A high-performance radio coverage prediction tool for mobile networks

Lucas Benedičič[a,*], Felipe A. Cruz[b], Tsuyoshi Hamada[b], Peter Korošec[c]

[a] *Research and Development Department, Telekom Slovenije, d.d., Cigaletova 15, SI-1000, Ljubljana, Slovenia*
[b] *Nagasaki Advanced Computer Center, Nagasaki University, 1-14 Bunkyo-machi, Nagasaki-city, Nagasaki, 852-8521, Japan*
[c] *Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, SI-1000, Ljubljana, Jamova cesta 39, SI-1000, Ljubljana, Slovenia*

Abstract

We present the design and implementation of a parallel radio-coverage prediction tool for GRASS GIS. Based on the serial implementation of a similar tool, we propose a master/slave programming model for our parallel implementation. We provide an extended analysis of the results of the experiments, which are based on real data from a UMTS network currently deployed in Slovenia. According to the experimental results, which are performed on a computer cluster, the parallel radio-coverage prediction tool has very good scalability properties, meaning it is able to calculate the radio-coverage prediction of real-world networks, greatly reducing processing time and maximizing performance. Moreover, we are able to solve problem instances, which sizes are out of reach of the serial implementation.

Keywords: Mobile networks, coverage, prediction, parallel, GRASS, GIS

1. Introduction

More than 20 years have passed since the world's first GSM mobile call was made in Finland. Still, the coverage planning of the radio network remains a key problem that all mobile operators have to deal with. Moreover, it has proven to be a fundamental issue not only in GSM networks but also in modern standards, such as the third generation (3G) UMTS and the fourth generation (4G) LTE Advanced [1, 2, 3, 4]. The complexity of the problem means that radio-coverage prediction can be a computationally-intensive and time-consuming task, hence the importance of fast and accurate prediction tools.

*Corresponding author
Email address: lucas.benedicic@telekom.si (Lucas Benedičič)
URL: http://csd.ijs.si/benedicic (Lucas Benedičič)

Despite various options of commercial tools specialized in radio-propagation modeling, the common thread among them is the restricted nature of its usage, mostly dominated by black-box implementations. This fact induces lack of adaptability, sometimes even combined with cumbersome user interfaces that are not suitable for big batch jobs, involving thousands of transmitters.

To tackle the afore-mentioned issues, we present a high-performance parallel radio-prediction tool for the open source Geographic Resources Analysis Support System (GRASS). For its design, we have focused on scalability and open nature of the tool, inspired by the GRASS geographic information system (GIS). These facts make it an ideal candidate for calculating radio-predictions of real mobile networks containing thousands of transmitters.

1.1. Objectives

The main goal of this work is to develop a high-performance parallel radio prediction tool (PRATO) to be used in large real-world network environments. To achieve this, we focus on the performance and scalability of PRATO, while other more dynamic aspects of radio networks are not considered. Among these aspects are code distributions, details of (soft) handover, and dynamics related to radio resource management.

Additionally, we introduce techniques to overcome several obstacles encountered during our research as well as in related work, which significantly improve the quality and performance of the presented implementation; e.g. inability to use GRASS in a threaded environment, lowering overhead of I/O operations, saving simulation results asynchronously and independently from GRASS, improving load balancing with a new message-passing technique.

The paper is organized as follows. Section 2 gives a description of the radio prediction tool, including the propagation model and GRASS GIS. 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. Description of the radio coverage prediction tool

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 Geographical Information System (for details of GRASS see Section 2.2). 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 by Hrovat et al. [5]. This work implements some well-known radio propagation models, e.g. Okumura-Hata and COST 231, the later is explained in more detail in Section 2.1. Regarding the accuracy of the predicted values, the authors report comparable results to those of a state-of-the-art commercial tool. To ensure that our implementation is completely compliant with the afore-mentioned reference, we have designed a comparison test that consists of running both the reference and PRATO with the same input parameters. The test results from PRATO and the reference implementation are identical.

