International Journal of Geographical Information Science
Vol. 00, No. 00, Month 200x, 1–25

Geographic information systems are ideal candidates for applying parallel programming techniques, mainly because of the large data sets they usually handle. To help dealing with complex calculations over such data sets, we investigate the challenges and advantages of applying a work-pool parallel paradigm over a message-passing communication model. To this end, we present the design and implementation of a parallel radio-coverage prediction tool for the GRASS environment. The prediction calculation employs digital elevation models and land-usage data in order to analyze the radio coverage of a geographic area. We provide an extended analysis of the experimental results, which are based on real data from an LTE network currently deployed in Slovenia. According to the experiments, which are performed on a computer cluster, the presented methods allow the parallel radio-coverage prediction tool to reach great scalability, meaning it is able to tackle real-world-sized data sets while greatly reducing the processing time and maximizing hardware utilization. Moreover, we are able to solve problem instances, which sizes are out of reach of the reference implementation.

Keywords:
GRASS, GIS, parallel, LTE, simulation.

ISSN: 1365-8816 print/ISSN 1362-3087 online

DOI: 10.1080/1365881YYxxxxxxxx
http://www.informaworld.com

International Journal of Geographical Information Science
Vol. 00, No. 00, Month 200x, 2–25

RESEARCH ARTICLE

A GRASS GIS parallel module for radio-propagation predictions

(*Received 00 Month 200x; final version received 00 Month 200x*)

1. Introduction

Although the well-known and often cited Gordon Moore's prediction still holds (Moore *et al.* 1998), the fact is that for the last few years, CPU speed has hardly been improving. Instead, the number of cores within a single CPU is growing. This fact poses a challenge for software development in general and research in particular: a hardware upgrade will, most of the time, fail to deliver twice the computing capacity of its predecessor. However, since this commodity hardware is present in practically all modern desktop computers, it creates an opportunity for the exploitation of these parallel computing resources to enhance the performance of complex algorithms over large data sets. The challenge is thus to deliver the computing power of multi-core systems into a geographic information systems (GIS). Moreover, by accessing many of such computing nodes through a network connection, the palette of possibilities broadens.

A traditional approach when dealing with computationally expensive problem solving is to simplify the models in order to be able to execute their calculations within a feasible amount of time. Clearly, this 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 (Huang *et al.* 2012, Yin *et al.* 2012). 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 physically installing the base stations and antennas that will cover a certain geographical area. A higher deviation of these results increases the probability of making the wrong decisions at installation time, which may unnecessarily increase costs or even cause losses to mobile network operators.

A group of studies has successfully deployed high-performance computing (HPC) systems and techniques to solve different problems dealing with spatial data (Armstrong *et al.* 2005, Li *et al.* 2010, Guan *et al.* 2011, Yin *et al.* 2012, Osterman 2012, Tabik *et al.* 2012, Widener *et al.* 2012). Unfortunately, despite some rare exceptions, most of these works are application-specific and do not introduce general principles for joining GIS and HPC. There are several reasons for this, but the most widely known fact is that parallel programming and HPC often call for area experts in order to integrate these practices in a given environment (Clematis *et al.* 2003). Moreover, the wide range of options currently available create even more barriers for general users willing to reach HPC.

In this paper, we combine proved principles of HPC and introduce some new approaches in order to improve the performance speed of a GIS module for radio-propagation predictions. To this end, we implement a parallel radio-prediction tool for the open source Geographic Resources Analysis Support System (GRASS) (Neteler and Mitasova 2002).

For its architecture, we have focused on scalability, clean design and openness of the tool, inspired by the GRASS GIS. These facts make it an ideal candidate for showing the benefits of the presented patterns, while tackling radio-coverage predictions of big problem instances, e.g. real mobile networks containing thousands of transmitters over high-resolution terrains, optimization problems which require a large number of coverage evaluations, and big scale whole-country-sized simulations.

Within the introduced context, the provided guidelines may be used as a template for parallelization of other computationally-expensive tasks within the GRASS environment. Namely, the radio-coverage prediction problem tackled in this work presents some common points with many different problems within geographic information sciences. Each transmitter within the radio network works as a point of interest, around which certain phenomena is analyzed. As an example, think about wild fire control, where the fire starts at several points over a given area (Yin *et al.* 2012). Some weather-related analysis also follows this pattern (Huang *et al.* 2012). Similarly, agent-based simulations present several analogies, where the points of interest represent the deployed agents (Gong *et al.* 2012).

1.1. *Parallel computation on computer clusters*

Considering the high computational power needed for predicting the radio-coverage of a real mobile network, the use of a computer cluster is required. A computer cluster is a group of interconnected computers that work together as a single system. To reach high levels of parallel performance and scalability, this work discusses the key steps of parallel decomposition of big data sets, and the distribution of the computational load among the computing nodes that belong to the cluster. As a practical instance for introducing the afore-mentioned principles, we use the radio-coverage prediction problem for real mobile networks.

Computer clusters typically consist of several commodity PCs connected through a high-speed local network with a distributed file system, like NFS (Shepler *et al.* 2003). One such system is the DEGIMA cluster (Hamada and Nitadori 2010) 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 held the third place of the Green 500 list[2] as one of the most energy-efficient supercomputers in the world.

It is worth pointing out that the condition of the cluster working as a single system is not necessary in the context of this study. The introduced principles work equally well over any set of networked computers. The decision of using the DEGIMA cluster is related with the requirement of simulating large runs in order to assert the benefits and drawbacks of the presented methodology.

1.2. *Objectives*

As a proof-of-concept of the presented patterns, we develop a high-performance parallel radio prediction tool (PRATO) targeted at large real-world network environments, such as the ones currently deployed by several mobile operators around the world. Therefore, our focus is on the performance and scalability of PRATO, while other more dynamic

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

aspects of radio networks are not considered. Among these aspects are code distributions, details of handover, and dynamics related to radio resource management (Khan 2009).

The performance evaluation of PRATO in a distributed computing environment is a major objective of this work, since its implementation is founded on the parallel methods presented in this study. Furthermore, by presenting a detailed description of the design and implementation of PRATO, we provide guidelines and patterns for achieving higher efficiency levels for general task parallelization in GRASS GIS. Additionally, we introduce techniques to overcome several obstacles encountered during our research, as well as in related work, which significantly improve the performance and lower the complexity of the presented implementation, e.g. the inability to use GRASS in a threaded environment, lowering overhead of I/O operations, saving simulation results asynchronously and independently from GRASS, and improving load balancing with an asynchronous message-passing technique.

The paper is organized as follows. Section 2 gives a description of the radio-coverage prediction problem, including the radio-propagation model. Section 3 concentrates on the design principles and implementation details of the radio propagation tool, for the serial and parallel versions. Section 4 discusses the experimental results and their analysis. Finally, Section 5 gives an overview of relevant publications, describing how they relate to our work, before drawing some conclusions.

2. Radio-coverage prediction for mobile networks

2.1. *Background*

The coverage planning of radio networks remains 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 (Saleh *et al.* 2010, Shabbir *et al.* 2011, Siomina and Yuan 2007, Valcarce *et al.* 2009). One of the primary objectives of mobile-network planning is to efficiently use the allocated frequency band to assure that some geographic area of interest can be satisfactorily reached with the base stations 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. Nevertheless, radio-coverage prediction is a complex task, mainly due to the wide range of various 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 prediction is a computationally-intensive and time-consuming task, hence the importance of using fast and accurate tools. Additionally, since the number of deployed transmitters keeps growing with the adoption of modern standards (Saleh *et al.* 2010), 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.

PRATO is a high-performance radio-prediction tool for GSM (2G), UMTS (3G) and LTE (4G) radio networks. It is implemented as a module for the GRASS GIS. 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 or upgrading.

As a reference implementation, we have used the publicly available radio coverage prediction tool, developed in (Hrovat *et al.* 2010). The authors of this work have 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. This work implements some well-known radio propagation models, e.g. Okumura-Hata Hata (1980) and COST 231 Cichon and Kurner (1995), the later is explained in more detail in Section 2.2. Regarding the accuracy of the predicted values, the authors (Hrovat *et al.* 2010) report comparable results to those of a state-of-the-art commercial tool. Furthermore, to ensure that our implementation is completely compliant with the afore-mentioned reference, we have designed a comparison test 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 tested cases.

