Urban Radio Network Planning for Mobile Phones

Patrice Calégari, Frédéric Guidec, Pierre Kuonen

URBAN RADIO NETWORK PLANNING FOR MOBILE PHONES

by Patrice Calégari, Frédéric Guidec, Pierre Kuonen, EPFL - DI - GRIP/LITH

La planification de réseaux radio pour la téléphonie mobile en milieu urbain est une nécessite de gros calculs, qu'il est préférable d'exécuter sur plusieurs processeurs en parallèle. D'une part, il faut simuler la propagation des ondes radio afin de prŽdire les zones qui peuvent être couvertes. Le programme ParFlow++ a été développé dans ce but. D'autre part, il faut choisir l'ensemble d'antennes qui couvre une ville au meilleur coût. Ce dernier point est un problème d'optimisation combinatoire d'une grande complexité. Nous le traitons avec un algorithme génétique, ainsi nommé pour ses similitudes avec des systèmes biologiques.

Urban radio network planning for mobile phones requires costly computation that are better to run on processors in parallel. First, radio wave propagation must be simulated in order to predict the area that can be covered by a Base Transceiver Station (BTS). The ParFlow++ piece of software was developed with this goal. Second, the set of BTSs that cover a city with the lowest cost must be found. The latter task is a hard combinatorial optimization problem, that we try to solve with a bio-inspired genetic algorithm.

RADIO NETWORK PLANNING

One of the issues telecommunication operators must face when deploying a cellular network in a city is the selection of good locations for Base Transceiver Stations (BTSs). The problem comes down to finding out the best possible sites for BTSs, while guaranteeing that all —or at least a given percentage of the surface of— the streets are covered, and that the global cost of the radio network is kept at a minimum. Assuming that a set of potential sites is available, our goal is to select the best subset of sites capable of satisfying the coverage requirements. The first step, that is, the computation for each BTS of the zone that it can cover, is achieved by a radio wave propagation simulation piece of software called ParFlow++. The second step, that is, the selection of the best BTS sites, is done by a bio-inspired piece of software called Paragene. These two parallel pieces of software are parts of the STORMS[1] project, which aims at the definition, the implementation, and the validation of a software tool to be used for the design and the planning of the future UMTS[2] network.

[1] STORMS (Software Tools for the Optimization of Resources in Mobile Systems) is an ACTS project funded by the EC and by the Swiss Government.

[2] UMTS: Universal Mobile Telecommunication System.

RADIO WAVE PROPAGATION SIMULATION

The ParFlow method

The ParFlow method was designed at the University of Geneva by Chopard, Luthi and Wagen [1]. It compares with the so-called Lattice Boltzmann Model, that describes a physical system in terms of the motion of fictitious microscopic particles over a lattice [2], and it permits the simulation of outdoor radio wave propagation in urban environment. As the ParFlow method operates on a 2D model of the terrain it is appropriate for simulating radio wave propagation when fixed antennas are placed below rooftops, as is the case in urban radio networks composed of micro-cells.

According to the Huygens principle, a wave front consists of a number of spherical wavelets emitted by secondary radiators. The ParFlow method is based on a discrete formulation of this principle. Space and time are represented in terms of finite elementary units Δr and Δt, related by the velocity of light C_0 [3]. Space is modeled by a grid with a mesh size of length Δr, and flow values are defined on the edges connecting neighboring grid points.

The flows entering a grid point at time t are scattered at time $t+\Delta t$ among the four neighboring points. For each grid point, outgoing flows at time t are a linear combination of incoming flows at that time, and an incoming flow at time t corresponds to the outgoing flow calculated on a neighboring grid point at time $t-\Delta t$ (see Figure 1).

