HIGH-PERFORMANCE COMPUTING FOR GRASS GIS: A PARALLEL RADIO COVERAGE PREDICTION TOOL

Lucas Benedičič
lucas.benedicic@telekom.si
Felipe Cruz
Tsuyoshi Hamada
Peter Korošec
Corresponding author.
Telephone number +386 1 472 2516.
Fax number +386 1 472 2590.
Telekom Slovenije, d.d., Cigaletova 15, SI-1000 Ljubljana, Slovenia
Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, SI-1000 Ljubljana, Slovenia
Nagasaki Advanced Computer Center, Nagasaki University,1–14 Bunkyo–machi, Nagasaki–city, Nagasaki, 852–8521, Japan

Abstract.

Mobile networks, GSM, UMTS, LTE, simulation, coverage, parallel, GRASS, GIS, MPI

1. Introduction

More than 20 years have passed since the world's first GSM call was made in Finland. Still, radio-coverage planning 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, but also in modern standards, such as the third generation (3G) UMTS and the fourth generation (4G) LTE Advanced [21, 17, 25, 19]. Clearly, one of the primary objectives of mobile-network planning is to efficiently use the allocated frequency band, to assure the necessary radio coverage is achieved. The complexity of this task arises from the wide range of various combinations of hardware and configuration parameters, which have to be analyzed in the context of different environments, certainly increasing their evaluation-time complexity.

Although different mathematical models have been proposed for radio propagation modeling, none of them excels in a network-wide scenario [19]. A combination of different models and parameters is generally needed in order to calculate radio-propagation predictions for particular environments. Moreover, since the number of deployed cells (transmitters) keeps growing with the adoption of modern standards [17], 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.

Despite various options of commercial tools, specialized in radio-propagation modeling, the common thread among all of the 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. Moreover, the evolution of any commercial tool is strictly bounded to its vendor, forcing the user to adapt its work-flow to it, when the opposite situation should be preferable.

To tackle the afore-mentioned issues, we present a high-performance parallel radio-prediction tool for GRASS GIS. For its design, we have focused on scalability, clean design and open nature of the tool, inspired by GRASS GIS. These facts make it an ideal candidate for calculating radio-predictions of real mobile networks containing thousands of transmitters. And also for the scientific research community, since our design may used as a template for parallelization of computationally-expensive tasks within the GRASS environment.

1.1. **Parallel computation on computer clusters.**

To reach high levels of performance and scalability, the presented work takes advantage of specialized hardware, e.g. a cluster of computers, that is set up to share, at least, login credentials for one user, and, ideally, also a networked file system like NFS [20]. The key step for reaching high-performance levels is to distribute the computational load among the computing nodes that belong to the cluster.

Such clusters typically consist of several commodity PCs connected through a high-speed local network. One such system is the DEGIMA cluster [9] at the Nagasaki Advanced Computing Center of the Nagasaki University, that currently holds the third place of the Green 500 list[1].

1.2. **Objectives .**

The main goal of this work is to develop a high-performance parallel radio prediction tool (PRATO) for 3G radio networks, which performance will allow its use in large real-world environments. Therefore, our focus is on the

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

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.

The performance evaluation of PRATO in a distributed computing environment is a major objective of this work. Additionally, we introduce some novel techniques that enable asynchronous parallelization of computationally-expensive tasks within the GRASS GIS environment, making it ideal for execution in heterogeneous computing environments. Furthermore, by presenting a detailed description of the design and implementation of the parallel version of PRATO, we intend to provide guidelines on how to achieve high efficiency levels of task parallelization in GRASS GIS.

The paper is organized as follows. Section 2 gives a description of the radio prediction tool, including the propagation model used and GRASS GIS. Section 3 concentrates on the design principles and implementation details of the radio propagation tool, for both 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 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. [10]. The authors of this work have developed a modular radio coverage tool, that performs separate calculations for radio-signal path loss, antenna radiation patterns, 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 cell (transmitter). This work implements some well-known radio propagation models (e.g. Okumura-Hata and COST 231). Regarding the accuracy of the predicted values, the authors report comparable results to those of a state-of-the-art commercial tool. Hence our decision to use this work as a reference implementation for PRATO.

To ensure that our implementation is completely compliant with the above-mentioned reference, we have designed a comparison test. It consists of running both the reference and our implementation with the same input parameters, hence resulting in two raster maps, one for each of the implementations. By using the GRASS module *r.mapcalc*, we calculate the arithmetic difference on every point of both raster maps. The resulting difference map should contain a raster grid with all its values set to zero for the test case to be successful. An example call of *r.mapcalc*, followed by an information display of a successful test, is depicted in Algorithm 1.

2.1. **Propagation modeling .**

The COST-231 Walfisch-Ikegami radio-propagation model was introduced as an extension of the well-known COST Hata model [19, 18], designed for frequencies above 2000 MHz. The suitability of this model comes

Algorithm 1 *Using r.mapcalc to compare the results with the reference implementation.*

```
GRASS 6.4.2 (data):~> r.mapcalc diff=reference_map-tested_map
GRASS 6.4.2 (data):~> r.info diff
...
| Range of data:  min = 0 max = 0 |
...
```

from the fact that it distinguishes between line-of-sight (LOS) and non-line-of-sight (NLOS) conditions. Equation (2.1) describes the path loss when there is LOS between the transmitter and the receiver.

$$PL_{LOS}(d) = 42.64 + 26\log(d) + 20log(F), \tag{2.1}$$

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 (2.2).

$$PL_{NLOS}(d) = L_0 + L_{RTS} + L_{MSD}, \tag{2.2}$$

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 in [10], 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 ETF artificial neural networks macro-cell model [12]. Consequently, both Equation (2.1) and Equation (2.2) have an extra term for signal loss due to clutter (L_{CLUT}), thus redefining the LOS and NLOS path losses as

$$PL_{LOS}(d) = 42.64 + 26\log(d) + 20log(F) + L_{CLUT}, \tag{2.3}$$

and

$$PL_{NLOS}(d) = L_0 + L_{RTS} + L_{MSD} + L_{CLUT}. \tag{2.4}$$

2.2. **GRASS GIS.** As the target environment we have chosen the open source Geographic Resources Analysis Support System (GRASS) [13]. This Geographic Information System (GIS) software was originally developed at the US Army Construction Engineering Research Laboratories (USA-CERL) and is a full-featured GIS system with a wide range of analytical, data-management, and visualization capabilities. Currently, the development of GRASS GIS is supported by a growing community of volunteer developers.

The use of GRASS GIS as an environment for PRATO presents many advantages. First, the current development of GRASS is primarily Linux-based. Since the field of high performance computing is dominated by Linux and UNIX systems, an environment with Linux support is critical for this work. Software licensing in another important consideration for choosing GRASS, since it is licensed under the GNU Public License (GPL; [23]) 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. Moreover, being an open system, GRASS provided us with a great deal of useful built-in functionality, capable of operating with raster and vector topological data that can be stored in an internal format or a relational database. For additional information about the GRASS, we refer the reader to the numerous guides and tutorials available online.

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 3.1.

Our design follows a similar internal organization as the radio planning tool developed by Hrovat et al. [10], but with some essential differences. Specifically, we have decided to avoid the modular design to prevent the overhead of I/O operations, that are needed to communicate data between the components of the modular architecture. Instead, we have chosen a monolithic design, in which all the steps for generating the radio coverage prediction are calculated inside one GRASS module. Regarding the result dumping and final aggregation, our approach employs a direct connection to an external database server, 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 we is simply "pipe through" to the database server. As for the result aggregation, it is achieved by issuing a query over the database tables containing 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 3.1), e.g. digital elevation model, clutter data, transmitter configurations, and other service dependent settings. They are all read from files, which 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 service-dependent options.

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

The INI file is split in two main sections: common and transmitter-specific. The common section ("[Common]" in Algorithm 2) contains parameters that are shared among all transmitters during a coverage-prediction run. The transmitter section ("[TX_1]" in Algorithm 2) contains the configuration of a specific transmitter. As it has been mentioned before, there many be many transmitter sections in one INI file.