2.1. Propagation modeling

The COST-231 Walfisch-Ikegami radio-propagation model was introduced as an extension of the well-known COST Hata model [6, 2], designed for frequencies above 2000 MHz. The suitability of this model comes from the fact that it distinguishes between line-of-sight (LOS) and non-line-of-sight (NLOS) conditions. Equation (1) describes the path loss when there is LOS between the transmitter and the receiver.

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

where d is the distance (in kilometers) from the transmitter to the receiver point, and F is the frequency, expressed in MHz.

On the other hand, while in NLOS conditions, the path loss is calculated as in Equation (2).

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

where L_0 is the attenuation in free space, L_{RTS} represents the diffraction from roof top to street, and L_{MSD} represents the diffraction loss due to multiple obstacles.

In this work, as well as in the reference implementation [5], the terrain profile is used for LOS determination. Besides, 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 by Neskovic et al. [7]. Consequently, both Equations (1) and (2) have an extra term for signal loss due to clutter (L_{CLUT}), thus redefining the LOS and NLOS path losses as

$$PL_{\mathrm{LOS}}(d) = 42.64 + 26\log(d) + 20\log(F) + L_{\mathrm{CLUT}}, \tag{3}$$

and

$$PL_{\mathrm{NLOS}}(d) = L_0 + L_{\mathrm{RTS}} + L_{\mathrm{MSD}} + L_{\mathrm{CLUT}}. \tag{4}$$

2.2. GRASS Geographical Information System

As the software environment for PRATO we have chosen GRASS (Geographic Resources Analysis Support System) [8], which is a free and open-source software (FOSS) project that implements a Geographical Information System (GIS).

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 high performance computing 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 [9] and imposes the availability of the source code. This allows us to make potential modifications to the system, thus adapting it for the parallel computation environment.

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 in the flow diagram represent input/output (I/O) operations.

Our approach explicitly avoids the modular design to prevent the overhead of I/O operations for communicating data between the components of the modular architecture. Instead, a monolithic design is used, in which all the steps for generating the radio coverage prediction are calculated inside one GRASS module. The module employs a direct connection to an external 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.

3.1.1. Read input parameters

All input data are read in the first step ("Read input data" in Figure 1), e.g. digital elevation model, clutter data, transmitter configurations, and other service-dependent settings. Their format differs based on the data they contain. Namely, GRASS raster files are used for the digital elevation model and clutter data, whereas a text file is used for the transmitter configurations and other simulation-dependent options.

3.1.2. Isotropic path-loss calculation

The first step here is to calculate which receiver points, r, are within the specified transmission radius ("transmission radius" in Figure 1). For these points, the LOS and NLOS conditions are calculated, with respect to the transmitter ("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 2.1, 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 (3) or Equation (4) is respectively applied ("Apply COST-231, LOS" or "Apply COST-231, NLOS" in Figure 1).

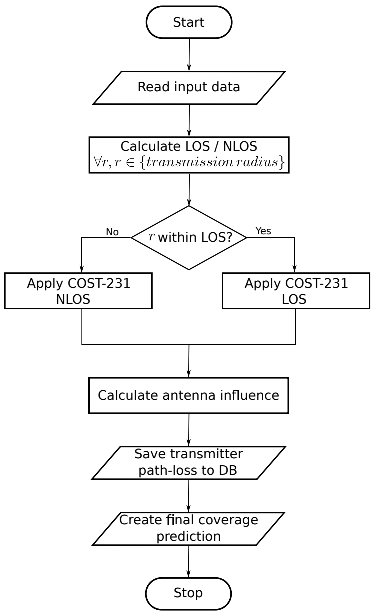

Start
Read input data
Calculate LOS / NLOS
$\forall r, r \in \{transmission\, radius\}$
r within LOS?
No
Yes
Apply COST-231 NLOS
Apply COST-231 LOS
Calculate antenna influence
Save transmitter path-loss to DB
Create final coverage prediction
Stop

Figure 1: *Flow diagram of the serial version.*

3.1.3. Antenna diagram influence

Working on the in-memory results generated by the previous step, this step considers the antenna radiation diagram of the current transmitter and its influence over the isotropic path-loss calculation ("Calculate antenna influence" in Figure 1).