2.2. *Propagation modeling*

The COST-231 Walfisch-Ikegami radio-propagation model was introduced as an extension of the well-known COST Hata model (Sarkar *et al.* 2003, Shabbir *et al.* 2011). The suitability of this model comes from the fact that it distinguishes between line-of-sight (LOS) and non-line-of-sight (NLOS) conditions.

In this work, as well as in the reference implementation (Hrovat *et al.* 2010), the terrain profile is used for LOS determination. The wave-guide effect in streets of big cities is not taken into account, because the building data is not available. In order to compensate the missing data, we include a correction factor, based on the land usage (clutter data). This technique is also adopted by other propagation models for radio networks, like the artificial neural networks macro-cell model developed in (Neskovic and Neskovic 2010). Consequently, we introduce an extra term for signal loss due to clutter (L_{CLUT}), thus defining the LOS and NLOS path losses as

$$PL_{\mathrm{LOS}}(d) = 42.64 + 26\log(d) + 20\log(F) + L_{\mathrm{CLUT}} \tag{1}$$

and

$$PL_{\mathrm{NLOS}}(d) = L_0 + L_{\mathrm{RTS}} + L_{\mathrm{MSD}} + L_{\mathrm{CLUT}}, \tag{2}$$

where d is the distance (in kilometers) from the transmitter to the receiver point, F is the frequency (in MHz), L_0 is the attenuation in free space (in dB), L_{RTS} represents the diffraction from roof top to street, and L_{MSD} represents the diffraction loss due to multiple obstacles.

Equation (1) describes the path loss when there is LOS between the transmitter and the receiver. On the other hand, in NLOS conditions, the path loss is calculated as in Equation (2).

3. Design and implementation

3.1. *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 1, where the parallelograms of the flow diagram represent input/output (I/O) operations.

Our design follows a similar internal organization as the radio planning tool presented in (Hrovat *et al.* 2010), but with some essential differences. Specifically, our approach 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 database vendor, the generated output is formatted in plain text, which is then forwarded to the database server. Any further processing is achieved by issuing a query over the database tables that contain the partial results for each of the processed transmitters.

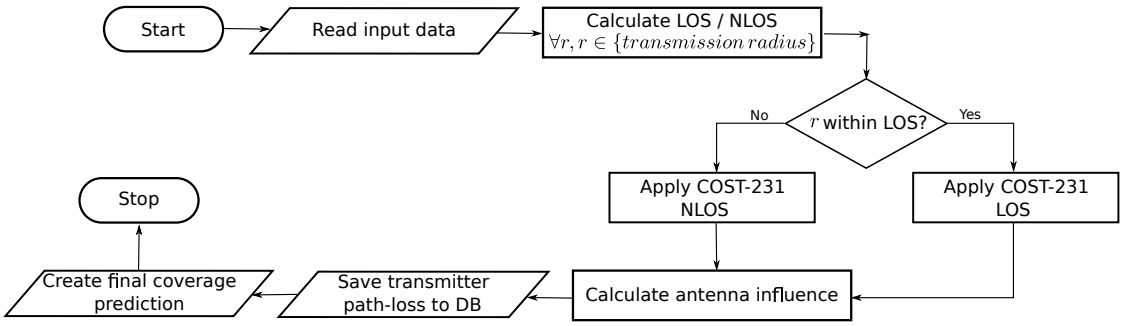

Figure 1. Flow diagram of the serial version.

3.1.1. Isotropic path-loss calculation

This step starts by calculating which receiver points, r, are within the specified transmission radius (see "transmission radius" in Figure 1). For these points, the LOS and NLOS conditions are calculated, with respect to the transmitter (see "Calculate LOS/NLOS" in Figure 1). The following step consists of calculating the path loss for an isotropic source (or omni antenna). This calculation is performed by applying the COST-231 path-loss model, which was previously introduced in Section 2.2, to each of the points within the transmission radius around the transmitter. Depending on whether the receiver point r is in LOS or NLOS, either Equation (1) or Equation (2) is respectively applied (see "Apply COST-231, LOS" or "Apply COST-231, NLOS" in Figure 1).

Figure 2 shows a portion of a raster map with an example result of the isotropic path-loss calculation. The color scale is given in dB, indicating the signal loss from the isotropic source, located in the center. Also, the hilly terrain is clearly distinguished due to LOS and NLOS conditions from the signal source.

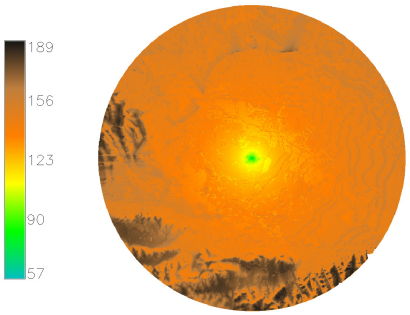

Figure 2. Example of raster map, showing the result of a path-loss calculation from an isotropic source.

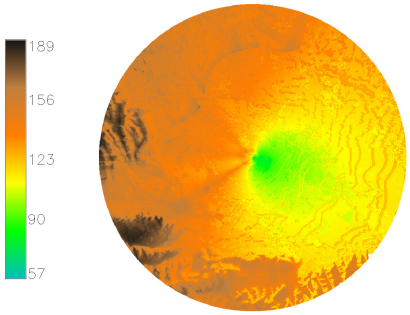

Figure 3. Example of raster map, showing the antenna influence over the isotropic path-loss result.

3.1.2. 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 1). Working on the in-memory results generated by the previous step, the radiation diagram of the antenna is taken into account, including beam direction, electrical and mechanical tilt. Figure 3 shows a portion of a raster map, where this calculation step has been applied to the results from Figure 2. Notice the distortion of the signal propagation that the antenna has introduced.

3.1.3. Transmitter path-loss prediction

In this step, the coverage prediction of the transmitter is saved in its own database table (see "Save transmitter path-loss to DB" in Figure 1), thus considerably enhancing the write performance during the result-dumping phase, which involves saving the path-loss results. This is accomplished by connecting the standard output of the developed module with the standard input of a database client. Naturally, the generated plain text should be understood by the database server itself.

3.1.4. Coverage prediction

The final radio coverage prediction, containing an aggregation of the partial path-loss results of the involved transmitters, is created in this step (see "Create final coverage prediction" in Figure 1). The received signal strength from each of the transmitters is calculated as the difference between its transmit power and path loss for the receiver's corresponding position. This is done for each point in the target area by executing an SQL query over the tables containing the path-loss predictions of each of the processed transmitters.

Finally, the output raster is generated, using the GRASS built-in modules *v.in.ascii* and *v.to.rast*, which create a raster map using the results of the above-mentioned query as input. The raster map contains the maximum received signal strength for each individual point, as shown in Figure 4. In this case, the color scale is given in dBm, indicating the received signal strength from the transmitters.

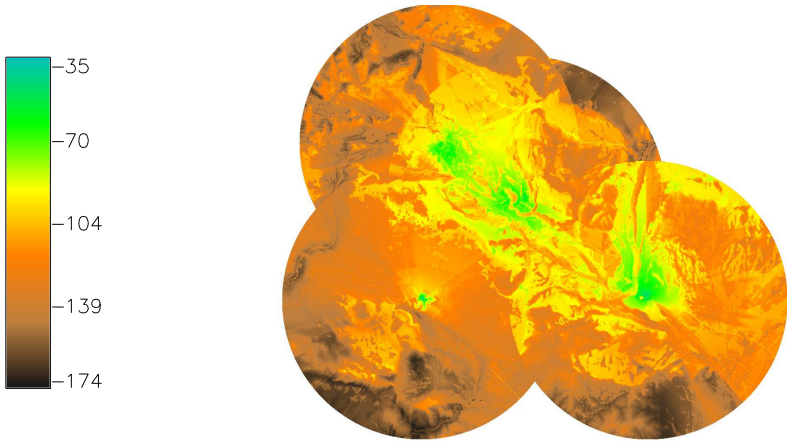

Figure 4. Example of a raster map, displaying the final coverage prediction of several transmitters over a geographical area. The color scale is given in dBm, indicating the received signal strength. Darker colors denote areas with reduced signal due to the shadowing effect of the hilly terrain.

3.2. *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 of the nodes in a computing cluster. To effectively use a shared memory multi-processor, PRATO uses POSIX threads to implement parallelism (Butenhof 1997). By using POSIX threads, multiple threads can exist within the same process while sharing its resources. For instance, an application using POSIX threads can execute multiple threads in parallel by using the cores of a multi-core processor, or use the system resources more effectively, thus avoiding process execution-halt due to I/O latency by using one thread for computing, while a second thread waits for an I/O operation to complete.

