to "Read input data" in Figure 6.6, and it is carried out in a similar way as in the serial version. The next step delivers the metadata that is common to all the transmitters to all the processes (see "Metadata broadcasting" in Figure 6.6). Before distributing the work among the worker processes, the master process proceeds to decompose the loaded raster data into 2D matrices of basic-data-type elements, e.g., *float* or *double*, before dispatching them to the multiple worker processes. In this case, the decomposition applies to the DEM and the clutter data only, but it could be applied to any point-based data set. In the next step, the master process starts an asynchronous message-driven processing loop (see "Processing loop" in Figure 6.6), the main task of which is to assign and distribute the sub-region and configuration data of different transmitters among the idle worker processes.

The flow diagram shown in Figure 6.7 illustrates the "Processing loop" step of the master process. In the processing loop, the master process starts by checking the available worker processes, that might calculate the radio-coverage prediction for the next transmitter. It is worth pointing out that this step also serves as a stopping condition for the processing loop itself (see "Any worker still on?" in Figure 6.7). The active worker processes inform

the master process that they are ready to compute by sending an idle message (see "Wait for idle worker" in Figure 6.7). The master process then announces to the idle worker process that it is about to receive new data for the next calculation, and it dispatches the complete configuration of the transmitter to be processed (see "Send keep-alive message" and "Send transmitter data" steps, respectively, in Figure 6.7). This is only done in the case that there are transmitters for which the coverage prediction has yet to be calculated (see "Any transmitters left?" in Figure 6.7). The processing loop of the master process continues distributing the transmitter data among the worker processes, which asynchronously become idle as they finish the radio-prediction calculations they have been assigned by the master process. When there are no more transmitters left, all the worker processes announcing they are idle will receive a shutdown message from the master process, indicating to them that they should stop running (see "Send stop message" in Figure 6.7). The master process will keep doing this until all the worker processes have finished (see "Any worker still on?" in Figure 6.7), thus fulfilling the stopping condition of the processing loop.

Finally, the last step of the master process is devoted to creating the final output of the calculation, e.g., a raster map (see "Create final coverage prediction" in Figure 6.6). The final coverage prediction of all the transmitters is an aggregation from the individual path-loss results created by each of the worker processes during the "Processing loop" phase in Figure 6.6, which provides the source data for the final raster map. The aggregation of the individual path-loss results is accomplished by issuing an SQL query over the database tables containing them, in a similar way as in the serial version.

6.4.5.2 Worker processes on CPU

An essential characteristic of the worker processes is that they are completely independent of the GRASS, i.e., they do not have to run within the GRASS environment nor use any of the GRASS libraries to work. This aspect significantly simplifies the deployment phase to run PRATO on a computer cluster, since no GRASS installation is needed on the computing nodes hosting the worker processes.

One possibility to overcome the thread-safety limitation of the GRASS is to save the transmitter path-loss predictions through the master process, thus avoiding concurrent access. However, for the workers to send intermediate results back to the master process, e.g., as in [5, 82], is a major bottleneck for the scalability of a parallel implementation. In such case, the scalability is limited by the master process, because it must serially process the received results in order to avoid inconsistencies due to concurrent access. Instead, PRATO allows each of the worker processes to output its intermediate results into a DB, i.e., each path-loss prediction in its own table. Additionally, worker processes do this from an independent thread, which runs concurrently with the calculation of the next transmitter received from the master process. In this way, the overlap between the calculation and communication significantly hides the latency created by the result-dumping task, thus making better use of the available system resources.

The computations of the worker processes, the flow diagram of which is given in Figure 6.8, begin by receiving metadata about the transmitters and the geographical area from the master process during the initialization time (see "Receive broadcasted metadata" in Figure 6.8).

After the broadcasted metadata are received by all the worker processes, each one proceeds to inform the master process that it is ready (i.e., in an idle state) to receive the transmitter-configuration data that define which transmitter path-loss prediction to perform (see "Send idle message" in Figure 6.8). If the master process does not give the instruction to stop processing (see "Has stop message arrived?" in Figure 6.8), the worker process collects