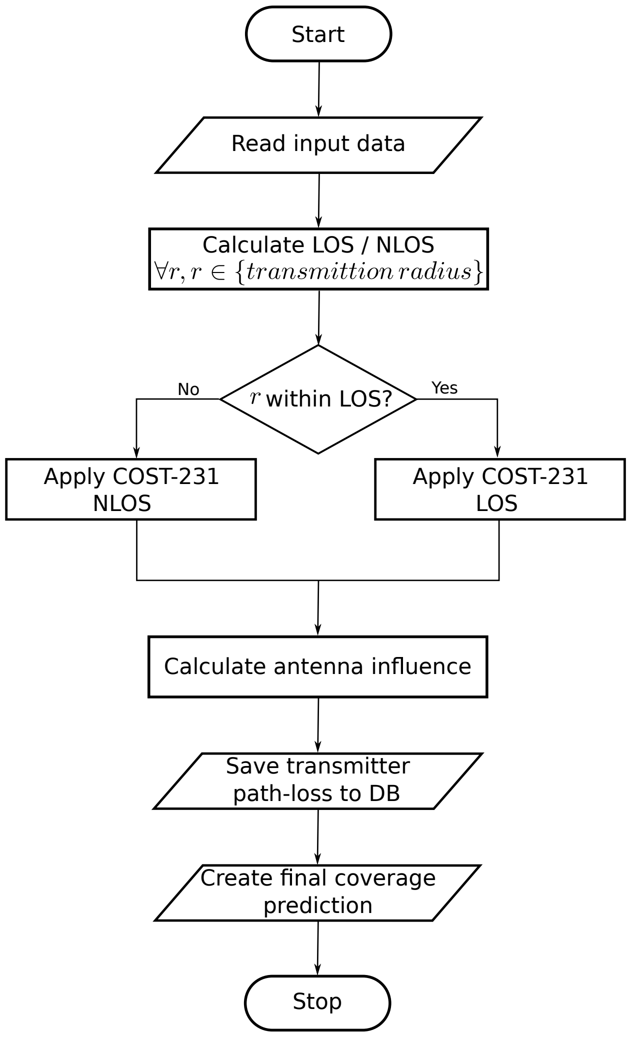
Start
Read input data
Calculate LOS / NLOS $\forall r, r \in \{transmittion\, radius\}$
r within LOS?
No
Yes
Apply COST-231 NLOS
Apply COST-231 LOS
Calculate antenna influence
Save transmitter path-loss to DB
Create final coverage prediction
Stop

FIGURE 3.1. *Flow diagram of the serial version.*

3.1.2. *Isotropic path-loss calculation* . The first step here is to calculate which points are within the specified transmission radius ("radius" in Algorithm 2). For these points, the LOS and NLOS conditions are calculated, with respect to the transmitter ("Calculate LOS/NLOS" in Figure 3.1). 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 (2.3) or Equation (2.4) is respectively applied ("Apply COST-231, LOS" or "Apply COST-231, NLOS" in Figure 3.1).

Figure 3.2 shows a portion of a raster map with an example of the isotropic path-loss calculation.

Algorithm 2 *Example of an INI file, containing input parameters for a coverage-prediction run.*

```
[Common]
DEMMapName = dem25@PERMANENT ; name of the digital elevation model raster map
clutterMapName = clut25@PERMANENT ; name of the clutter map
receiverHeightAGL = 1.5 ; receiver's height above ground level
frequency = 2040.0 ; transmitter frequency in MHz
radius = 20 ; calculation radius around a transmitter (in km)
antennaDirectory = ~/r.coverage/antenna ; directory containing antenna files
[TX_1]
cellName = TX_1 ; name of the transmitter
beamDirection = 20 ; antenna beam angle
electricalTiltAngle = 2 ;
mechanicalTiltAngle = 3 ;
heightAGL = 23.9 ; antenna height above ground level
antennaFile = 742212_2140_X_CO_M45_02T.MSI ; antenna diagram file
positionEast = 501152 ; coordinate of the transmitter
positionNorth = 142449 ; coordinate of the transmitter
power = 30.2 ; transmitter pilot power in dBm
...
```

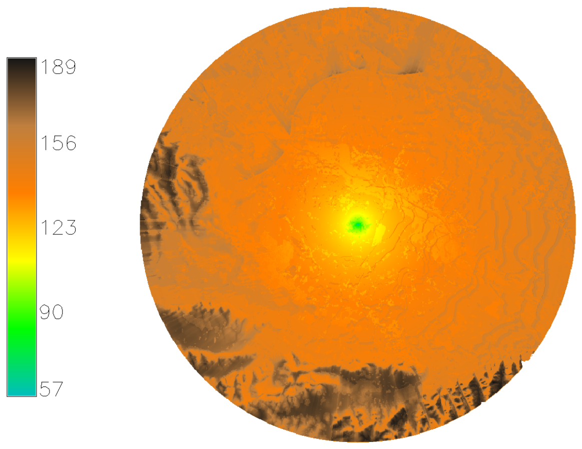

FIGURE 3.2. *Example of the isotropic path-loss calculation.*

3.1.3. *Antenna diagram influence* . 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 3.1). Working on the in-memory results generated by the previous step, the radiation diagram of the antenna is taken into account ("antennaFile" in Algorithm 2), including beam direction, electrical and mechanical tilt. Figure 3.3 shows a portion of a raster map with an example calculation at this step. Notice the distortion of the signal propagation that the antenna has introduced, and compare it with Figure 3.2.

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 3.1), thus considerably enhancing the write performance during result dumping. This is accomplished by connecting the standard output of the

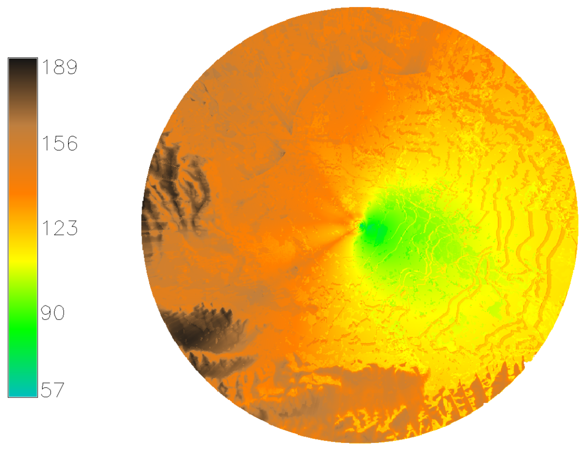

FIGURE 3.3. *Influence of the antenna over the isotropic path-loss calculation.*

Algorithm 3 *Plain-text results of the path-loss prediction in PostgreSQL dialect.*

```
GRASS 6.4.2 (data):~> r.coverage ini_file=./parameters.ini tx_ini_section=TX_1
CREATE TABLE IF NOT EXISTS coverage_TX_1 (east int, north int, pl float);
\COPY coverage_TX_1 (east, north, pl) FROM STDIN WITH DELIMITER ' '
589312 179388 181.01559
589338 179388 180.98627
589362 179388 172.67766
589388 179388 180.91296
589412 179388 182.16579
589438 179388 189.72215
...
```

Algorithm 4 *Connecting the standard output of the GRASS module with a PostgreSQL client on Linux.*

```
GRASS 6.4.2 (data):~> r.coverage ini_file=./parameters.ini tx_ini_section=TX_1 |
psql -h server_name -U usr_name database_name
```

developed module with the standard input of a database client. Naturally, the generated plain text should be understood by the database server itself, as the example in Algorithm 3 shows for the PostgreSQL dialect. Notice the backslash in front of the COPY command. Algorithm 4 shows a short example of how to connect the standard output of the module with the standard input of a PostgreSQL 9.1 database client on Linux. Clearly, the values for *server_name*, *usr_name* and *database_name* should be adapted according to the database-server installation.

3.1.5. *Coverage prediction* . The final radio coverage prediction, containing an aggregation of the partial predictions of the involved transmitters, is created in this step ("Create final coverage prediction" in Figure 3.1). 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. A sample of such aggregation for two transmitters (TX_1 and TX_2), using a SQL query, is shown in Algorithm 5.

Algorithm 5 *Aggregation of partial path-loss predictions by means of SQL.*

```
database_name=> SELECT east, north, MAX(rscp)
FROM (SELECT east, north, rscp FROM coverage_TX_1
UNION
SELECT east, north, rscp FROM coverage_TX_2) agg
GROUP BY agg.east, agg.north;
```

Algorithm 6 *Final raster map creation, containing the radio coverage prediction for all transmitters.*

```
GRASS 6.4.2 (data):~> r.in.xyz input=/tmp/aggregate.xyz fs=space output=temp
--overwrite
...
GRASS 6.4.2 (data):~> r.resamp.rst input=temp ew_res=25 ns_res=25
elev=final_coverage --overwrite
```

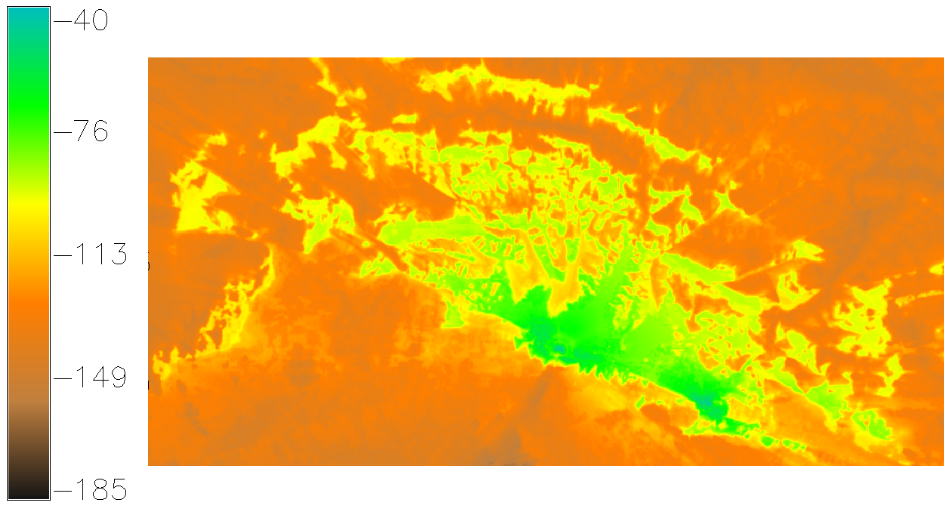

FIGURE 3.4. *Example of raster map, displaying the final coverage prediction.*

Finally, the output raster is generated, using the commands shown in Algorithm 6, where */tmp/aggregate.xyz* contains the query results from Algorithm 5. The raster map contains the maximum received signal strength for each individual point, which can be visualized in GRASS, as Figure 3.4 shows.