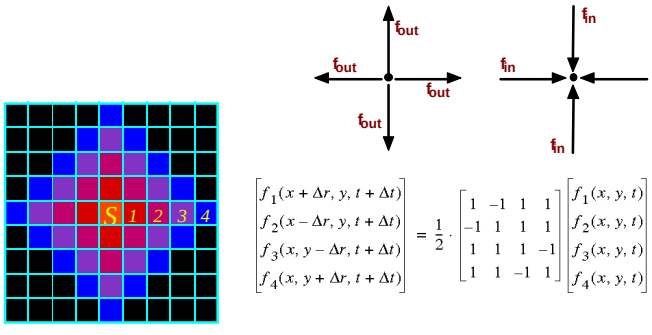

Fig. 1: Discretization of space and time in the ParFlow model

These rules apply for all grid points but those modeling the source (i.e., the BTS) and obstacles. The source point radiates a signal through its four outgoing flows, but it does not propagate incoming flows. Obstacles (typically, city

[3] $\Delta t = \Delta r / (C_0 . \sqrt{2})$

buildings) are modeled by two kinds of grid points: wall points, and indoor points (see Figure 2).

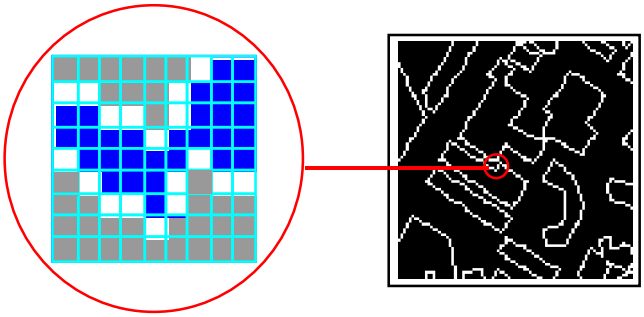

Fig. 2: Obstacles modelling in the ParFlow method. The input is a digital terrain model that permits to distinguish between outdoor points (blue), wall points (white), and indoor points (gray)

In the current version of the ParFlow model, it is assumed that radio waves do not penetrate buildings. As a consequence, indoor points are not involved in the computation: wall points are perfectly reflecting points that return any incident wave with opposite sign, and whose reflection coefficient and matrix elements can be modified in order to model different kinds of walls [1].

Irregular implementation in ParFlow++

ParFlow++ is an object-oriented, irregular implementation of the ParFlow method, targeted at distributed memory parallel platforms. Its main originality with respect to former implementations of the ParFlow method [1] is that it was implemented as an irregular application. Since the method does not allow for radio wave propagation through buildings, ParFlow++ does not model indoor points. Experience shows that this approach permits a significant reduction of memory space and of computational power (when modeling an urban area, buildings can represent up to 30 % of the surface considered). But such an approach inevitably leads to the creation and the management of an irregular data structure. As they provide powerful features to describe and to manipulate complex, irregular data structures, object-oriented languages are perfectly suited to an irregular implementation of the ParFlow method. This is the reason why ParFlow++ was implemented in C++. All kinds of non-indoor points are described in a hierarchy of C++ classes, whose instances are assembled at runtime so as to constitute an irregular structure that preserves the neighborhood relationships.

Data parallelism

A major advantage of the ParFlow method is that, although the calculations made during each iteration step are theoretically synchronous, updates of points require independent computation. The ParFlow algorithm is therefore a good candidate to parallel implementation.

The first parallel version of the ParFlow method was implemented by Chopard, Luthi and Wagen on Thinking Machine Corporation's CM-200, whose SIMD[4] architecture provides thousands of synchronous processors [1]. The ParFlow method can be readily and efficiently implemented on SIMD platforms, because it is possible to take advantage of the regular grid data structure and of the synchronous progress of the computation. However, the main disadvantage of such an implementation is its lack of scalability. An irregular data structure such as that discussed in the former paragraph can hardly be mapped on the synchronous regular architecture of the CM-200. The consequence is that, on such a platform, many processing units (those modeling indoor points) remain idle throughout the entire computation. Moreover, since each point of the grid must be allocated to one processing element, the size of the grid is constrained by the size of the parallel machine.

MIMD platforms are more versatile than SIMD platforms when it comes to implementing irregular applications. Actually, the ParFlow method has already been implemented on several MIMD platforms by Chopard, Luthi and Wagen [1]. Yet, every time it was implemented as a regular algorithm, in C or in Fortran. ParFlow++ contrasts sharply with these former implementations, not only because of its object-orientation, but also because it results from the first attempt to implement the ParFlow method for MIMD-DM platforms while operating on an irregular data structure with an irregular algorithm. Another characteristic of ParFlow++ is that it was developed so as to be easily portable on any kind of MIMD-DM platform.