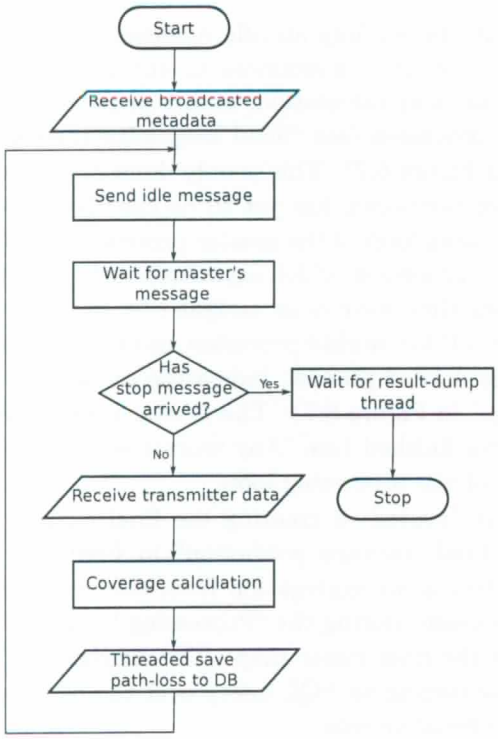

Figure 6.8: Flow diagram of a worker process.

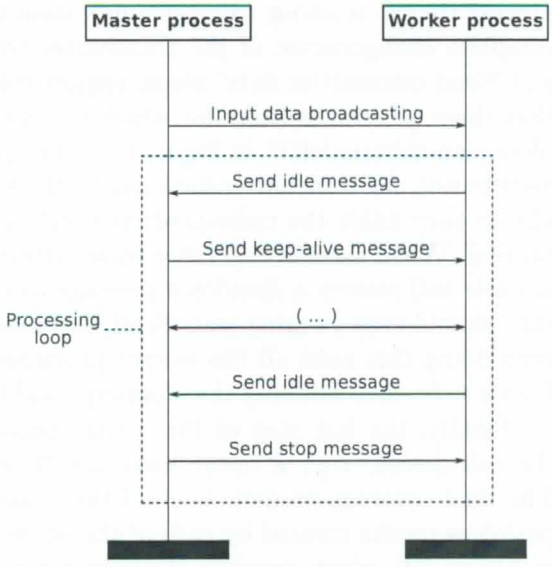

Figure 6.9: Communication diagram, showing the message passing between the master and a worker process.

the sub-region spatial data and the transmitter configuration (see "Receive transmitter data" in Figure 6.8). In the event that a stop message is received, the worker process will wait for any result-dumping thread to finish (see "Wait for result-dump thread" in Figure 6.8) before shutting down. The coverage calculation itself follows a similar design as the serial version (see "Coverage calculation" in Figure 6.8).

As mentioned before, the worker process launches an independent thread to save the path-loss prediction of the target transmitter into a DB table (see "Threaded save path-loss to DB" in Figure 6.8). It is important to note that there is no possibility of data inconsistency due to the saving task being executed inside a thread, since path-loss data from different workers belong to different transmitters and are, at this point of the process, mutually exclusive.

6.4.5.3 Worker processes on GPU

PRATO provides multi-GPU support for improving the execution performance on the computing nodes hosting the worker processes. The algorithmic adaptation from a CPU to a GPU is not a trivial task. This section focuses on the main modifications made to the radio-propagation algorithm in order for it to work on GPU hardware.

It is well known that the bandwidth of the PCI Express bus can cause a throughput bottleneck when a significant amount of data is transferred between a CPU and a GPU in a heterogeneous system [69]. Some researchers acknowledged that unless a full working set of data can fit into the memory on a GPU, the PCI Express will become the bottleneck of the system [120, 136]. For this reason, it is imperative to have as much data as possible allocated on the GPU itself.

In order to minimize the CPU-to-GPU memory transfers, the spatial data used by the radio-propagation algorithm was organized as explained in Section 6.4.5, i.e., using geographically-located, offset-based 2D matrices. However, the internal representation of the matrix elements was changed to use less memory. To this end, the clutter-matrix elements were represented as *unsigned char*, since they express signal loss in dB. It follows that for the radio-propagation prediction of one transmitter, the following matrices should be allocated on the GPU:

- one 2D matrix containing DEM data for the target subregion, the elements of which are *float* or *double*;
- one 2D matrix containing clutter data for the target subregion, the elements of which are *unsigned char*; and
- one 2D matrix containing the resulting path-loss prediction.

The dimension of all matrices is based on the transmission radius, within which the radio-propagation prediction should be calculated. The contents of the DEM and clutter matrices is constant throughout the calculation process. For this reason they were saved into read-only texture memory to take advantage of the faster access time (see Section 5.1, Chapter 5). Regarding the resulting path-loss matrix, each step of the radio-prediction algorithm is applied over the results of the previous step (see Figure 6.1 for a flow diagram of the steps involved), thus avoiding extra allocation on global memory or a data-transfer from/to the CPU.