3.2. **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 network instances [10].

Some authors have already published their work on implementing parallel versions of GRASS modules for solving different time-consuming tasks [**?** 22, 4, 16]. However, 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, 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. 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. Also, a message-passing technique is proposed, that distributes 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.

3.2.1. *Master process* . As it has been suggested before, the parallel version of PRATO follows a master-worker model. The master process is the only component that should be run from within the GRASS environment. As soon as it starts, the input parameters are read ("Read input data" in Figure 3.5). This is done in a similar way as in the serial version, which has been introduced in Section 3.1.1. In the next step, it dynamically initiates the worker processes in the computing nodes ("Dynamic worker-process spawning" in Figure 3.5), taking into account the amount of transmitters for which the coverage prediction should be calculated. This means that master never starts more worker processes than the number of transmitters to be processed. The opposite is clearly possible. The master process then 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 ("Input data broadcasting" in Figure 3.5). This decomposition applies to the digital-elevation and the clutter data only. In the next step, the master process starts a message-driven processing loop ("Processing loop" in Figure 3.5), which main task is to distribute the calculation work, i.e. the configuration data of the different transmitters, among idle worker processes.

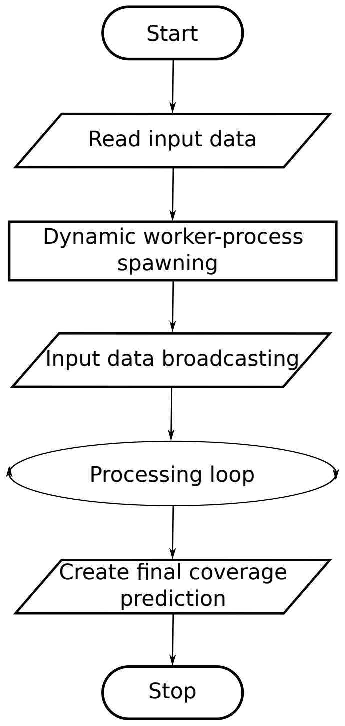
Start
Read input data
Dynamic worker-process spawning
Input data broadcasting
Processing loop
Create final coverage prediction
Stop

FIGURE 3.5. *Flow diagram of the master process.*

The flow diagram shown in Figure 3.6 depicts the steps taking part of the processing loop of the master process. It starts by checking which worker processes are still running, so that they may calculate the radio coverage prediction for the next transmitter. It is worth pointing out that this step also serves as a stop condition for the processing loop itself ("Any worker still on?" in Figure 3.6). The active worker processes inform master they are ready to process by sending an idle message ("Wait for idle worker" in Figure 3.6). 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 ("Send keep-alive message" and "Send transmitter data", respectively, in Figure 3.6). This is only done if there are still transmitters, for which the coverage prediction has yet to be calculated ("Any transmitters left?" in Figure 3.6). 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 ("Send stop message" in Figure 3.6). The master process will keep doing this until all worker processes have finished ("Any worker still on?" in Figure 3.6), thus fulfilling the stop 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 ("Create final coverage prediction" in Figure 3.5). The final result is an aggregation from the individual results created by each

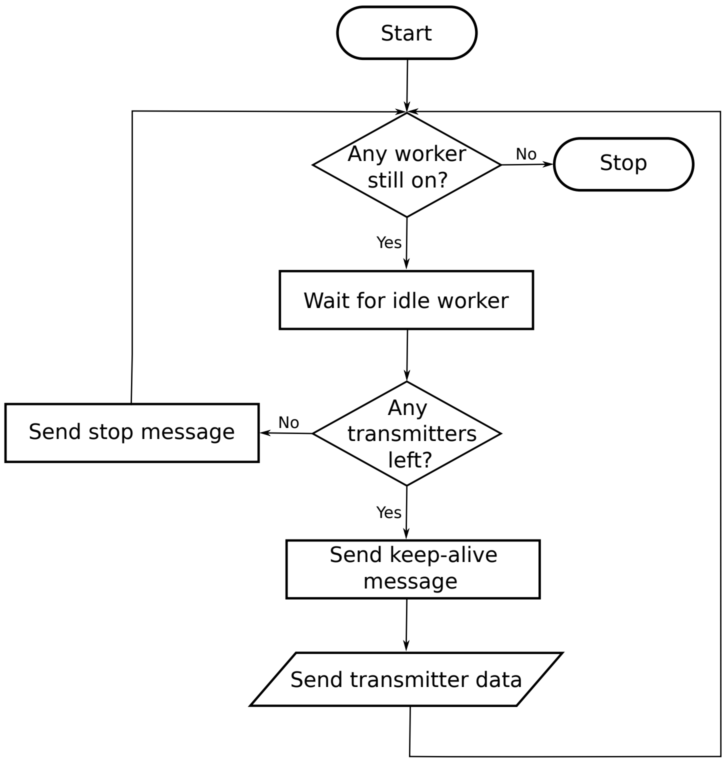

FIGURE 3.6. *Flow diagram of the processing loop of the master process.*

of the worker processes during the "Processing loop" phase in Figure 3.5, which provide the source data for the final raster map. This is accomplished in a similar way as in the serial version, which has been introduced in Section 3.1.5.

3.2.2. *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. Neither the GRASS libraries have to be installed. All their computations are run over arrays of basic element types, like floats or doubles, which are received from the master process at initialization time ("Receive broadcasted data" in Figure 3.7). Moreover, these data is common to all of the transmitters for which coverage predictions have to be calculated afterwards. This key feature make each worker process capable of processing any of the transmitters.

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 [3]. 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.

Because concurrent access to data within GRASS shields 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. One possible solution would be to save them through the master process, thus avoiding concurrent access. But 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 processed by the master process alone, in a serial fashion. Instead, our approach allows each of the worker processes to output its results into an external database server, following an asynchronous and decoupled design. The results are saved in separate tables, following a similar design as the serial version, which has been introduced in Section 3.1.4. Moreover, worker processes do this from an independent thread, running concurrently with the calculation of the next transmitter received from the master process. When compared to the serial version, this approach completely hides the latency created by the result dumping task.

After received the broadcasted data, the worker process continues by checking if the stop message from master has arrived (first “Has stop message arrived?” in Figure 3.7), in which case it waits for result-dumping threads that may still be running, to finish (“Wait for result-dump threads” in Figure 3.7), before shutting down. Otherwise it informs the master process it is idle and ready to receive transmitter-configuration data for the next coverage prediction (“Send idle message” in Figure 3.7). If master does not instruct to stop processing (second “Has stop message arrived?” in Figure 3.7), the worker process collects the transmitter configuration sent (“Receive transmitter data” in Figure 3.7), before starting the coverage-prediction calculation for the target transmitter (“Coverage calculation” in Figure 3.7). The coverage calculation itself follows a similar design as the serial version, which has been introduced in Sections 3.1.2 and 3.1.3.

As it was mentioned before, the worker process launches an independent thread to save the coverage prediction of the target transmitter to a database table (“Threaded save path-loss to DB” in Figure 3.7). This is done in the last step of the loop.

3.2.3. *Master-worker communication* . The selected message-passing technique introduced in this work might seem too elaborated at first sight, but important reasons lay behind each of the messages passed between master and worker processes. These decisions are 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.