Because of the large amount of outdoor points that must be considered in a simulation (typically, several thousands of outdoor points for a single city district), an appropriate policy must be chosen to allocate each point to a processing element of the target platform. Many partitioning policies can be considered for an irregular structure such as that of ParFlow++. Yet, exotic partitioning policies often require costly mechanisms for locating remote data, and for ensuring efficient data exchanges. For these reasons, and in order to obtain good load balancing, ParFlow++ relies on a simple, yet efficient static data distribution. The simulation zone is split in horizontal stripes, which are then allocated to processors based on a round-robin policy, as shown in Figure 3.

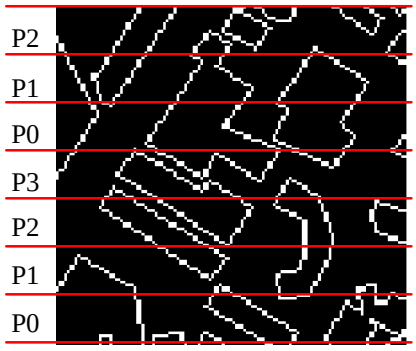

Fig. 3: Simulation zone partitioning and mapping

[4] SIMD: Single Instruction stream, Multiple Data stream.

In ParFlow++ each stripe basically consists of a structured collection of points. It also manages two `frontiers' internally. A frontier can be perceived as a collection of references to points that are either on the northern edge or on the southern edge of a partition. The behavior of frontier points differs slightly from that of other points, for they have to interact with `neighboring' points that are actually stored on remote processors. A frontier thus ensures the propagation of flows between a partition and one of its neighbors. Communications between neighboring partitions are based on calls to routines of the PVM library [3].

Experimental results and performances

ParFlow++ was compiled and tested on the Cray T3D of the Swiss Federal Institute of Technology[5]. We ran an 800 step radio wave propagation simulation on a 500x500 point zone modeling a 1 km^2 district of the city of Geneva. Figure 4 shows snapshots of the simulation. Each picture is a path-loss map.

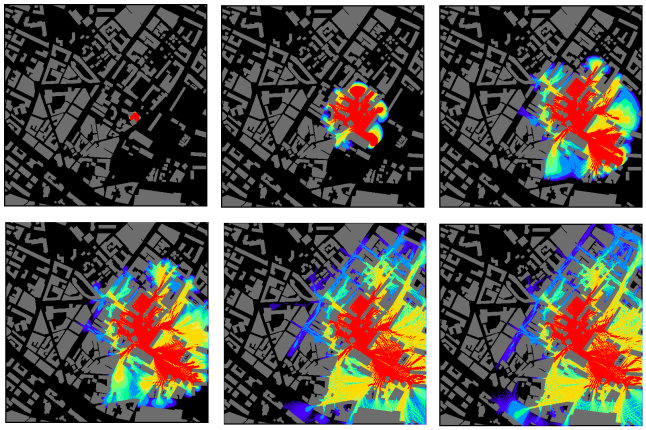

Fig. 4: Snapshots of a radio wave propagation simulation achieved with the ParFlow method. These pictures show how the wave propagates and covers a growing surface

On the Cray T3D we measured the speedups observed for various simulation zone sizes. Figures 5 and 6 confirm the scalability of the parallel implementation. The table in Figure 5 shows the actual computation times observed on each test case, and Figure 6 shows the speedups calculated from these times. (For the 1024x1024 and 2048x2048 zones, the simulation was not possible on a single processor because of memory limitation, so we had to estimate the sequential reference times).

[5] EPFL, CH-1015 Lausanne.