6.4.5.4 Master-worker communication

Similar to [156, 157], the message-passing technique used in this work enables a better use of the available computing resources, both in terms of scalability and load balancing, while introducing a negligible overhead. This last point is supported by the experimental results, introduced in Section 6.5.3.

The first reason to implement the message-passing technique is to support heterogeneous computing environments. In particular, our approach focuses on taking full advantage of the hardware of each computing node, thus explicitly avoiding the bottlenecks introduced by the slowest computing node in the cluster. This problem appears when evenly distributing the data among the worker processes on disparate hardware, e.g., as in [6, 82], being more noticeable with a larger number of computing nodes and processes. In other words, computing nodes that deliver better performance have more calculations assigned to them. Moreover, in real-world scenarios, it is often the case that a large number of dedicated computing nodes featuring exactly the same configuration is difficult to find, i.e., not every organization owns a computer cluster.

A second reason for selecting a message-passing technique is related to the flexibility it provides for load balancing, which is of greater importance when dealing with extra data or information besides spatial data [77]. This can be seen in Figure 6.7, where the master process, before delivering the spatial subset and transmitter-configuration data, sends a message to the worker process, indicating that it is about to receive more work. This a priori meaningless message plays a key role in correctly supporting the asynchronous process communication. Notice that the subset of spatial data that a worker process receives is directly related to the transmitter for which the prediction will be calculated. Similar to [156, 157], this problem-specific property enables the use of a data-decomposition technique based on a block partition of spatial data, e.g., the DEM and clutter data.

In general, there are many different ways a parallel program can be executed, because the steps from the different processes can be interleaved in various ways and a process can make non-deterministic choices [139], which may lead to situations such as race conditions [37] and deadlocks. A deadlock occurs whenever two or more running processes are waiting for each other to finish, and thus neither ever does. To prevent PRATO from deadlocking, message sending and receiving should be paired, i.e., an equal number of send and receive messages on the master and worker sides [139]. Figure 6.9 depicts the master-worker message passing, from which the transmitter-data transmission has been excluded for clarity. Notice how each idle message sent from the worker process is paired with an answer from the master process, whether it is a keep-alive or a stop message.

6.5 Simulations

Considering the large computational power needed for predicting the radio-coverage of a real radio network, the use of a computer cluster is recommended. A computer cluster is a group of interconnected computers that work together as a single system. Computer clusters typically consist of several commodity PCs connected through a high-speed local-area network (LAN) with a distributed file system, like NFS [138]. One such system is the DEGIMA cluster [72] at the Nagasaki Advanced Computing Center (NACC) of the Nagasaki University in Japan. This system ranked in the TOP 500 list of supercomputers until June 2012[1], and in June 2011 it held the third place in the Green 500 list[2] as one of the most energy-efficient supercomputers in the world.

This section presents the simulations, and an exhaustive analysis of the performance and scalability of the parallel implementation of PRATO. The most common usage case for PRATO is to perform a radio-coverage prediction for multiple transmitters. Therefore, a straight-forward parallel decomposition is to divide a given problem instance by transmitter, for which each coverage prediction is calculated by a separate worker process.

The following simulations were carried out on 34 computing nodes of the DEGIMA cluster. The computing nodes are connected by a LAN, over a Gigabit Ethernet interconnect. As mentioned before, the reason for using a high-end computer cluster such as DEGIMA is to explore, by experimentation, the advantages and drawbacks of the considered methods. However, this does not imply any loss of generality if applying these principles over a different group of networked computers that do not operate as a computer cluster. Moreover, PRATO also supports parallel calculation of radio-propagation predictions for multiple cells by distributing the processes among the individual cores of a single CPU.

Each computing node of DEGIMA features one of two possible configurations, namely:

- Intel Core i5-2500T quad-core processor CPU, clocked at 2.30 GHz, with 16 GB of RAM; and
- Intel Core i7-2600K quad-core processor CPU, clocked at 3.40 GHz, also with 16 GB of RAM.

During the simulation runs, the nodes equipped with the Intel i5 CPU hosted the worker processes, whereas the master process and the PostgreSQL database server (version 9.1.4) each run on a different computing node, featuring an Intel i7 CPU. The DB server performed all its I/O operations on the local file system, which was mounted on an 8 GB RAM disk. During the simulations, the path-loss predictions of 5,120 transmitters occupied less than 4 GB of this partition. No GPU hardware was used for the following simulation sets.

[1] http://www.top500.org
[2] http://www.green500.org