To use the computing resources of a distributed memory system, such as a cluster of processors, PRATO uses the Message Passing Interface (MPI) (Gropp *et al.* 1999). MPI is a message-passing standard, which defines syntax and semantics designed to function on a wide variety of parallel computers. 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. Its development is supported by a broadly-based committee of vendors, developers, and users.

In order to make the text more clear and to differentiate between the programming paradigms used from here on, we will refer to a POSIX thread simply as a 'thread' and a MPI process as a 'process'.

3.3. *Design of the parallel version*

Keeping our focus on the usability and performance of PRATO, we are introducing a new distributed implementation to overcome computational-time constraints that prevented the reference implementation from tackling big problem instances (Hrovat *et al.* 2010).

Some authors have already published their work on implementing parallel versions of GRASS modules for solving different time-consuming tasks (Akhter *et al.* 2007, Campos *et al.* 2012, Sorokine 2007). However, one major drawback of GRASS as a parallelization environment is that it is not thread-safe, meaning that concurrent changes to the same data set have undefined behavior. To overcome this problem, we present a technique that saves the simulation results asynchronously and independently from the GRASS environment, e.g. into an external database system. This database system works also as an input source, serving data to GRASS, whether it is used to aggregate the partial results of the path-loss prediction or to visualize them. It is worth pointing out that any database system may be used. By this we mean relational, distributed (Özsu and Valduriez 2011) or even those of the NoSQL type (Stonebraker 2010). Nevertheless, in this study we use a central relational database system, since they are the most popular and widely available ones.

We also introduce a methodology that allows the parallel implementation to be almost completely GRASS independent. This means that a GRASS installation is needed on only one of the nodes, i.e. the master node of the target computer cluster, thus improving the usability of the introduced methods. Also, a message-passing technique is proposed to distribute the work-load among nodes hosting the worker processes. Using this technique, 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.

3.3.1. Master process

As it has been suggested before, the parallel version of PRATO follows a master-worker model. The master process, for which the flow diagram is given in Figure 5, is the only component that should be run from 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 5, and it is done in a similar way as in the serial version. In the next step, the master process dynamically initiates the worker processes using the available computing nodes (see "Dynamic worker-process spawning" in Figure 5), based on the amount of transmitters (or points of interest) for which the coverage prediction should be calculated. This means that master process never starts more worker processes than there are transmitters to be processed. However, if the number of transmitters is larger than the amount of available computing nodes, the master process can assign several transmitters to each of the worker processes. For distributing the work among the worker processes, the master process proceeds to decompose the loaded raster data into arrays of basic-data-type elements, e.g. floats or doubles, before dispatching them to the multiple worker processes (see "Input data broadcasting" in Figure 5). In this case, the decomposition of the data applies to the digital-elevation and the clutter data only, but it could be applied to any point-based data set, vector or raster. In the next step, the master process starts a message-driven processing loop (see "Processing loop" in Figure 5), which main task is to assign and distribute the configuration data of different transmitters among idle worker processes. If using points of interest instead of transmitters, this configuration data translates into local properties that differ among points of interest.

The flow diagram shown in Figure 6 depicts in more detail the steps inside the "Processing loop" step of the master process. In the processing loop, the master process starts by checking the available worker processes, which will 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). The active worker processes inform the master process they are ready to compute by sending an idle message (see "Wait for idle worker" in Figure 6). The master process then announces

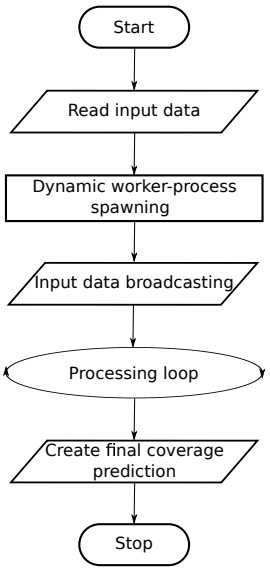

Figure 5. Flow diagram of the master process.

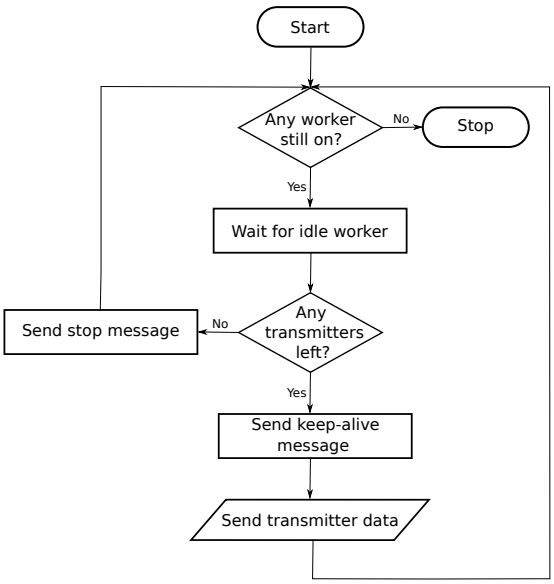

Figure 6. Flow diagram of the "Processing loop" step of the master process.

the idle worker process 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). This is only done in case there are transmitters for which the coverage prediction has yet to be calculated (see "Any transmitters left?" in Figure 6). The processing loop of the master process continues to distribute transmitter data among worker processes, which asynchronously become idle as they finish the coverage-prediction calculations for the transmitters 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 them to stop running (see "Send stop message" in Figure 6). The master process will keep doing this until all worker processes have finished (see "Any worker still on?" in Figure 6), 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 5). The final coverage prediction of all transmitters is an aggregation from the individual path-loss results created by each of the worker processes during the "Processing loop" phase in Figure 5, which provides the source data for the final raster map. The aggregation of the individual transmitter path-loss results is accomplished by issuing an SQL query over the database tables containing the partial results, in a similar way as in the serial version.

3.3.2. Worker processes

An essential characteristic of the worker processes is that they are completely independent from 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.

The computations of the worker processes, for which the flow diagram is given in Figure 7, are initialized by data that are received from the master process at initialization time (see "Receive broadcasted data" in Figure 7). It is important to note that the received data contain the transmitter and terrain-profile information which is common to all the coverage-prediction calculations, therefore making each worker process capable of processing any given transmitter.

The reason for the worker processes to be independent from GRASS arises from the design of GRASS itself. Specifically, the existing GRASS library, distributed with the GRASS GIS package, is not thread-safe, because GRASS was designed as a system of small stand-alone modules and not as a library for multi-threaded programs (Blazek and Nardelli 2004). Because of this limitation, it is not an option for a parallel implementation to create separate threads for each worker process, since this would mean worker processes should wait for each other to finish, before accessing the target data. Consequently, the scalability of such implementation would be very limited.

One possible solution to overcome this limitation would be to save the transmitter path-loss prediction result through the master process, thus avoiding concurrent access. However, sending intermediate results back to the master process from the workers would represent a major bottleneck for the scalability of the parallel version, since the results generated by a parallel computation would have to be serially processed by the master process alone. Instead, our approach allows each of the worker processes to output its results into an external database server, following an asynchronous and decoupled design. Each of the transmitter path-loss prediction results are saved in separate tables, i.e. one for each transmitter or point of interest. Moreover, worker processes do this from an

independent thread, which runs concurrently with the calculation of the next transmitter received from the master process. The overlap between calculation and communication achieved by the use of an auxiliary thread completely hides the latency created by the result dumping task, and makes better use of the system resources.

After the broadcasted data are received by all the worker processes, each worker process proceeds to inform the master process that it is ready (i.e. in an idle state) to receive the transmitter-configuration data that defines which transmitter path-loss prediction to perform (see "Send idle message" in Figure 7). If the master process does not instruct to stop processing (see "Has stop message arrived?" in Figure 7), the worker process collects the transmitter configuration sent (see "Receive transmitter data" in Figure 7). However, in case a stop message is received, the worker process will wait for result-dumping threads to finish (see "Wait for result-dump threads" in Figure 7) before shutting down. The coverage calculation itself follows a similar design as the serial version (see "Coverage calculation" in Figure 7) and it is executed for the received transmitter or point of interest.

As it was mentioned before, the worker process launches an independent thread to save the path-loss prediction of the target transmitter to a database table (see "Threaded save path-loss to DB" in Figure 7). 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 least at this point of the process, mutually exclusive.