# procs	512 x 512	1024 x 1024	2048 x 2048
1	1366.04	-	-
2	641.46	-	-
4	363.50	2993.71	-
16	172.96	1382.78	-
16	105.58	739.14	5676.88
32	52.30	371.87	2891.31
64	27.85	209.14	1537.96
128	16.16	111.69	799.27
256	9.32	63.80	449.33

Fig. 5: Parallel simulation duration on the Cray T3D, for 512 x 512, 1024 x 1024, and 2048 x 2048 point simulation zones (times given in seconds)

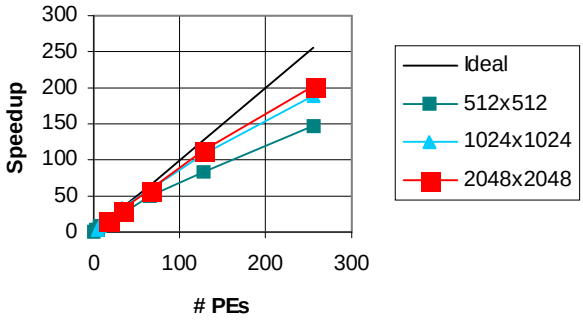

Fig. 6: Speedups observed on the Cray T3D, for 512 x 512, 1024 x 1024, and 2048 x 2048 point simulation zones

Covered cells computation

Once a propagation simulation is complete for a given BTS, the geographical area in which the coverage is good for mobile phone communication must be delimited. This area is called the *covered cell* of the BTS. A threshold value is determined based on different radio quality criteria (guessed traffic demand, etc.), and each grid point where the power of the received signal is bigger than the threshold value belongs to the covered cell. Such a covered cell is shown in Figure 7.

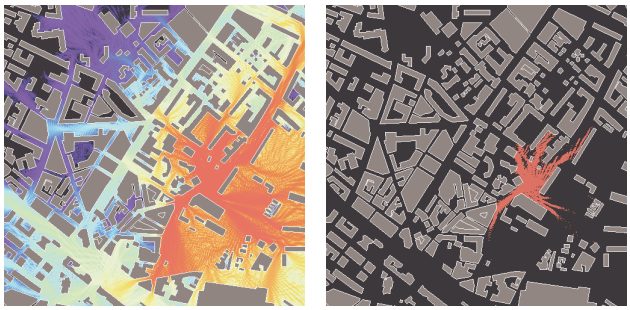

Fig. 7: Result of a radio wave propagation simulation on a district of the city of Geneva (left), and shape of the covered cell hence identified (right).

BTS SITING

Modelling

The aim of the first step was to generate the set of covered cells associated to a set of potential BTSs. These data will be used to find the best subset of BTSs that covers an area. In order to store the huge amount of information

about BTSs and covered cells, a data structure was developed. This structure connects a subset of BTSs to the *intercells* they can cover, where an *intercell* is a set of locations that are potentially covered by exactly the same BTSs. A simple example of such a structure is shown in Figure 8.

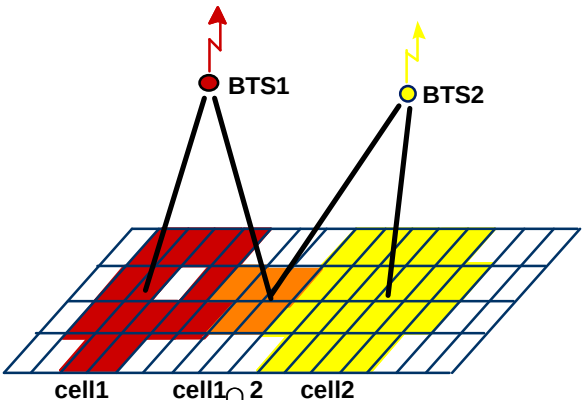

Fig. 8 The covered cells associated to the BTSs are discretized on a grid. Each BTS is connected to the intercells it covers.

Complexity

Definition: A set system $(X,\Re)$ is a set X, with a collection $\Re$ of subsets of X

Let us consider:

- X is the set of all intercells.
- $\Re$ is the set of all subsets R of X such that:
  $\bigcup_{x_i \in R} x_i = $ One covered cell

Definition: a set cover is a subcollection $C \subseteq \Re$, whose union is X, and the size of C is the number of sets in C [4].