Before delivering the transmitter-configuration data, the master process sends a message to a worker process, indicating that it is about to receive more work. This a-priory meaningless message has a key role in correctly supporting computer clusters. Namely, parallel computation within such environments is subject to race conditions [6]. To avoid a dead-lock of the parallel version of PRATO, message

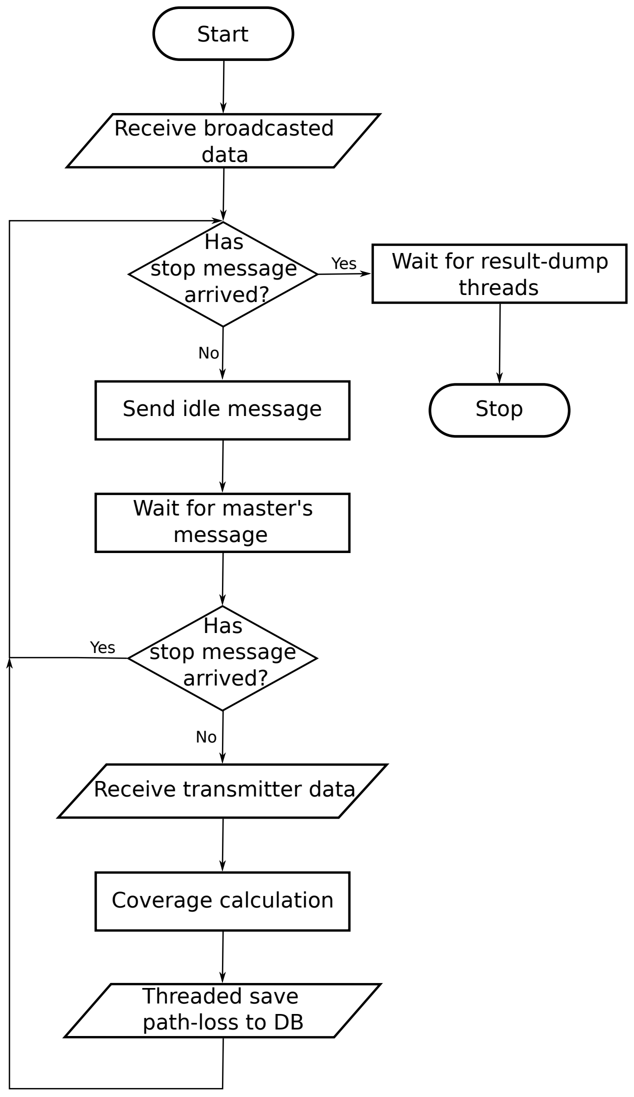
Start
Receive broadcasted data
Has stop message arrived?
Yes
Wait for result-dump threads
No
Send idle message
Stop
Wait for master's message
Has stop message arrived?
Yes
No
Receive transmitter data
Coverage calculation
Threaded save path-loss to DB

FIGURE 3.7. *Flow diagram of the worker process.*

sending and receiving should be paired, being equal number of send and receive operations on the master and worker sides.

Figure 3.8 depicts a diagram of the master-worker message passing. The time line runs vertically. 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.

3.3. **Implementation.**

3.3.1. *The MPI framework.* The implementation methodology adopted for the parallel version of the simulator uses the Message Passing Interface or MPI [8]. MPI is a standardized and portable library of functions, designed to function on a wide

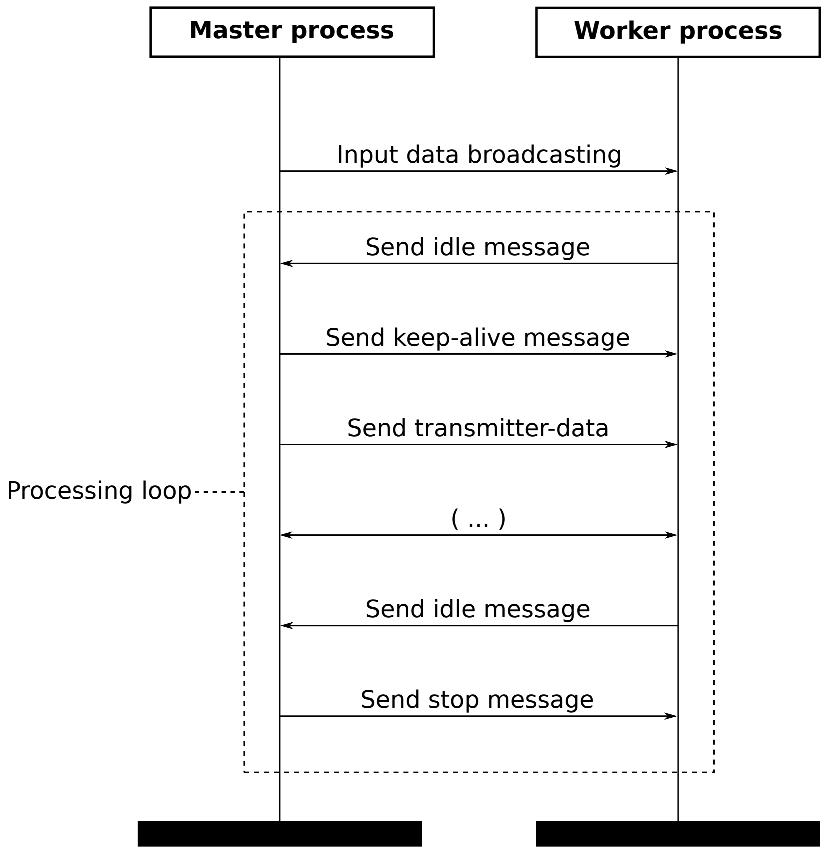

FIGURE 3.8. *Communication diagram, between master and one worker process.*

variety of parallel computers, which syntax and semantics are defined by an open standard. The library allows implementing portable message-passing programs in Fortran and C programming languages. MPI was designed for high performance on both massively parallel machines and on workstation clusters, and it has been developed by a broadly based committee of vendors, developers, and users.

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 other to determine if the objectives of this work, previously introduced in Section 1.2, are fullfiled. We follow the principle of segmenting a problem instance by transmitter, for which the coverage prediction is calculated by a separate worker process.

The following simulations were carried out on 34 computing nodes of the DEGIMA cluster. DEGIMA is a computer cluster located at the Nagasaki Advanced Computing Center (NACC), in the University of Nagasaki, Japan. The computing nodes are connected by a LAN, over a 1 Gigabit Ethernet interconnect, and share a NFS partition, from which all input and intermediate files were accessed.

Table 1. *Common simulation parameters.*

Parameter	Value
Resolution (m^2)	25.0
Frequency (MHz)	2040.0
Receiver height above ground level (m)	1.5
Effective radius around transmitter (km)	20.0

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. The data sets are created by randomly replicating and distributing a group of 10 transmitters over the whole target area. The configuration parameters of these 10 transmitters is taken from the UMTS network deployed in Slovenia by Telekom Slovenije, d.d.. The path-loss predictions are calculated using the COST-231, which was introduced in Section 2.1. The digital elevation model used has a resolution of 25 m^2, the same as the clutter data, containing different levels of signal loss based on the land usage. For all the points within the 20 km radius around each transmitter, the receiver is positioned $1.5\,m$ above the ground, and the frequency is set to 2040 MHz. Table 1 summarizes these parameters, which are used for all further simulations.

4.2. **Weak scalability.** This set of simulations is meant to analyze the scalability of the parallel implementation in cases where the size of the problem, i.e. the number of transmitters deployed over the target area, changes with the number of 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.

Problems particularly well-suited for parallel computing exhibit computational costs that are linearly dependent on the problem size. This property, also referred

TABLE 2. *Simulation results for weak scalability.*

	Number of cores							
Transmitters per 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

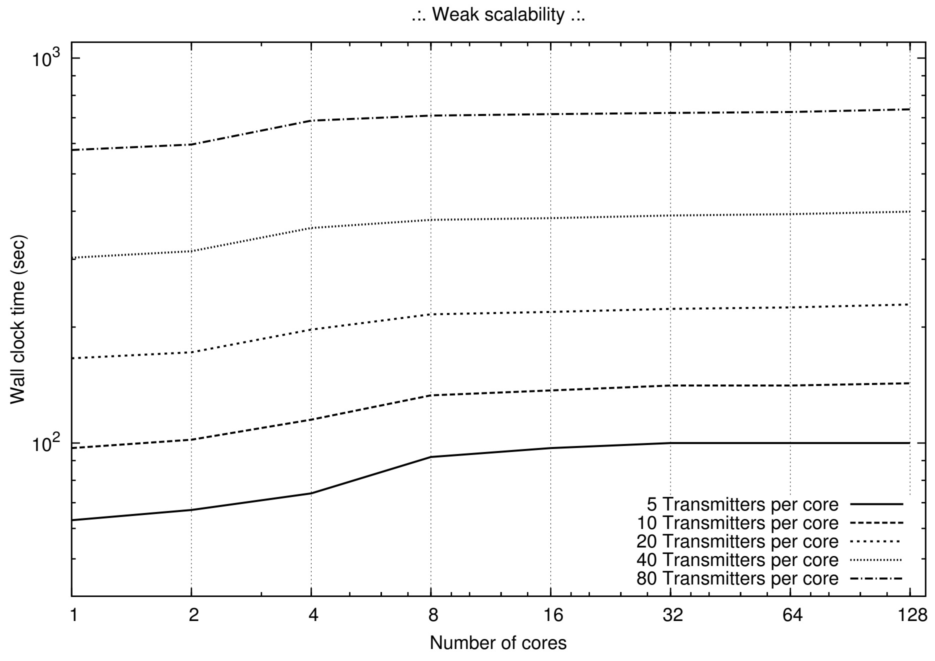

FIGURE 4.1. *Measured time for weak-scalability experiments.*

to as algorithmic scalability, means that proportionally increasing both the problem size and the number of cores, results in a roughly constant wall-clock 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 of the time measurements collected after the simulations for the weak-scalability experiments are shown in Table 2. All times are expressed in seconds. The wall-clock time measurements are plotted in Figure 4.1, where the time axis is expressed in base-10 logarithmic scale, whereas the axis representing the number of transmitters per core is expressed in base-2 logarithmic scale.