3.1.4. Transmitter path-loss prediction

In this step, the coverage prediction of the transmitter is saved in its own database table ("Save transmitter path-loss to DB" in Figure 1), thus considerably enhancing the write performance during result dumping. This is accomplished by connecting the standard output of the developed module with the standard input of a database client.

3.1.5. Coverage prediction

The final radio coverage prediction, containing an aggregation of the partial path-loss predictions of the involved transmitters, is created in this step ("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.

3.2. Multi-paradigm parallel programming

The implementation methodology adopted for PRATO follows a multi-paradigm parallel programming approach in order to fully use the resources of a computing cluster. To effectively use a shared memory multi-processor, PRATO uses POSIX threads to implement parallelism [10].

Consideration of the high computational-intensity of predicting the radio-coverage of a real mobile network, the use of a computer cluster is required, i.e. a group of interconnected computers that work together as a single system.

Such computer clusters typically consist of several commodity PCs connected through a high-speed local network with a distributed file system, like NFS [11]. One such system is the DEGIMA cluster [12] at the Nagasaki Advanced Computing Center of the Nagasaki University. 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 of the world.

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 proccess as a 'process'.

3.3. Design of the parallel version

Keeping our focus on the 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 [5].

One major drawback of GRASS as a parallelization environment is that it is not thread-safe, meaning that concurrent changes to a 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.Also, a message-passing technique is proposed to distribute the workload 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 Fig. 2, 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 Fig. 2, and it is done in a

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

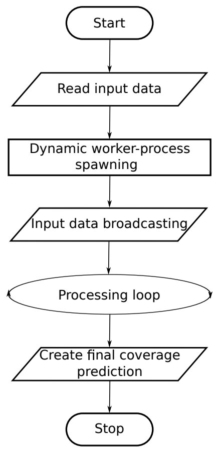

Start
Read input data
Dynamic worker-process spawning
Input data broadcasting
Processing loop
Create final coverage prediction
Stop

Figure 2: *Flow diagram of the master process.*

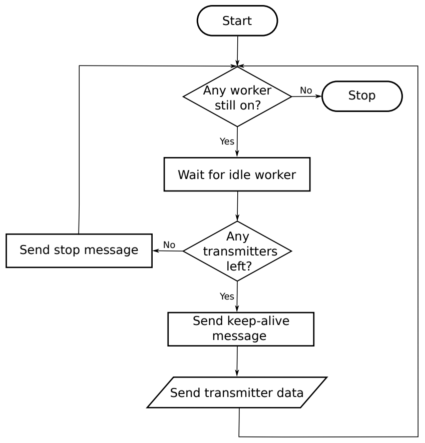


Figure 3: *Flow diagram of the "Processing loop" step of the master process.*

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 the "Dynamic worker-process spawning" step in Fig. 2), based on the amount of transmitters for which the coverage prediction should be calculated. In other words, this means that master process never starts more worker processes than the number of transmitters to be processed. However, most often there are more transmitters than available computing nodes, therefore, 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 the "Input data broadcasting" step in Fig. 2). The decomposition of the data applies to the digital-elevation and the clutter data only. In the next step, the master process starts a message-driven processing loop (see the "Processing loop" step in Fig. 2), which main task is to assign and distribute the configuration data of the different transmitters among idle worker processes.

The flow diagram shown in Fig. 3 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 the "Any worker still on?" step in Fig. 3). The active worker processes inform the master process they are ready to compute by sending an idle message (see the "Wait for idle worker" step in Fig. 3). The master process then announces 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 the "Send keep-alive message" and "Send transmitter data" steps, respectively, in Fig. 3). This is only done in case there are transmitters for which the coverage prediction has yet to be calculated (see the "Any transmitters left?" step in Fig. 3). 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 the "Send stop message" step in Fig. 3). The master process will keep doing this until all worker processes have finished (see the "Any worker still on?" step in Fig. 3), thus fulfilling the stopping condition of the processing loop.