With such a modelling, the problem of finding the minimum number of BTSs that cover an area comes down to finding a set cover of minimum size. Unfortunately, finding a set cover of minimum size is NP-complete [5]. Thus unless P=NP, if we want a polynomial time algorithm, we must use an approximation algorithm and look for possibly sub-optimal —yet satisfactory— solutions. To achieve this goal, we investigate several approaches, which are said to be 'bio-inspired' because they use heuristics that have some analogies with natural or social systems [6].

The genetic approach

To date, we most especially focus on the so-called genetic approach. This work is done in cooperation with the LEOPARD[6] project. A genetic algorithm is a population-based algorithm, which means that its state at any time is a population of candidate solutions. A population is a set of individuals that represent candidate solutions for a given problem, and that are generally encoded as chromosome-like bit strings. The algorithm uses selection and recombination operators to generate new sample points in a search space [7]. The bit strings we consider represent the whole set of possible BTS locations. Whether a location is actually selected in a potential solution depends on the value of the corresponding entry in the bit string. Figure 9 shows how a possible solution to our problem is encoded as the bit string of an individual.

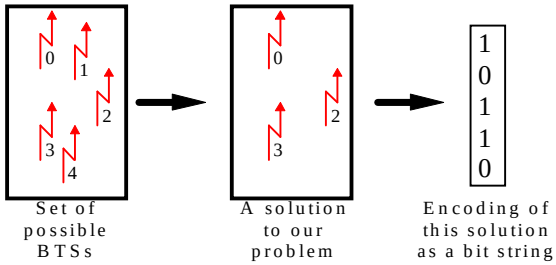

Fig. 9 Encoding of a candidate solution in an individual bit string.

The first step in the execution of a genetic algorithm consists in the creation of an initial population. In our implementation, the initial population is built randomly. Each iteration step of the algorithm on the population (see Figure 10) is called a generation. During the execution of a genetic algorithm, a generation can be thought of as a two stage process. First, selection is applied to the current population to create an intermediate population. Then, crossover and mutation operators are applied to the intermediate population to create the next population. The execution terminates when a satisfactory individual has been produced or when a predefined number of generations has been run through. Figure 10 describes the different steps in the evolution of a population.

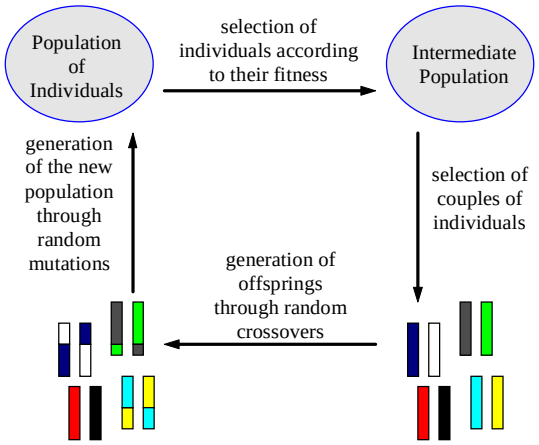

Fig. 10 Global overview of a genetic algorithm. One iteration of the loop is called a generation.

The selection is achieved based on the fitness value associated with each individual, which may be perceived as an allocation of reproductive opportunities: the higher the fitness value of an individual, the likelier it is to ` reproduce' . The function returning the fitness value is:

[6] The LEOPARD project (Parallel population-based methods for combinatorial optimisation) is funded by the Swiss National Science Foundation (project # 21-45070.95/1)

$$fitness(individual) = \frac{CoverRate^{\alpha}}{\text{number of selected BTSs}}$$

where *CoverRate* is the percentage of locations covered by the selected BTSs (in relation to the number of locations covered by the initial set of BTSs). The parameter α can be tuned to favor the cover rate with respect to the number of transmitters. Figure 11 shows the influence of the parameter α on the characteristics of the results. It shows that for $\alpha<1.5$ the coverage is unacceptable, for $1.5<\alpha<4$ the coverage grows quickly while the number of BTS grows slowly, and for $4<\alpha<10$ the coverage remains almost constant while the number of BTSs keeps growing. A good value for α is thus 4 (this value will be used in the remainder of this article).