The time measurements observed from the weak-scalability results show that the achieved level of scalability gets close-to-linear as the amount of transmitters-per-core increases. Moreover, 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 clearly 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 [14].

Not being to achieve perfect weak scalability is due to a number of factors, namely:

(1) the largest fraction of the run-time is spent on the parallel processing of transmitters, which scales notably well with the increasing number of nodes;
(2) 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;
(3) communication overhead grows linearly with the number of cores used.

To confirm the above-mentioned claims, 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 5, 20 and 80 transmitters per core, which are shown in Figures 4.2, 4.3 and 4.4 respectively. We have explicitly excluded the "Read input data" step (see Figure 3.5) since its time is constant for all problem instances, thus not adding extra information about the scalability of the parallel version itself. Consequently, for the relative-processing-time plots we use the formula

$$RT(np) = \frac{t_{ps} + t_{db} + t_{pl} + t_{cp}}{t_{total} - t_{rd}}, \tag{4.1}$$

where 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, t_{total} is the total wall-clock processing time, and t_{rd} is the "Read input data" wall-clock time. For a reference of the different steps taking part of the master process, see Figure 3.5.

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 clearly see the bounce of the relative time for the "Input data broadcasting" step, from 4 to 8 cores, i.e. from one to more computing nodes to more. This is due to the use of Ethernet communication for more than 1 computing node hosting worker processes. Moreover, we may conclude that the network infrastructure has not been saturated with the message-passing load, since the relative times for input data broadcasting are almost constant as we double the amount of shared data among the worker processes. Regarding the final coverage prediction creation, we may see that the relative times grow proportionally with the problem sizes. This is directly related to the time the database server takes to aggregate the partial path-loss predictions, which grow relative to the problem size, as expected.

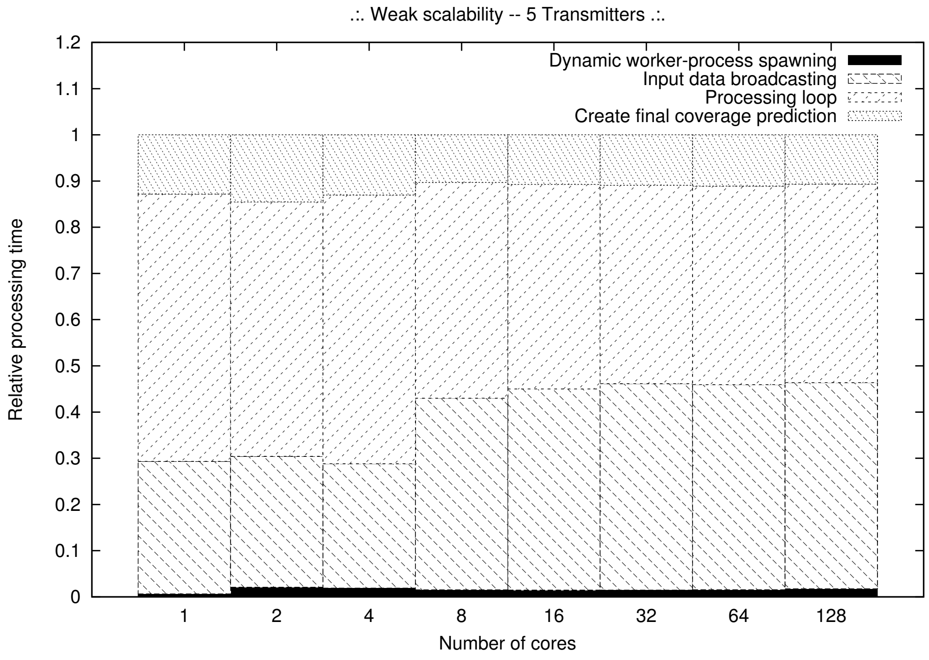

FIGURE 4.2. *Relative times for the weak-scalability experiments, considering 5 transmitters per core.*

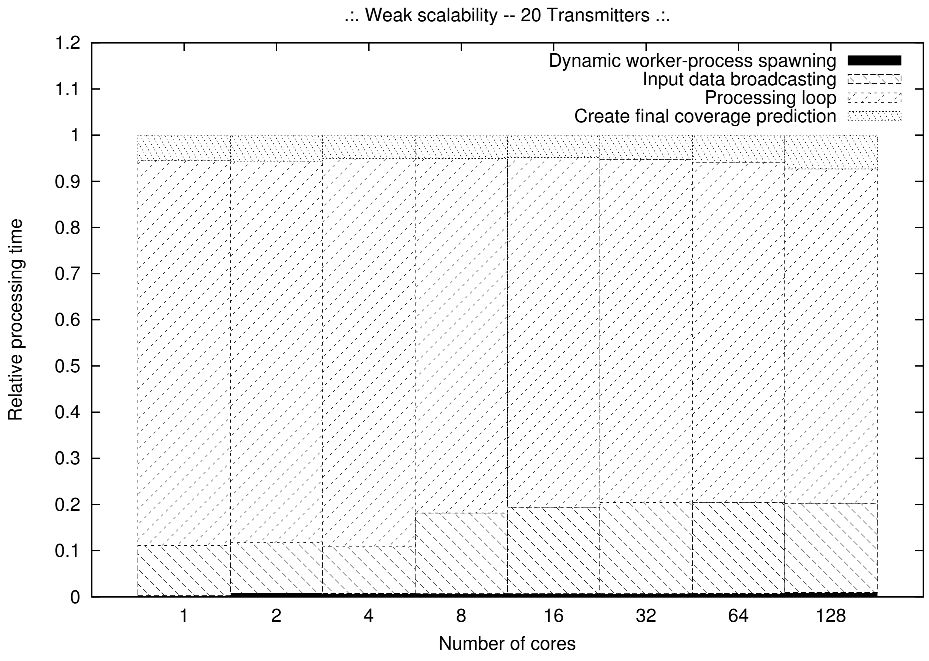

FIGURE 4.3. *Relative times for the weak-scalability experiments, considering 20 transmitters per core.*

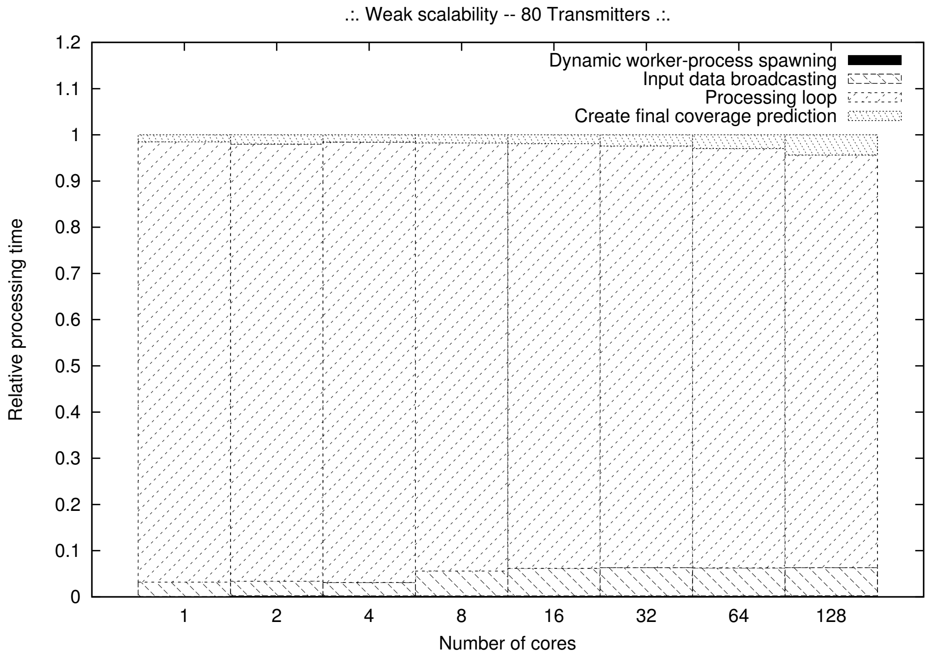

FIGURE 4.4. *Relative times for the weak-scalability experiments, considering 80 transmitters per core.*

4.3. **Strong scalability .** This set of simulations is meant to analyze the impact of a changing number of transmitters per core. In other words, we test the scalability of the parallel implementation in cases where the size of the problem is fixed, i.e. the number of transmitters deployed over the target area does not change, while only the number of cores used is increased.