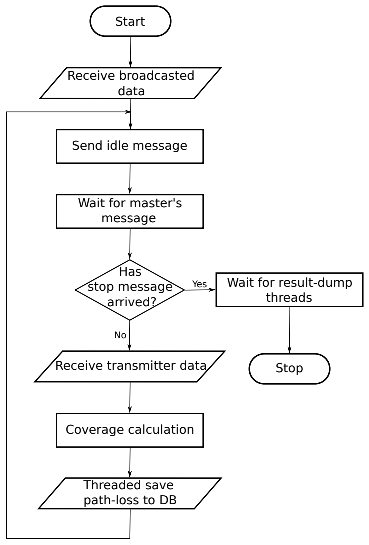
Finally, the last step of the master process is devoted to creating the final output of the calculation, e.g. a raster map (see the "Create final coverage prediction" step in Fig. 2). The final coverage prediction of all transmitters is an aggregation from the individual path-loss results created by each of the worker processes during the "Processing loop" phase in Fig. 2, which provides the source data for the final raster map. The aggregation of the individual transmitter path-loss results is accomplished 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 Fig. 4, are initialized by data that are received from the master process at initialization time (see the "Receive broadcasted data" step in Fig. 4). 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.

Because concurrent access to data within GRASS by multiple processes yields undefined behavior, i.e. it is not thread-safe, the results generated by the worker processes cannot be directly saved into the GRASS data set. 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, following a similar design as the serial version. Moreover, worker processes do this from an independent thread (see the "Threaded save path-loss to DB" step in Fig. 4), which runs concurrently with the calculation of the next transmitter received from the master process. When compared to the serial version, the overlap between calculation and communication achieved by the use of an auxiliary thread completely hides the latency created by the result dumping task,



Start
Receive broadcasted data
Send idle message
Wait for master's message
Has stop message arrived?
Yes
No
Wait for result-dump threads
Stop
Receive transmitter data
Coverage calculation
Threaded save path-loss to DB

Figure 4: *Flow diagram of one worker process.*

Table 1: Wall-clock times (in seconds) of the simulation results for weak scalability.

	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

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 (in an idle state) to receive the transmitter-configuration data that defines which transmitter path-loss prediction to perform (see the “Send idle message” step in Fig. 4). If the master process does not instruct to stop processing (see the “Has stop message arrived?” step in Fig. 4), the worker process collects the transmitter configuration sent (see the “Receive transmitter data” step in Fig. 4). However, in case a stop message is received, the worker process will wait for result-dumping threads to finish (see the “Wait for result-dump threads” step in Fig. 4) before shutting down. The coverage calculation itself follows a similar design as the serial version (see the “Coverage calculation” step in Fig. 4) and it is executed for the received transmitter.

4. Simulations

This section presents the simulations and analysis of the parallel version 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, and share a NFS partition, from which all input and intermediate files are accessed.

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 is the only node not writing or reading data from the common NFS partition. Instead, all I/O is done 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 are taken from the UMTS 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 same as the clutter data, which contains different levels of signal loss based on the land usage. For all the points within a transmission 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 2040 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: $\{5, 10, 20, 40, 80\}$, and we increase the number of workers/cores from 1 to 128 in powers of 2.

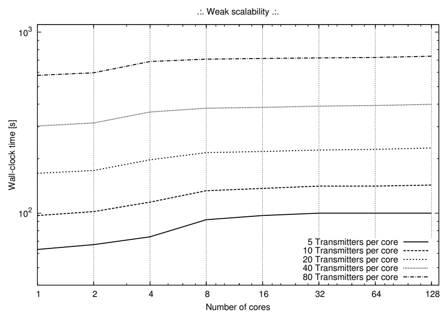


Figure 5: *Measured wall-clock time for weak-scalability experiments as shown in Table 1. Experiments performed assigned one MPI worker process per available core. The wall-clock time axis is expressed in base-10 logarithmic scale, whereas the axis representing the number of cores is expressed in base-2 logarithmic scale.*