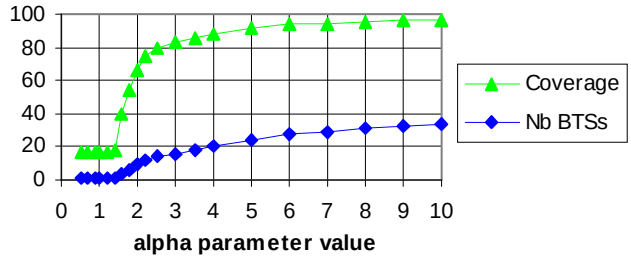

Fig. 11: Average number of BTSs and average coverage of solutions found with different values of the parameter α.

The crossover operator we use achieves a 'one-point-crossover'. It splits the bit strings of two randomly selected individuals at a randomly chosen position, and it then recombines the bit strings by exchanging their ends, hence producing two new bit strings. In the current implementation of Paragene, each couple of individuals taken from the intermediate population has a probability of 0.9 to achieve a crossover. In the case no crossover is performed, the couple of individuals is put back into the new population without changes. Our mutation operator simply flips the value of a randomly chosen bit of the bit string with a probability of 0.6[7]. The execution terminates after a predefined number of generations.

First experimental results

Experiments were conducted for the district of Geneva introduced in the first part of this article. Starting with 99 user-provided potential BTS locations, 99 cells were computed with ParFlow++. Figure 12 shows the coverage that would be obtained if these 99 BTSs were all selected in the final radio network. Locations that are covered exactly once are colored in blue, those covered twice are colored in green, those covered three times are colored in red and those covered four times and more are colored in pink. It can be noticed that the gray locations cannot be covered by any of the user-provided potential BTSs. As a lot of points are covered many times, it is clear that some of the proposed BTSs are useless. Moreover, too many transmitters covering a same zone can lead to radio interference. On the other hand it can also be useful to cover a same location with two different BTSs, as this permits roaming: when a user moves from one cell to another.

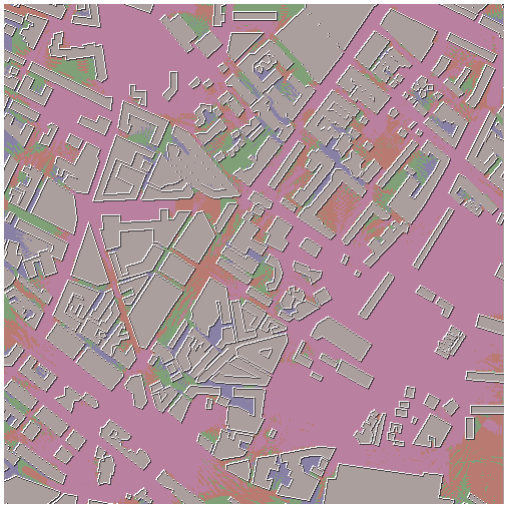

Fig.12 Coverage obtained with the 99 potential BTSs. Pink locations are covered 4 or more times.

Figure 13 shows a solution returned by Paragene after 320 generations of a 160 individual population. The algorithm meets our demand by returning solutions with large coverage and few BTSs. It has however two major shortcomings. First, it is too slow (3mn31s for this solution) to be used interactively, as required by telecommunication operators. Second, the solutions still contain quite too many BTSs in the operators opinion.

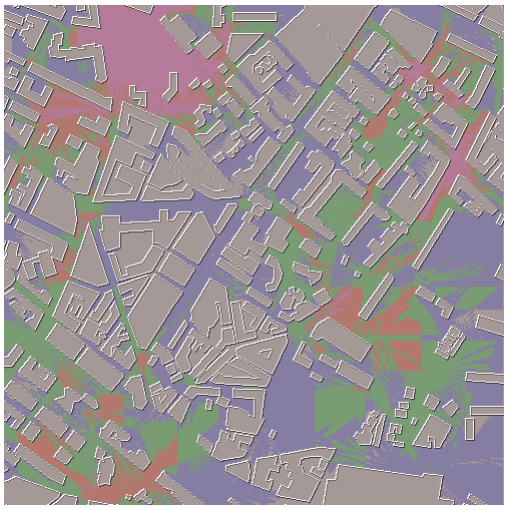

Fig. 13 A solution obtained, with a 160 individual population, after 320 generations. The 36 selected BTSs cover 94.09% of the initially covered surface.