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 3. All times are expressed in seconds. These wall-clock time measurements are plotted in Figure 4.5, where the 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 show that small problem sizes per core are correlated with a relatively large proportion of serial work and communication overhead. Therefore, the performance deteriorates as the number of transmitters per core approaches 1. This fact may be deduced from the slope tilts, for which it is more significant with smaller problem instances, e.g. 64, 128 and 512 transmitters. However, the relative contribution of each of the non-parallel steps is smaller for larger problem sizes, as a larger proportion of the input data is reused when calculating the coverage prediction for different transmitters.

We have also measured the performance of the parallel implementation in terms of the speedup, which is defined as follows:

$$S(NTX, NP) = \frac{execution\,time\,for\,base\,case}{execution\,time\,for\,NP\,cores}, \tag{4.2}$$

Table 3. *Simulation results for strong scalability.*

	Number of transmitters						
Number of 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

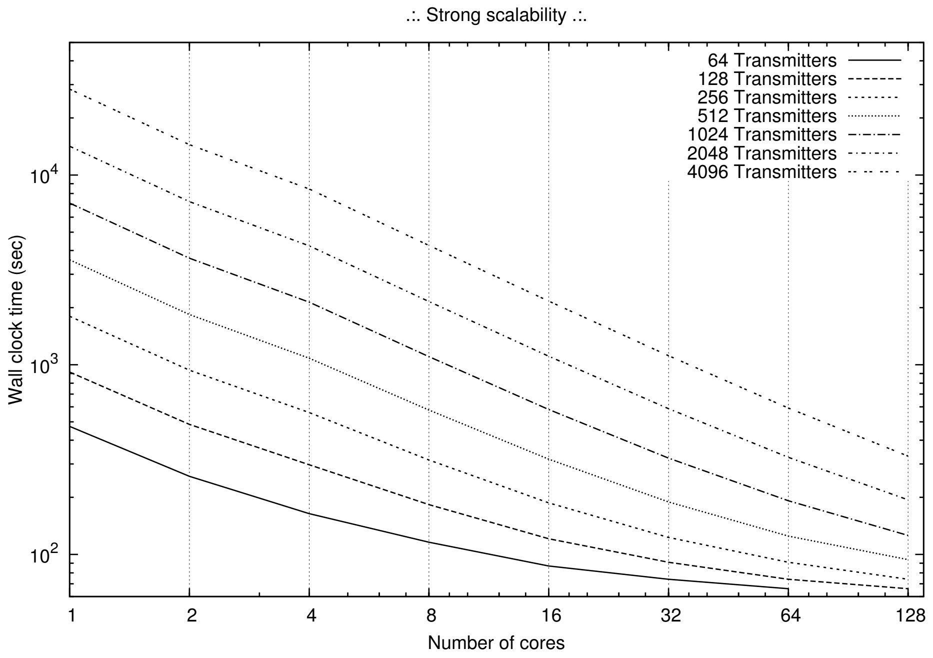

Figure 4.5. *Measured time for strong-scalability experiments.*

where NTX is the number of transmitters processed, i.e. the problem size, and NP is the number of cores executing the worker processes. As the base case we have chosen the parallel implementation running on only one core. The reason for choosing this base case and not the serial implementation is fair comparison. Namely, several concatenated runs of the serial version would be considerably slower, because it introduces a large I/O overhead. Such comparison would be entirely biased towards the parallel implementation, showing speed-ups which would not be real.

Linear scaling is achieved when the obtained speed-up is equal to the total number of processors used. However, it should be noted that perfect speed-up is almost never achieved, due to the existence of serial stages within an algorithm and communication overheads of the parallel implementation [7]. Therefore, the ideal case would be for the parallel execution time to be inversely proportional to the number of cores.

Figure 4.6 shows the speed-up of the parallel implementation for up to 128 cores, running the same number of worker processes, and compares seven different problem sizes: 64, 128, 256, 512, 1024, 2048 and 4096 transmitters deployed over the target area. These problem sizes are comparable to several operational radio networks that have already been deployed in England, e.g. Bedfordshire County, Cheshire County, Hampshire County, West Midlands, and Greater London Authority [14]. The speed-up axis is expressed in base-10 logarithmic scale, whereas the axis representing the number of cores is expressed in base-2 logarithmic scale.

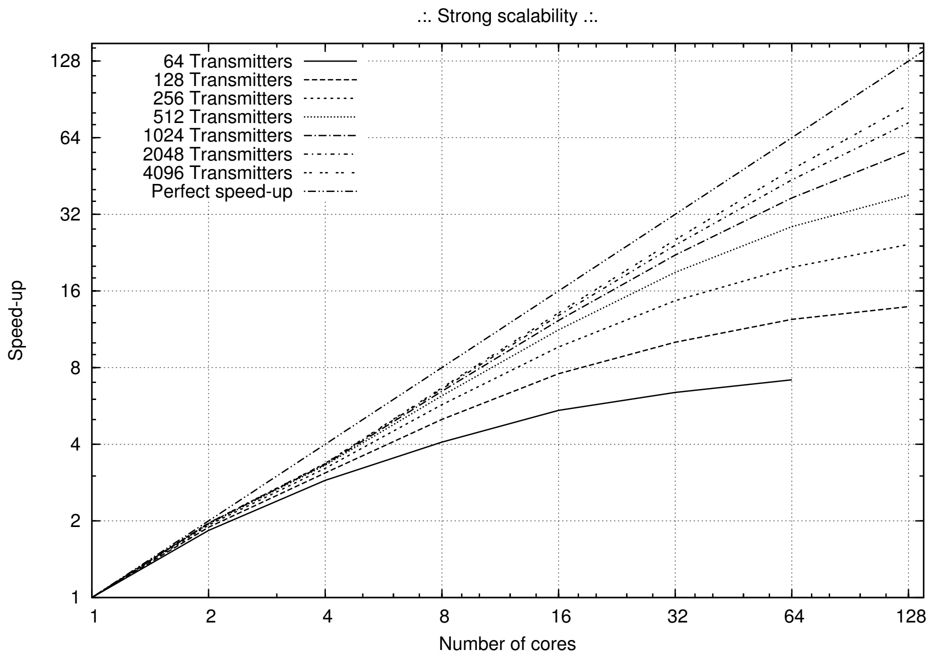

FIGURE 4.6. *Measured speed-up for strong-scalability experiments.*

We can see that the significant reductions in wall-clock time for large problem sizes shown in Figure 4.5 are correlated with the speed-up factors shown in Figure 4.6.

The last aspect we consider is the parallel efficiency of the implementation, i.e. how well the parallel implementation makes use of the available resources. The definition of parallel efficiency is as follows:

$$E(NTX, NP) = \frac{S(NTX, NP)}{NP}, \tag{4.3}$$

where $S(NTX, NP)$ is the speed-up as defined in Equation (4.2), and NP is the number of cores executing worker processes. Figure 4.7 shows the parallel efficiency of the parallel implementation. The parallel-efficiency axis is expressed in linear scale, whereas the axis representing the number of cores is expressed in base-2 logarithmic scale.

The ideal situation would be to utilize all available resources, in which case the parallel efficiency would constantly be 1 as we increase the core count. From the plot in Figure 4.7, we may see that the computational resources are not being completely used. In accordance to the previous analysis, this is more significant in smaller problem sizes, where number of transmitters per core approaches 1. 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 weight of these overheads is significantly reduced as the work-load per core increases. Unsurprisingly, these results confirm that it is not worth parallelizing small problem instances.