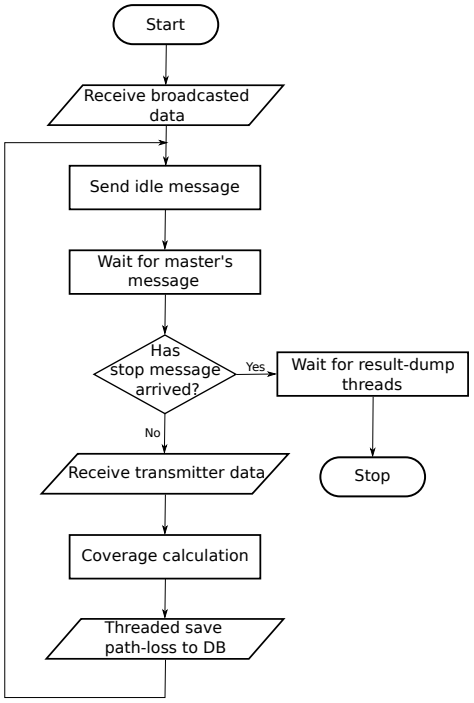

Figure 7. Flow diagram of a worker process.

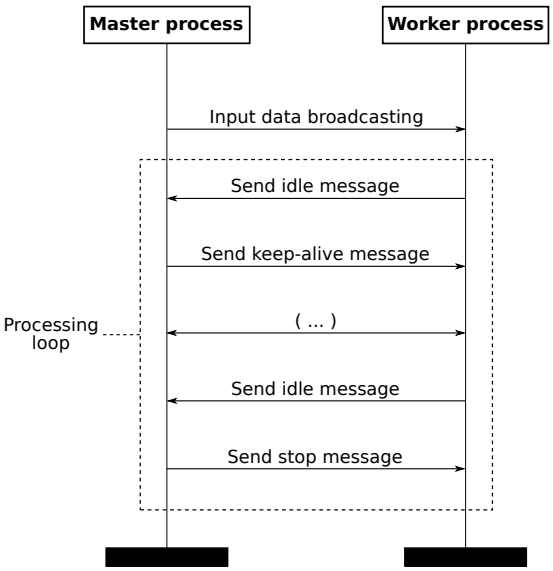

Figure 8. Communication diagram, showing message passing between master and one worker process.

3.3.3. Master-worker communication

The selected message-passing technique introduced in this work enables a better use of the available computing resources, both in terms of scalability and load balancing. This

argument is supported by the experimental results, introduced in Section 4.

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 possible bottlenecks introduced by the slowest computing node in the cluster. In other words, computing nodes that deliver better performance get more calculations assigned to the worker processes they host. The main advantages of this technique are simplicity and negligible overhead, which contrast with more elaborated approaches for parallel-task allocation in heterogeneous clusters (Bosque and Pastor 2006).

A second reason for selecting a message-passing technique is related to the flexibility it provides for load balancing, which is of great importance on heterogeneous clusters. This can be seen in Figure 7 where the master process, before delivering the transmitter-configuration data, sends a message to the worker process indicating that it is about to receive more work. This a priori meaningless message has a key role in correctly supporting computer clusters. 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 (Siegel and Avrunin 2007), which may lead to situations such as race conditions (Clemencon *et al.* 1995) 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, being equal number of send and receive messages on the master and worker sides (Siegel and Avrunin 2007).

Figure 8 depicts a diagram of the master-worker message passing, from which the transmitter-data transmission has been excluded for clarity. Note 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.

4. Simulations

This section presents the simulations and analysis of the parallel version of PRATO. Our aim is to provide an exhaustive analysis of the performance and scalability of the parallel implementation in order to determine if the objectives of this work are fulfilled. 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 it has been mentioned before, the reason for using a high-end computer cluster as DEGIMA is to explore by experimentation the advantages and drawbacks of the introduced methods. However, this does not imply any loss of generality when applying these principles over a different group of networked computer, acting as a computer cluster.

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 host the worker processes, whereas the master process and the PostgreSQL database server (version 9.1.4) run each on a different computing node, featuring an Intel i7 CPU. The database server performs all its I/O operations on the local file system, which is mounted on a 8 GB RAM disk.

All nodes are equipped with a Linux 64-bit operating system (Fedora distribution). As the message passing implementation we use OpenMPI, version 1.6.1, which has been manually compiled with the distribution-supplied gcc compiler, version 4.4.4.

4.1. *Test networks*

To test the parallel performance of PRATO, we have prepared different problem instances that emulate real radio networks of different sizes. In order to create synthetic test data-sets with an arbitrary number of transmitters, we use the data of a group of 10 transmitters, which we randomly replicate and distribute over the whole target area. The configuration parameters of these 10 transmitters were taken from the LTE network deployed in Slovenia by Telekom Slovenije, d.d. The path-loss predictions are calculated using the COST-231. The digital elevation model has an area of 20,270 km^2, with a resolution of 25 m^2. The clutter data extends over the same area with the same resolution, and it contains different levels of signal loss based on the land usage. For all the points within a radius of 20 km around each transmitter, we assume that the receiver is positioned 1.5 m above the ground, and the frequency is set to 1,843 MHz.

4.2. *Weak scalability*

This set of simulations is meant to analyze the scalability of the parallel implementation in cases where the workload assigned to each process (one MPI process per processor core) remains constant as we increase the number of processor cores and the total size of the problem, i.e. the number of transmitters deployed over the target area is directly proportional to the number of processor cores and worker processes. We do this by assigning a constant number of transmitters per core while increasing the number of cores hosting the worker processes. Consequently, we tackle larger radio-network instances as we increase the number of cores. Here we test for the following numbers of transmitters per worker/core: $\{5, 10, 20, 40, 80\}$, by progressively doubling the number of workers per core from 1 to 128.

Problems particularly well-suited for parallel computing exhibit computational costs that are linearly dependent on the problem size. This property, also referred to as algorithmic scalability, means that proportionally increasing both the problem size and the number of cores results in a roughly constant time to solution. Therefore, with this set of experiments, we would like to investigate how well-suited the coverage-prediction problem is for parallel computing environments.

4.2.1. Results and discussion

The results collected after the simulations for the weak-scalability experiments are shown in Table 1. All measurements express wall-clock times in seconds for each problem instance, defined as number of transmitters per core (TX/core). Wall-clock time represents real time that elapses from the start of the master process to its end, including time that passes waiting for resources to become available. They are plotted in Figure 9.

The time measurements observed from the weak-scalability results show that the wall-

Table 1. Execution wall-clock times (in seconds) of the simulations for the weak-scalability experiments.

	Number of cores							
TX/core	1	2	4	8	16	32	64	128
5	92	99	118	122	123	124	125	126
10	140	152	171	175	177	179	180	182
20	244	260	278	282	284	285	287	290
40	451	470	491	497	500	502	504	509
80	865	892	920	925	928	931	937	948

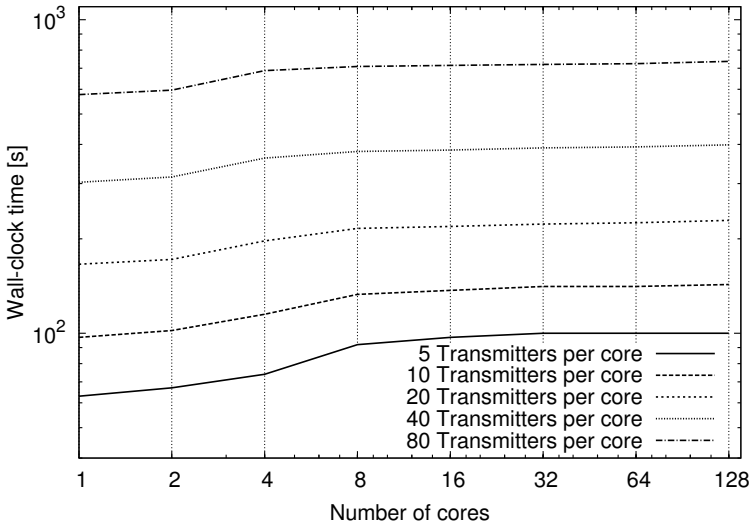

Figure 9. Measured wall-clock time for weak-scalability experiments as shown in Table 1. Experiments performed assigned one MPI worker process per available core. The wall-clock time axis is expressed in base-10 logarithmic scale, whereas the axis representing the number of cores is expressed in base-2 logarithmic scale.

clock times do not grow rapidly, especially when the number of cores is more than 8. Moreover, these times are almost constant for bigger problem instances, revealing that the achieved level of scalability gets close-to-linear as the amount of transmitters-per-core increases. Certainly, the parallel version of PRATO scales especially well when challenged with a big number of transmitters (10,240 for the biggest instance) over 128 cores. This fact shows PRATO would be able to calculate the radio coverage prediction for real networks in a feasible amount of time, since many operational radio networks have already deployed a comparable number of transmitters, e.g. the 3G network within the Greater London Authority area, in the UK (Ofcom 2012).