[7] These probability rates have been carefully chosen, after a comprehensive study of their impact on the quality of the results returned by Paragene.

Island concept and parallelism

In order to improve the quality of the results obtained with the genetic algorithms, the population can be distributed in subpopulations that evolve independently. These subpopulations, called islands, increase the chances for the algorithm to find good solutions by having many genetic algorithms competing, each of them operating on small populations. In order to let some islands benefit from the information found by others, some individuals are allowed to move from an island to another. This mechanism is called migration.

In order to speedup the execution, the parallelisation of the island-based algorithm was investigated. The fact that the islands evolve almost independently suggests that the algorithm is parallel in essence. The computation time can thus be decreased by distributing the islands on several processors. This MIMD-DM[8] approach enables to run the program on clusters of workstations, which are more commonly available and cheaper than parallel supercomputers. The islands are virtually positioned on an oriented ring, and migrations are only allowed along that ring (see Figure 14).

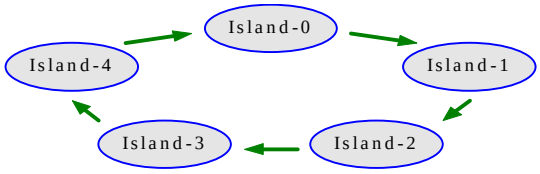

Fig. 14 Distribution of islands on an oriented ring. Migrations are only allowed along the oriented ring.

This topology was chosen so as to minimize the amount of migrations, and thus to minimize the communication overhead due to migrations between remote islands. Every time a new generation is computed, a copy of the best individual (that with the best fitness value) ever met by each island is sent to the next island on the ring. Each island thus receives a new individual that replaces one of its individual selected randomly. The only requirement is that the number of islands be greater or equal to the number of available processors because each processor must be in charge of at least one island (see Figure 15). Our parallel implementation of Paragene [8] was designed using object-oriented techniques, and it was written in C++. It uses the PVM library [3] for inter-processor communications.

[8] MIMD: Multiple Instruction stream Multiple Data stream - Distributed Memory

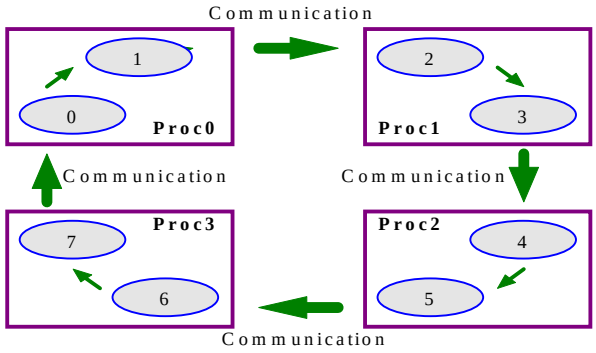

Fig. 15 Parallel version of our island-based genetic algorithm.

Quality of the results and speedup

Figure 16 shows a solution returned by our island-based genetic algorithm after 320 generations when running on 40 workstations that were in charge of one island each. The total population size (40 * 4 = 160 individuals) was the same as that used for computing the result shown in Figure 13. This new result is of better quality: its associated fitness value is 0.03051, whereas that of the previous solution was only 0.02177, and it was found in 10 seconds (against 3mn31s. for the previous solution).