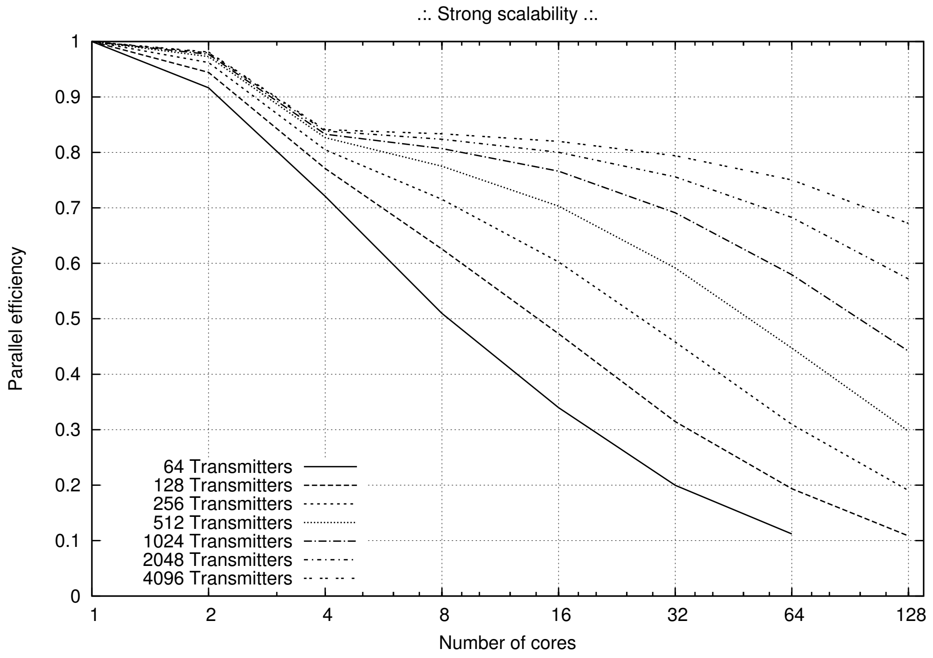

FIGURE 4.7. *Measured parallel efficiency for strong-scalability experiments..*

Similarly as before, we show 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. Again, we have explicitly excluded the “Read input data” step because it does not contribute any information about the scalability of the parallel version itself. In this case, we have created plots for four problem instances namely 64, 256, 1024, and 4096 transmitters, which are shown in Figures 4.8, 4.9, 4.10 and 4.11 respectively. The relative times shown are calculated using the formula depicted in Equation (4.1).

We may observe the non-parallel steps comprising “Dynamic worker-process spawning”, “Input data broadcasting” and “Final coverage prediction” contribute

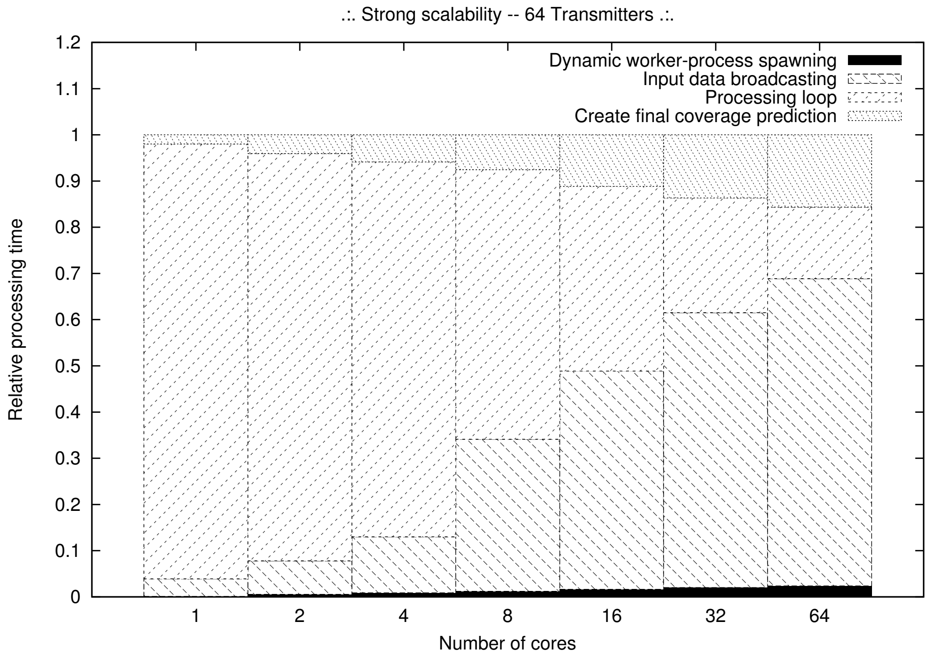

FIGURE 4.8. *Relative times for the strong-scalability experiment, considering 64 transmitters.*

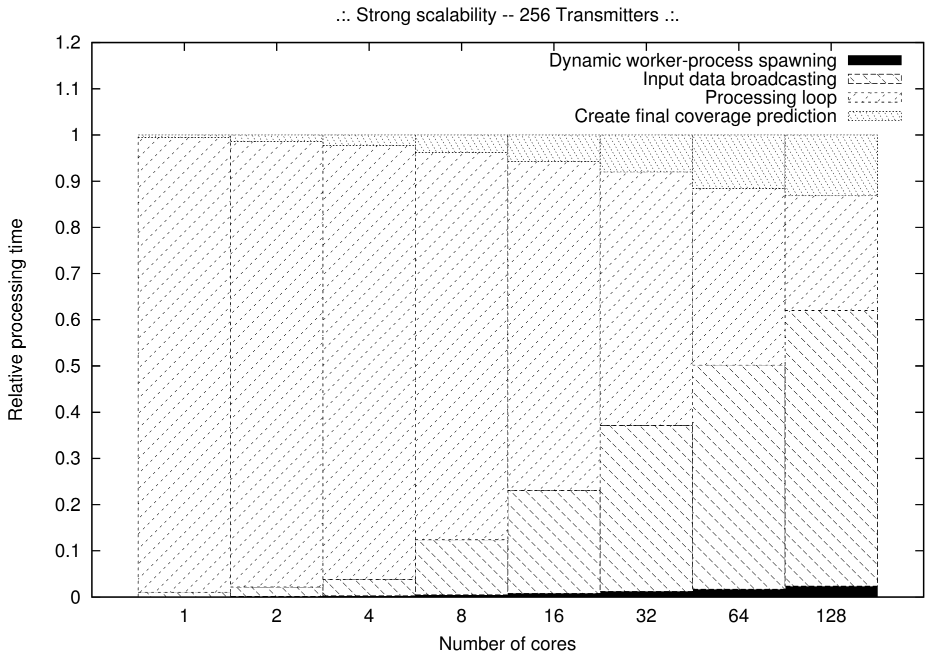

FIGURE 4.9. *Relative times for the strong-scalability experiment, considering 256 transmitters.*

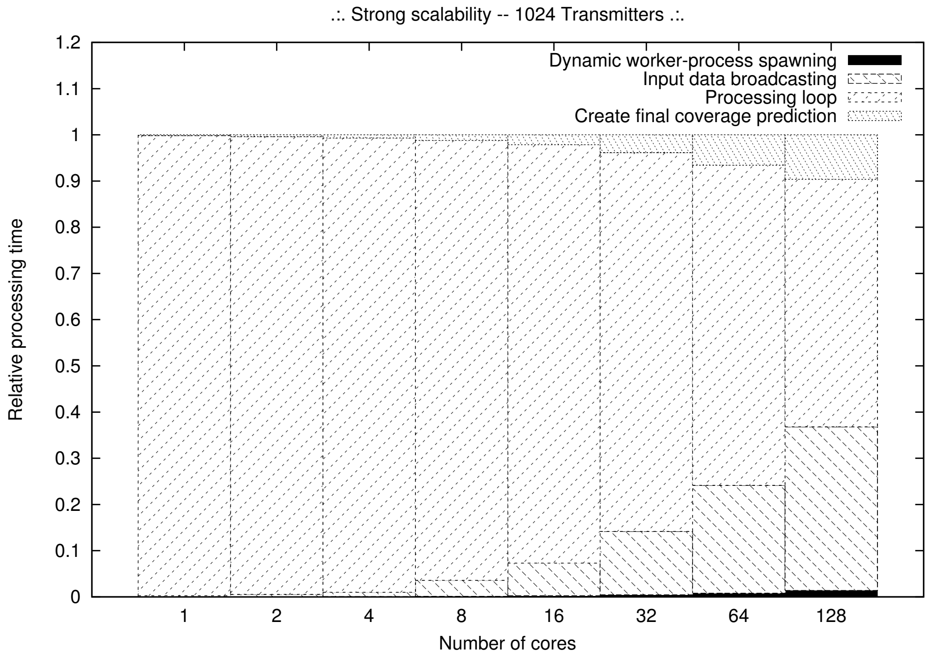

FIGURE 4.10. *Relative times for the strong-scalability experiment, considering 1024 transmitters.*