Not being able to achieve perfect weak scalability is due to a number of factors. Specifically, the overhead time of the serial sections of the master process grow proportionally with the number of cores, although the total contribution of this overhead remains low for large problem sizes. Moreover, the communication overhead grows linearly with the number of cores used.

To confirm these arguments, we analyze the times of each of the steps taken by the master process relative to the total processing time. To this end, we have created plots for four problem instances 10, 20, 40 and 80 transmitters per core, which are shown in Figure 10. The relative-processing time plots follow the formula

$$RT = \frac{t_{\mathrm{rd}} + t_{\mathrm{ps}} + t_{\mathrm{db}} + t_{\mathrm{pl}} + t_{\mathrm{cp}}}{t_{\mathrm{total}}}, \tag{3}$$

where t_{rd} is the "Read input data" wall-clock time, t_{ps} is the wall-clock time of the "Dynamic worker-process spawning" step, t_{db} is the wall-clock time of the "Input data broadcasting" step, t_{pl} is the wall-clock time of the "Processing loop" step, t_{cp} is the wall-clock time of the "Create final coverage prediction" step, and t_{total} is the total wall-clock processing time. For a reference of the different steps taking part of the master process, see Figure 5.

From the relative-times plots, we see that, as we increase the number of nodes, the largest fraction of the run-time is spent on the parallel processing of transmitters, which scales notably well for larger problem instances. The plotted relative times show that there is no dependency between the relative processing times and the number of cores used, confirming the good weak-scalability properties noted before. Additionally, in all three plots we may observe a "jump" in the relative time for the "Input data broadcasting" step that takes place when comparing the result from 4 to 8 cores, i.e. from one to two computing nodes, as each node hosts "1 worker per core" or a total of "4 workers per node". This "jump" is due to the use of network communication when more than one computing node participates in the parallel processing. In addition, we may also conclude that the network infrastructure has not been saturated with the data-passing load, since the relative times for input-data broadcasting do not grow exponentially from 8 cores onward. Regarding the "Create final coverage prediction" step, we may see that as we

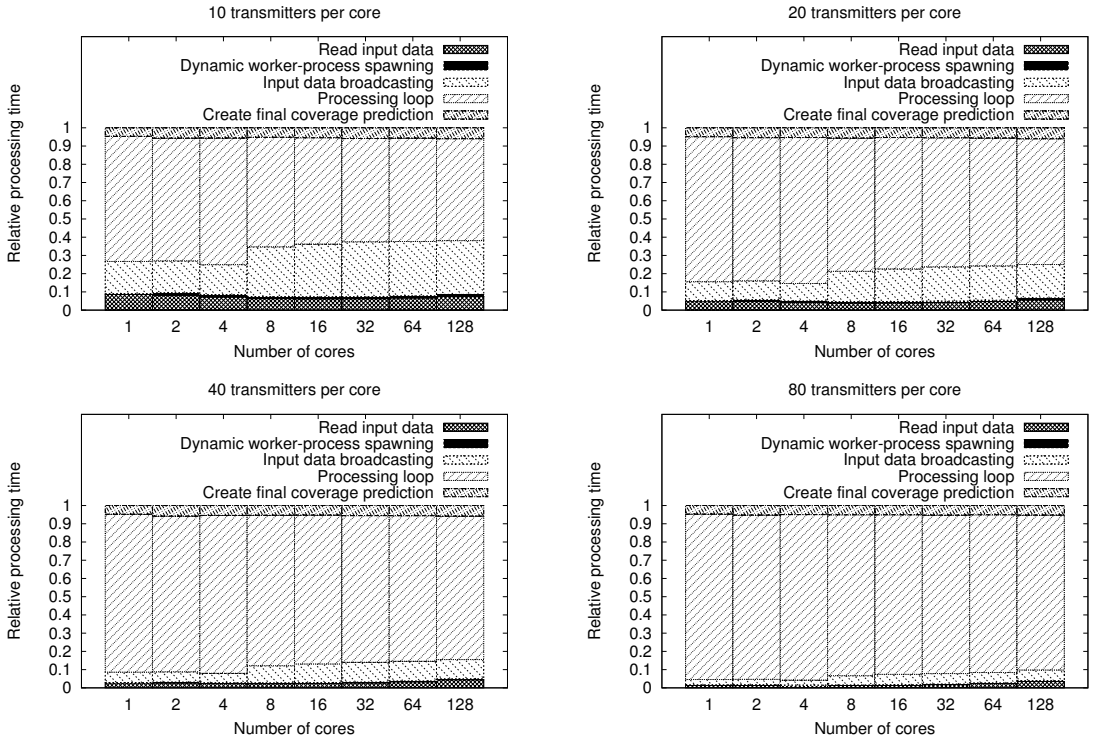

Figure 10. Relative times for the weak-scalability experiments. The relative-processing time axes are expressed in linear scale, whereas the axes representing the number of cores are expressed in base-2 logarithmic scale.

increase the number of cores the relative times grow proportionally for all three problem sizes.

4.3. *Strong scalability*

This set of simulations is meant to analyze the impact of increasing the number of computing cores for a given problem size, i.e. the number of transmitters deployed over the target area does not change, while only the number of cores used is increased. Here we test for the following number of transmitters: $\{64, 128, 256, 512, 1024, 2048, 4096\}$, by gradually doubling the number of workers per core from 1 to 128 for each problem size.

4.3.1. Results and discussion

The results of the time measurements collected after the simulations for the strong-scalability experiments are shown in Table 2. All times are expressed in seconds. These wall-clock time measurements are plotted in Figure 11.

Table 2. Execution wall-clock times (in seconds) of the simulations for the strong-scalability experiments.

	Number of transmitters						
No. cores	64	128	256	512	1024	2048	4096
1	714	1392	2740	5437	10830	21562	43217
2	386	734	1419	2791	5535	10996	21987
4	232	408	751	1432	2811	5549	11042
8	155	242	409	754	1441	2817	5549
16	113	156	244	414	759	1447	2821
32	92	114	159	245	414	760	1449
64	82	94	115	159	245	420	764
128	-	83	94	116	159	248	423

The time measurements show that small problem sizes per core have a relatively large proportion of serial work and communication overhead. Therefore, the performance deteriorates as the number of transmitters per core approaches one. It can be observed in Figure 11 that as we increase the number of transmitters used to solve a given problem size, the slope of the curve generated by the progression of wall-clock times tends to a flat line, i.e. as we increase the number of transmitters there is no reduction in computational time. This idea is more clearly noted in the test with smaller problem instances, e.g. 64, 128 and 256 transmitters. In contrast, for the problems with a number of transmitters larger than 512, the relative contribution of the non-parallel steps to the wall-clock time is smaller, and a larger portion of the time is spent on computing the transmitters coverage in parallel (see Section 3.3 for details on the steps of PRATO algorithm). A more detailed discussion of the reasons for the loss of parallel efficiency will be presented towards the end of this section.