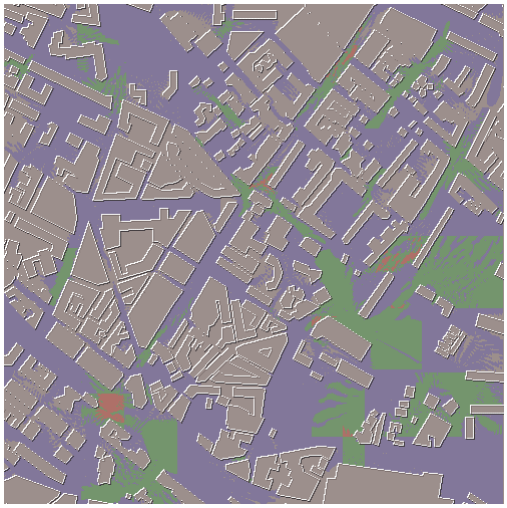

Fig. 16: A solution obtained with 40 islands of 4 individuals, after 320 generations. The 19 selected BTSs cover 87.26% of the initial covered surface.

Figure 17 shows the speedup observed on a network of 40 workstations, when running 320 generations over 40 islands of 4 individuals each. It can be noticed that the speedup is almost linear for less than 10 workstations, and that the efficiency on 40 workstations amounts to 52.75%, which is satisfying.

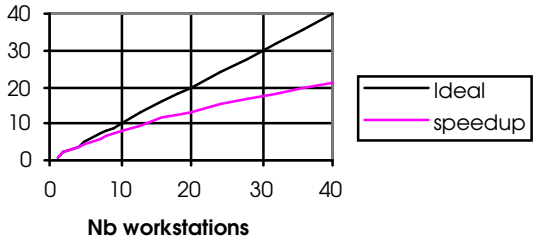

Fig. 17 Speedup of the parallel algorithm observed when running 320 generations over 40 islands of 4 individuals each.

CONCLUSION

In this article we have presented a cellular automaton approach to simulate radio wave propagation in urban environment on a MIMD-DM computer. A very good speedup was achieved by using an original static load balancing technique. The development was realized using object-oriented techniques, leading to a strict distinction between what belongs to the physical problem (radio wave propagation) and what belongs to the computing model (cellular automaton). In the future we intend to reuse this computing model to simulate other physical phenomena such as flowing fluids.

Using radio wave propagation predictions computed by the former algorithm, we tackled the problem of selecting optimal BTS sites for urban mobile phone networks. A parallel genetic algorithm was implemented on a network of up to 40 workstations. Taking into account that fact the links between the workstations are rather slow the efficiency of the parallel execution measured on 40 workstations (52.75%) is good. It was obtained with an island-based genetic algorithm which tries to minimize the communication between the workstations.

Again, the development was realized using object-oriented techniques, because we intend to reuse this island-based parallel genetic algorithm to solve other difficult combinatorial optimization problems.

BIBLIOGRAPHY

[1] P. O. Luthi, B. Chopard, and J.-F. Wagen. Wave Propagation in Urban Micro-Cells: a Massively Parallel Approach using the TLM Method. In Proceedings of PARA'95, Workshop on Applied Parallel Scientific Computing, Copenhagen, Aug. 1995. Also in COST 231 TD(95) 33.

[2] R. Benzi, S. Succi, and M. Vergassola. The Lattice Boltzmann Equation: Theory and Applications. Physics Reports, 222(3):145-197, 1992.

[3] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V. ‘‘PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing.’’, The MIT Press, 1994.

[4] H. Brönnimann and M. T. Goodrich. Almost Optimal Set Covers in Finite VC-Dimension. In Proceedings of the 10th Annual ACM Symposium of Computational Geometry, pages 293-302, 1994.

[5] M. Garey, D.S. Johnson, ‘‘Computers and intractability: a guide to the theory of NP-Completeness’’, Freeman, 1979, IBSN 0-7167-1045-5.

[6] A. Colorni et al., ‘‘Heuristics from nature for hard combinatorial optimization problems’’, Dipartimento di Elettronica - Politecnico di Milano, Report no 93025.

[7] J. Holland, ‘‘Adaptation in natural and artificial systems’’, University of Michigan Press, 1975.

[8] Calégari P., Guidec F., Kuonen P, Chamaret B., Josselin S., Wagner D. ‘‘Radio Network Planning with Combinatorial Optimization Algorithms’’, in Proceedings of the ACTS Mobile Telecommunications Summit 96, volume 2, pages 707-713, Nov. 96.