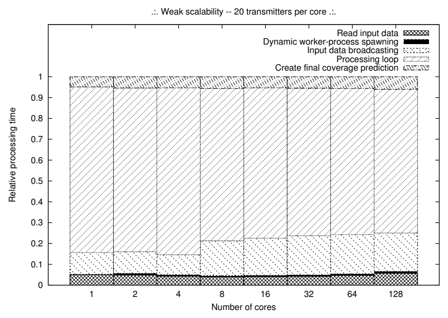


Figure 6: *Relative times for the weak-scalability experiments for the 20 transmitters per worker/core case.*

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. 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 Fig. 5, where 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.

The time measurements observed from the weak-scalability results show that the wall-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 (10240 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 [13].

Not being able to achieve perfect weak scalability is due to a number of factors. Namely, 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. Also, 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 three problem instances 20 and 80 transmitters per core, which are shown in Fig. 6 and Fig. 7, respectively. The relative-processing-time plots follow the formula

$$RT = \frac{t_{\mathrm{rd}} + t_{\mathrm{ps}} + t_{\mathrm{db}} + t_{\mathrm{pl}} + t_{\mathrm{cp}}}{t_{\mathrm{total}}}, \tag{5}$$

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 Fig. 2.

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

4.3. 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 [14].

The parallel implementation of PRATO performs load-balancing using point-to-point messages between master and worker processes. When a worker process issues an idle message (see the "Send idle message" step in Fig. **??**), the worker process will block until the message arrives to the master process. A similar situation occurs when the master process signals a worker back, whether

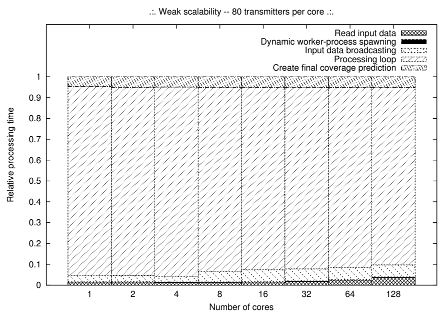


Figure 7: *Relative times for the weak-scalability experiments for the 80 transmitters per worker/core case.*

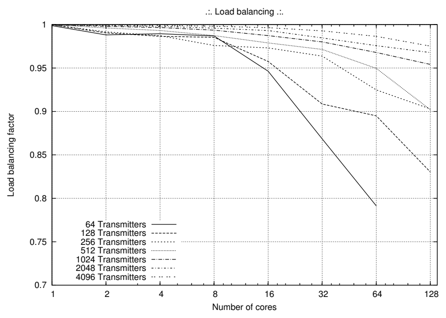


Figure 8: *Load balancing among worker processes.*

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 [15]. Specifically, we would like to verify the penalties that such synchronization technique has on the scalability of the parallel implementation.

We use the following metric as an indicator of the load balancing among processes:

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

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 at the same 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.

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.

4.3.1. Results and discussion

For this set of experiments, we have chosen problem sizes with 64, 128, 256, 512, 1024, 2048 and 4096 transmitters deployed over the target area, where the coverage predictions are calculated up-to 128 cores, running on 32 computing nodes.

From the plot shown in Fig. 8, 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 10% 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 to one, latencies introduced by several hardware and OS-specific factors (e.g. TurboBoost, process affinity, etc.) are influential over the total process 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, thus improving the load-balancing factors.

5. Related work

As it has been mentioned before, the reference implementation for PRATO is the work done by Hrovat et al. [5]. The reported results 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 performed, because it only deals with serial implementations.

A different example of a GIS-based open-source radio planning tool, called Q-Rap, has been presented in [16]. Developed by the University of Pretoria and the Meraka Institute of South Africa, the software was made publicly available in May 2010. Its design is geared towards an end-user tool with a graphical user interface, not appropriate for big batch jobs involving thousands of transmitters, or even parallel job execution. It is implemented as a plug-in for the Quantum GIS (QGIS) open source system [17].