In order to observe how well the application scales when compared against a base case, we have also measured the performance of the parallel implementation in terms of the speedup, which is defined as

$$S(NP) = \frac{execution\,time\,for\,base\,case}{execution\,time\,for\,NP\,cores}, \tag{4}$$

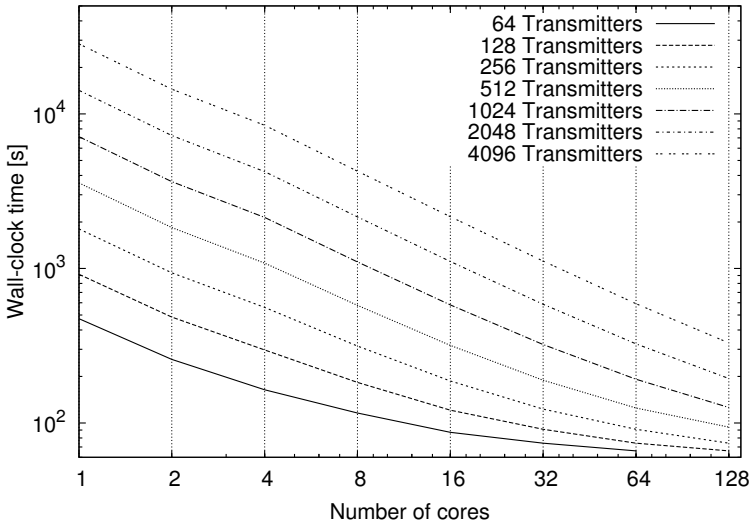

Figure 11. Measured wall-clock time for strong-scalability experiments as shown in Table 2. Experiments performed assigned one MPI worker process per available core. The wall-clock time axis is expressed in base-10 logarithmic scale, whereas the axis representing the number of cores is expressed in base-2 logarithmic scale.

where NP is the number of cores executing the worker processes. As the base case for comparisons we have chosen the parallel implementation running on only one core and decided against using the serial implementation. We consider that the serial implementation is not a good base comparison for the parallel results as it does not reuse resources between each transmitter coverage calculation and it does not overlap I/O operations with transmitter computations. In practice, this means that several concatenated runs of the serial version would be considerably slower than the parallel but single worker implementation, because the serial implementation is not able to use all of the memory bandwidth and computing resources simultaneously. Therefore such comparison would be entirely biased towards the parallel implementation, showing super-linear scaling and speedups which would not be real, as the parallel version is better equipped to make use of the system resources by means of multiple threads.

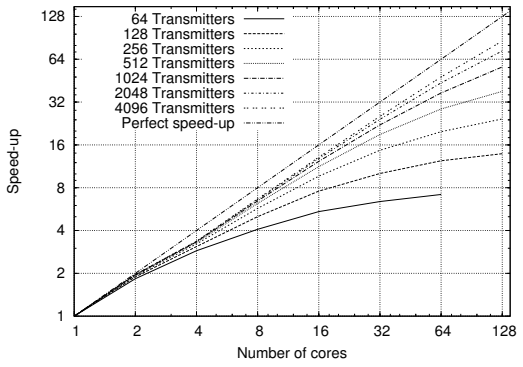

Figure 12. Measured speedup for strong-scalability experiments. The speedup axis is expressed in base-2 logarithmic scale, whereas the axis representing the number of cores is expressed in base-2 logarithmic scale.

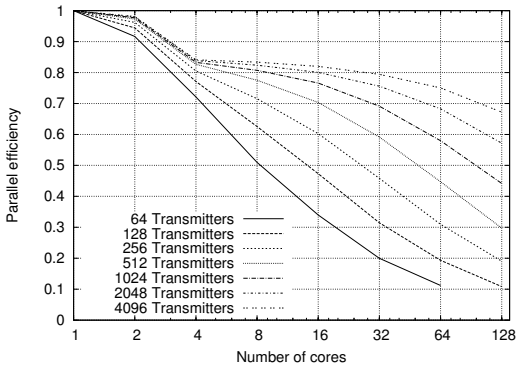

Figure 13. Measured parallel efficiency for strong-scalability experiments. The parallel-efficiency axis is expressed in linear scale, whereas the axis representing the number of cores is expressed in base-2 logarithmic scale.

Using the speedup metric, linear scaling is achieved when the obtained speedup is equal to the total number of processors used. However, it should be noted that perfect speedup is almost never achieved, due to the existence of serial stages within an algorithm and communication overheads of the parallel implementation (Cruz Villaroel 2010).

Figure 12 shows the speedup of the parallel implementation for up to 128 cores (running one worker process per node), and compares seven different problem sizes with 64, 128, 256, 512, 1024, 2048 and 4096 transmitters deployed over the target area. The number of transmitters used in these problem sizes are comparable to several operational radio networks that have already been deployed in England, e.g. Bedfordshire County with 69 base stations, Cheshire County with 132 base stations, Hampshire County with 227 base stations, West Midlands with 414 base stations, and Greater London Authority with 1086 base stations (Ofcom 2012). Moreover, consider that it is common for a single base station to host multiple transmitters.

We can see that the significant reductions in wall-clock time for large problem sizes shown in Figure 11 are directly correlated with the speedup factors shown in Figure 12.

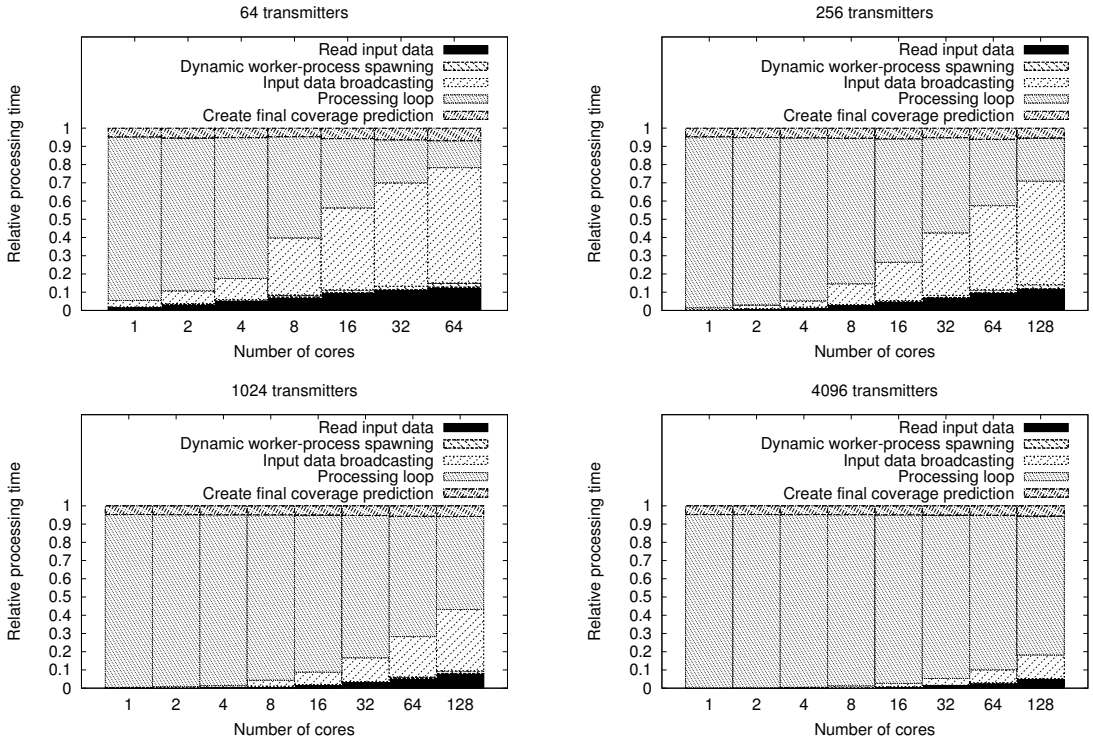

Figure 14. Relative times for the strong-scalability experiments. The relative-processing time axes are expressed in linear scale, whereas the axes representing the number of cores are expressed in base-2 logarithmic scale.

To study how well PRATO utilizes the available computing resources we consider the parallel efficiency of the implementation, i.e. how well the parallel implementation makes use of the available processor cores. The definition of parallel efficiency is as follows:

$$E(NP) = \frac{S(NP)}{NP}, \tag{5}$$

where $S(NP)$ is the speedup as defined in Equation (4), and NP is the number of cores executing worker processes. Figure 13 shows the parallel efficiency of the parallel implementation for different problem sizes as we increase the number of processing cores.

The ideal case for a parallel application would be to utilize all available resources, in which case the parallel efficiency would be constantly equal to one as we increase the core count. From the plot in Figure 13, we may observe that the efficiency is less than one, hence the computational resources are under utilized. In accordance to the previous analysis, the under utilization of the computing resources is more significant for the smaller problem sizes, where number of assigned transmitters per core approaches one. This is due to the increased relative influence introduced by serial and communication overheads, without which the parallel implementation would not be feasible. On the other hand, the relative time contribution of the serial and communication overheads is significantly reduced as the work-load per core increases. Unsurprisingly, these results confirm what it has previously been suggested during the weak-scaling analysis, i.e. it is not worth parallelizing small problem instances over a large number of nodes, since the time reduction due to computations that make use of the extra parallel resources is surpassed by the extra parallel initialization and communication overhead.

Similarly as in the weak-scaling test, we study the relative contribution of each of the steps of the master process as we increase the number of cores used for a fixed problem size. In this case, we have created plots for four problem instances, namely 64, 256, 1024 and 4096 transmitters, which are shown in Figure 14. The relative times shown are calculated using the formula depicted in Equation (3).