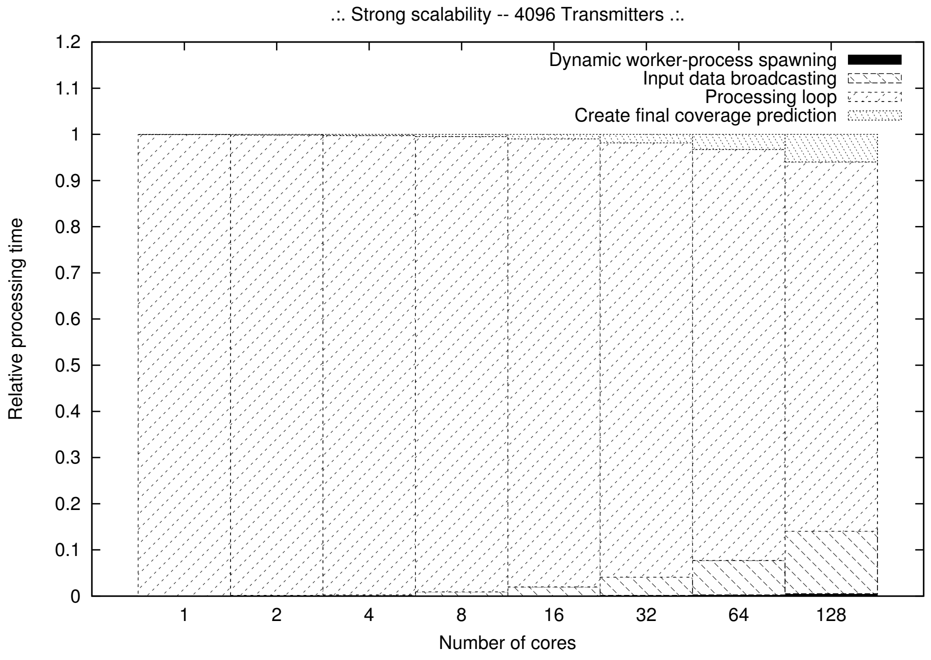

FIGURE 4.11. *Relative times for the strong-scalability experiment, considering 4096 transmitters.*

with a larger portion of time as we increase the number of cores, since the total wall-clock processing time decreases. Additionally, the low parallel efficiency for small problem sizes, particularly for 64 (Figure 4.8) and 256 transmitters (Figure 4.9), is validated as we see 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 unevenly across all processors. Workload distribution strategies should be based on the processor speed, memory hierarchy and communication network [5].

The parallel implementation of PRATO performs load-balancing using point-to-point messages (see Section 3.2.3) between master and worker processes. When a worker process issues an idle request ("Send idle message" in Figure 3.8), it blocks until the requested 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 [2]. Specifically, we would like to verify the penalties that such synchronization technique has on the scalability of the parallel implementation.

Based on the definition given in [7], we use the following metric as an indicator of the load balancing among processes:

$$LB(NP) = \frac{minimum\,execution\,time\,among\,NP\,cores}{processing\,loop\,time\,of\,master\,process}, \tag{4.4}$$

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, while 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 has been broadcasted to all worker processes ("Input data broadcasting" in Figure 3.5), until just before the beginning of the last step ("Create final coverage prediction" in Figure 3.5).

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 of Section 4.3, which coverage predictions are calculated by up-to 128 cores, running on 32 computing nodes.

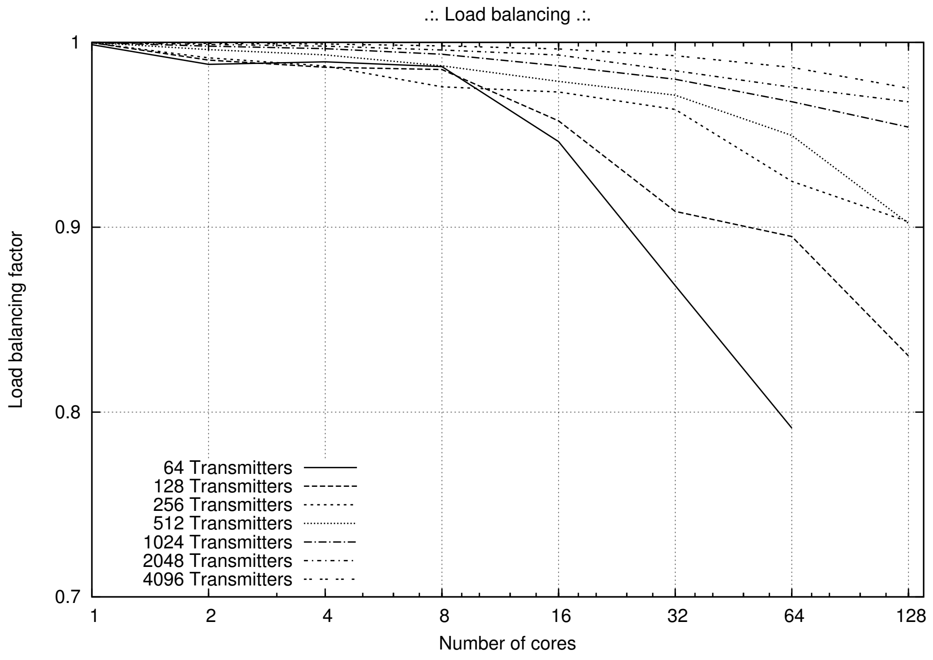

Figure 4.12. *Load balancing among worker processes.*

From the plot shown in Figure 4.12, 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 256 and 512 transmitters, showing a very good performance of the dynamic task assignment, driven by our message-passing technique;
- for problem sizes 64 and 128 transmitters, the parallel-process times are within 80% a the perfect load balancing, showing that when the number of transmitters per core approaches one, latencies introduced by several hardware and OS-specific factors, e.g. TurboBoost, process affinity, is 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 before 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. Hrovat et al. [10]. Although the reported results show a comparable quality to those of a professional radio-planning tool, there is an explicit concern about the performance of the aggregation module, *r.MaxPower*. The authors note that due to its memory inefficiency, it is not possible to consider large regions, since the afore-mentioned module runs out of memory. When comparing the wall-clock reported, PRATO clearly shows improved performance for comparable problem sizes. Additionally, the parallel implementation of PRATO is not subject to the memory problem, making calculation over large regions perfectly possible.

A different example of a GIS-based open-source radio planning tool, called Q-Rap, has been presented in [24]. 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 driven as 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 [15].

The task-parallelization problem within the GRASS environment has been addressed by several authors in different works. Campos et al. [4] 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.

Showing that GIS applications may also take advantage of GPU hardware, Osterman [16] has presented a CUDA-based module to calculate optical visibility (or line of sight - LOS) based on the digital elevation model. The experimental results report that the module performance could be up-to three size classes faster than the serial implementation.

On the field of high-performance computing, Akhter et al. [1] have presented implementation examples of the *r.vi* module 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 range, since the input data is 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 by distributing additional work to computationally more capable nodes.

Similarly, Huang et al. [11] use the parallel inverse distance weighting (IDW) interpolation algorithm as a parallel-pattern example. 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.

6. Conclusion

References

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

[2] M. Bhandarkar, L. Kale, E. de Sturler, and J. Hoeflinger. Adaptive load balancing for MPI programs. *Computational Science-ICCS 2001*, pages 108–117, 2001.

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

[4] I. Campos, I. Coterillo, J. Marco, A. Monteoliva, and C. Oldani. Modelling of a Watershed: A Distributed Parallel Application in a Grid Framework. *Computing and Informatics*, 27(2):285–296, 2012.

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

[6] C. Clemencon, J. Fritscher, M. Meehan, and R. Rühl. An implementation of race detection and deterministic replay with MPI. *EURO-PAR'95 Parallel Processing*, pages 155–166, 1995.

[7] F.A. Cruz Villaroel. *Particle flow simulation using a parallel FMM on distributed memory systems and GPU architectures*. PhD thesis, Faculty of Science, University of Bristol, November 2010.

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

[9] T. Hamada and 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*, pages 1–9. IEEE Computer Society, 2010.

[10] A. Hrovat, I. Ozimek, A. Vilhar, T. Celcer, I. Saje, and T. Javornik. An open-source radio coverage prediction tool. In *Proceedings of the 14th WSEAS international conference on Communications*, pages 135–140. World Scientific and Engineering Academy and Society (WSEAS), 2010.

[11] F. Huang, D. Liu, X. Tan, J. Wang, Y. Chen, and B. He. Explorations of the implementation of a parallel IDW interpolation algorithm in a Linux cluster-based parallel GIS. *Computers & Geosciences*, 37(4):426–434, 2011.

[12] A. Neskovic, N. Neskovic, and G. Paunovic. Modern approaches in modeling of mobile radio systems propagation environment. *Communications Surveys & Tutorials, IEEE*, 3(3):2–12, 2000.

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

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

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

[16] A. Osterman. 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, 2012.

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

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

[19] N. Shabbir, M.T. Sadiq, H. Kashif, and R. Ullah. Comparison of Radio Propagation Models for Long Term Evolution (LTE) Network. *arXiv preprint arXiv:1110.1519*, 2011.

[20] S. Shepler, M. Eisler, D. Robinson, B. Callaghan, R. Thurlow, D. Noveck, and C. Beame. Network file system (nfs) version 4 protocol. *Network*, 2003.

[21] Iana Siomina and Di Yuan. Minimum pilot power for service coverage in WCDMA networks. *Wireless Networks*, 14(3):393–402, June 2007.

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

[23] R. Stallman et al. GNU general public license. *Free Software Foundation, Inc., Tech. Rep*, 1991.

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

[25] A. Valcarce, G. De La Roche, Á. Jüttner, D. López-Pérez, and J. Zhang. Applying FDTD to the coverage prediction of WiMAX femtocells. *EURASIP Journal on Wireless Communications and Networking*, 2009:1–13, 2009. ISSN 1687-1472.