On the field of high-performance computing, Akhter et al. [18] have presented implementation examples of a GRASS raster module, used to process vegetation indexes for satellite images, for MPI and Ninf-G environments. A performance restriction appears due to the computing nodes being fixed to a specific range, since the input data are equally distributed among worker processes, creating an obstacle for load balancing in heterogeneous environments.

6. Conclusion

We have presented the design and implementation of PRATO, a parallel radio-coverage prediction tool for GRASS GIS. Extensive simulations were run in the DEGIMA computer cluster of the Nagasaki Advanced Computing Center. The results have been analyzed to determine the level of scalability of the implementation.

The conducted analysis shows that PRATO is able to calculate the radio-coverage prediction of real-world radio networks in a reduced amount of time with a high scalability level. Particularly, the gathered results suggest that our approach would be also beneficial in the area of mobile network optimization, where thousands of simulations take part of the evaluation step during an optimization process. Still, further research is needed on how this method may be exploited.

Nevertheless, as PRATO is FOSS, it can be 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.

Acknowledgment

This project was co-financed by the European Union, through the European Social Fund. Cruz and Hamada acknowledge support from the Special Coordination Funds for Promoting Science and Technology, Ministry of Education, Culture, Sports, Science and Technology, Japan (MEXT).

References

[1] A. Saleh, S. Redana, J. Hämäläinen, B. Raaf, On the coverage extension and capacity enhancement of inband relay deployments in LTE-Advanced networks, Journal of Electrical and Computer Engineering 2010 (2010) 4.

[2] N. Shabbir, M. Sadiq, H. Kashif, R. Ullah, Comparison of Radio Propagation Models for Long Term Evolution (LTE) Network, arXiv preprint arXiv:1110.1519.

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

[4] A. Valcarce, G. De La Roche, Á. Jüttner, D. López-Pérez, J. Zhang, Applying FDTD to the coverage prediction of WiMAX femtocells, EURASIP Journal on Wireless Communications and Networking 2009 (2009) 1–13.

[5] A. Hrovat, I. Ozimek, A. Vilhar, T. Celcer, I. Saje, T. Javornik, Radio coverage calculations of terrestrial wireless networks using an open-source GRASS system, WSEAS Transactions on Communications 9 (10) (2010) 646–657.

[6] T. Sarkar, Z. Ji, K. Kim, A. Medouri, M. Salazar-Palma, A survey of various propagation models for mobile communication, Antennas and Propagation Magazine, IEEE 45 (3) (2003) 51–82.

[7] A. Neskovic, N. Neskovic, D. Paunovic, A field strength prediction model based on artificial neural networks, in: Electrotechnical Conference, 1998. MELECON 98., 9th Mediterranean, Vol. 1, IEEE, 1998, pp. 420–424.

[8] M. Neteler, H. Mitasova, Open source GIS: a GRASS GIS approach, Vol. 689, Kluwer Academic Pub, 2002.

[9] R. Stallman, et al., GNU general public license, Free Software Foundation, Inc., Tech. Rep.

[10] D. Butenhof, Programming with POSIX threads, Addison-Wesley Professional, 1997.

[11] S. Shepler, M. Eisler, D. Robinson, B. Callaghan, R. Thurlow, D. Noveck, C. Beame, Network file system (nfs) version 4 protocol, Network.

[12] T. Hamada, K. Nitadori, 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, IEEE Computer Society, 2010, pp. 1–9.

[13] Ofcom, Table of base station totals (accessed October 2012).
URL `http://stakeholders.ofcom.org.uk/sitefinder/table-of-totals`

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

[15] M. Bhandarkar, L. Kale, E. de Sturler, J. Hoeflinger, Adaptive load balancing for MPI programs, Computational Science-ICCS 2001 (2001) 108–117.

[16] University of Pretoria, Q-Rap: Radio Planning Tool (accessed October 2012).
URL `http://www.qrap.org.za/home`

[17] Open Source Geospatial Foundation, Quantum GIS (accessed October 2012).
URL `http://www.qgis.org`

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