We may observe the non-parallel steps comprising "Read input data", "Dynamic worker-process spawning", "Input data broadcasting" and "Final coverage prediction" contribute with a larger portion of time as we increase the number of cores, because the total wall-clock processing time decreases. Additionally, the low parallel efficiency for small problem sizes, particularly for 64 and 256 transmitters (left-most plots in Figure 14), is validated with the relative small proportion of the radio-coverage calculation ("Processing loop") compared to the serial steps of the process.

4.4. *Load balancing*

In this section, we analyze the level of utilization of the computing resources available at the computing nodes hosting the worker processes. Computing-resource utilization is achieved by partitioning the computational workload and data across all processors. Efficient workload distribution strategies should be based on the processor speed, memory hierarchy and communication network (Clarke *et al.* 2011).

The parallel implementation of PRATO performs load-balancing using point-to-point messages (see Section 3.3.3) between master and worker processes. When a worker process issues an idle message (see "Send idle message" in Figure 8), the worker process will not continue until the message arrives to the master process. A similar situation occurs when the master process signals a worker back, whether to indicate it to shutdown or to continue working. Since the process-to-core mapping is one-to-one, blocking messages typically waste processor cycles on a computing node (Bhandarkar *et al.* 2001). Specifically, we would like to verify the penalties that such synchronization technique has on the scalability of the parallel implementation.

We evaluate the load empirically (Watts and Taylor 1998) by using the following metric as an indicator of the load balancing among processes:

$$LB(NP) = \frac{minimum\,execution\,time\,among\,NP\,cores}{processing\,loop\,time\,of\,master\,process}, \quad (6)$$

where NP is the number of cores executing worker processes. Taking the processing-loop time of the master process ensures we measure the overhead of the message passing during the time the coverage prediction is being executed by the workers. This means that the time measurement is performed excluding the serial parts of the process, i.e. after the common data have been broadcasted to all worker processes ("Input data broadcasting" in Figure 5), until the beginning of the last step ("Create final coverage prediction" in Figure 5).

High performance is achieved when all cores complete their work within the same time, hence showing a load-balancing factor of one. On the other hand, lower values indicate disparity between the run times of the various worker processes sharing the parallel task, thus reflecting load imbalance.

4.4.1. Results and discussion

For this set of experiments, we have chosen the same problem sizes as for strong scalability in Section 4.3, where the coverage predictions are calculated up-to 128 cores, running on 32 computing nodes.

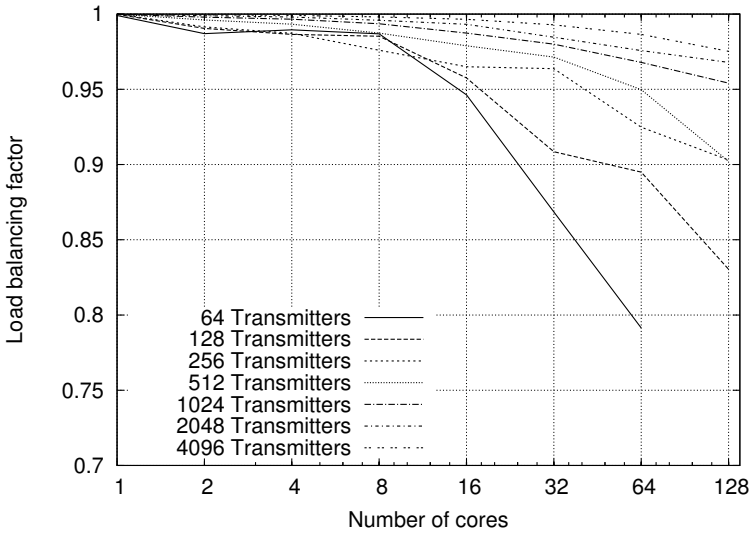

Figure 15. Load balancing among worker processes.

From the plot shown in Figure 15, it is clear that the influence of the message-passing overhead over the processing time is inversely proportional to the amount of work each worker process receives. Additionally, for the biggest problem instances (1024, 2048 and 4096 transmitters), parallel-process execution times are within 95% of a perfect load-balancing factor, and within 90% for problem sizes with 256 and 512 transmitters, showing a very good performance of the dynamic task assignment, driven by our message-passing technique. For problem instances of 64 and 128 transmitters, the parallel-process times are within 80% of the perfect load balancing, showing that, as the number of transmitters per core approaches one, latencies introduced by several hardware and OS-specific factors (e.g. TurboBoost, process affinity, etc.) are influential over the total processing time. Particularly, message-passing is not able to compensate these latencies as it is executed only once per worker process.

It is worth pointing out that the very good load-balancing factors shown here are not only merit of the message-passing technique. The result dumping of partial path-loss predictions, performed by the worker processes in a separate thread into an external database server, prevents data synchronization from occurring at each iteration of the parallel process, consequently improving the load-balancing factors significantly.

5. Related work

The reported results in (Hrovat *et al.* 2010) show a comparable quality to those of a professional radio-planning tool. Since the results of the conducted comparison tests showed identical results between PRATO and this work, we may conclude that PRATO reaches solutions of comparable quality to those of a professional tool. However, a performance comparison with this work has not been carried out, because it only deals with serial implementations.

The task-parallelization problem within the GRASS environment has been addressed by several authors in different works. For example, in (Campos *et al.* 2012), the authors present a collection of GRASS modules for watershed analysis. Their work concentrates on different ways of slicing raster maps to take advantage of a potential MPI implementation, but there are no guidelines for work replication. Moreover, the hardware specification, on which the experiments have been run, is missing, making it very difficult to build upon this work.

On the field of high-performance computing, the authors of (Akhter *et al.* 2007) have presented implementation examples of a GRASS raster module, used to process vegetation indexes for satellite images, for MPI and Ninf-G environments. The main drawback with their methodology is the compulsory use of GRASS libraries in all the computing nodes that take part in the parallel calculation, making them more difficult to setup. Moreover, the authors explicitly 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. It is worth pointing out that in the parallel implementation of PRATO we specifically address this problem with our message-passing technique.

Similarly, in (Huang *et al.* 2011), the parallel implementation of the inverse distance weighting interpolation algorithm is presented. Although it is not explicitly noted, it can be concluded that the computing nodes make use of the GRASS environment, again making them more difficult to setup. Moreover, since the amount of work is evenly distributed among all processes (including the master one), their approach would also show decreased efficiency in heterogeneous environments.

Some years ago, grid computing received a lot of attention as a way of accessing the extra computational power needed for spatial analysis of large data sets (Armstrong *et al.* 2005, Vouk 2008, Wang 2010). However, several obstacles are still preventing this technology from being widely used. Namely, its adoption requires not only hardware and software compromises of the involved parts, but also a behavioral change at the human level (Armstrong *et al.* 2005).

In (Yin *et al.* 2012), the authors present a parallel framework for GIS integration. Based on the principle of spatial dependency, they lower the calculation processing time by backing it with 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 is

an additional effort to achieve the presented goals, since the implementation of a fully functional GIS (or "thick GIS" as the authors call it) is required on both the desktop client and in the parallel environment. On the other hand, their implementation uses proprietary software, somewhat reducing the possibilities of adaptation for different GIS.

In (Gong *et al.* 2012), the authors present an agent-based approach for simulating spatial interactions. Their approach decomposes the entire landscape into equally-sized regions, which are in turn processed by a different core of a multi-core CPU. Despite this, the agent interaction at the region border greatly affects the efficiency of the algorithm, showing that an asynchronous approach, as the one used by PRATO, would greatly improve the overall efficiency of the system. Although their work is geared towards CPUs with a high number of cores instead of a computing cluster, it would be interesting to apply a methodology as the one presented in this study in order to push all agent communications to shared memory, thus improving the efficiency at region borders.

6. Conclusion

We have presented PRATO, a functional parallel radio-coverage prediction tool for radio networks. The tool, as well as the patterns for exploiting the computing power of a group of networked computers, namely a computer cluster, are intended to be used for spatial analysis and decision support. The introduced techniques, combined with the use of a database system, deliver a platform for parallel and asynchronous computation, that is completely independent from the GIS used, in this case the GRASS environment. Consequently, a GIS installation is needed on only one of the nodes, thus simplifying the required system setup and greatly enhancing the applicability of this methodology in different environments.

Also, the proposed message-passing technique fairly distributes the work-load among nodes hosting the worker processes. Hence, computing nodes featuring more capable hardware receive more work than those with weaker configurations, thus maximizing utilization of the available computing resources.

The extensive simulations results, performed on the DEGIMA cluster of the Nagasaki Advanced Computing Center, have been analyzed to determine the level of scalability of the implementation, as well as the impact of the presented methods for parallel-algorithm design aimed at spatial-data processing.

The conducted analysis shows that PRATO is able to calculate the radio-coverage prediction of real-world mobile networks in a reduced amount of time with a high scalability level. The promising results also show the great potential of our approach to parallelize different time-consuming tasks for GIS. Optimization problems, where thousands of simulations take part of the evaluation step during an optimization process, are also very good candidates for this approach. Still, further research is needed to fully demonstrate this point.

Encouraged by the favorable results, further work will include abstracting the introduced principles and methodology into a multi-purpose library for GRASS GIS, which shall be published as free and open source software. By implementing such tool for spatial problem solving and decision support, we aim at completely validating the suitability and usefulness of the presented methods.

Nevertheless, as PRATO is also a free and open-source software project[1], it can be

[1]The source code is available for download from http://cs.ijs.si/benedicic/

readily modified and extended to support, for example, other propagation models and post-processing algorithms. This characteristic defines a clear advantage when compared to commercial and closed-source tools.

References

Akhter, S., Chemin, Y., and Aida, K., 2007. Porting a GRASS raster module to distributed computing Examples for MPI and Ninf-G. *OSGeo Journal*, 2 (1).

Armstrong, M.P., Cowles, M.K., and Wang, S., 2005. Using a computational grid for geographic information analysis: a reconnaissance. *The Professional Geographer*, 57 (3), 365–375.

Bhandarkar, M., *et al.*, 2001. Adaptive load balancing for MPI programs. *Computational Science-ICCS 2001*, 108–117.

Blazek, R. and Nardelli, L., 2004. The GRASS server. *In*: *Proceedings of the Free/Libre and Open Source Software for Geoinformatics: GIS-GRASS Users Conference*.

Bosque, J. and Pastor, L., 2006. A parallel computational model for heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems*, 17 (12), 1390.

Butenhof, D., 1997. *Programming with POSIX threads*. Addison-Wesley Professional.

Campos, I., *et al.*, 2012. Modelling of a Watershed: A Distributed Parallel Application in a Grid Framework. *Computing and Informatics*, 27 (2), 285–296.

Cichon, D. and Kurner, T., 1995. Propagation prediction models. *COST 231 Final Rep.*

Clarke, D., Lastovetsky, A., and Rychkov, V., 2011. Dynamic Load Balancing of Parallel Computational Iterative Routines on Highly Heterogeneous HPC Platforms. *Parallel Processing Letters*, 21 (02), 195–217.

Clematis, A., Mineter, M., and Marciano, R., 2003. Guest editorial: high performance computing with geographical data. *Parallel Computing*, 29 (10), 1275–1279.

Clemencon, C., *et al.*, 1995. An implementation of race detection and deterministic replay with MPI. *EURO-PAR'95 Parallel Processing*, 155–166.

Cruz Villaroel, F., 2010. Particle flow simulation using a parallel FMM on distributed memory systems and GPU architectures. Thesis (PhD). Faculty of Science, University of Bristol.

Gong, Z., *et al.*, 2012. Parallel agent-based simulation of individual-level spatial interactions within a multicore computing environment. *International Journal of Geographical Information Science*, (ahead-of-print), 1–19.

Gropp, W., Lusk, E., and Skjellum, A., 1999. *Using MPI: portable parallel programming with the message passing interface*. Vol. 1. MIT press.

Guan, Q., Kyriakidis, P.C., and Goodchild, M.F., 2011. A parallel computing approach to fast geostatistical areal interpolation. *International Journal of Geographical Information Science*, 25 (8), 1241–1267.

Hamada, T. and Nitadori, K., 2010. 190 TFlops astrophysical N-body simulation on a cluster of GPUs. *In*: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–9.

Hata, M., 1980. Empirical formula for propagation loss in land mobile radio services. *Vehicular Technology, IEEE Transactions on*, 29 (3), 317–325.

Hrovat, A., *et al.*, 2010. Radio coverage calculations of terrestrial wireless networks using an open-source GRASS system. *WSEAS Transactions on Communications*, 9 (10),

646–657.

Huang, F., *et al.*, 2011. Explorations of the implementation of a parallel IDW interpolation algorithm in a Linux cluster-based parallel GIS. *Computers & Geosciences*, 37 (4), 426–434.

Huang, Q., *et al.*, 2012. Using adaptively coupled models and high-performance computing for enabling the computability of dust storm forecasting. *International Journal of Geographical Information Science*, (ahead-of-print), 1–20.

Khan, F., 2009. *LTE for 4G Mobile Broadband: Air Interface Technologies and Performance*. Cambridge University Press.

Li, X., *et al.*, 2010. Parallel cellular automata for large-scale urban simulation using load-balancing techniques. *International Journal of Geographical Information Science*, 24 (6), 803–820.

Moore, G.E., *et al.*, 1998. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86 (1), 82–85.

Neskovic, A. and Neskovic, N., 2010. Microcell electric field strength prediction model based upon artificial neural networks. *AEU-International Journal of Electronics and Communications*, 64 (8), 733–738.

Neteler, M. and Mitasova, H., 2002. *Open source GIS: a GRASS GIS approach*. Vol. 689. Kluwer Academic Pub.

Ofcom, 2012. *Table of base station totals* [online]. : Independent regulator and competition authority for the UK communications industries. Available from: http://stakeholders.ofcom.org.uk/sitefinder/table-of-totals/ [Accessed October 2012].

Osterman, A., 2012. Implementation of the r.cuda.los module in the open source GRASS GIS by using parallel computation on the NVIDIA CUDA graphic cards. *Elektrotehniški Vestnik*, 79 (1-2), 19–24.

Özsu, M.T. and Valduriez, P., 2011. *Principles of distributed database systems*. Springer.

Saleh, A., *et al.*, 2010. On the coverage extension and capacity enhancement of inband relay deployments in LTE-Advanced networks. *Journal of Electrical and Computer Engineering*, 2010, 4.

Sarkar, T., *et al.*, 2003. A survey of various propagation models for mobile communication. *Antennas and Propagation Magazine, IEEE*, 45 (3), 51–82.

Shabbir, N., *et al.*, 2011. Comparison of Radio Propagation Models for Long Term Evolution (LTE) Network. *arXiv preprint arXiv:1110.1519*.

Shepler, S., *et al.*, 2003. Network file system (NFS) version 4 protocol. *Network*.

Siegel, S. and Avrunin, G., 2007. Verification of halting properties for MPI programs using nonblocking operations. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 326–334.

Siomina, I. and Yuan, D., 2007. Minimum pilot power for service coverage in WCDMA networks. *Wireless Networks*, 14 (3), 393–402.

Sorokine, A., 2007. Implementation of a parallel high-performance visualization technique in GRASS GIS. *Computers & geosciences*, 33 (5), 685–695.

Stonebraker, M., 2010. SQL databases v. NoSQL databases. *Communications of the ACM*, 53 (4), 10–11.

Tabik, S., Zapata, E., and Romero, L., 2012. Simultaneous computation of total viewshed on large high resolution grids. *International Journal of Geographical Information Science*.

Valcarce, A., *et al.*, 2009. Applying FDTD to the coverage prediction of WiMAX femtocells. *EURASIP Journal on Wireless Communications and Networking*, 2009, 1–13.

Vouk, M.A., 2008. Cloud Computing–Issues, Research and Implementations. *Journal of*

Computing and Information Technology, 16 (4), 235–246.

Wang, S., 2010. A cyberGIS framework for the synthesis of cyberinfrastructure, GIS, and spatial analysis. *Annals of the Association of American Geographers*, 100 (3), 535–557.

Watts, J. and Taylor, S., 1998. A practical approach to dynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 9 (3), 235–248.

Widener, M.J., Crago, N.C., and Aldstadt, J., 2012. Developing a parallel computational implementation of AMOEBA. *International Journal of Geographical Information Science*, 26 (9), 1707–1723.

Yin, L., *et al.*, 2012. A framework of integrating GIS and parallel computing for spatial control problems–a case study of wildfire control. *International Journal of Geographical Information Science*, 26 (4), 621–641.